VCU Bioinformatics and Bioengineering Summer Institute

Making a model of DNA sequences

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 genes

You are now familiar with using the program Hamlet.pl and have explored how what the Markov model it creates when it analyzes 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 whether a DNA sequence does or does not encode a gene. 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 recognizable about gene sequences? We have two steps to consider in going from Hamlet.pl to a way of deciding on the genehood of a DNA sequence:

1. How do we modify Hamlet.pl so that it will use a set of known gene sequences as the raw material to build the Markov model? *(Today's notes)*
2. How do we use the Markov model produced by Hamlet.pl on unknown sequences to assess the likelihood that those sequences are parts of genes? *(Tomorrow's notes)*

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.

SQ1. Try running Hamlet.pl using 6803_training_set.nt as the input file (you can download this file from our web site). What happens?

SQ2. What's the problem? Compare an input file that works (e.g. HamletSpeech.txt) to one that doesn't (e.g. 6803_training_set.nt).

It would be nice if we could just run some genes through Hamlet.pl, but this is not to be. Examine both HamletSpeech.txt and 6803_training_set.nt and you'll see why (the error message is informative also). Each textual unit in HamletSpeech.txt is enclosed in brackets ({...}), while the genes in 6803_training_set.nt are given in FastA format.

SQ3. What is FastA format? Don't run to the web. Just look at the file 6803_training_st.nt and deduce its format.

You ***could*** solve this problem by altering 6803_training_set.nt, as you did before with a file of French or German text, but now is not the time for quick fixes. Your intent is to change Hamlet.pl so that it runs on DNA sequences.

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

II.A. Teach your program to read in the training set

You might think of other ways to go about modifying the program than what I proposed (e.g. updating the Markov model immediately as you read in a sequence), but I chose this way to exploit some code that already exists, a Perl module called FastA_module.pm that knows how to read in a file of FastA-formatted genes. A module is a self-contained set of programs that can be incorporated into your own program.

To make use of a module, it is necessary to:

- Download the module to your space (FastA_module.pm is available on our web site)
- Add to the Libraries section of your program the statement:

  `use` *name-of-module*

  (*name-of-module* should not include ".pm")
- Call one or more functions included in the module

The first two tasks are straightforward, but it is not clear what function would be appropriate to call. Time to look at the directions. Open FastA_module.pm so that you can examine its documentation.

SQ4. Download FastA_module.pm and add the appropriate use statement to the Libraries section of Hamlet.pl. When you save Hamlet.pl, give it a new name, perhaps something descriptive like Make_markov.pl, to preserve the old program.

It's generally a good idea to test a program after making any change to it, before moving on to the next change. That way any failure in the program can be attributed to the minmal change you just made.

SQ5. Run the new program. What happens?

Adding a module should have no effect on the program (until you call a function). So, you should have gotten the same error message Hamlet.pl gave you before.

If you instead got an error message like

```
Can't locate ...
```

Then you either misspelled the name of the module or did not copy it to the same directory where Make_markov.pl lives. If you got an error message like

```
syntax error at make_markov.pl line 20, near "use ..."
```

then you may have erroneously left "`.pm`*" as part of the name of the module. The* `use` *command presumes that the module has a .pm extension and doesn't allow you to specify it.*

SQ6. Which of the four subroutines described within FastA_module.pm seem appropriate for the needs of the program you are building from Hamlet.pl?

SQ7. Add a your growing program a call to an appropriate function within FastA_module.pm, one that reads all the sequences within 6803_training_set.nt and

saves in a variable the number of sequences read. Add another statement to test whether the function call works. The statement should print the number of sequences read. Run the program.

I would put the following statements in the FILES section of the program, since it is used to read the input file (but good arguments could be made for putting them in the MAIN PROGRAM section):

```
$number_of_sequences = Read_FastA_sequences ("6803_training_set.nt");
print $number_of_sequences, $LF;
```

If you get the following error:

```
Global symbol "$number_of_sequences" requires explicit package...
```

then you need to recall that strict Perl demands that every variable first be declared with `my` *variable. Add*

```
my $number_of_sequences;
```

to the Variables section of the program and include a brief comment as to the meaning of the variable.

If the output of the program begins with a number (plausibly the number of sequences in 6803_training_set.nt), then all seems to be OK.

SQ8. Test FastA_module.pm further by using one of its functions to access and print the first sequence within the training set. Compare what is printed with the actual first sequence within the training set.

The documentation of FastA_module.pm says that Get_sequence_info gives you the header and the sequence of one of the sequences read by Read_FastA_sequences. Accessing the first sequence and saving the returned information in two variables could be done this way:

```
($header, $sequence) = Get_sequence_info (1);
print $header, $LF, $sequence, $LF;
```

(Again, be sensitive to declaring and describing in the Variables section any new variables you use). Since Get_sequence_info returns ***two*** pieces of information, two variables need to be in place to capture what the function returns.

Once you're satisfied that the training set has been read in correctly, remove the test lines (accessing and printing the first sequence) from the program.

II.B. Teach your program to add each sequence to the Markov model

Take a look at the Main Program of Make_markov.pl (still unchanged from Hamlet.pl) and notice how the program reads in text. It reads a line from the input file, and so long as that line exists (the end of the file hasn't been reached), it does various things aimed at analyzing the line. One line of text is analogous to one DNA sequence, but you intend to get those DNA sequences not from a file but from the function Get_sequence_info. Your strategy is this:

- Consider in turn each sequence read in by Read_FastA_sequences
- Analyze each sequence the same way that Hamlet analyzes each line of text

"Consider in turn each..." generally means a loop, a block of computer code that cycles through the same instructions repeatedly. Perl provides many ways of doing this. Here's one:

```
foreach my $sequence_index (1 .. $number_of_sequences) {
```
(same instructions as in Hamlet.pl's `while` *loop)* }

This means that the variable $sequence_index will be given the value 1, then 2, then 3, and so forth, up until whatever value $number_of_sequences contains. For each value, Perl will run through the all the instructions between { and }, i.e. the contents of the loop.

SQ9. Incorporate this loop into Make_markov.pl and run the resulting program. Delete the line beginning `while` (which began the old loop).

If all goes splendidly, you'll still get the same error message you've been getting for the past four pages. Clearly, the time has come to tell the program to stop worrying about {}. How?

SQ10. Where in the program does it specify that lines must begin and end with brackets?
NB: You don't have to understand hardly any Perl to answer this question!

Obviously, Hamlet had already read in a line before complaining about it, so the critical instruction must be somewhere within the main loop given in the Main Program. That leaves only seven possible statements that might be responsible, the seven within the loop:

```
$line = <INPUT_FILE>;
while (defined $line) {          # Analyze text
1.  @letters = Break_up($line);  # Make sure text conforms to proper format
                                 # Also split line into individual letters
2.  $size_of_line = Size(@letters);
3.  $last_starting_letter = $size_of_line - 1 - $order;
                                 # Move window of size order+1 over line
                                 #   incrementing every window encountered
4.  foreach $starting_letter (0 .. $last_starting_letter) {
5.     @window = @letters[$starting_letter .. $starting_letter+$order];
6.     Update_entry(@window);
    }
7.  $line = <INPUT_FILE>;
}
```

Lines 3 and 5 are unappealing as they contain only variables (tagged by $), arrays (tagged by @), and constants. The only possibilities are lines that contain functions (tagged by nothing). And the standout amongst those is line 1, because the documentation says it concerns itself with format. So visit that function. You should find statements that look very much like they deal with brackets. Delete those lines and run the program.

If you and I are running the equivalent program, then you're pleased that there are no more errors, but you may be getting bored staring at:

```
Please wait a few seconds for Hamlet to be created...

UPDATE COMPLETE
```

You're waiting for the program to produce a very long pseudosequence. Put it out of its misery by breaking out of the program with Ctrl-C if you're in a DOS window or Run/Stop if your in

PerlEditor). In fact, now is the time to accept that this program is no longer Hamlet and chop out the portion of the program that produces new text.

SQ11. Where in the program is new pseudotext made? Get rid of that section in the Main Program and rerun the program. If you want, you can also chop out the subroutines for producing text (but be careful you don't chop out too much).

The program should now work without visible error, although it may take several seconds to finish (the training set is BIG!). You should now be able to use Display_hash.pl to examine the model produced.

SQ12. What do you expect the model to look like? What kind of entries will it have? What will be the first entries (those preceded by □…) (what did □ mean again?). Be sure you base your answer on a good acquaintance with the input file. *Look at the raw data and know what it is when you write or modify a program!*

SQ12. What do you see in the model? Why?

You shouldn't be too surprised that the program isn't working perfectly yet. You spent all that time testing whether Read_FastA_sequences worked (see SQ8), but those test lines are gone. You haven't actually USED any FastA sequences. And just as bad, the program is still reading the file the old way: line-by-line.

SQ13: Put in the loop a statement that accesses one sequence. It will be almost the same as the test statement you used in SQ8. Also, get rid of the three statements that are thus rendered obsolete, two because the program should no longer be reading from an input file and one because you therefore no longer need to open the file.

Now examine the resulting model by Display_hash.pl. If you put in the Get_sequence_info line correctly, then the model should be ALMOST what you expect. Examining the training set told you that all the sequences begin with ATG, GTG, or TTG, so you that should inform your expectations of the model. Unfortunately, TG yes, ATG no.

> *This error is rather subtle. Hamlet.pl read a line and lopped off the initial character, a "{". Now the first character is part of the sequence, and the program should NOT lop it off. To fix this, go to the subroutine Break_up and delete the line that (per its annotation) removes the first character.*

SQ14: It would also make sense (just for the sake of intelligibility) to change references to "line" to "sequence", which you can do through a global replace (but look at each change before confirming).

SQ15. Examine the model produced by the working program.

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

b. What is the ratio of the frequency of the most common tetranucleotide to the frequency of the least common in the training set? *Natural DNA is NOT well approximated by a random sequence!*