Chapter 6 Utilisation of PERL and Statistics for Analysis of Bioinformation

6.1 Introduction

PERL is a programming language. After survey of various computer languages we found that PER is best for analysing bioinformation. PERL has several features for working with bioinformation. It can install on Computer. It is available at free of cost and runs on all the operating systems like Unix, Linux, Macintosh and Windows. The web site that is used for downloading PERL is http://www.perl.com/

The main aim to select this language can be justified using following illustration.

6.2 Illustration

Suppose you discovered a segment of DNA sequence and suspect it may hold a clue to the cancer. After sequencing the DNA you perform a search of Genbank and other data sources using web based sequence alignment tools such as BLAST. You find a few related sequences but you do not get a direct match or any information that indicates a link to the cancer so picture will not become clear. You know that the
public genetic databases are growing daily and rapidly. You would like to perform your searches every day and comparing the results to the previous searches to check whether anything new appears in the databases. This process requires too much time.

But using PERL that automatically performs a daily BLAST search of Genbank for your DNA sequence and match the results with the previous day's results and sends you confirmation if there has been any change. This technique is useful and it can be worked for other sequences also. This facility will save our time. So it is suggested and recommended for research work.

6.3 Working with sample sequence

Using PERL we can perform following task with bioinformation and sequences. It will used for disease probability for drug design.

- Transcribe DNA to RNA
- Concatenate sequences
- Make the reverse complement of sequences
- Read sequence data from files

We take some sample sequences and apply PERL to that for analysis.

Sample Program to Store a DNA Sequence
Following program stores DNA in a variable and prints it to the screen. The DNA is written in form of string made of the letters A, C, G, and T and variable $DNA used for that.

Example: Putting DNA into the computer

#!/usr/bin/perl

# Storing DNA in a variable and printing it out
# First we store the DNA in a variable called $DNA
$DNA = 'ACGGAGGACGGGAAAATTACTACGGCATTAGC';
# Next, we print the DNA onto the screen
print $DNA;
exit;

Concatenating DNA sequence

Concatenation of DNA sequence is attaching two sequences. This is common task for research in sequence. For example when a clone is inserted into a cell vector or when splicing exons together during the expression of a gene.

Example: Concatenating DNA

#!/usr/bin/perl

# Concatenating DNA
# Store two DNA fragments into two variables called $DNA1 and $DNA2
$DNA1 = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
$DNA2 = 'ATAGTGCCGTGAGAGTGATGTAGTA';

# Print the DNA onto the screen
print " Here are the original two DNA fragments:\n\n";
print $DNA1, "\n";
print $DNA2, "\n\n";
# Concatenate the DNA fragments into a third variable and print them
# Using "string interpolation"
$DNA3 = "$DNA1$DNA2";
print "The concatenation of the first two fragments:\n\n";
print "$DNA3\n\n";
exit;

Here are the original two DNA fragments:
ACGGGAGGACGGGAAAATTACTACGGCATTAGC
ATAGTGCCGTGAGAGTGATGTAGTA

Here is the concatenation of the first two fragments:
ACGGGAGGACGGGAAAATTACTACGGCATTAGC
ATAGTGCCGTGAGAGTGATGTAGTA
CGTGAGAGTGATGTAGTA
Transcription of DNA to RNA

It transcribes DNA to RNA. In cell this transcription of DNA to RNA is the process of the workings of a delicate, complex, and error correcting molecular machinery. When DNA is transcribed to RNA, all the T's are changed to U's.

Example: Transcribing DNA into RNA

```perl
#!/usr/bin/perl

# Transcribing DNA into RNA

# The DNA
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';

# Print the DNA onto the screen
print "Here is the starting DNA:\n\n";
print "$DNA\n\n";

# Transcribe the DNA to RNA by substituting all T's with U's.
$RNA = $DNA;
$RNA =~ s/T/U/g;

# Print the RNA onto the screen
print "Here is the result of transcribing the DNA to RNA:\n\n";
print "$RNA\n";

# Exit the program.
exit;
```

Output:
Here is the starting DNA:

ACGGGAGGACGGGAAAAATTACTACGCCATTAGC

Here is the result of transcribing the DNA to RNA:

ACGGGAGGACGGGAAAAUUACUACGGCAUUAAGC

Calculating the Reverse Complement in Perl

A DNA polymer is composed of nucleotides. Given the close relationship between the two strands of DNA in a double helix it turns out given one strand prints out the other. Such a calculation is an important part of many applications. When searching a database with some query DNA it is common to automatically search for the reverse complement of the query because some time we have the opposite strand of some known gene. So, we can made prediction of disease or functionality of the living things.

Example: Calculating the reverse complement of a strand of DNA

#!/usr/bin/perl -w

# Calculating the reverse complement of a strand of DNA

# The DNA

$DNA = 'ACGGGAGGACGGGAAAAATTACTACGCCATTAGC';

# Print the DNA onto the screen

print "Here is the starting DNA:\n\n";
print "$DNA\n\n";
$revcom = reverse $DNA;

# Next substitute all bases by their complements,
# A->T, T->A, G->C, C->G
#
$revcom =~ s/A/T/g;
$revcom =~ s/T/A/g;
$revcom =~ s/G/C/g;
$revcom =~ s/C/G/g;

# Print the reverse complement DNA onto the screen
print "Here is the reverse complement DNA:\n\n";
$revcom = reverse $DNA;
$revcom =~ tr/ACGTacgt/TGCAtgca/;
# Print the reverse complement DNA onto the screen
print "Here is the reverse complement DNA:\n\n";
print "$revcom\n";
exit;

Here's what the output of Example 4 should look like on your
screen:

Here is the reverse complement DNA:

GCTAATGCCGTAGTAATTTTCCCGTCTCCCGT
Example: Reading protein sequence data from a file

#!/usr/bin/perl -w

# Reading protein sequence data from a file

# The filename of the file containing the protein sequence data
$proteinfilename = 'NM_021964fragment.pep';

# First we have to "open" the file, and associate
# a "filehandle" with it. We choose the filehandle
# PROTEINFILE for readability.

open(PROTEINFILE, $proteinfilename);

# Now we do the actual reading of the protein sequence data from the
# file,
# by using the angle brackets < and > to get the input from the
# filehandle. We store the data into our variable $protein.

$protein = <PROTEINFILE>;

# Now that we've got our data, we can close the file.

close PROTEINFILE;

# Print the protein onto the screen

print "Here is the protein:\n\n";

print $protein;
exit;

Here's the output of Example:

Here is the protein:

MNIDDKLEGLFLKCGIDEMQSSRTMVVMGGVSGQSTVSGELQD

#In a loop, ask the user for a motif, search for the motif,
#and report if it was found.
#Exit if no motif is entered.

do {
    print "Enter a motif to search for: "; $motif = <STDIN>;
    #Remove the newline at the end of $motif chomp $motif;
    #Look for the motif
    if ( $protein =~ /$motif/ ) { print "I found it!\n
";
    } else {
        print "I couldn't find it.\n
";
    }
    #exit on an empty user input } until ( $motif =~ /^s*$/ );
    #exit the program
Example: #!/usr/bin/perl -w

# Example Counting the number of G's in some DNA on the command line

use strict;

# Collect the DNA from the arguments on the command line when the user calls the program.

# If no arguments are given, print a USAGE statement and exit.
#$0 is a special variable that has the name of the program.
my($USAGE) = "$0 DNA\n\n";

# @ARGV is an array containing all command-line arguments.
#

# If it is empty, the test will fail and the print USAGE and exit statements will be called.
unless(@ARGV) { print $USAGE;
    exit;
}

# Read in the DNA from the argument on the command line.
my($dna) = $ARGV[0];

# Call the subroutine that does the real work, and collect the result.
my($num_of_Gs) = countG ($dna);
Report the result and exit.

print "The DNA $dna has $num_of_Gs G's in it!

# Subroutines for Example 2-1

sub countG {
  # return a count of the number of G's in the argument $dna
  # initialize arguments and variables
  my($dna) = @_;  
  my($count) = 0;
  # Use the tr on the regular expression for counting nucleotides in DNA
  $count = ($dna =~ tr/Gg//); return $count;
}

Example

#!/usr/bin/perl -w

# Example 2-2 Mutate DNA
# using a random number generator to randomly select bases to mutate
use strict;
use warnings;

# Declare the variables
The DNA is chosen to make it easy to see mutations:

```perl
my $DNA = 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
```

$i is a common name for a counter variable, short for "integer"

```perl
my $i;
my $mutant;
```

Seed the random number generator.

```perl
# time|$$ combines the current time with the current process id srand(time|$$);
```

```perl
$mutant = mutate($DNA); print "\nMutate DNA\n\n";
```

```perl
print "\nHere is the original DNA:\n\n"; print "$DNA\n";
```

```perl
print "\nHere is the mutant DNA:\n\n"; print "$mutant\n";
```

Let's put it in a loop and watch that bad boy accumulate mutations:

```perl
# Let's put it in a loop and watch that bad boy accumulate mutations:
for ($i=0 ; $i < 10 ; ++$i) { $mutant = mutate($mutant); print "$mutant\n";
}
```

exit;
Example

#!/usr/bin/perl -w

# Example : Translate DNA into protein

use lib '../ModLib/'; use strict;
use warnings;

use BeginPerlBioinfo; # This does not require the '.pm' in the 'use' command

# Initialize variables

my $dna = 'CGACGTCTTCGTACGGGACTAGCTCGTGTCGGTCGC'; my $protein = ''; my $codon;

# Translate each three-base codon into an amino acid, and append to a protein
for(my $i=0; $i < (length($dna) - 2) ; $i += 3) {
    $codon = substr($dna,$i,3); $protein .= codon2aa($codon);
}

print "I translated the DNA
$dna

into the

$protein

";
exit;
6.4 Computing Facility for Statistical Analysis for Bioinformation

PSPP is a statistical analysis tool and it is free and kind of open source software. So it is freely available and researcher can download it for managing bioinformation. There is no limit for variables and no need to purchase any license. We can perform linear regression and other statistical methods in PSPP.

In this chapter we present the study of analysis of sequences of DNA or amino acids using computer based statistical methods.

Bioinformation and data we get as a DNA sequence like a sequence of A, C, G and T from a certain organism or a sequence of amino acids that represents a protein synthesized by an organism. Many of this data are publicly available and searchable database called GenBank.

6.5 Single DNA Sequence Analysis

DNA analyses will have to do with analysing a single DNA sequence. Such data can represent a gene or a non-coding region in an organism. For example here is the DNA sequence from GenBank is given
ttatgctttcgaagactgcatgggaacc

This sequence consists of 30 bases.

So we analyses that there is four bases are equally represented in the DNA sequence or not.

For the DNA sequence above, the relative proportions of the bases are as under:

<table>
<thead>
<tr>
<th>Base</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>23.33</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>20.00</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>23.33</td>
</tr>
<tr>
<td>T</td>
<td>10</td>
<td>33.33</td>
</tr>
</tbody>
</table>

This suggests that there are more T nucleotides and fewer C nucleotides than the 25% that would have been expected if there were an equal distribution. So we can conclude that base T is higher in this sequence.

This situation suggests in these genes the probability of T is high as compared to other bases.

6.7 Measures of Central Tendency in Bioinformation
The most important descriptive statistics for central tendency are the mean and the median. Thus the sample mean is simply the average of the n data values. Since it is the sum of all data values divided by the sample size a few extreme data values may largely influence its size. In other words, the mean is not robust against outliers.

The median is defined as the second quartile or the 50th percentile. When the data are symmetrically distributed around the mean then the mean and the median are equal. Since extreme data values do not influence the size of the median it is very robust against outliers.

Robustness is important in bioinformatics because data are frequently contaminated by extreme or otherwise influential data values.

5.8 Measures of Spread in Bioinformation

The most important measures of spread are the standard deviation, the interquartile range and the median absolute deviation. It is the average of the squared differences between the data values and the sample mean. The sample standard deviation is the square root of the sample variance and may be interpreted as the distance of the data values to the mean. The variance and the standard deviation are not robust against outliers.