Bioinformática Estructural

Amaranta Manrique de Lara y Ramirez, Valeria Erendira Mateo Estrada

4 de marzo de 2016

Tarea 2: Predicción de promotores

Este módulo consiste en familiarizarse con el uso de modelos para la predicción de promotores a través de un enfoque estructural.

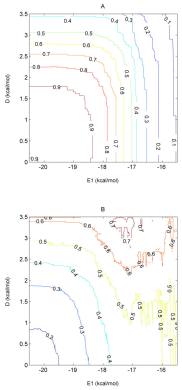
Para esta prática contamos con coordenadas de una colección de open reading frames y se pretenden encontrar las posiciones de sus secuencias promotoras.

Las secuencias para analizar se tomaron del fichero K12_400_50_sites, con coordenadas -400,+50 de ORFs de *Escherichia coli*.

2.1) Completar el código fuente del programa 1.1 para implementar el predictor de Kanhere y Bansal, justificando los valores de cutoff1 y cutoff2 de acuerdo con las figuras de su artículo.

El algoritmo de Kanhere y Bansal, desarrollado en 2005, que se ha utilizado para resolver problemas de búsqueda de promotores según su estabilidad molecular en la secuencia del DNA.

Se completó el código fuente utilizando el lenguaje de programación Perl. Los valores de cutoff se establecieron según la Figura 4 del artículo, en la que se grafican los niveles de sensibilidad (A) y precisión (B). Los valores se escogieron para mantener los rangos más altos de estos dos criterios (de amarillo a café).



Así se definieron como valores: $D > 3.4$ y $E1 > -16.7$.

A continuación se presenta el código completado:

```
#!/usr/bin/perl -w
# amanriqu_vmateo_promotores Amaranta Manrique de Lara y Ramirez Valeria Erendira
# Mateo Estrada Nearest Neighbor dG calculator
use strict;
#Subrutina para buscar el complemento de una secuencia
sub complement{ $_[0] =~ tr/ATGC/TACG/; return $_[0]}
# global variables
my $T = 37; # temperature(C)
my $windowL = 15;
my $window1 = '';
my $window2 = '';
```

```
my $nt='';
my $tK='';
my @sequence=();
#variable para guardar la otra mitad de la ventana
my %NNparams = (
        # SantaLucia J (1998) PNAS 95(4): 1460-1465. [NaCl] 1M, 37C & pH=7
        # H(enthalpy): kcal/mol        , S(entropy): cal/k·mol stacking dinucleotides
        'AA/TT' , {'H',-7.9, 'S',-22.2},
        'AT/TA' , {'H',-7.2, 'S',-20.4},
        'TA/AT' , {'H',-7.2, 'S',-21.3},
        'CA/GT' , {'H',-8.5, 'S',-22.7},
        'GT/CA' , {'H',-8.4, 'S',-22.4},
        'CT/GA' , {'H',-7.8, 'S',-21.0},
        'GA/CT' , {'H',-8.2, 'S',-22.2},
        'CG/GC' , {'H',-10.6,'S',-27.2},
        'GC/CG' , {'H',-9.8, 'S',-24.4},
        'GG/CC' , {'H',-8.0, 'S',-19.9},
        # initiation costs
        'G' , {'H', 0.1, 'S',-2.8 },
        'A' , {'H', 2.3, 'S',4.1 },
        # symmetry correction
        'sym' , {'H', 0, 'S',-1.4 } );

my $infile = $ARGV[0] || die "# usage: $0 <promoters file>\n";
print "# parameters: Temperature=$T\C Window=$windowL\n\n";
open(SEQ, $infile) || die "# cannot open input $infile : $!\n";
while(<SEQ>) {
        if(/^(b\d{4}) \\ ([ATGC]+)/)
        {
                my ($name,$seq) = ($1,$2);

                printf("sequence %s (%d nts)\n",$name,length($seq));
                ###AQUI EMPIEZA EL CODIGO AÑADIDO. El resto son modificaciones
                ###de la subrutina###
                duplex_deltaG($seq, $T, $name);
        }
}
close(SEQ);
# calculate NN free energy of a DNA duplex , dG(t) = (1000*dH - t*dS) / 1000
# parameters: 1) DNA sequence string; 2) Celsius temperature returns; 1) free
# energy scalar uses global hash %NNparams utiliza variables globales window1
# y window2
sub duplex_deltaG{
           my ($seq,$tCelsius,$name) = @_;

        my ($DNAstep,$nt,$dG) = ('','',0);
        #DNAstep: contiene dinucleotido nt: utilizado para correciones dG:
        #intermediario en energia libre
        my @sequence = split(//,uc($seq));
        my $tK = 273.15 + $tCelsius;
        my @deltaG=(); #vector para guardar las energías libres de Gibbs

        # add dG for overlapping dinucleotides
        ###AQUI EMPIEZA EL CODIGO MODIFICADO###
```

```
        for(my $n=0;$n<$#sequence-14;$n++){
#bucle que recorre la secuencia para encontrar las ventanas
                for(my $i=0; $i<$windowL-1; $i++){
#bucle que recorre las 15 posiciones para formar la ventana
                        $dG=0;
            $DNAstep='';
                        my $temporal=$sequence[$n+$i].$sequence[$n+$i+1];
#variable temporal para no afectar la variable seq original
                        $temporal= complement($temporal);
            $DNAstep = $sequence[$n+$i].$sequence[$n+$i+1].'/'.$temporal;

                        if(!defined($NNparams{$DNAstep})){
                                $DNAstep = reverse($DNAstep);
#encontrar reversa
                        }

                        $dG = ((1000*$NNparams{$DNAstep}{'H'})-($tK*$NNparams{$DNAstep}{'S'}))/ 1000 ;
                        $deltaG[$n] += $dG; #vector de energías libres
                }

                # add correction for helix initiation
                $nt = $sequence[$n]; # first pair
                if(!defined($NNparams{$nt})){
                        $nt = complement($nt) }
                $deltaG[$n] += ((1000*$NNparams{$nt}{'H'})-($tK*$NNparams{$nt}{'S'}))/ 1000;
                $nt = $sequence[$#sequence]; # last pair
                if(!defined($NNparams{$nt})){
                        $nt = complement($nt) }
                $deltaG[$n] += ((1000*$NNparams{$nt}{'H'})-($tK*$NNparams{$nt}{'S'}))/ 1000;

                # please complete for symmetry correction secuencias
                # palindromicas
                for(my $j=0; $j<7; $j++){

            $window1 .= $sequence[$n+$j];
            $window2 .= $sequence[$n+14-$j];
        }
                $window2=complement($window2);
#complementaria de la segunda mitad
                if($window1 eq $window2){ #comparar fragmentos de secuencia
                        $deltaG[$n] += ((1000*$NNparams{'sym'}{'H'})-($tK*$NNparams{'sym'}{'S'}))/1000;
 #agregar correciones a las ventanas correspondientes
                }

        }

        ###CALCULAR D, E1 y E2###
        my $cutoff_1= -16.7; #valor obtenido de la figura 4
        my $cutoff_2= 3.4; #valor obtenido de la figura 4
        my @promoter; #vector para guardar la posicion de la ventana
        my $E1n=0;
        my $E2n=0;
        my $Dn=0;
        my @E1=();
```

```
        my @E2=();
        my @D=();
        my $l=0;
        for( my $i=0; $i<((scalar(@deltaG))-199); $i++){
#bucle que recorre todas las ventanas para calcular D
                #calcula E1
                for (my $k=0; $k<49; $k++){
                        $E1+=$deltaG[$k+$i];
                }
                $E1n/=50; #promedio
                for (my $k=99; $k<199; $k++){
                                $E2n+=$deltaG[$k+$i];
                        }
                $E2n/=100; #promedio
                $Dn=$E1n-$E2n; #diferencia
                $E1[$i]=$E1n;
                $E2[$i]=$E2n;
                $D[$i]=$Dn;
                if($E1n>$cutoff_1){
#si se respeta el valor de corte, se siguen haciendo los cálculos
                        if($Dn>$cutoff_2){
#checar que se respete el segundo valor de corte
                                if(($i-350)>-150 && ($i-350)<50){
#se encuentra entre las posiciones -150,+50
                                        $promoter[$l]=$i;
#guardar posición del promotor
                                        $i+=24;
#aumentar las posiciones que no se aumentan en el bucle for
                                        $l++;
                                }
                        }
                }
                #reiniciar variables
                $E1n=0;
                $E2n=0;
                $Dn=0;
        }

        for(my $i=0; $i<scalar(@promoter); $i++){
#bucle para imprimir el vector en el que se guardan los inicios de la secuencias predichas donde los va
                print "Inicio\tFinal\n";
                print $promoter[$i]-400,"\t",$promoter[$i]-350,"\n";
 #imprime el inicio y fin del posible promotor
        }
        for(my $i=0; $i<scalar(@E1); $i++){
#bucle para imprimir el vector en el que se guardan los E1
                print "E1\t";
                print $E1[$i],"\n";
        }
        for(my $i=0; $i<scalar(@E2); $i++){
#bucle para imprimir el vector en el que se guardan los E2
                print "E2\t";
                print $E2[$i],"\n";
        }
```

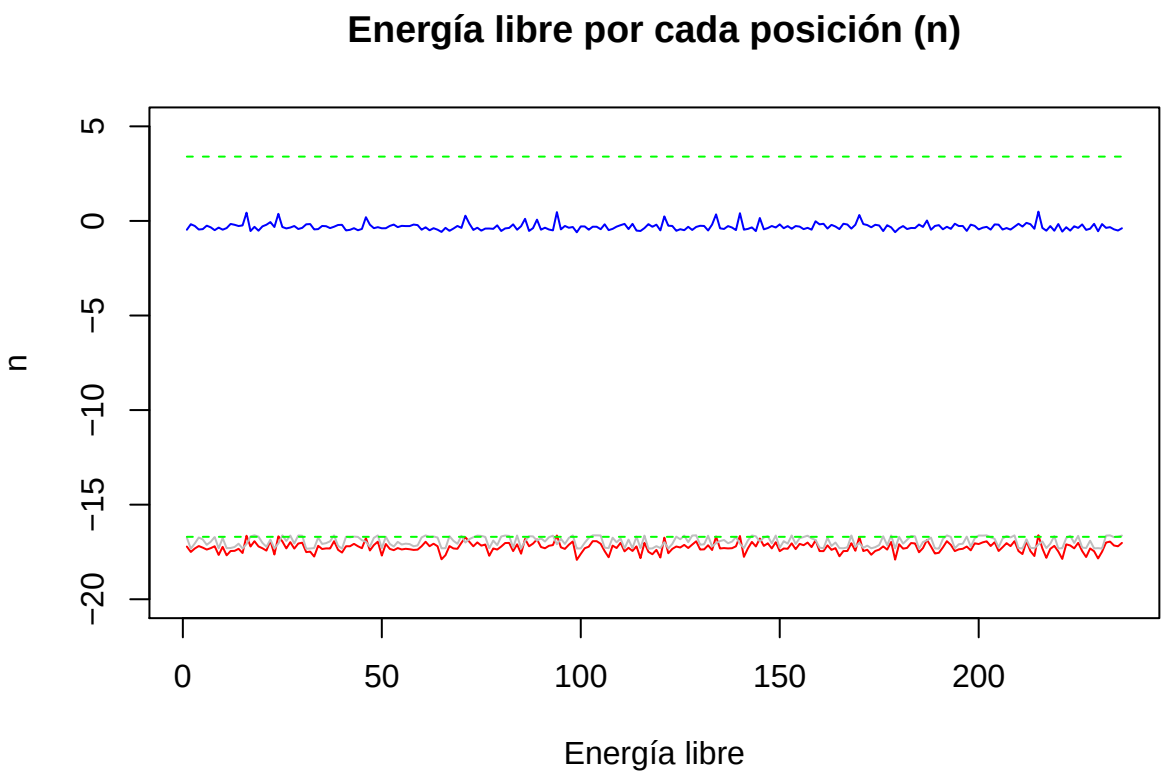
```
        for(my $i=0; $i<scalar(@D); $i++){
#bucle para imprimir el vector en el que se guardan los D
                print "D\t";
                print $D[$i],"\n";
        }


        print "\n\n";
        print "Energía libre\n";
        printf("%s\tPosicion\n",$name);
        for (my $i=0; $i<scalar(@deltaG); $i++){
                print $deltaG[$i],"\t",$i-400,"\n";
        }
}
```

2.2) Diseñar una figura donde se muestre gráficamente D, E1 y E2 para una posición n.

Se crearon archivos de salida a partir del código anterior, para poder luego leer los valores de D, E1 y E2 en un código en R.

Se muestran los valores promedio de D, E1 y E2 de todas las posiciones n para observar el comportamiento a lo largo de la secuencia.

```
#Crear vectores a partir de los datos obtenidos
E1<-as.vector(scan("C:/Users/USUARIO/Documents/TODO - Lenovo backup/LCG/Semestre IV/Bioinformática/Brun
E2<-as.vector(scan("C:/Users/USUARIO/Documents/TODO - Lenovo backup/LCG/Semestre IV/Bioinformática/Brun
D<-as.vector(scan("C:/Users/USUARIO/Documents/TODO - Lenovo backup/LCG/Semestre IV/Bioinformática/Bruno

#Crear vectores de los valores cutoff repetidos para facilitar graficarlos
cutoff1<-rep(3.4,236)
cutoff2<-rep(-16.7,236)

#Graficar
plot(E1, ylim=c(-20,5), xlim=c(1,236), ylab="n", xlab="Energía libre", main="Energía libre por cada pos
lines(E2, type='l', col="grey")
lines(D, type='l', col="blue")
lines(cutoff1, type='l', lty=2, col="green")
lines(cutoff2, type='l', lty=2, col="green")
```



E1 - rojo. E2 - gris. D - azul.

Los valores de corte se muestran en verde.

En los promedios, parece que en ninguna posición se pasa de los valores de corte. Sin embargo, observemos qué pasa si tomamos una única secuencia.

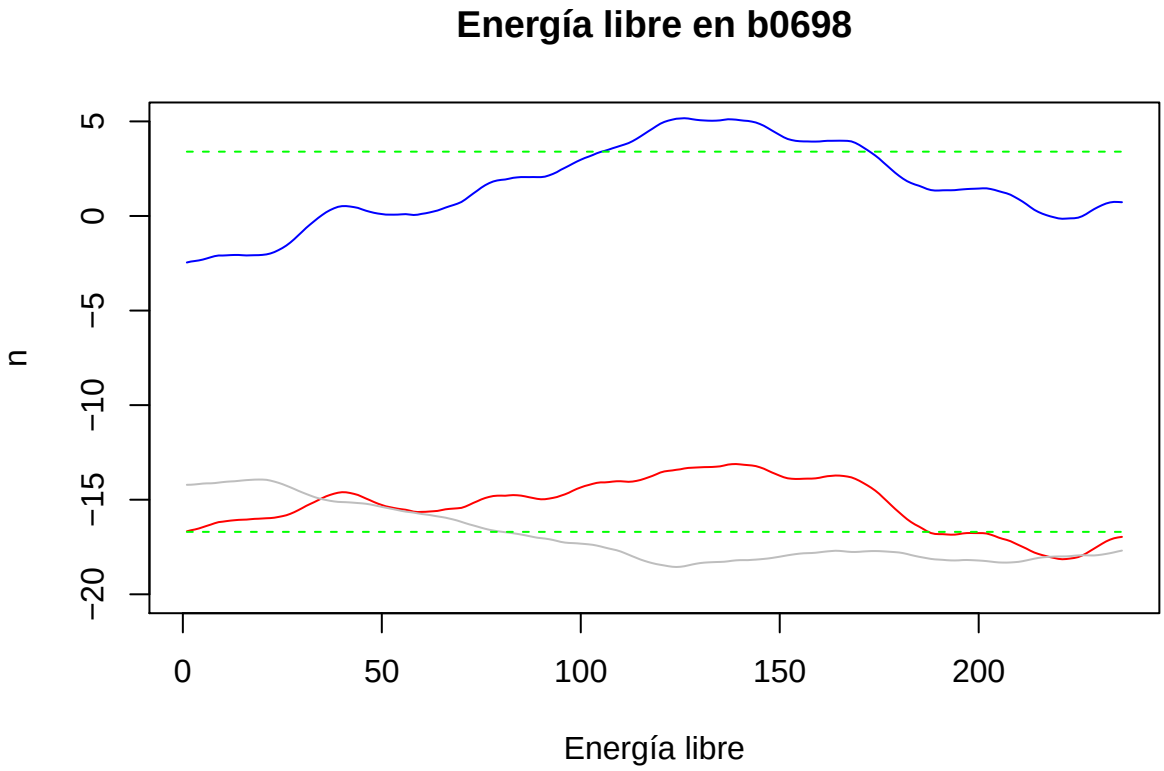
Se escogió la secuencia b0698 porque se encontraron tres posibles promotores:

```
b0698 \ CTTTTCTCCCCCTTTTGTATCTACCCGGTGAATGGCACCGGAAAAATGAATTTGTTTATCTGATGAAAATAG
TACCGCCTTTTGTGTAATTTTACTACTCATCCGACCACTTATTTTTGCTTATTGATGGTTTATTTACATTCATCCTGTAA
TTAAGTTACACAAAAGTTAAATTAATACTAAACATTAGTTAAATCATGGCTTTTGCCATTTTTATACTTTTTTTACACCC
CGCCCGCAGATTTTTGCGAAATCTTTGCAGCCAGAATTCTACCCTTCCGGTATCACTTTTAGGCCACTGGAGGTGCACTG
TGAGTGCAGGCGTGATAACCGGCGTATTGCTGGTGTTTTTATTACTGGGTTATCTGGTTTATGCCCTGATCAATGCGGAG
GCGTTCTGATGGCTGCGCAAGGGTTCTTACTGATCGCCACGTTTTTACTGGTGTTAATG \
```

A continuación se muestran los valores de D, E1 y E2 en esta secuencia.

```r
#Crear vectores a partir de los datos obtenidos
EjemE1<-as.vector(scan("C:/Users/USUARIO/Documents/TODO - Lenovo backup/LCG/Semestre IV/Bioinformática/
EjemE2<-as.vector(scan("C:/Users/USUARIO/Documents/TODO - Lenovo backup/LCG/Semestre IV/Bioinformática/
EjemD<-as.vector(scan("C:/Users/USUARIO/Documents/TODO - Lenovo backup/LCG/Semestre IV/Bioinformática/Bi

#Graficar
plot(EjemE1, ylim=c(-20,5), xlim=c(1,236), ylab="n", xlab="Energía libre", main="Energía libre en b0698
lines(EjemE2, type='l', col="grey")
lines(EjemD, type='l', col="blue")
```

```
lines(cutoff1, type='l', lty=2, col="green")
lines(cutoff2, type='l', lty=2, col="green")
```



E1 - rojo. E2 - gris. D - azul.

Los valores de corte se muestran en verde.

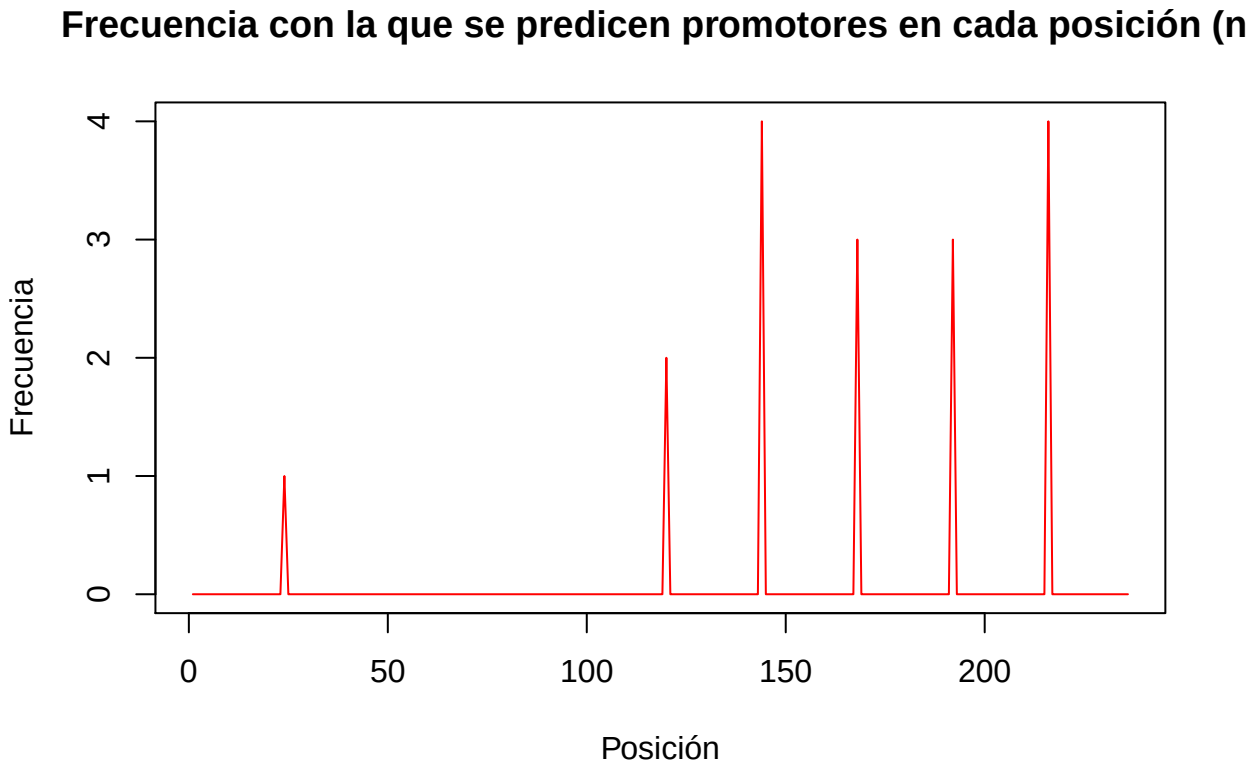
2.3) Predecir promotores en todas las secuencias del fichero K12_400_50_sites.

```
sequence b0585 (451 nts)
Inicio  Final
-134    -84
sequence b0972 (451 nts)
Inicio  Final
-206    -156
Inicio  Final
-182    -132
Inicio  Final
-158    -108
sequence b0894 (451 nts)
Inicio  Final
-158    -108
Inicio  Final
```

```
-134    -84
sequence b0592 (451 nts)
Inicio  Final
-326    -276
Inicio  Final
-206    -156
sequence b0584 (451 nts)
Inicio  Final
-182    -132
Inicio  Final
-158    -108
sequence b0698 (451 nts)
Inicio  Final
-230    -180
Inicio  Final
-206    -156
Inicio  Final
-182    -132
sequence b0116 (451 nts)
Inicio  Final
-206    -156
sequence b0388 (451 nts)
Inicio  Final
-134    -84
sequence b0889 (451 nts)
Inicio  Final
-230    -18
sequence b0827 (451 nts)
Inicio  Final
-134    -84
```

2.4) Graficar con qué frecuencia se predicen promotores en el intervalo -400,+50.

```
#Crear un vector a partir de los datos obtenidos
Pos<-as.vector(scan("C:/Users/USUARIO/Documents/TODO - Lenovo backup/LCG/Semestre IV/Bioinformática/Bru

Freqn<-as.data.frame(table(Pos))

elems<-as.vector(Freqn$Pos)

freqs<-as.vector(Freqn$Freq)

Freq<-data.frame(pos=c(1:236), freq=rep(0,236))

Freq[elems,]<-freqs

graph_freq<-Freq$freq

graph_pos<-Freq$pos

plot(graph_freq, xlab="Posición", ylab="Frecuencia", type='l', col="red", main="Frecuencia con la que s
```



¿Hay manera de validar sus resultados y calcular la tasa de FPs, usando RSAT::matrix-scan?

Una manera para descubrir falsos positivos sería creando un tipo de control negativo.

El fichero K12_400_50_sites contiene coordenadas de marcos abiertos de lectura de *E. coli*, como ya se ha mencionado.

Para esto, un control negativo de los promotores sería encontrar las secuencias río arriba de genes aleatorios (*random gene selection* en RSAT) y comparar las frecuencias.

Se puede recolectar el set completo de promotores del genoma de interés y ver si los obtenidos existen en este conjunto.

Matrix-scan hace comparación de patrones. Los resultados muestran hasta qué punto los motivos tienen matches en secuencias río arriba; es decir, cuántos de los promotores predichos realmente están en secuencias río arriba de *coli*. Se esperaría que los valores fueran mejores para loss observados/predichos que para los controles negativos.
