Ke et al.

Saturation mutagenesis reveals manifold determinants of exon definition

Supplemental Methods: Perl and Python scripts

TABLE OF CONTENTS

```
################################################################################
########
# IpBarCodeParse.pl
#
# Strictly screens sequencing data by demanding that bar code for protein of interest
is
# matched perfectly
################################################################################
########
use warnings;
use strict;
my %bases = ();
$bases{"A"} = "T";
$bases{"C"} = "G";
$bases{"G"} = "C";
$bases{"T"} = "A";

my %bar_codes = ();
open(FILE, "exp_2_rna_quantities.csv");
while(<FILE>){
    chomp $_;
    my @props = split(/\,/, $_);
    $bar_codes{$props[2]} = $props[0];

}
close(FILE);

my %groups = ();
my %phreds = ();
my %ids = ();
my $total_sequences = 0;
for(my $number = 1; $number < 5; $number++){
    open(FILE, "20150410_S1_L00".$number."_R1_001.fastq");
    my $counter = 0;
    my $current_phred = "";
    my $current_id = "";
    my $is_good = 0;
    my $current_membership = "";
    while(<FILE>){
      chomp $_;
      if(($counter % 4) == 0){
          $current_id = $_;
      }elsif(($counter % 4) == 1){
          my $seq_bar = substr($_, 0, 8);
          if(exists($bar_codes{$seq_bar})){
            if(!exists($groups{$bar_codes{$seq_bar}})){
                $groups{$bar_codes{$seq_bar}} = [];
                $ids{$bar_codes{$seq_bar}} = [];
                $phreds{$bar_codes{$seq_bar}} = [];
            }
            push(@{$groups{$bar_codes{$seq_bar}}}, $_);
            push(@{$ids{$bar_codes{$seq_bar}}}, $current_id);
            $current_membership = $bar_codes{$seq_bar};
            $is_good = 1;
          }else{
            $is_good = 0;
          }
          $total_sequences++;
      }elsif(($counter % 4) == 3 && $is_good){
```

```
            push(@{$phreds{$current_membership}}, $_);
        }
        $counter++;
    }
    close(FILE);
}
foreach my $group (sort keys %groups){
    open(WRITE, ">".$group."_Perfect");
    my $size = scalar(@{$groups{$group}});
    for(my $i = 0; $i < $size; $i++){
      print WRITE $ids{$group}[$i]."\n";
      print WRITE $groups{$group}[$i]."\n";
      print WRITE $phreds{$group}[$i]."\n";
    }
    close(WRITE);
    print $group."\t".$size."\t".$total_sequences."\t".($size/$total_sequences)."\n";
}

sub rc{
    my $seq = $_[0];
    my $reverse_seq = reverse($seq);
    my @seq_bases = split(//, $reverse_seq);
    my $rc_seq = "";
    foreach my $base (@seq_bases){
      $rc_seq = $rc_seq.$bases{$base};
    }
    return $rc_seq;
}
```

```
################################################################################
########
# IpSequenceParse.pl
#
# 1)Demands that read sequences already screened for perfect bar codes by
ParseData.pl contains
# a perfect match to one of our known mutants at the expected nt distance from the
bar code
# 2) Calculates the read count and frequency for all sequences of interest
################################################################################
########
use warnings;
use strict;
my %real_seqs = ();
my %bases = ();
$bases{"A"} = "T";
$bases{"C"} = "G";
$bases{"G"} = "C";
$bases{"T"} = "A";
$bases{"N"} = "N";

open(FILE, "HM90summVNewLEISC");
while(<FILE>){
    chomp $_;
    my @props = split(/\t/, $_);
    my $mut_area = substr($props[3], 24, 47);
    $real_seqs{$mut_area} = 1;
}
close(FILE);
my @classes;
open(FILE, "exp_2_rna_quantities2.csv");
while(<FILE>){
    chomp $_;
    my @props = split(/\,/, $_);
    push(@classes, $props[0]);
}
close(FILE);

open(WRITE0, ">UH_Normalized_Frequencies/summary_part_2");
foreach my $class (@classes){
    my $start = time();
    open(FILE, $class."_Perfect");
    my $total_good_reads = 0;
    my %read_count = ();
    my $is_kept = 0;
    my $kept_index = -1;
    my @seq_phreds;
    my $bad = 0;
    my $line_count = 0;
    while(<FILE>){
      chomp $_;
      my $l = length($_);
      if(($line_count % 3) == 1){
          if($l >= 47){
            my $seq = rc($_);
            my $already_in = 0;
            for(my $i = 0; $i < ($l-46) && !$already_in; $i++){
                my $sub_seq = substr($seq, $i, 47);
                if(exists($real_seqs{$sub_seq})){
```

```
                if(!exists($read_count{$sub_seq})){
                    $read_count{$sub_seq} = 0;
                }
                $read_count{$sub_seq} += 1;
                $total_good_reads++;
                $already_in = 1;
                $is_kept = 1;
                $kept_index = $i;
              }else{
                $is_kept = 0;
              }
            }
          }else{
            $is_kept = 0;
          }
      }elsif(($line_count % 3) == 2 && $is_kept){
          my $seq_quality = reverse($_);
          $seq_quality = substr($seq_quality, $kept_index, 47);
          my @scores;
          for(my $i = 0; $i < 47; $i++){
            push(@scores, ord(substr($seq_quality, $i, 1))-33);
          }
          my $avg_phred = average(\@scores);
          if($avg_phred < 21){
            $bad++;
          }
          push(@seq_phreds, $avg_phred);
      }
      $line_count++;
    }
    close(FILE);
    my @real_read_counts;
    my $real = 0;
    open(WRITE1, ">UH_Normalized_Frequencies/".$class."_Frequency");
    foreach my $key (sort {$read_count{$b}<=>$read_count{$a}} keys %read_count){
      push(@real_read_counts, $read_count{$key});
      print WRITE1
$key."\t".$read_count{$key}."\t".($read_count{$key}/$total_good_reads)."\n";
      $real++;
    }
    close(WRITE1);
    my $average_avg_phred = average(\@seq_phreds);
    my $staddev_avg_phred = staddev(\@seq_phreds);
    my $average_real = average(\@real_read_counts);
    my $staddev_real = staddev(\@real_read_counts);
    print WRITE0
$class."\t".$real."\t".$total_good_reads."\t".$average_real."\t".$staddev_real."\t".$
average_avg_phred."\t".$staddev_avg_phred."\t".$bad."\n";
    my $end = time();
    my $elapsed = $end-$start;
    print $class."\t".$elapsed."\n";
}
close(WRITE0);

sub average{
    my @values = @{$_[0]};
    my $length = scalar(@values);
    my $total = 0;
    foreach my $value (@values){
```

```
      $total = $total + $value;
    }
    my $avg = $total/$length;
    return $avg
}
#function to calculate the standard deviation of a set of values in an array
#given that there exists an average function
sub staddev{
    my @values = @{$_[0]};
    my $avg = average(\@values);
    my $total = 0;
    foreach my $value (@values){
      $total = $total + ($value-$avg)**2;
    }
    my $length = scalar(@values);
    my $stdd = ($total/($length-1))**.5;
    return $stdd;
}

sub rc{
    my $seq = $_[0];
    my $reverse_seq = reverse($seq);
    my @seq_bases = split(//, $reverse_seq);
    my $rc_seq = "";
    foreach my $base (@seq_bases){
      if(exists($bases{$base})){
          $rc_seq = $rc_seq.$bases{$base};
      }else{
          die $base."\n";
      }
    }
    return $rc_seq;
}
```

```
################################################################################
########
# CalculatePPD.pl
#
# For a given protein pull down, calculates a PPD by using the frequency of a
sequence in the
# pull down library of interest and its frequency in the input library, controlling
for
# the amount of RNA used and for artificial pull down per immunoglobulin used (mouse,
rabbit,
# or goat)
################################################################################
########

use warnings;
use strict;

my $READ_MIN = $ARGV[0];
my %quantities = ();

my %classes = ();
my %label_to_class = ();
my @classes_array;
open(FILE, "bar_codes");
while(<FILE>){
    chomp $_;
    my @props = split(/\,/, $_);
    $classes{$props[0]} = [$props[0], $props[1], $props[3]];
    $label_to_class{$props[0]} = $props[0];
    $quantities{$props[0]} = $props[4];
    push(@classes_array, $props[0]);
}
close(FILE);

#find controls

my %controls = ();
my %control_totals = ();
my %experimental_totals = ();
foreach my $key (keys %classes){
    my @control_type = split("_", $classes{$key}[0]);
    if($control_type[0] eq "MOUSE" || $control_type[0] eq "RABBIT" ||
$control_type[0] eq "GOAT"){
      $controls{$key} = {};
      open(FILE, $key."_Frequency");
      while(<FILE>){
          chomp $_;
          my @props = split(/\t/, $_);
          $controls{$key}{$props[0]} = $props[1];
          $control_totals{$key} += $props[1];
      }
      close(FILE);
    }else{
      $experimental_totals{$key} = 0;
      open(FILE, $key."_Frequency");
      while(<FILE>){
          chomp $_;
          my @props = split(/\t/, $_);
          $experimental_totals{$key} += $props[1];
```

```
      }
      print $key."\t".$experimental_totals{$key}."\n";
      close(FILE);
    }
}


for(my $exp = 1; $exp < 3; $exp++){
    my %input_class = ();
    my %input_class_raw = ();
    my $current_input = "I_".$exp;
    open(FILE, "I_".$exp."_Frequency");
    while(<FILE>){
      chomp $_;
      my @props = split(/\t/, $_);
      $input_class{$props[0]} = $props[2];
      $input_class_raw{$props[0]} = $props[1];
    }
    close(FILE);
    foreach my $class (@classes_array){
      my @file_props = split("_", $class);
      my $control_label = $classes{$class}[0];
      my @control_props = split("_", $control_label);
      if($file_props[0] ne "I" && $control_props[0] ne "MOUSE" && $control_props[0]
ne "RABBIT" && $control_props[0] ne "GOAT" && $file_props[1] == $exp){
          print $control_props[0]."\n";
          my %output_class = ();
          open(FILE, $class."_Frequency");
          my $not_present = 0;
          my $not_present_freqs = "";
          my $total_seqs = 0;
          open(WRITE,
">Experiment_Controlled_Fractions_Modified_".$READ_MIN."/".$class."_Controlled_Fracti
ons");
          my $pseudo = 0;
          while(<FILE>){
            chomp $_;
            my @props = split(/\t/, $_);
            $output_class{$props[0]} = $props[1];
            $total_seqs += $props[1];

            if(!exists($input_class{$props[0]})){
                $not_present++;
                $not_present_freqs = $not_present_freqs."\t".$props[1];
            }elsif($input_class_raw{$props[0]} >= $READ_MIN){
                my $controlled_ei_num;
                my $out_prop;
                my $cont_prop;
                my $out_quant;
                my $cont_quant;

                if(!exists($controls{$classes{$class}[1]}{$props[0]})){
                  $controlled_ei_num =
(($output_class{$props[0]}/$experimental_totals{$class})*$quantities{$classes{$class}
[0]})-
((1/$control_totals{$classes{$class}[1]})*$quantities{$classes{$classes{$class}[1]}[0
]});
                  $out_prop = ($output_class{$props[0]}/$experimental_totals{$class});
                  $cont_prop = (1/$control_totals{$classes{$class}[1]});
```

```
                $out_quant = $out_prop*$quantities{$classes{$class}[0]};
                $cont_quant =
$cont_prop*$quantities{$classes{$classes{$class}[1]}[0]};
                $pseudo++;
              }else{
                $controlled_ei_num =
(($output_class{$props[0]}/$experimental_totals{$class})*$quantities{$classes{$class}
[0]})-
(($controls{$classes{$class}[1]}{$props[0]}/$control_totals{$classes{$class}[1]})*$qu
antities{$classes{$classes{$class}[1]}[0]});
                $out_prop = ($output_class{$props[0]}/$experimental_totals{$class});
                $cont_prop =
($controls{$classes{$class}[1]}{$props[0]}/$control_totals{$classes{$class}[1]});
                $out_quant = $out_prop*$quantities{$classes{$class}[0]};
                $cont_quant =
$cont_prop*$quantities{$classes{$classes{$class}[1]}[0]};
              }
              my $controlled_ei =
$controlled_ei_num/($input_class{$props[0]}*$quantities{$current_input});
              print WRITE
$props[0]."\t".$controlled_ei."\t".$out_prop."\t".$out_quant."\t".$cont_prop."\t".$co
nt_quant."\t".$input_class{$props[0]}."\t".($input_class{$props[0]}*$quantities{$curr
ent_input})."\n";
            }
          }
          close(FILE);

          foreach my $key (keys %input_class){
            if(!exists($output_class{$key}) && $input_class_raw{$key} >= $READ_MIN){
              my $controlled_ei_num;
              my $out_prop;
              my $cont_prop;
              my $out_quant;
              my $cont_quant;
              if(exists($controls{$classes{$class}[1]}{$key})){
                $controlled_ei_num =
((1/$experimental_totals{$class})*$quantities{$classes{$class}[0]})-
(($controls{$classes{$class}[1]}{$key}/$control_totals{$classes{$class}[1]})*$quantit
ies{$classes{$classes{$class}[1]}[0]});
                $out_prop = (1/$experimental_totals{$class});
                $cont_prop =
($controls{$classes{$class}[1]}{$key}/$control_totals{$classes{$class}[1]});
                $out_quant = $out_prop*$quantities{$classes{$class}[0]};
                $cont_quant =
$cont_prop*$quantities{$classes{$classes{$class}[1]}[0]};
              }else{
                $controlled_ei_num =
((1/$experimental_totals{$class})*$quantities{$classes{$class}[0]})-
((1/$control_totals{$classes{$class}[1]})*$quantities{$classes{$classes{$class}[1]}[0
]});
                $out_prop = (1/$experimental_totals{$class});
                $cont_prop = (1/$control_totals{$classes{$class}[1]});
                $out_quant = $out_prop*$quantities{$classes{$class}[0]};
                $cont_quant =
$cont_prop*$quantities{$classes{$classes{$class}[1]}[0]};
              }
              my $controlled_ei =
$controlled_ei_num/($input_class{$key}*$quantities{$current_input});
```

```
                print WRITE
$key."\t".$controlled_ei."\t".$out_prop."\t".$out_quant."\t".$cont_prop."\t".$cont_qu
ant."\t".$input_class{$key}."\t".($input_class{$key}*$quantities{$current_input})."\n
";
                $pseudo++;
             }
          }
          print $pseudo."\n";
          close(WRITE);
      }
    }
}
```

```
################################################################################
########
# Molecule.pm
#
# Molecule object containing essential parameters to define an exon molecule and
store
# key properties
################################################################################
########


package Molecule;
sub new{
    my $class = shift;
    my $self = {
      _serial => shift,
      _hm => shift,
      _mut_pos => shift,
      _wt_bases => shift,
      _mutated_bases => shift,
      _seq => shift,
      _EI => shift,
      _rel_EI => shift,
      _PUP => shift,
      _mut_type => shift,
      _LEI => undef,
      _PSI => undef,
      _LEISC => undef,
      _LEIe =>undef,
      _t_stat => undef,
      _p_val => undef,
      _protein_positions => {},
      _sig => undef,
      _control => undef,
      _proteins => {},
      _excess => undef,
      _average => undef,
      _motifs => [],
      _clusters => [],
      _max_wt_motifs => [],
      _sig_motifs => [],
      _insig_motifs => [],
      _motif_scores => [],
      _esr_scores_left => [],
      _esr_scores_right => [],
      _esr_sig_left => [],
      _esr_sig_right => [],
      _rbpss => [],
      _rbpss_dis => [],
    };
    bless $self, $class;
    return $self;
}
1;
```

```
################################################################################
########
# RegressionsFDR.pl
#
# 1. Uses heptamer Z-scores for RNA binding proteins (CISBP-RNA) and experimental
exon
# inclusion data to perform regressions at each position along the mutated exon,
correlating
# exon inclusion changes created by mutations at a 7mer focus along the exon with the
# consequential estimated affinity change for a given protein
# 2. Controls for multiple comparisons by the BH procedure (alpha = 0.05) for a given
hexmut
# set
################################################################################
########
use warnings;
use strict;
use Statistics::Regression;
use Statistics::Distributions;
use Molecule;


#GATHER Z SCORE AND RBP INFORMATION#
open(FILE, "Zscores.txt") or die "ERROR: $!\n";
my @lines = <FILE>;
close(FILE);
chomp @lines;
my $id_row = shift(@lines);
my @ids = split(/\t/, $id_row);

shift @ids;
my %proteins = ();
my $id_index = 0;
my @translator;
foreach my $id (@ids){
    $translator[$id_index] = $id;
    $proteins{$id} = {};
    $id_index++;
}

my $n_protein_ids = scalar(@translator);
my $hept_counter = 0;
foreach my $line (@lines){
    my @z_intensities = split(/\t/, $line);
    my $motif = shift(@z_intensities);
    $motif =~ s/U/T/g;
    my $n_proteins = scalar(@z_intensities);
    $hept_counter++;
    for(my $i = 0; $i < $n_proteins; $i++){
      if(exists($proteins{$translator[$i]}{$motif})){
          die "ERROR: Repeated sequence\n";
      }
      $proteins{$translator[$i]}{$motif} = $z_intensities[$i];
    }
}

my %protein_translator = ();
open(FILE, "RBP_Information.txt") or die "ERROR: $!\n";
my @lns = <FILE>;
```

```
close(FILE);
shift @lns;
chomp @lns;
foreach my $ln (@lns){
    my @props = split(/\t/, $ln);
    if(!exists($protein_translator{$props[3]})){
          $protein_translator{$props[3]} = "";
    }
    $protein_translator{$props[3]} = $protein_translator{$props[3]}.$props[6].",";
}
$protein_translator{"Cons"} = "Cons";
#INITIALIZE MOLECULES##

my %mol_secondary = ();
open(SECONDARY, "All_HM_Heptamer_Secondary_Structure");
while(<SECONDARY>){
    chomp $_;
    my @props = split(/\,/, $_);
    my $id = shift(@props);
    my $seq_id = substr($id, 4);
    $mol_secondary{$seq_id} = \@props;
}
close(SECONDARY);

open(FILE, "HM90summVNewLEISC");
my @molecules;
while(<FILE>){
    chomp $_;
    my @props = split(/\t/, $_);
    if($props[1] != 12){ #gets rid of an HM when needed
      if($props[2] == 0){
          my $temp_mol = new Molecule($props[0], $props[1], $props[2], "", "",
$props[3], $props[4], $props[5], $props[6], "WT");
          $temp_mol->{_LEISC} = $props[7];
          push(@molecules, $temp_mol);
      }else{
          my @sub_bases = split("-", $props[10]);
          my $sub_type = 1;
          if(length($sub_bases[0]) == 1){
            $sub_type = 0;
          }
          my $wt_bases = $sub_bases[0];
          my $mut_bases = $sub_bases[1];
          my $temp_mol = new Molecule($props[0], $props[1], $props[2], $wt_bases,
$mut_bases, $props[3], $props[4], $props[5], $props[6], $sub_type);
          $temp_mol->{_LEISC} = $props[7];
          for(my $i = 0; $i < 100; $i++){
            push(@{$temp_mol->{_rbpss}}, []);
          }
          push(@molecules, $temp_mol);
      }
    }
}
close(FILE);
for(my $hm = 1; $hm < 2; $hm++){
    my $n_molecules_array = scalar(@molecules);
    my %regressions = ();
    my %all_heptamers_z = ();
```

```
    for(my $i = 20; $i < 69; $i++){
      my $exon_position;
      if($i > 22){
          $exon_position = $i-22;
      }else{
          $exon_position = $i-23;
      }
      my $wt_heptamer; my $wt_left; my $wt_right;
      foreach my $molecule (@molecules){

          if($molecule->{_hm} == $hm){
            if($molecule->{_mut_type} eq "WT"){
                $wt_heptamer = substr($molecule->{_seq}, $i, 7);
                $wt_left = substr($molecule->{_seq}, ($i-1), 1);
                $wt_right = substr($molecule->{_seq}, ($i+7), 1);
                foreach my $protein_key (sort keys %proteins){
                  if(exists($proteins{$protein_key}{$wt_heptamer})){
                      if(!exists($regressions{$protein_key})){
                        $regressions{$protein_key} = {};
                      }
                      if(!exists($regressions{$protein_key}{$exon_position})){
                        $regressions{$protein_key}{$exon_position} = [[], [], [], [],
[], [], []];
                      }
                      push(@{$regressions{$protein_key}{$exon_position}[0]},
log2($molecule->{_EI}));
                      push(@{$regressions{$protein_key}{$exon_position}[1]},
$molecule->{_serial});
                      push(@{$regressions{$protein_key}{$exon_position}[2]},
$molecule->{_hm});
                      push(@{$regressions{$protein_key}{$exon_position}[3]},
($mol_secondary{$molecule->{_serial}}[$i]/7));
                      push(@{$regressions{$protein_key}{$exon_position}[4]},
$wt_heptamer);
                      push(@{$regressions{$protein_key}{$exon_position}[5]},
$molecule->{_seq});
                      push(@{$regressions{$protein_key}{$exon_position}[6]},
$proteins{$protein_key}{$wt_heptamer});
                      my $z = sprintf("%.2f", $proteins{$protein_key}{$wt_heptamer});
                      $all_heptamers_z{$z} = 1;
                  }
                }
            }else{
                my $mut_heptamer = substr($molecule->{_seq}, $i, 7);
                my $mut_left = substr($molecule->{_seq}, ($i-1), 1);
                my $mut_right = substr($molecule->{_seq}, ($i+7), 1);
                if($mut_heptamer ne $wt_heptamer && $mut_left eq $wt_left &&
$mut_right eq $wt_right){
                  foreach my $protein_key (sort keys %proteins){
                      if(exists($proteins{$protein_key}{$mut_heptamer})){
                        if(!exists($regressions{$protein_key})){
                            $regressions{$protein_key} = {};
                        }
                        if(!exists($regressions{$protein_key}{$exon_position})){
                            $regressions{$protein_key}{$exon_position} = [[], [], [],
[], [], [], [], []];
                        }
                        push(@{$regressions{$protein_key}{$exon_position}[0]},
log2($molecule->{_EI}));
```

```
                        push(@{$regressions{$protein_key}{$exon_position}[1]},
$molecule->{_serial});
                        push(@{$regressions{$protein_key}{$exon_position}[2]},
$molecule->{_hm});
                        push(@{$regressions{$protein_key}{$exon_position}[3]},
($mol_secondary{$molecule->{_serial}}[$i]/7));
                        push(@{$regressions{$protein_key}{$exon_position}[4]},
$mut_heptamer);
                        push(@{$regressions{$protein_key}{$exon_position}[5]},
$molecule->{_seq});
                        push(@{$regressions{$protein_key}{$exon_position}[6]},
$proteins{$protein_key}{$mut_heptamer});
                        my $z = sprintf("%.2f",
$proteins{$protein_key}{$mut_heptamer});
                        $all_heptamers_z{$z} = 1;
                      }

                  }
                }
            }
          }
      }
    }

    my @z_cutoffs = sort {$a<=>$b} keys %all_heptamers_z;
    my $n_cutoffs = scalar(@z_cutoffs);
    my %significant_regressions = ();
    my $iteration = 0;
    my @first_regressions;
    my @sig_regressions;
    my @insig_regressions;
    open(WRITE,
">No_Cutoff_Regressions_FDR_5/Significant_Cutoff_LEI_Z_Regressions_".$hm);
    open(WRITE2,
">No_Cutoff_Regressions_FDR_5/Significant_Cutoff_LEI_Z_Regressions_".$hm."_Values");
    foreach my $protein (keys %regressions){
      foreach my $exon_position (keys %{$regressions{$protein}}){
          my $n_points = scalar(@{$regressions{$protein}{$exon_position}[0]});
          my @leiscs; my @z_scores;
          for(my $i = 0; $i < $n_points; $i++){
            push(@leiscs, $regressions{$protein}{$exon_position}[0][$i]);
            push(@z_scores, $regressions{$protein}{$exon_position}[6][$i]);
          }

          $n_points = scalar(@leiscs);
          if($n_points >= 4){
            my $n_pass_cutoff = 0;
            my $reg = Statistics::Regression->new("", ["y-int", "slope"]);
            for(my $i = 0; $i < $n_points; $i++){
                $reg->include($leiscs[$i], [1, $z_scores[$i]]);
                print WRITE2
$protein."\t".$protein_translator{$protein}."\t".$exon_position."\t".$z_scores[$i]."\
t".$leiscs[$i]."\n";
                if($z_scores[$i] >= .45){
                  $n_pass_cutoff++;
                }
            }
            print WRITE2 "\n";
            my $r2 = $reg->rsq();
```

```
            my @betas = $reg->theta();
            my $rss = ($reg->sigmasq())*($n_points-2);
            my $sst = $reg->sst();
            my $seed_var = 2;
            my $average_z = average(\@z_scores);
            my $median_z = median(\@z_scores);
            my $f_for_model = (($sst-$rss)/($seed_var-1))/(($rss)/($n_points-
$seed_var));
            my $p_for_model = Statistics::Distributions::fprob(($seed_var-1),
($n_points-$seed_var), $f_for_model);
            my @regression_props = ($protein, $protein_translator{$protein},
$exon_position, $betas[0], $betas[1], $r2, $p_for_model, $f_for_model, $average_z,
$median_z, $n_pass_cutoff, $n_points);
            push(@first_regressions, \@regression_props);

          }
      }
    }
    my $alpha = .05;
    my $prev_p = -10;
    my $rank = 0;
    my $total_regs = 0;
    my $sig_regs = 0;
    my $avg_r2 = 0;
    my $m = scalar(@first_regressions);


    foreach my $regression (sort sortByP @first_regressions){
      my @regression_arr = @{$regression};
      if($regression_arr[6] != $prev_p){
          $rank++;
          $prev_p = $regression_arr[6];
      }
      if($regression_arr[6] <= $alpha*($rank/$m)){
          if(!exists($significant_regressions{$regression_arr[0]})){
            $significant_regressions{$regression_arr[0]} = {};
          }

if(!exists($significant_regressions{$regression_arr[0]}{$regression_arr[2]})){
            $significant_regressions{$regression_arr[0]}{$regression_arr[2]} = 1;
          }
          my $id = shift(@regression_arr);
          my $name = shift(@regression_arr);
          my $position = shift(@regression_arr);
          print WRITE $id."\t".$position."\t".$name;
          foreach my $element (@regression_arr){
            print WRITE "\t".$element;
          }
          print WRITE "\n";
      }
    }
    close(WRITE);
    close(WRITE2);
}

sub average{
    my @values = @{$_[0]};
    my $length = scalar(@values);
    my $total = 0;
```

```
    foreach my $value (@values){
      $total = $total + $value;
    }
    my $avg = $total/$length;
    return $avg
}
#function to calculate the standard deviation of a set of values in an array
#given that there exists an average function
sub staddev{
    my @values = @{$_[0]};
    my $avg = average(\@values);
    my $total = 0;
    foreach my $value (@values){
      $total = $total + ($value-$avg)**2;
    }
    my $length = scalar(@values);
    my $stdd = ($total/($length-1))**.5;
    return $stdd;
}


sub sortByP{
    my @a_arr = @{$a};
    my @b_arr = @{$b};
    $a_arr[6]<=>$b_arr[6];
}

sub log2{
    my $n = shift;
    return log($n)/log(2);
}

sub median{
    my @values = @{$_[0]};
    @values = sort {$a<=>$b} @values;
    my $median = 0;
    my $n_values = scalar(@values);

    if($n_values % 2 == 0){
      my $m = ($n_values/2);
      $median = ($values[$m-1]+$values[$m])/2;
    }else{
      my $m = int($n_values/2);
      $median = $values[$m];
    }
    return $median;

}
```

```
################################################################################################
########
# Stepwise.pl
#
# 1) Performs stepwise regression (alpha = 0.01) using proteins that were significant
#    from RegressionsFDR.pl as a starting set
# 2) Used to build a multiple linear model that includes a combination of protein-
positions
#    which have been determined to be significant independent predictors of our exon
inclusion
#    data
# 3) Performs leave-one-out cross validation on models generated
################################################################################################
########

use warnings;
use strict;
use Molecule;
use Statistics::Distributions;
use Statistics::Regression;
my %protein_translator = ();
open(FILE, "RBP_Information.txt") or die "ERROR: $!\n";
my @lns = <FILE>;
close(FILE);
shift @lns;
chomp @lns;
foreach my $ln (@lns){
    my @props = split(/\t/, $ln);
    if(!exists($protein_translator{$props[3]})){
          $protein_translator{$props[3]} = "";
    }
    $protein_translator{$props[3]} = $protein_translator{$props[3]}.$props[6].",";
}


open(FILE, "Zscores.txt") or die "ERROR: $!\n";
my @lines = <FILE>;
close(FILE);
chomp @lines;
my $id_row = shift(@lines);
my @ids = split(/\t/, $id_row);

shift @ids;
my %proteins = ();
my $id_index = 0;
my @translator;
foreach my $id (@ids){
    $translator[$id_index] = $id;
    $proteins{$id} = ();
    $id_index++;
}
my $n_protein_ids = scalar(@translator);
my $hept_counter = 0;
foreach my $line (@lines){
    my @z_intensities = split(/\t/, $line);
    my $motif = shift(@z_intensities);
    $motif =~ s/U/T/g;
    my $n_proteins = scalar(@z_intensities);
    $hept_counter++;
```

```
    for(my $i = 0; $i < $n_proteins; $i++){
      if(exists($proteins{$translator[$i]}{$motif})){
          die "ERROR: Repeated sequence\n";
      }
      $proteins{$translator[$i]}{$motif} = $z_intensities[$i];
    }
}


for(my $hm = 1; $hm < 11; $hm++){#loop can be eliminated when only interested in
performing stepwise on one set of proteins
    my %sig_proteins = ();
    open(FILE, "Significant_Cutoff_LEI_Z_Regressions_".$hm);#file changes depending
on which hexmut is of interest
    while(<FILE>){
      chomp $_;
      my @props = split(/\t/, $_);
      if($props[0] ne "Cons"){
          my @var_props = ($props[0], $props[1]);
          if(!exists($sig_proteins{$var_props[1]})){
            $sig_proteins{$var_props[1]} = [];
          }
          push(@{$sig_proteins{$var_props[1]}}, $var_props[0]);
      }
    }
    close(FILE);

    foreach my $pos_key (keys %sig_proteins){
      my @sorted_proteins = sort(@{$sig_proteins{$pos_key}});
      $sig_proteins{$pos_key} = \@sorted_proteins;
    }


    open(FILE, "HM90summVNewLEISC");
    my @molecules;
    while(<FILE>){
      chomp $_;
      my @props = split(/\t/, $_);
      if($props[1] == $hm){
          if($props[2] == 0){
            my $temp_mol = new Molecule($props[0], $props[1], $props[2], "", "",
$props[3], $props[4], $props[5], $props[6], "WT");
            $temp_mol->{_LEISC} = $props[7];
            push(@molecules, $temp_mol);
          }else{
            my @sub_bases = split("-", $props[10]);
            my $sub_type = 1;
            if(length($sub_bases[0]) == 1){
                $sub_type = 0;
            }
            my $wt_bases = $sub_bases[0];
            my $mut_bases = $sub_bases[1];
            my $temp_mol = new Molecule($props[0], $props[1], $props[2], $wt_bases,
$mut_bases, $props[3], $props[4], $props[5], $props[6], $sub_type);
            $temp_mol->{_LEISC} = $props[7];
            for(my $i = 0; $i < 100; $i++){
                push(@{$temp_mol->{_rbpss}}, []);
            }
            push(@molecules, $temp_mol);
```

```
      }
  }
}
close(FILE);
my @matrix_ids;
push(@matrix_ids, "Cons");
foreach my $position (sort {$a<=>$b} keys %sig_proteins){
  foreach my $prot (@{$sig_proteins{$position}}){
      push(@matrix_ids, ($prot.",".$position));
  }
}
my @leisc_vector;
my @z_score_matrix;
my $forced_zero_counter = 0;
foreach my $molecule (@molecules){
  my $leisc = log2($molecule->{_EI});
  my @z_scores;
  push(@z_scores, 1);
  foreach my $position (sort {$a<=>$b} keys %sig_proteins){
      my $pos = 0;
      if($position > 0){
        $pos = $position+22;
      }else{
        $pos = $position+23;
      }
      foreach my $prot (@{$sig_proteins{$position}}){
        my $heptamer = substr($molecule->{_seq}, $pos, 7);
        if(exists($proteins{$prot}{$heptamer})){
            push(@z_scores, $proteins{$prot}{$heptamer});
        }else{
            push(@z_scores, 0);
            $forced_zero_counter++;
        }
      }
  }
  push(@leisc_vector, $leisc);
  push(@z_score_matrix, \@z_scores);
}

my $start = time();
my $n_ob = scalar(@leisc_vector);
my $n_params = 1;
my @final_model;
my @final_model_thetas;
my @final_model_stdes;
my @final_model_ids;
for(my $i = 0; $i < $n_ob; $i++){
  push(@final_model, [1]);
}
push(@final_model_ids, "Cons");
my $min_p = 0;
my $r2 = 0;
my $adj_r2 = 0;
my $rss = 0;
my $sst = 0;
my $f_for_model = 0;
my $p_for_model = 1;
my $added = 0;
my $alpha = .01;
```

```
    while($n_ob > $n_params && $min_p <= $alpha){
      my $max_t = 0;
      my $max_t_index = -1;
      my $max_t_p = 1;
      for(my $p = 1; $p < scalar(@matrix_ids); $p++){
          my @temp_model = @{copy_matrix(\@final_model)};
          my @temp_model_ids = @{copy_array(\@final_model_ids)};
          @temp_model = @{copy_column(\@temp_model, \@z_score_matrix, $p)};
          push(@temp_model_ids, $matrix_ids[$p]);
          my $reg = Statistics::Regression->new("", \@temp_model_ids);
          for(my $i = 0; $i < $n_ob; $i++){
            $reg->include($leisc_vector[$i], $temp_model[$i]);
          }
          my @temp_model_thetas = $reg->theta();
          my @temp_model_stdes = $reg->standarderrors();
          my $t_stat = abs($temp_model_thetas[(scalar(@temp_model_thetas)-
1)]/$temp_model_stdes[(scalar(@temp_model_thetas)-1)]);
          my $p_val = Statistics::Distributions::tprob($n_ob-scalar(@temp_model_ids),
$t_stat);
          $p_val = $p_val*2;
          if($t_stat > $max_t){
            $max_t = $t_stat;
            $max_t_index = $p;
            $max_t_p = $p_val;
          }
      }
      $min_p = $max_t_p;
      if($min_p <= $alpha){
          @final_model = @{copy_column(\@final_model, \@z_score_matrix,
$max_t_index)};
          push(@final_model_ids, $matrix_ids[$max_t_index]);
          @matrix_ids = @{delete_from_array(\@matrix_ids, $max_t_index)};
          @z_score_matrix = @{delete_column(\@z_score_matrix, $max_t_index)};
          my $reg2 = Statistics::Regression->new("", \@final_model_ids);
          for(my $i = 0; $i < $n_ob; $i++){
            $reg2->include($leisc_vector[$i], $final_model[$i]);
          }
          @final_model_thetas = $reg2->theta();
          @final_model_stdes = $reg2->standarderrors();
          $n_params = scalar(@final_model_ids);
          my @now_insig_prots;
          for(my $i = 1; $i < $n_params; $i++){
            my $t_stat = abs($final_model_thetas[$i]/$final_model_stdes[$i]);
            my $p_val = Statistics::Distributions::tprob($n_ob-
scalar(@final_model_ids), $t_stat);
            $p_val = $p_val*2;
            if($p_val > $alpha){
                push(@now_insig_prots, $i);
            }
          }
          my $eliminated = scalar(@now_insig_prots);
          @final_model = @{delete_columns(\@final_model, \@now_insig_prots)};
          @final_model_ids = @{delete_many_from_array(\@final_model_ids,
\@now_insig_prots)};
          $n_params = scalar(@final_model_ids);
          $r2 = $reg2->rsq();
          $adj_r2 = $reg2->adjrsq();
          $rss = ($reg2->sigmasq())*($n_ob-$n_params);
          $sst = $reg2->sst();
```

```
          $f_for_model = (($sst-$rss)/($n_params-1))/($rss/($n_ob-$n_params));
          $p_for_model = Statistics::Distributions::fprob(($n_params-1), ($n_ob-
$n_params), $f_for_model);
          $added++;
          print $added."\t".$min_p."\n";

      }
    }

    open(WRITE,
">Full_Exon_Models_Stepwise_Redone/Full_Exon_Model_Sig_Only_Variables_LEI_Stepwise_".
$hm."_0.01");
    for(my $i = 0; $i < $n_params; $i++){
      my $t_stat = $final_model_thetas[$i]/$final_model_stdes[$i];
      my $abs_t_stat = abs($t_stat);
      my $p_val = Statistics::Distributions::tprob($n_ob-scalar(@final_model_ids),
$abs_t_stat);
      print WRITE
$final_model_ids[$i]."\t".$final_model_thetas[$i]."\t".$final_model_stdes[$i]."\t".$t
_stat."\t".($p_val*2)."\t".$r2."\t".$adj_r2."\t".$p_for_model."\n";
    }
    close(WRITE);
    my $end = time();
    my $elapsed = $end-$start;
    print $elapsed."\n";


    my @predicted;
    my @observed;
    open(WRITE,
">Full_Exon_Models_Stepwise_Redone/Full_Exon_Model_LEI_Sig_Only_Variables_Added_".$hm
."_CV_Vals_0.01");
    for(my $o = 0; $o < $n_ob; $o++){
      my @cv_matrix = @{copy_matrix(\@final_model)};
      my @cv_leiscs = @{copy_array(\@leisc_vector)};
      my $tp_leis = $cv_leiscs[$o];
      my @tp_zs = @{$cv_matrix[$o]};
      @cv_leiscs = @{delete_from_array(\@cv_leiscs, $o)};
      @cv_matrix = @{delete_from_array(\@cv_matrix, $o)};
      my $cv_reg = Statistics::Regression->new("", \@final_model_ids);
      for(my $j = 0; $j < scalar(@cv_leiscs); $j++){
          $cv_reg->include($cv_leiscs[$j], $cv_matrix[$j]);
      }

      my @thetas = $cv_reg->theta();

      my $cv_predicted = 0;

      for(my $p = 0; $p < scalar(@final_model_ids); $p++){
          $cv_predicted += $tp_zs[$p]*$thetas[$p];
      }

      print WRITE $tp_leis."\t".$cv_predicted."\n";
      push(@predicted, $cv_predicted);
      push(@observed, $tp_leis);

    }

    close(WRITE);
```

```
    my $MSE = calc_mse(\@predicted,\@observed);
    print $hm."\t".$MSE."\n";
}

sub delete_from_array{
    my @array = @{$_[0]};
    my $index_to_delete = $_[1];
    my @new_array;
    my $n = scalar(@array);
    for(my $v = 0; $v < $n; $v++){
      if($v != $index_to_delete){
          push(@new_array, $array[$v]);
      }
    }
    return \@new_array;
}

sub copy_array{
    my @array = @{$_[0]};
    my @new_array;
    foreach my $value (@array){
      push(@new_array, $value);
    }
    return \@new_array;
}

sub copy_matrix{
    my @matrix = @{$_[0]};
    my @new_matrix;
    my $n_rows = scalar(@matrix);
    for(my $r = 0; $r < $n_rows; $r++){
      my $n_columns = scalar(@{$matrix[$r]});
      $new_matrix[$r] = [];
      for(my $c = 0; $c < $n_columns; $c++){
          $new_matrix[$r][$c] = $matrix[$r][$c];
      }
    }
    return \@new_matrix;
}

sub copy_column{
    my @to_matrix = @{$_[0]};
    my @from_matrix = @{$_[1]};
    my $from_index = $_[2];
    for(my $r = 0; $r < scalar(@to_matrix); $r++){
      push(@{$to_matrix[$r]}, $from_matrix[$r][$from_index]);
    }
    return \@to_matrix;
}

sub add_protein{
    my @matrix = @{$_[0]};
    my @column = @{$_[1]};
    my $n_rows = scalar(@matrix);
    for(my $r = 0; $r < $n_rows; $r++){
      push(@{$matrix[$r]}, $column[$r]);
    }
    return \@matrix;
}
```

```
sub delete_column{
    my @matrix = @{$_[0]};
    my $column_to_delete = $_[1];
    my @new_matrix;
    my $n_rows = scalar(@matrix);
    my $n_columns= scalar(@{$matrix[0]});
    for(my $r = 0; $r < $n_rows; $r++){
      $new_matrix[$r] = [];
      for(my $c = 0; $c < $n_columns; $c++){
          if($c != $column_to_delete){
            push(@{$new_matrix[$r]}, $matrix[$r][$c]);
          }
      }
    }
    return \@new_matrix;
}

sub correl{
    my @puppy_array = @{$_[0]};
    my @d_cs_array = @{$_[1]};
    my $mean_1 = average(\@puppy_array);
    my $mean_2 = average(\@d_cs_array);
    my $stddev_1 = staddev(\@puppy_array);
    my $stddev_2 = staddev(\@d_cs_array);
    my $n = scalar(@puppy_array);
    my @z_scores_1; my @z_scores_2;
    for(my $l = 0; $l < $n; $l++){
      my $z_1 = ($puppy_array[$l]-$mean_1)/$stddev_1;
      push(@z_scores_1, $z_1);
      my $z_2 = ($d_cs_array[$l]-$mean_2)/$stddev_2;
      push(@z_scores_2, $z_2);
    }
    my $product_sum = 0;
    for(my $l = 0; $l < $n; $l++){
      $product_sum = $product_sum + ($z_scores_1[$l]*$z_scores_2[$l]);
    }
    my $r = $product_sum/($n-1);
    return $r;
}

#function to calculate the average of a set of values in an array
sub average{
    my @values = @{$_[0]};
    my $length = scalar(@values);
    my $total = 0;
    foreach my $value (@values){
      $total = $total + $value;
    }
    my $avg = $total/$length;
    return $avg
}
#function to calculate the standard deviation of a set of values in an array
#given that there exists an average function
sub staddev{
    my @values = @{$_[0]};
    my $avg = average(\@values);
    my $total = 0;
    foreach my $value (@values){
```

```
      $total = $total + ($value-$avg)**2;
    }
    my $length = scalar(@values);
    my $stdd = ($total/($length-1))**.5;
    return $stdd;
}

sub calc_mse{
    my @y_pred = @{$_[0]};
    my @y = @{$_[1]};
    my $n = scalar(@y_pred);
    my $ssd = 0;
    for(my $i = 0; $i < scalar(@y_pred); $i++){
      $ssd += (($y_pred[$i]-$y[$i])**2)
    }
    my $mse = $ssd/$n;
    return $mse;
}

sub log2{
    my $n = shift;
    return log($n)/log(2);
}

sub delete_many_from_array{
    my @array = @{$_[0]};
    my @indeces_to_delete = @{$_[1]};
    my %to_delete = ();
    foreach my $index (@indeces_to_delete){
      $to_delete{$index} = 1;
    }
    my @new_array;
    my $n = scalar(@array);
    for(my $v = 0; $v < $n; $v++){
      if(!exists($to_delete{$v})){
          push(@new_array, $array[$v]);
      }
    }
    return \@new_array;
}

sub delete_columns{
    my @matrix = @{$_[0]};
    my @columns_to_delete = @{$_[1]};
    my %to_delete = ();
    foreach my $index (@columns_to_delete){
      $to_delete{$index} = 1;
    }
    my @new_matrix;
    my $n_rows = scalar(@matrix);
    my $n_columns= scalar(@{$matrix[0]});

    for(my $r = 0; $r < $n_rows; $r++){
      $new_matrix[$r] = [];
      for(my $c = 0; $c < $n_columns; $c++){
          if(!exists($to_delete{$c})){
            push(@{$new_matrix[$r]}, $matrix[$r][$c]);
          }
      }
```

```
    }
    return \@new_matrix;
}
```

```
################################################################################
########
# StepwiseCombined.pl
#
# Performs the same task as Stepwise.pl but builds a single multiple linear model for
all
# hexmuts combined, using only proteins that were found significant in each hexmut
multiple
# linear model (as selected by prior stepwise procedure) as a starting set
################################################################################
########
use warnings;
use strict;
use Molecule;
use Statistics::Distributions;
use Statistics::Regression;
my %protein_translator = ();
open(FILE, "RBP_Information.txt") or die "ERROR: $!\n";
my @lns = <FILE>;
close(FILE);
shift @lns;
chomp @lns;
foreach my $ln (@lns){
    my @props = split(/\t/, $ln);
    if(!exists($protein_translator{$props[3]})){
          $protein_translator{$props[3]} = "";
    }
    $protein_translator{$props[3]} = $protein_translator{$props[3]}.$props[6].",";
}

open(FILE, "Zscores.txt") or die "ERROR: $!\n";
my @lines = <FILE>;
close(FILE);
chomp @lines;
my $id_row = shift(@lines);
my @ids = split(/\t/, $id_row);

shift @ids;
my %proteins = ();
my $id_index = 0;
my @translator;
foreach my $id (@ids){
    $translator[$id_index] = $id;
    $proteins{$id} = ();
    $id_index++;
}
my $n_protein_ids = scalar(@translator);
my $hept_counter = 0;
foreach my $line (@lines){
    my @z_intensities = split(/\t/, $line);
    my $motif = shift(@z_intensities);
    $motif =~ s/U/T/g;
    my $n_proteins = scalar(@z_intensities);
    $hept_counter++;
    for(my $i = 0; $i < $n_proteins; $i++){
      if(exists($proteins{$translator[$i]}{$motif})){
          die "ERROR: Repeated sequence\n";
      }
      $proteins{$translator[$i]}{$motif} = $z_intensities[$i];
```

```
    }
}
my %sig_proteins = ();
open(FILE, "All_HMs_Proteins_0.01_no_3");
while(<FILE>){
    chomp $_;
    my @props = split(/\t/, $_);
    if($props[0] ne "Cons"){
      my @var_props = split(",", $props[0]);
      if(!exists($sig_proteins{$var_props[1]})){
          $sig_proteins{$var_props[1]} = [];
      }
      push(@{$sig_proteins{$var_props[1]}}, $var_props[0]);
    }
}
close(FILE);

foreach my $pos_key (keys %sig_proteins){
    my @sorted_proteins = sort(@{$sig_proteins{$pos_key}});
    $sig_proteins{$pos_key} = \@sorted_proteins;
}

my @molecules;

open(FILE, "HM90summVNewLEISC");
while(<FILE>){
    chomp $_;
    my @props = split(/\t/, $_);
    if($props[1] != 3){ #gets rid of HM3
      if($props[2] == 0){
          my $temp_mol = new Molecule($props[0], $props[1], $props[2], "", "",
$props[3], $props[4], $props[5], $props[6], "WT");
          $temp_mol->{_LEISC} = $props[7];
          push(@molecules, $temp_mol);
      }else{
          my @sub_bases = split("-", $props[10]);
          my $sub_type = 1;
            if(length($sub_bases[0]) == 1){
                $sub_type = 0;
            }
          my $wt_bases = $sub_bases[0];
          my $mut_bases = $sub_bases[1];
          my $temp_mol = new Molecule($props[0], $props[1], $props[2], $wt_bases,
$mut_bases, $props[3], $props[4], $props[5], $props[6], $sub_type);
          $temp_mol->{_LEISC} = $props[7];
          for(my $i = 0; $i < 100; $i++){
                push(@{$temp_mol->{_rbpss}}, []);
          }
          push(@molecules, $temp_mol);
          }
    }
}
close(FILE);


my @matrix_ids;
push(@matrix_ids, "Cons");
foreach my $position (sort {$a<=>$b} keys %sig_proteins){
    foreach my $prot (@{$sig_proteins{$position}}){
```

```
      push(@matrix_ids, ($prot.",".$position));
    }
}
my @leisc_vector;
my @z_score_matrix;
my $forced_zero_counter = 0;
foreach my $molecule (@molecules){
    my $leisc = log2($molecule->{_EI});
    my @z_scores;
    push(@z_scores, 1);
    foreach my $position (sort {$a<=>$b} keys %sig_proteins){
      my $pos = 0;
      if($position > 0){
          $pos = $position+22;
      }else{
          $pos = $position+23;
      }
      foreach my $prot (@{$sig_proteins{$position}}){
          my $heptamer = substr($molecule->{_seq}, $pos, 7);
          if(exists($proteins{$prot}{$heptamer})){
            push(@z_scores, $proteins{$prot}{$heptamer});
          }else{
            push(@z_scores, 0);
            $forced_zero_counter++;
          }
      }
    }
    push(@leisc_vector, $leisc);
    push(@z_score_matrix, \@z_scores);
}


print "Zero forced: ".$forced_zero_counter."\n";

my $start = time();

my $n_ob = scalar(@leisc_vector);
my $n_params = 1;
my @final_model;
my @final_model_thetas;
my @final_model_stdes;
my @final_model_ids;
for(my $i = 0; $i < $n_ob; $i++){
    push(@final_model, [1]);
}
push(@final_model_ids, "Cons");
my $min_p = 0;
my $r2 = 0;
my $adj_r2 = 0;
my $rss = 0;
my $sst = 0;
my $f_for_model = 0;
my $p_for_model = 1;
my $added = 0;
my $alpha = 0.01;
while($n_ob > $n_params && $min_p <= $alpha){
    my $max_t = 0;
    my $max_t_index = -1;
    my $max_t_p = 1;
```

```
    for(my $p = 1; $p < scalar(@matrix_ids); $p++){
      my @temp_model = @{copy_matrix(\@final_model)};
      my @temp_model_ids = @{copy_array(\@final_model_ids)};
      @temp_model = @{copy_column(\@temp_model, \@z_score_matrix, $p)};
      push(@temp_model_ids, $matrix_ids[$p]);
      my $reg = Statistics::Regression->new("", \@temp_model_ids);
      for(my $i = 0; $i < $n_ob; $i++){
          $reg->include($leisc_vector[$i], $temp_model[$i]);
      }
      my @temp_model_thetas = $reg->theta();
      my @temp_model_stdes = $reg->standarderrors();
      my $t_stat = abs($temp_model_thetas[(scalar(@temp_model_thetas)-
1)]/$temp_model_stdes[(scalar(@temp_model_thetas)-1)]);
      my $p_val = Statistics::Distributions::tprob($n_ob-scalar(@temp_model_ids),
$t_stat);
      $p_val = $p_val*2;
      if($t_stat > $max_t){
            $max_t = $t_stat;
            $max_t_index = $p;
            $max_t_p = $p_val;
      }
    }
    $min_p = $max_t_p;
    if($min_p <= $alpha){
      @final_model = @{copy_column(\@final_model, \@z_score_matrix, $max_t_index)};
      push(@final_model_ids, $matrix_ids[$max_t_index]);
      @matrix_ids = @{delete_from_array(\@matrix_ids, $max_t_index)};
      @z_score_matrix = @{delete_column(\@z_score_matrix, $max_t_index)};
      my $reg2 = Statistics::Regression->new("", \@final_model_ids);
      for(my $i = 0; $i < $n_ob; $i++){
          $reg2->include($leisc_vector[$i], $final_model[$i]);
      }
      @final_model_thetas = $reg2->theta();
      @final_model_stdes = $reg2->standarderrors();
      $n_params = scalar(@final_model_ids);
      my @now_insig_prots;
      for(my $i = 1; $i < $n_params; $i++){
          my $t_stat = abs($final_model_thetas[$i]/$final_model_stdes[$i]);
          my $p_val = Statistics::Distributions::tprob($n_ob-
scalar(@final_model_ids), $t_stat);
          $p_val = $p_val*2;
          if($p_val > $alpha){
            push(@now_insig_prots, $i);
          }
      }
      my $eliminated = scalar(@now_insig_prots);
      @final_model = @{delete_columns(\@final_model, \@now_insig_prots)};
      @final_model_ids = @{delete_many_from_array(\@final_model_ids,
\@now_insig_prots)};
      $n_params = scalar(@final_model_ids);
      $r2 = $reg2->rsq();
      $adj_r2 = $reg2->adjrsq();
      $rss = ($reg2->sigmasq())*($n_ob-$n_params);
      $sst = $reg2->sst();
      $f_for_model = (($sst-$rss)/($n_params-1))/($rss/($n_ob-$n_params));
      $p_for_model = Statistics::Distributions::fprob(($n_params-1), ($n_ob-
$n_params), $f_for_model);
      $added++;
      print $added."\t".$min_p."\n";
```

```
    }
}

open(WRITE,
">Full_Exon_Models_Stepwise_Redone/Full_Exon_Model_LEI_Stepwise_0.01_no_3");
for(my $i = 0; $i < $n_params; $i++){
    my $t_stat = $final_model_thetas[$i]/$final_model_stdes[$i];
    my $abs_t_stat = abs($t_stat);
    my $p_val = Statistics::Distributions::tprob($n_ob-scalar(@final_model_ids),
$abs_t_stat);
    print WRITE
$final_model_ids[$i]."\t".$final_model_thetas[$i]."\t".$final_model_stdes[$i]."\t".$t
_stat."\t".($p_val*2)."\t".$r2."\t".$adj_r2."\t".$p_for_model."\n";
}
close(WRITE);
my $end = time();
my $elapsed = $end-$start;
print $elapsed."\n";


my @predicted;
my @observed;
open(WRITE,
">Full_Exon_Models_Stepwise_Redone/Full_Exon_Model_LEI_CV_Vals_0.01_no_3");
for(my $o = 0; $o < $n_ob; $o++){
    my @cv_matrix = @{copy_matrix(\@final_model)};
    my @cv_leiscs = @{copy_array(\@leisc_vector)};
    my $tp_leis = $cv_leiscs[$o];
    my @tp_zs = @{$cv_matrix[$o]};
    @cv_leiscs = @{delete_from_array(\@cv_leiscs, $o)};
    @cv_matrix = @{delete_from_array(\@cv_matrix, $o)};
    my $cv_reg = Statistics::Regression->new("", \@final_model_ids);
    for(my $j = 0; $j < scalar(@cv_leiscs); $j++){
      $cv_reg->include($cv_leiscs[$j], $cv_matrix[$j]);
    }

    my @thetas = $cv_reg->theta();

    my $cv_predicted = 0;

    for(my $p = 0; $p < scalar(@final_model_ids); $p++){
      $cv_predicted += $tp_zs[$p]*$thetas[$p];
    }

    print WRITE $tp_leis."\t".$cv_predicted."\n";
    push(@predicted, $cv_predicted);
    push(@observed, $tp_leis);

}

close(WRITE);
my $MSE = calc_mse(\@predicted,\@observed);
print $MSE."\n";

sub delete_from_array{
    my @array = @{$_[0]};
    my $index_to_delete = $_[1];
    my @new_array;
```

```
    my $n = scalar(@array);
    for(my $v = 0; $v < $n; $v++){
      if($v != $index_to_delete){
          push(@new_array, $array[$v]);
      }
    }
    return \@new_array;
}

sub copy_array{
    my @array = @{$_[0]};
    my @new_array;
    foreach my $value (@array){
      push(@new_array, $value);
    }
    return \@new_array;
}

sub copy_matrix{
    my @matrix = @{$_[0]};
    my @new_matrix;
    my $n_rows = scalar(@matrix);
    for(my $r = 0; $r < $n_rows; $r++){
      my $n_columns = scalar(@{$matrix[$r]});
      $new_matrix[$r] = [];
      for(my $c = 0; $c < $n_columns; $c++){
          $new_matrix[$r][$c] = $matrix[$r][$c];
      }
    }
    return \@new_matrix;
}

sub copy_column{
    my @to_matrix = @{$_[0]};
    my @from_matrix = @{$_[1]};
    my $from_index = $_[2];
    for(my $r = 0; $r < scalar(@to_matrix); $r++){
      push(@{$to_matrix[$r]}, $from_matrix[$r][$from_index]);
    }
    return \@to_matrix;
}

sub add_protein{
    my @matrix = @{$_[0]};
    my @column = @{$_[1]};
    my $n_rows = scalar(@matrix);
    for(my $r = 0; $r < $n_rows; $r++){
      push(@{$matrix[$r]}, $column[$r]);
    }
    return \@matrix;
}

sub delete_column{
    my @matrix = @{$_[0]};
    my $column_to_delete = $_[1];
    my @new_matrix;
    my $n_rows = scalar(@matrix);
    my $n_columns= scalar(@{$matrix[0]});
    for(my $r = 0; $r < $n_rows; $r++){
```

```
      $new_matrix[$r] = [];
      for(my $c = 0; $c < $n_columns; $c++){
          if($c != $column_to_delete){
            push(@{$new_matrix[$r]}, $matrix[$r][$c]);
          }
      }
    }
    return \@new_matrix;
}

sub correl{
    my @puppy_array = @{$_[0]};
    my @d_cs_array = @{$_[1]};
    my $mean_1 = average(\@puppy_array);
    my $mean_2 = average(\@d_cs_array);
    my $stddev_1 = staddev(\@puppy_array);
    my $stddev_2 = staddev(\@d_cs_array);
    my $n = scalar(@puppy_array);
    my @z_scores_1; my @z_scores_2;
    for(my $l = 0; $l < $n; $l++){
      my $z_1 = ($puppy_array[$l]-$mean_1)/$stddev_1;
      push(@z_scores_1, $z_1);
      my $z_2 = ($d_cs_array[$l]-$mean_2)/$stddev_2;
      push(@z_scores_2, $z_2);
    }
    my $product_sum = 0;
    for(my $l = 0; $l < $n; $l++){
      $product_sum = $product_sum + ($z_scores_1[$l]*$z_scores_2[$l]);
    }
    my $r = $product_sum/($n-1);
    return $r;
}

#function to calculate the average of a set of values in an array
sub average{
    my @values = @{$_[0]};
    my $length = scalar(@values);
    my $total = 0;
    foreach my $value (@values){
      $total = $total + $value;
    }
    my $avg = $total/$length;
    return $avg
}
#function to calculate the standard deviation of a set of values in an array
#given that there exists an average function
sub staddev{
    my @values = @{$_[0]};
    my $avg = average(\@values);
    my $total = 0;
    foreach my $value (@values){
      $total = $total + ($value-$avg)**2;
    }
    my $length = scalar(@values);
    my $stdd = ($total/($length-1))**.5;
    return $stdd;
}

sub calc_mse{
```

```
    my @y_pred = @{$_[0]};
    my @y = @{$_[1]};
    my $n = scalar(@y_pred);
    my $ssd = 0;
    for(my $i = 0; $i < scalar(@y_pred); $i++){
      $ssd += (($y_pred[$i]-$y[$i])**2)
    }
    my $mse = $ssd/$n;
    return $mse;
}

sub log2{
    my $n = shift;
    return log($n)/log(2);
}

sub delete_many_from_array{
    my @array = @{$_[0]};
    my @indeces_to_delete = @{$_[1]};
    my %to_delete = ();
    foreach my $index (@indeces_to_delete){
      $to_delete{$index} = 1;
    }
    my @new_array;
    my $n = scalar(@array);
    for(my $v = 0; $v < $n; $v++){
      if(!exists($to_delete{$v})){
          push(@new_array, $array[$v]);
      }
    }
    return \@new_array;
}

sub delete_columns{
    my @matrix = @{$_[0]};
    my @columns_to_delete = @{$_[1]};
    my %to_delete = ();
    foreach my $index (@columns_to_delete){
      $to_delete{$index} = 1;
    }
    my @new_matrix;
    my $n_rows = scalar(@matrix);
    my $n_columns= scalar(@{$matrix[0]});

    for(my $r = 0; $r < $n_rows; $r++){
      $new_matrix[$r] = [];
      for(my $c = 0; $c < $n_columns; $c++){
          if(!exists($to_delete{$c})){
            push(@{$new_matrix[$r]}, $matrix[$r][$c]);
          }
      }
    }
    return \@new_matrix;
}
```

```
################################################################################################
########
# TenFoldCV.pl
#
# Performs a ten-fold cross validation on a multiple linear model as specified
################################################################################################
########
use warnings;
use strict;
use Molecule;
use Statistics::Distributions;
use Statistics::Regression;
use Math::Random::Secure qw(irand);


open(WRITE2, ">HM_10FCV_Selected_Full_Result_0.01");
#for(my $hm = 1; $hm < 11; $hm++){
#for(my $count = 0; $count < 10; $count++){
open(FILE, "Full_Exon_Model_LEI_Stepwise_0.01");
my %sig_proteins = ();
while(<FILE>){
    chomp $_;
    my @props = split(/\t/, $_);
    if($props[0] ne "Cons"){
      my @var_props = split(",", $props[0]);
      if(!exists($sig_proteins{$var_props[1]})){
          $sig_proteins{$var_props[1]} = [];
      }
      push(@{$sig_proteins{$var_props[1]}}, $var_props[0]);
    }
}
close(FILE);

foreach my $pos_key (keys %sig_proteins){
    my @sorted_proteins = sort(@{$sig_proteins{$pos_key}});
    $sig_proteins{$pos_key} = \@sorted_proteins;
}
open(FILE, "Zscores.txt") or die "ERROR: $!\n";
my @lines = <FILE>;
close(FILE);
chomp @lines;
my $id_row = shift(@lines);
my @ids = split(/\t/, $id_row);

shift @ids;
my %proteins = ();
my $id_index = 0;
my @translator;
foreach my $id (@ids){
    #print $id."\n";
    $translator[$id_index] = $id;
    $proteins{$id} = ();
    $id_index++;
}
my $n_protein_ids = scalar(@translator);
my $hept_counter = 0;
foreach my $line (@lines){
    my @z_intensities = split(/\t/, $line);
    my $motif = shift(@z_intensities);
```

```
    $motif =~ s/U/T/g;
    my $n_proteins = scalar(@z_intensities);
    $hept_counter++;
    for(my $i = 0; $i < $n_proteins; $i++){
      if(exists($proteins{$translator[$i]}{$motif})){
          die "ERROR: Repeated sequence\n";
      }
      $proteins{$translator[$i]}{$motif} = $z_intensities[$i];
    }
}


open(FILE, "HM90summVNewLEISC");
my @molecules;
while(<FILE>){
    chomp $_;
    my @props = split(/\t/, $_);
    #if($props[1] == $hm){ #gets rid of HM3
      if($props[2] == 0){
          my $temp_mol = new Molecule($props[0], $props[1], $props[2], "", "",
$props[3], $props[4], $props[5], $props[6], "WT");
          $temp_mol->{_LEISC} = $props[7];
          push(@molecules, $temp_mol);
      }else{
          my @sub_bases = split("-", $props[10]);
          my $sub_type = 1; #since most are DB subs, it's the default
          if(length($sub_bases[0]) == 1){
            $sub_type = 0;
          }
          my $wt_bases = $sub_bases[0];
          my $mut_bases = $sub_bases[1];
          my $temp_mol = new Molecule($props[0], $props[1], $props[2], $wt_bases,
$mut_bases, $props[3], $props[4], $props[5], $props[6], $sub_type);
          $temp_mol->{_LEISC} = $props[7];
          for(my $i = 0; $i < 100; $i++){
            push(@{$temp_mol->{_rbpss}}, []);
          }
          push(@molecules, $temp_mol);
      }
    #}
}
close(FILE);
my @model_ids;
push(@model_ids, "Cons");
foreach my $position (sort {$a<=>$b} keys %sig_proteins){
    foreach my $prot (@{$sig_proteins{$position}}){
      push(@model_ids, ($prot.",".$position));
    }
}
my @leisc_vector;
my @z_score_matrix;
my $forced_zero_counter = 0;
foreach my $molecule (@molecules){
    my $leisc = $molecule->{_LEISC};
    my @z_scores;
    push(@z_scores, 1);
    foreach my $position (sort {$a<=>$b} keys %sig_proteins){
      my $pos = 0;
      if($position > 0){
```

```
            $pos = $position+22;
        }else{
            $pos = $position+23;
        }
        foreach my $prot (@{$sig_proteins{$position}}){
            my $heptamer = substr($molecule->{_seq}, $pos, 7);
            if(exists($proteins{$prot}{$heptamer})){
              push(@z_scores, $proteins{$prot}{$heptamer});
            }else{
              push(@z_scores, 0);
              $forced_zero_counter++;
            }
        }
    }
    push(@leisc_vector, $leisc);
    push(@z_score_matrix, \@z_scores);
}

my $n_ob = scalar(@leisc_vector);
my $n_params = scalar(@model_ids);
print $n_params."\n";
my $reg = Statistics::Regression->new("", \@model_ids);
for(my $i = 0; $i < $n_ob; $i++){
    $reg->include($leisc_vector[$i], $z_score_matrix[$i]);
}
my $r2 = $reg->rsq();
my $adj_r2 = $reg->adjrsq();
my @model_thetas = $reg->theta();
my @model_stdes = $reg->standarderrors();
my $rss = ($reg->sigmasq())*($n_ob-$n_params);
my $sst = $reg->sst();
my $f = (($sst-$rss)/($n_params-1))/($rss/($n_ob-$n_params));
my $p = Statistics::Distributions::fprob(($n_params-1), ($n_ob-$n_params), $f);

open(WRITE, ">Full_Exon_Model_Selected_All_0.01_CV_Vals");
my @matrix_folds;
my @leisc_folds;
my $fold_size = ($n_ob/10);
my $current_size = $n_ob;
my $has_next = 1;
$fold_size = sprintf("%.0f", $fold_size);
for(my $f = 0; $f < 10; $f++){
    push(@matrix_folds, []);
    push(@leisc_folds, []);
    for(my $e = 0; $e < $fold_size && $has_next; $e++){
      my $rand_index = irand($current_size);
      push(@{$leisc_folds[$f]}, $leisc_vector[$rand_index]);
      push(@{$matrix_folds[$f]}, $z_score_matrix[$rand_index]);
      @leisc_vector = @{delete_from_array(\@leisc_vector, $rand_index)};
      @z_score_matrix = @{delete_from_array(\@z_score_matrix, $rand_index)};
      $current_size = scalar(@leisc_vector);
      if($current_size == 0){
          $has_next = 0;
      }

    }
}
```

```
my @predicted;
my @observed;
my $g_cv_reg = Statistics::Regression->new("", ["b0","b1"]);
for(my $f = 0; $f < 10; $f++){
    my @tp_leiscs;
    my @tp_zs;
    my @tu_leiscs;
    my @tu_zs;
    #leave one fold out, and merge the other nine
    for(my $add_f = 0; $add_f < 10; $add_f++){
      if($add_f == $f){
          @tp_leiscs = @{copy_array($leisc_folds[$add_f])};
          @tp_zs = @{copy_matrix($matrix_folds[$add_f])};
      }else{
          for(my $o = 0; $o < scalar(@{$leisc_folds[$add_f]}); $o++){
            push(@tu_leiscs, $leisc_folds[$add_f][$o]);
            push(@tu_zs, $matrix_folds[$add_f][$o]);
          }
      }
    }
    #now use the nine folds to make a models
    my $cv_reg = Statistics::Regression->new("", \@model_ids);
    for(my $j = 0; $j < scalar(@tu_leiscs); $j++){
      $cv_reg->include($tu_leiscs[$j], $tu_zs[$j]);
    }
    my @thetas = $cv_reg->theta();

    for(my $tp = 0; $tp < scalar(@tp_leiscs); $tp++){
      my $cv_predicted = 0;
      for(my $p = 0; $p < scalar(@model_ids); $p++){
          $cv_predicted += $tp_zs[$tp][$p]*$thetas[$p];
      }
      $g_cv_reg->include($tp_leiscs[$tp], [1, $cv_predicted]);
      print WRITE $tp_leiscs[$tp]."\t".$cv_predicted."\n";
      push(@predicted, $cv_predicted);
      push(@observed, $tp_leiscs[$tp]);
    }
}
my $r_2 = $g_cv_reg->rsq();
my @thetas = $g_cv_reg->theta();
my $MSE = calc_mse(\@predicted,\@observed);
print WRITE2 $thetas[0]."\t".$thetas[1]."\t".$r_2."\t".$MSE."\n";
close(WRITE);

#}
#}
close(WRITE2);

sub delete_from_array{
    my @array = @{$_[0]};
    my $index_to_delete = $_[1];
    my @new_array;
    my $n = scalar(@array);
    for(my $v = 0; $v < $n; $v++){
      if($v != $index_to_delete){
          push(@new_array, $array[$v]);
      }
    }
    return \@new_array;
```

```
}

sub copy_array{
    my @array = @{$_[0]};
    my @new_array;
    foreach my $value (@array){
      push(@new_array, $value);
    }
    return \@new_array;
}

sub copy_matrix{
    my @matrix = @{$_[0]};
    my @new_matrix;
    my $n_rows = scalar(@matrix);
    for(my $r = 0; $r < $n_rows; $r++){
      my $n_columns = scalar(@{$matrix[$r]});
      $new_matrix[$r] = [];
      for(my $c = 0; $c < $n_columns; $c++){
          $new_matrix[$r][$c] = $matrix[$r][$c];
      }
    }
    return \@new_matrix;
}


sub copy_column{
    my @to_matrix = @{$_[0]};
    my @from_matrix = @{$_[1]};
    my $from_index = $_[2];
    for(my $r = 0; $r < scalar(@to_matrix); $r++){
      push(@{$to_matrix[$r]}, $from_matrix[$r][$from_index]);
    }
    return \@to_matrix;
}

sub add_protein{
    my @matrix = @{$_[0]};
    my @column = @{$_[1]};
    my $n_rows = scalar(@matrix);
    for(my $r = 0; $r < $n_rows; $r++){
      push(@{$matrix[$r]}, $column[$r]);
    }
    return \@matrix;
}

sub delete_column{
    my @matrix = @{$_[0]};
    my $column_to_delete = $_[1];
    my @new_matrix;
    my $n_rows = scalar(@matrix);
    my $n_columns= scalar(@{$matrix[0]});
    for(my $r = 0; $r < $n_rows; $r++){
      $new_matrix[$r] = [];
      for(my $c = 0; $c < $n_columns; $c++){
          if($c != $column_to_delete){
            push(@{$new_matrix[$r]}, $matrix[$r][$c]);
          }
      }
```

```
    }
    return \@new_matrix;
}

sub correl{
    my @puppy_array = @{$_[0]};
    my @d_cs_array = @{$_[1]};
    my $mean_1 = average(\@puppy_array);
    my $mean_2 = average(\@d_cs_array);
    my $stddev_1 = staddev(\@puppy_array);
    my $stddev_2 = staddev(\@d_cs_array);
    my $n = scalar(@puppy_array);
    my @z_scores_1; my @z_scores_2;
    for(my $l = 0; $l < $n; $l++){
      my $z_1 = ($puppy_array[$l]-$mean_1)/$stddev_1;
      push(@z_scores_1, $z_1);
      my $z_2 = ($d_cs_array[$l]-$mean_2)/$stddev_2;
      push(@z_scores_2, $z_2);
    }
    my $product_sum = 0;
    for(my $l = 0; $l < $n; $l++){
      $product_sum = $product_sum + ($z_scores_1[$l]*$z_scores_2[$l]);
    }
    my $r = $product_sum/($n-1);
    return $r;
}

#function to calculate the average of a set of values in an array
sub average{
    my @values = @{$_[0]};
    my $length = scalar(@values);
    my $total = 0;
    foreach my $value (@values){
      $total = $total + $value;
    }
    my $avg = $total/$length;

    return $avg
}
#function to calculate the standard deviation of a set of values in an array
#given that there exists an average function
sub staddev{
    my @values = @{$_[0]};
    my $avg = average(\@values);
    my $total = 0;
    foreach my $value (@values){
      $total = $total + ($value-$avg)**2;
    }

    my $length = scalar(@values);
    my $stdd = ($total/($length-1))**.5;

    return $stdd;
}

sub calc_mse{
    my @y_pred = @{$_[0]};
    my @y = @{$_[1]};
    my $n = scalar(@y_pred);
```

```
    my $ssd = 0;
    for(my $i = 0; $i < scalar(@y_pred); $i++){
      $ssd += (($y_pred[$i]-$y[$i])**2)
    }
    my $mse = $ssd/$n;
    return $mse;
}
```

```
################################################################################
########
# VarKmerContribution.pl
#
# Takes a k-mer length as an argument and estimates the effect of all unique k-mers
of the
# specified length in each hexmut set; A window of length k is started at the
beginnig of the
# sequence potentially mutated, the LEI of all unique k-mers created at that position
along
# the exon is averaged to estimate the net effect of mutating the sequence in that
space, and
# this value is subtracted from the individual LEI of each k-mer at that position,
yielding
# a net contribution of each unique k-mer to the LEI of the molecule it is in. The
window
# is then slid by one base, until the end of the exon. The contribution of any k-mers
that
# occurs more than once along the exon/set of mutations is averaged at the end of the
procedure,
# yielding the eLEI
################################################################################
########

use warnings;
use strict;
use Statistics::Regression;
use Statistics::Distributions;
use Molecule;
open(FILE, "HM90summVNewLEISC");
my $WINDOW = $ARGV[0];
my @molecules;
while(<FILE>){
    chomp $_;
    my @props = split(/\t/, $_);
    if($props[1] != 12){ #to exclude an HM as needed
      if($props[2] == 0){
          my $temp_mol = new Molecule($props[0], $props[1], $props[2], "", "",
$props[3], $props[4], $props[5], $props[6], "WT");
          $temp_mol->{_LEISC} = $props[7];
          push(@molecules, $temp_mol);
      }else{
          my @sub_bases = split("-", $props[10]);
          my $sub_type = 1;
          if(length($sub_bases[0]) == 1){
            $sub_type = 0;
          }
          my $wt_bases = $sub_bases[0];
          my $mut_bases = $sub_bases[1];
          my $temp_mol = new Molecule($props[0], $props[1], $props[2], $wt_bases,
$mut_bases, $props[3], $props[4], $props[5], $props[6], $sub_type);
          $temp_mol->{_LEISC} = $props[7];
          for(my $i = 0; $i < 100; $i++){
            push(@{$temp_mol->{_rbpss}}, []);
          }
          push(@molecules, $temp_mol);
      }
    }
}
```

```
close(FILE);

my %motif_pos = ();
my %hm_groups = ();
for(my $hm = 1; $hm < 11; $hm++){
    my $n_molecules_array = scalar(@molecules);
    my %regressions = ();
    for(my $i = 24; $i < (72-$WINDOW); $i++){
      my $exon_position;
      if($i > 22){
          $exon_position = $i-22;
      }else{
          $exon_position = $i-23;
      }
      my $wt_heptamer; my $wt_left; my $wt_right;
      foreach my $molecule (@molecules){
          if($molecule->{_hm} == $hm){
            if($molecule->{_mut_type} eq "WT"){
                $wt_heptamer = substr($molecule->{_seq}, $i, $WINDOW);
                $wt_left = substr($molecule->{_seq}, ($i-1), 1);
                $wt_right = substr($molecule->{_seq}, ($i+$WINDOW), 1);
                if(!exists($regressions{$exon_position})){
                  $regressions{$exon_position} = [[], [], [], [], []];
                }
                push(@{$regressions{$exon_position}[0]}, log2($molecule->{_EI}));
                push(@{$regressions{$exon_position}[1]}, $molecule->{_serial});
                push(@{$regressions{$exon_position}[2]}, $molecule->{_hm});
                push(@{$regressions{$exon_position}[3]}, $wt_heptamer);
                push(@{$regressions{$exon_position}[4]}, $molecule->{_seq});
            }else{
                my $mut_heptamer = substr($molecule->{_seq}, $i, $WINDOW);
                my $mut_left = substr($molecule->{_seq}, ($i-1), 1);
                my $mut_right = substr($molecule->{_seq}, ($i+$WINDOW), 1);
                if($mut_heptamer ne $wt_heptamer && $mut_left eq $wt_left &&
$mut_right eq $wt_right){
                  if(!exists($regressions{$exon_position})){
                      $regressions{$exon_position} = [[], [], [], [], []];
                  }
                  push(@{$regressions{$exon_position}[0]}, log2($molecule->{_EI}));
                  push(@{$regressions{$exon_position}[1]}, $molecule->{_serial});
                  push(@{$regressions{$exon_position}[2]}, $molecule->{_hm});
                  push(@{$regressions{$exon_position}[3]}, $mut_heptamer);
                  push(@{$regressions{$exon_position}[4]}, $molecule->{_seq});

                }
            }
          }
      }
    }
    my %results = ();

    foreach my $pos (sort {$a<=>$b} keys %regressions){
      my $g_avg = average($regressions{$pos}[0]);
      my $n = scalar(@{$regressions{$pos}[0]});
      print $n."\n";
      for(my $i = 0; $i < scalar(@{$regressions{$pos}[0]}); $i++){
          if(!exists($results{$regressions{$pos}[3][$i]})){
            $results{$regressions{$pos}[3][$i]} = {};
          }
```

```
          $results{$regressions{$pos}[3][$i]}{$pos} = $regressions{$pos}[0][$i]-
$g_avg;
      }
    }
    $hm_groups{$hm} = {};
    foreach my $key (sort keys %results){
      my @base_pos;
      for(my $pos = 1; $pos < 49; $pos++){
          if(exists($results{$key}{$pos})){
            push(@base_pos, $results{$key}{$pos});
          }
      }
      $hm_groups{$hm}{$key} = average(\@base_pos);
      $motif_pos{$key} = 1;

    }
}
open(WRITE, ">All_HM_".$WINDOW."NT_Avg_Effects");
print WRITE "HM";
foreach my $m_key (sort keys %motif_pos){
    print WRITE "\t".$m_key;
}
print WRITE "\n";
foreach my $hm_key (sort {$a<=>$b} keys %hm_groups){
    print WRITE $hm_key;
    foreach my $group (sort keys %motif_pos){
      if(exists($hm_groups{$hm_key}{$group})){
          print WRITE "\t".$hm_groups{$hm_key}{$group};
      }else{
          print WRITE "\t";
      }
    }
    print WRITE "\n";
}
close(WRITE);
sub average{
    my @values = @{$_[0]};
    my $length = scalar(@values);
    my $total = 0;
    foreach my $value (@values){
      $total = $total + $value;
    }
    my $avg = $total/$length;
    return $avg
}
#function to calculate the standard deviation of a set of values in an array
#given that there exists an average function
sub staddev{
    my @values = @{$_[0]};
    my $avg = average(\@values);
    my $total = 0;
    foreach my $value (@values){
      $total = $total + ($value-$avg)**2;
    }
    my $length = scalar(@values);
    my $stdd = ($total/($length-1))**.5;
    return $stdd;
}
```

```
sub sortByP{
    my @a_arr = @{$a};
    my @b_arr = @{$b};
    $a_arr[6]<=>$b_arr[6];
}

sub log2{
    my $n = shift;
    return log($n)/log(2);
}

sub median{
    my @values = @{$_[0]};
    @values = sort {$a<=>$b} @values;
    my $median = 0;
    my $n_values = scalar(@values);

    if($n_values % 2 == 0){
      my $m = ($n_values/2);
      $median = ($values[$m-1]+$values[$m])/2;
    }else{
      my $m = int($n_values/2);
      $median = $values[$m];
    }
    return $median;

}
```

```
#################################################################
# BuildForest.py
#
# Builds a random forest using all available RBPs and their
# respective Z-score based affinities for heptamers and
# the eLEI from our exon inclusion experiment
# Uses random forest generated to predict the eLEIsc of
# heptamers not generated by our mutations
# ################################################################

from sklearn import ensemble
from sklearn import tree
from sklearn.externals.six import StringIO
import os

min_oobs = {}
min_oobs_split = {}


f = open("HA7Esc_RF_Vals", "r")
lines = [line.strip() for line in f]
f.close()
leiscs = (lines.pop(0)).split('\t')
for i in range (0, len(leiscs)):
    leiscs[i] = float(leiscs[i])
ids = (lines.pop(0)).split('\t')
z_matrix = [line.split('\t') for line in lines]
for i in range (0, len(z_matrix)):
    for j in range (0, len(z_matrix[0])):
        z_matrix[i][j] = float(z_matrix[i][j])

f2 = open("HA7Esc_RF_To_Predict_All", "r")
lines = [line.strip() for line in f2]
f2.close()
z_tp = [line.split('\t') for line in lines]
for i in range (0, len(z_tp)):
    for j in range (0, len(z_tp[0])):
        z_tp[i][j] = float(z_tp[i][j])


fw = open("HA7Esc_Forest_OOBS_mean", "w")
oob_scores = []
params = []
fw3 = open("HA7Esc_RF_Predictions_All", "w")
for min_samp in range(10, 11):
    clf = ensemble.RandomForestRegressor(n_estimators=100,
min_samples_split=min_samp, oob_score=True)
    clf = clf.fit(z_matrix, leiscs)
    oob_s = clf.oob_score_
    oob_score_2 = clf.score(z_matrix, leiscs)
    params = clf.feature_importances_
    predictions = clf.predict(z_tp)
    for i in range(len(predictions)):
        fw3.write(str(predictions[i])+'\n')

    fw.write(str(min_samp)+'\t'+str(oob_s)+'\t'+str(oob_score_2)+'\n')
    oob_scores.append(oob_s)
min_oobs_l = 1000;
min_oobs_split_l = -1;
```

```
fw3.close()
for iteration in range(0, 1):
    if(oob_scores[iteration] < min_oobs_l):
        min_oobs_l = oob_scores[iteration]
        min_oobs_split_l = iteration+2
min_oobs[0] = min_oobs_l
min_oobs_split[0] = min_oobs_split_l
fw.close()
minwrite = open("HA7Esc_Random_Forest_Min_OOBS_All", "w")
for k in min_oobs.keys():
    minwrite.write(str(k)+"\t"+str(min_oobs[k])+"\t"+str(min_oobs_split[k])+"\n")
minwrite.close()
```