Predicting HIV progression

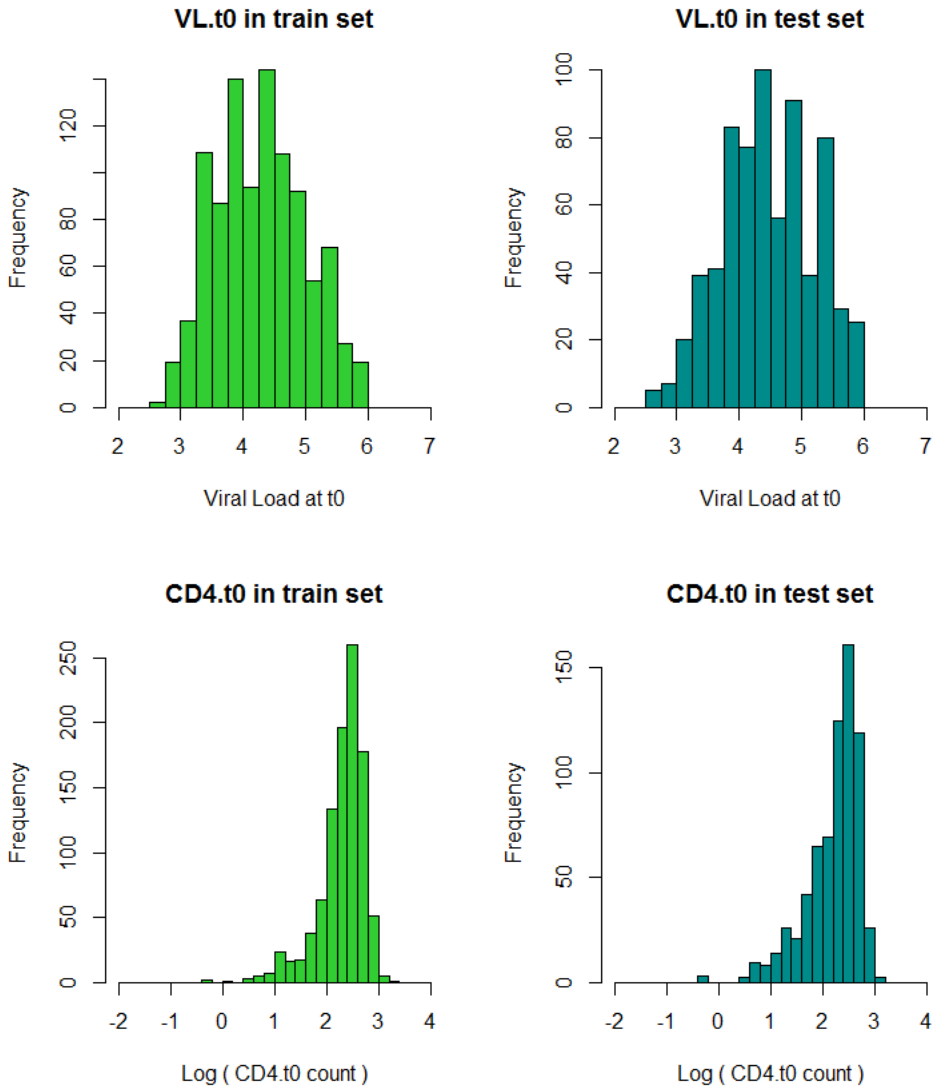
The goal of this project is to predict patients response to the anti-HIV treatment based on a very limited number of measured parameters. The dataset was retrieved from Kaggle (https://www.kaggle.com/c/hivprogression).

- Dataset includes information about a patient's viral load, CD4 count and the nucleotide sequences of their Reverse Transcriptase (RT) and their Protease (PR) HIV genes.
- In this project, false positives and a false negatives are weighted equally.
- The goal is to manage to extract useful information from the DNA sequences. These features may be easy to process using specific software, but may be difficult to annotate in a large corpus of sequences like this one. Datasets were downloaded from: https://www.kaggle.com/c/hivprogression/data

Data preparation and cleaning.

Initial data preparation was performed in R. The main goal of this step was to extract features from the nucleotide sequences of *pr* and *tr* genes.

Briefly, viral load and Log(CD4+ count) were analyzed in both datasets downloaded from Kaggle

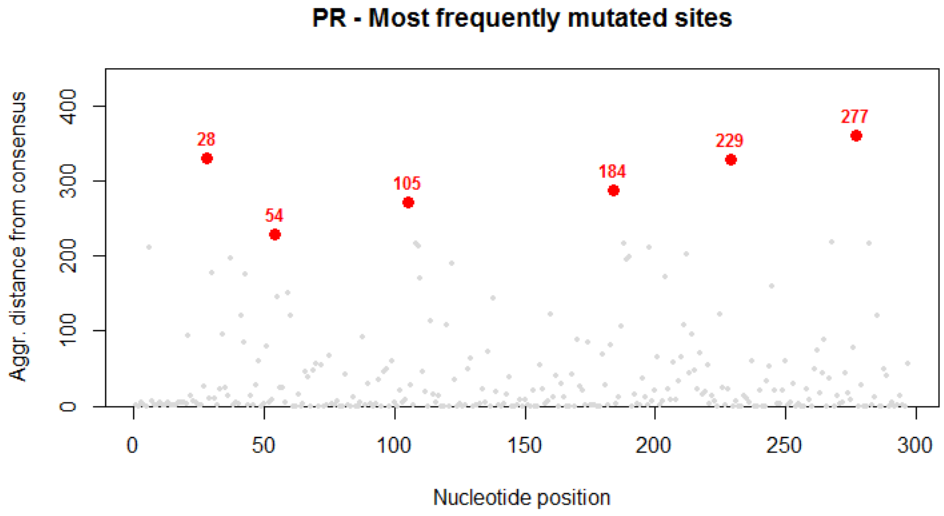


Distributions look fine. These values were not further modified.

PR sequence analysis

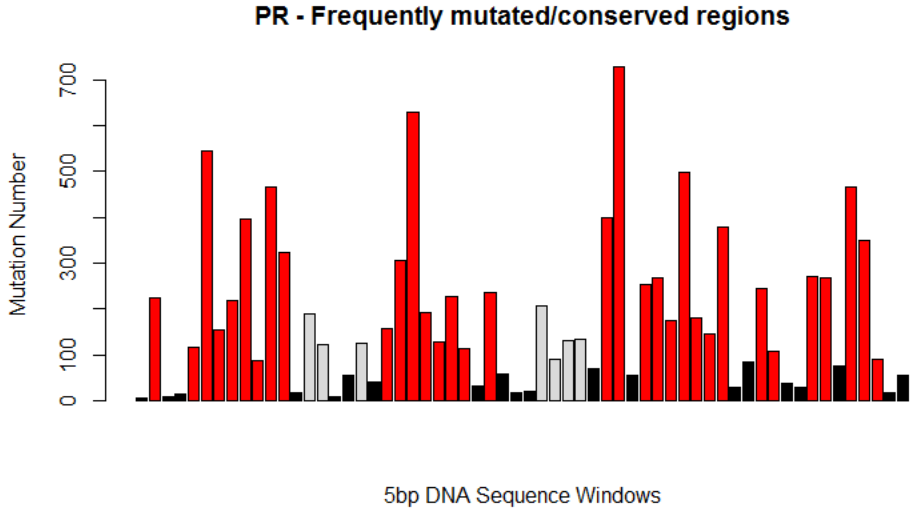
Most of the samples have a 297 bp long PR sequence. Only few sequences bearing deletions were found. Some of the PR sequences are missing in the dataset. I reasoned that this may be due to a problem with the PCR (mutations occurring in the sequence corresponding to the primers used for the sequence). The full length (297 bp) PR sequences were used to generate a consensus sequence (which nucleotide is the most frequent at each position?). The distance of each sequence from the consensus was calculated and included in the final dataset (pr_cons_dist). Distances were grouped (based on the 75%, 90%, 95% and

99% quantiles) and the resulting value (categorical) was saved as (pr_d_quant). Each sequence was then compared against the consensus in order to identify the positions corresponding to the highest frequency of mutations.



Each PR sequence in the dataset was aligned to the consensus and then the nucleotides at positions 28, 54, 105, 184, 229 and 277 were extracted. The letters corresponding to the nucleotides at these positions were added to the final dataset in the pr_nucl_X fields. Nucleotides for the missing sequences were imputed based on the most frequent value in the rest of the sequences.

I then looked at the aggregated number of mutations in 5bp- windows in the PR window.

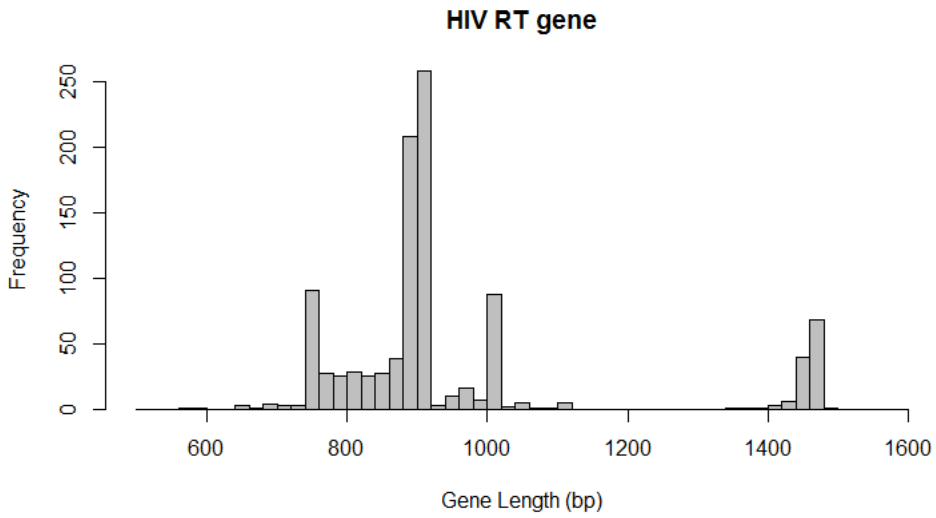


In this plot we can identify the most variable (red) and the most conserved (black) 5-bp regions in the PR sequence. For each sequence, I computed the number of mutations occurring in each large “red” region (one or more consecutive red 5-bp windows). The corresponding integer values were stored in the final dataset in the pr_wdist_X columns.

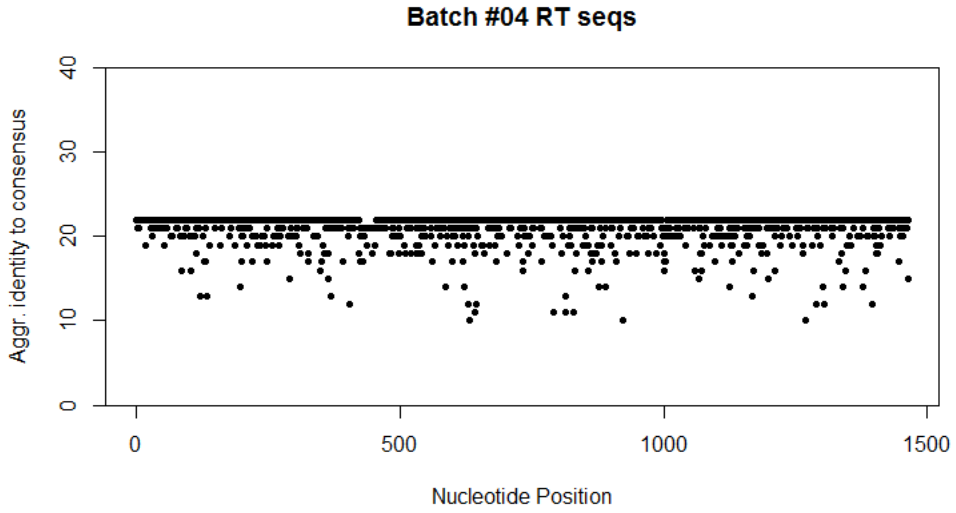
The percentage of GC nucleotides (%GC / %ATCG) in each large red region was also calculated and stored in the final dataset in the pr_wgc_X columns.

RT sequence analysis

RT gene is very different compared to PR. The variability in the sequence length is huge. In the plot, we can see that while some sequences are as short as 600 bp, some were as long as 1500 bp.



Because of this, a unique RT consensus sequence could not be extracted. Therefore, I grouped the sequences based on the gene length (4 groups were defined) and I calculated a consensus sequence for each group. I then aligned all sequences of the group to the corresponding consensus and I counted the sequence-to-consensus matches (nucleotide identity) at each position. The plot shows the results of such analysis on the RT sequences in the range 1200bp to 1600bp (batch n.4, corresponding to the right peak in the previous barplot).



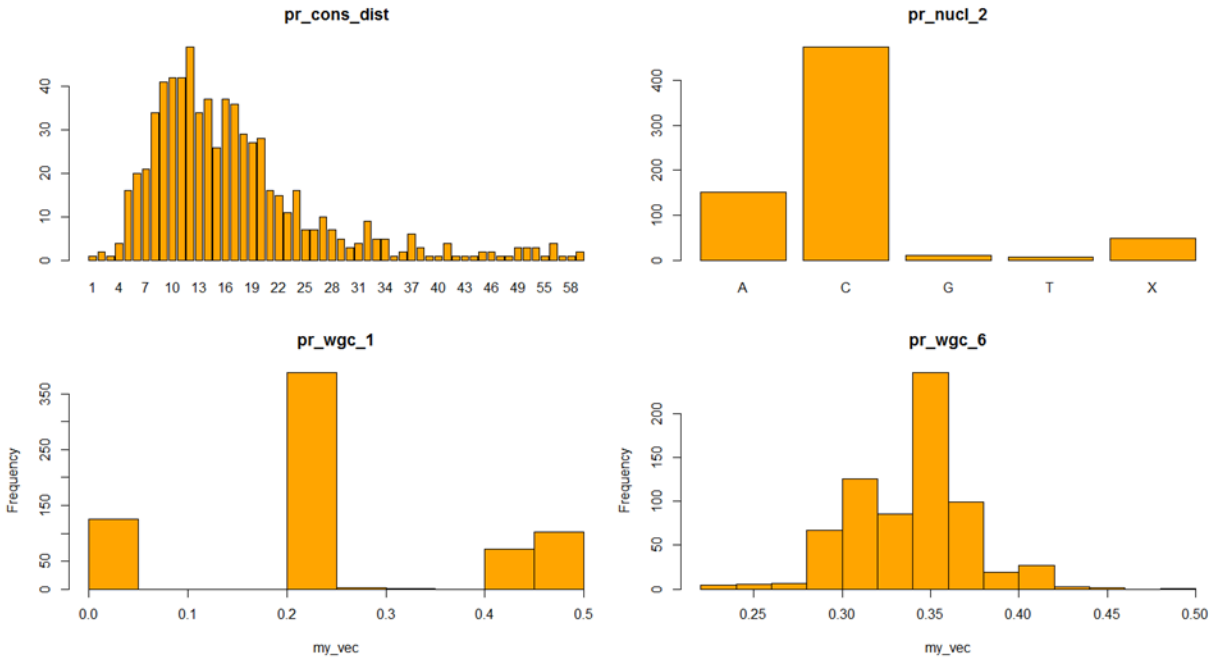
Mutations are scattered everywhere along the sequence. I therefore decided to limit the features extracted from each PR sequence to 4 variables:

- sequence length in bp (rt_len)
- rt_Len_group (feature extracted from the previous one, 4 groups)
- rt_gc_ratio: %GC of each sequence
- rt_other_ratio: % of mismatches/non assigned bases in the sequence (% of non-A,T,C,G letters in the sequence)

Data cleaning

There were no missing values for CD4+ count, viral load and TR sequences. Features based on missing PR sequences were imputed using the medians (for categorical variables, the most frequent value was used) of the rest of the cases. rt_other_ratio was also transformed. The negative logarithm of rt_other_ratio was calculated and used for replacing the original value

Barplots or histograms of each variable were generated before saving the final dataset (examples showed below). The train dataset (downloaded from Kaggle) was processed applying the same operations and transformations described above. This generated a csv file that will be used for the deployment phase.



The R code used for processing the files is available in appendix I and II.

Modeling and Deployment

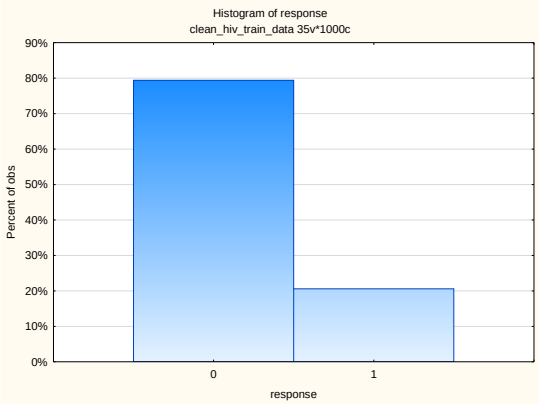
- Open Statistica13. From the Data Mining Panel, click on the Workspaces button and select All Procedures
- Click the Data Sources button. Click on Files button and select the file containing the dataset to analyze (f2_hiv_train_data.csv, f2_hiv_test_data.csv). Make sure to import as “delimited file” and select the checkbox for “taking variable name from the first row”.
- Open each file (double click on the icon in the workspace) and remove the first column (right click on the first column header and select delete variable). Save each dataset as a *.sta file

Univariate and Bivariate Data Exploration

Many univariate variable analyses were performed during the Data preparation and Cleaning Steps. Therefore, I will mainly use Statistica for bi- and multi-variate analyses, with the aim of detecting relations among variables. These steps will be performed on the TRAIN dataset only.

- In the Graphs panel, click on the 2D Histogram button. Connect the dataset to the 2D Histogram icon. Double click on the Box Plot Icon in order to define the settings.
- Click on the Variables button. Select “2-response” as the variable and click OK. In the Advanced settings tab select Fit type = OFF and Y-axis = %.
- Click on the Run button (green arrow) on the 2D Histogram icon. A Reporting Documents icon will automatically appear in the Workspace. Click on the “View Document” button on the Reporting Documents icon (a tiny white rectangle).

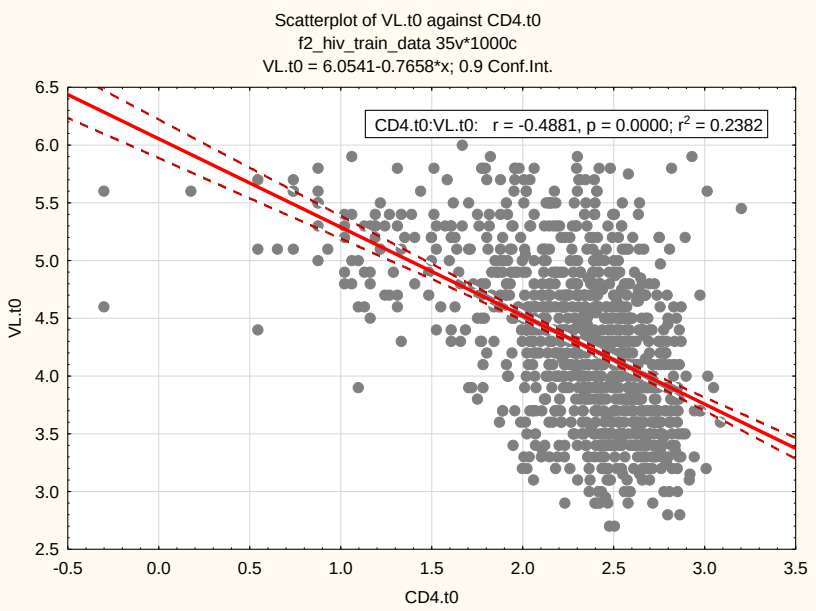
We can see that the target variable is not homogenously distributed in the dataset. Value 0 (no response) is by far more frequent than the value 1 (response). We will need to balance the dataset based on the target variable before generating the model.



I am interested in checking the relation between viral Load vs CD4+ (immune cells) count.

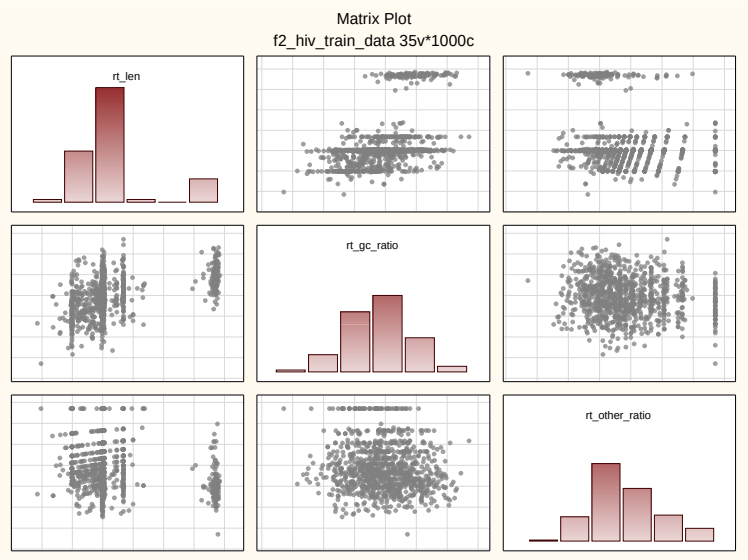
- Click on the Scatterplot button in the “Graph tab”. Connect the dataset to the plot and then double click to open the settings window. Click the Variables button and select. X = 4, Y = 3. In the “Advanced” tab, tick the “R square” box and “Corr. and p” box. Also, click on the Confidence radio button and set the confidence level at 0.90.
- Click the OK button and run the node.

An inverse correlation can be appreciated between viral load and CD4+ count. The trend is probably not very well fitted by a linear model, however higher CD4+ counts (a stronger immune system) goes together with a lower viral load.



Next, we can analyze relations among several variables in a pairwise fashion. I am interested in analyzing relations among variables I extracted from the sequences of TR and PR genes.

- Click on the Matrix button in the Graphs panel.
- Connect the Dataset to the Matrix Plots icon.
- Double click on the icon, open the Variable selection window. We will visualize the following variables: 32, 34, 35. Leave the rest of the settings as is and then run the node.
- Click on the Reporting Documents icon to view the results.



The variables we are analyzing here are: rt_len is length (in bp) of the RT DNA sequence; rt_gc_ratio: % of G,C in the sequence; rt_other_ratio: % of mismatched/unmatched bases in the RT DNA sequence. The distribution of rt_gc_ratio (range: 0 to 1) and rt_other_ratio (-Log(ratio): range 1 to 3) look similar (both are centered on a median value with a narrow tail).

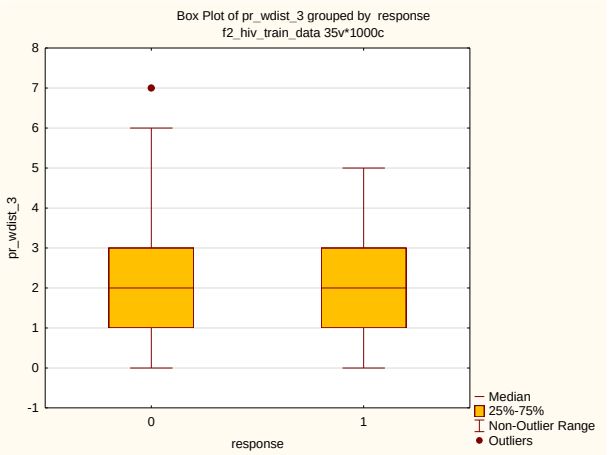
rt_other_ratio looks randomly distributed, but there is an interesting trend emerging from the plot. It looks like longer versions of the RT gene are much richer in GC nucleotides. The biological significance of this is not clear, however this may be something to further explore in the future.

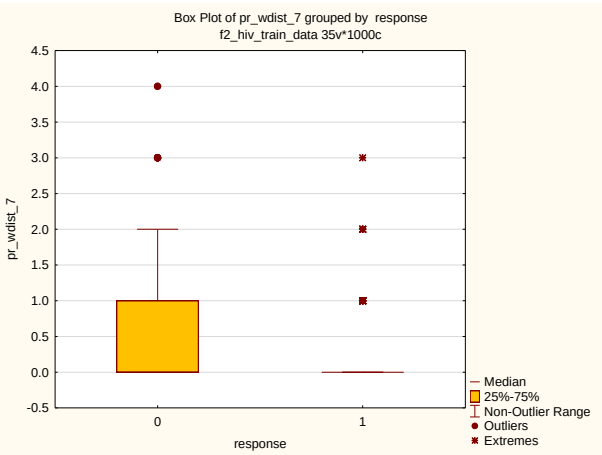
Bring a 2D Box Plot into the Workspace and connect it to the dataset.

Select the variables to display: Dependent Variable: 16-pr_wdist_3; grouping variable: 2- response. Run the node and then visualize the results. There is no difference in the pr_wdist_3 variable among the two groups: responders and non-responders. Chances are this variable won't be included for generating the predictive model.

Now, double click on the 2D Box Plot icon in the Workspace.

Re-set the dependent variable and select 20-pr_wdist_7. Here there is a striking difference. Responders may have viruses whose PR gene DNA sequence accumulates less mutations in the region 7 (as defined in the Data Preparation section).





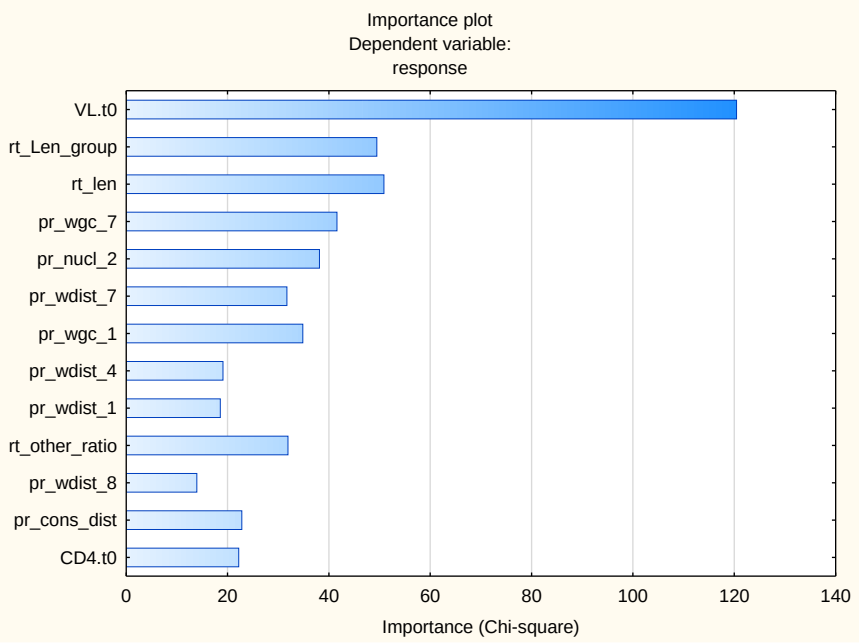
Before proceeding, I want to balance the dataset for the target variable.

- From the Data panel, click on Sampling and select the Random Sample Filtering (or via the node selector: Data > Manage > Sampling).
- Bring the sampling node into the Workspace and connect it to the data source. Double click on the icon and set the parameters as follows:
- Variables: select all variables. Stratified Sampling tab: select Strata Variables: 2-response; click on the Codes button and select All. Set the following percentages: Stratification group 1 = 100%, stratification group 0 : 25%. Click OK and then run the node. This will generate a subset containing less than 500 cases.

Even if smaller, the subset is now balanced. We can confirm this by copying & pasting a new 2D Histogram node in the workspace and connecting it to the Random Sample Filtering node. Running the node will generate a new plot.

Feature Selection

- Add a Feature Selection Node in the Workspace (data mining >> feature selection. Connect it to the Random Sample Filtering Node and then double click to define the following parameters:
    - Categorical dependent: 2
    - Continuous predictors: 3 4 6 14-32 34-35
    - Categorical predictors: 5 7 8 9 10 11 12 13 33
    - Results tab, Display 15 best predictors, sorted by p.
- Click OK




```
Best predictors for categorical dependent var: response

Best continuous predictors: 3 32 29 20 23 17 14 35 21 6 4
Best categorical predictors: 33 9
```

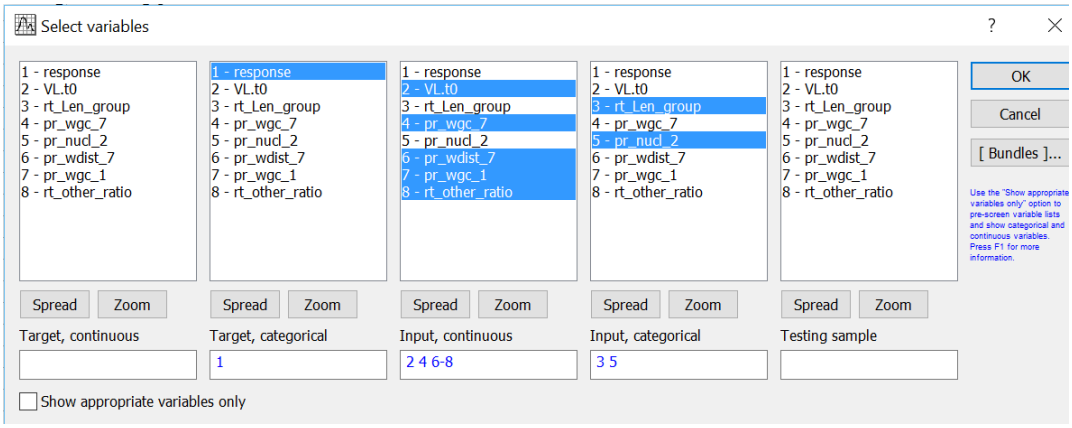
The initial viral load is the best predictor of treatment response. The length of the RT gene is also a very good predictor (rt_Len_group and rt_len are a numerical and a categorical variable respectively; I will select rt_Len_group for further analysis). CD4+ cell count is also in the list, but it is not a very strong predictor. Interestingly, pr_wdist_7 (described above) is among the predictors.

Since the number of cases is not huge (less than 500 total cases in the balanced subset), I decided to build a model using the 7 top variables: VL.t0, rt_Len_group, pr_wgc_7, pr_nucl_2, pr_wdist_7, pr_wgc_1, rt_other_ratio.

- In the Data panel, click on Sampling and add a Random Sample Filtering to the Workspace.
- Connect the new Filtering icon to the one we added previously. While we filtered cases before, we are filtering variables now. Double click on the icon and then select the variables indicated above: 2 3 33 29 9 20 23 35. Click OK and run the node. This will generate a subset matrix of 8 columns and 416 rows).
- Add a second Random Sample Filtering to the workspace and connect it to the TEST dataset
- Select the same variables and the patient ID (1 2 3 33 29 9 20 23 35) and click OK.
- Run both Filtering nodes
- Double click each Filtering node described in the last step and save the resulting dataset as a STA file

I will use a DM Recipe for quickly comparing different modeling algorithms.

- Open Data Miner Recipes (Statistica icon in the bottom left corner >> Data Mining >> Recipes)
- Click on New and connect the data file corresponding to the TRAIN dataset you just saved
- Select variables as illustrated below



- Click OK and proceed to the Next Step
- In the next panel, select Testing sample by defining a % of cases equal to 40% and then proceed to the next step.
- Do not process variables for correlation and ignore important variable selection (select none and proceed to the next step)
- Select the following three algorithms for the modeling phase: Neural network, SVM and Boosted tree. Statistica13 will now build and evaluate each selected model. Check all the methods and click on Evaluate models. This will generate a summary outlining the performance differences among the chosen algorithms.

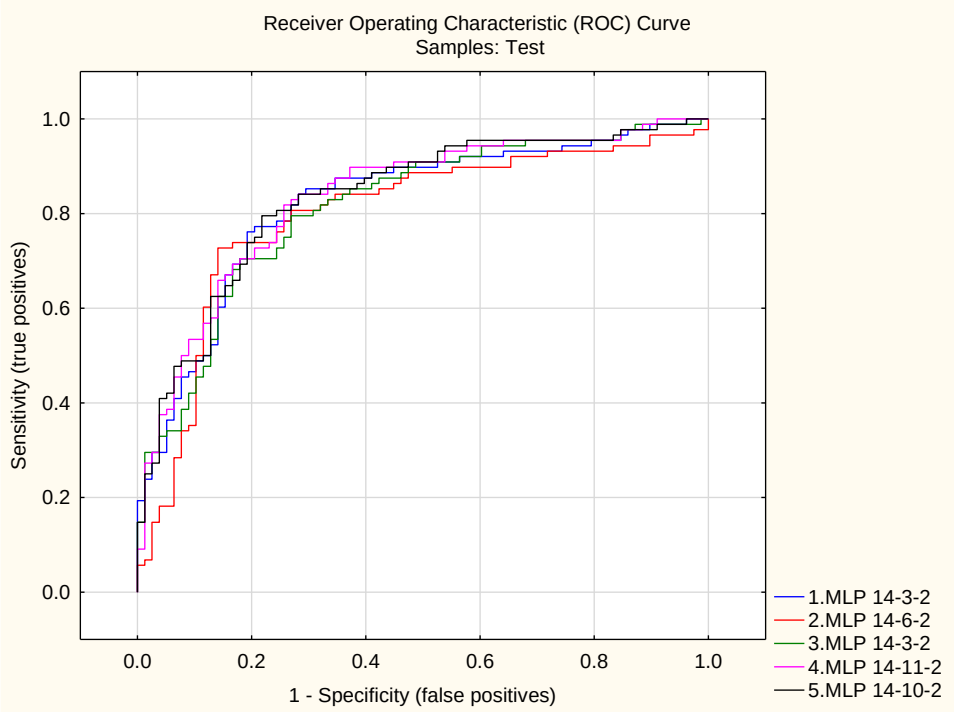
ID	Name	Error rate (%) (Testing sample)
☑ 3	Neural ne...	24.86
☑ 2	SVM	26.52
☑ 1	Boosted t...	32.04

In this case, the neural network looks like the most accurate model.

We can move back to our workspace and continue with the development of the project.

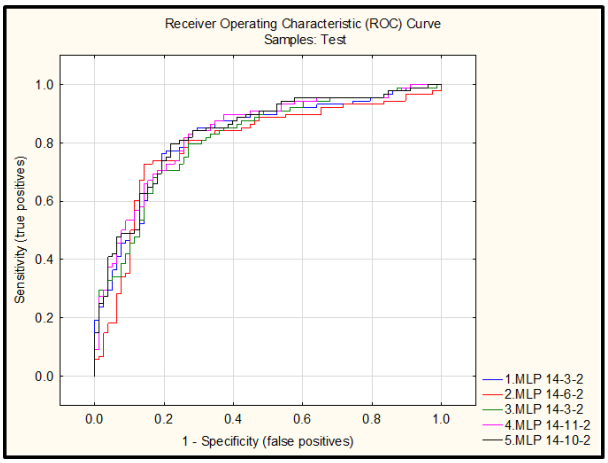
- Add a Classification Neural Network to the workspace, link it to the Filter icon corresponding to the Train set and double click to define parameters.
- Select variables as indicated above.
    - Sampling tab: Train %: 60
    - Test%: 40
    - Results>Samples tab
    - Samples: Test
    - Predictions> Include Targets, Output, Accuracy
    - Lift chrats> select ROC curve
    - Lift chart and Cumulative Lift Chart
    - Code generator: PMML only
- Click OK and run the SANN node.

The ROC curve generated by this model is encouraging good, as shown below.

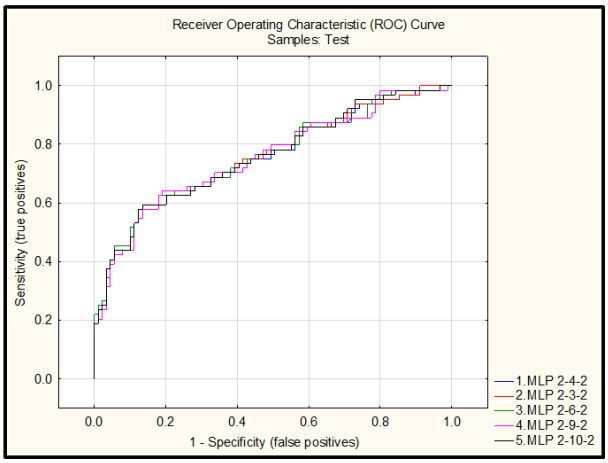


Interestingly, the neural network model we just generated performs better than a similar model that does not include features extracted from the PR and RT nucleotide sequences.

Model including seq data



Model w/o seq data



We can now deploy our model.

- Connect the Filter icon corresponding to the TEST set to the rapid deployment beta icon. Double click to edit parameters. Allow “writing back to the input data” and Assign SANNModelPRED to 2-response
- Adjust configuration about output and graphs (if needed)
- Click OK and RUN

A summary of the deployment is now added in Reporting Document node. Moreover, if we double click the TEST set Filter node, we will notice that the model has written its predictions in column 2 (before we only had “H”). Done! Success!

Conclusions

We were able to generate a model for predicting HIV virus load reduction based on 2 DNA sequences, initial viral load and CD4+ cell counts in a patient’s blood. The model we generated is based on a neural network and has an error rate of about 25% in the test set.

Interestingly, the model was based on the initial viral load (major predictor) as well as several features extracted by annotating HIV PR and TR gene sequences. For example, we showed that the higher variability in specific regions of the PR gene sequence (for example, r7: 240-250bp) correlate with a poor response (it could be interesting to further analyze the underlying biological mechanisms, as there could be some potential for drug development).

We also managed to deploy the model on a set of “unseen” cases that were classified using our neural network model. Hopefully, this model could be useful for improving the management of the disease. For example, our model could help adjusting the timing of routine tests for each patient or help physicians in selecting an appropriate personalized therapeutic regimen for each patient.

Appendix I – R code – Data preparation

```
library(msa)
#
###### Functions #######
#
prep_cd4_var <- function(cd4_vector) {
  log(cd4_vector + 0.5, base = 10)
}
#
detect_indels <- function(seq_vect, expected_len) {
  as.numeric(sapply(seq_vect, (function(seq){
    if (nchar(seq) > expected_len) {
      1
    } else if (nchar(seq) < expected_len) {
      -1
    } else {
      0
    }
  })))
}
#
build_consensus <- function(full_length_seqs) {
  consensus_rt <- paste(sapply(1:nchar(full_length_seqs[1]), (function(i) {
    tmp_tab <- table(sapply(1:length(full_length_seqs), (function(j){
      substr(full_length_seqs[j], i,i)
    })))
    names(tmp_tab[tmp_tab == max(tmp_tab)])[1]
  })), collapse = "")
}
#
scale_seqs <- function(seq_vector, consensus_seq) {
  v1 <- seq_vector
  consensus_rt <- consensus_seq
  scaled_seq <- sapply(v1, (function(seq){
    if(nchar(seq) > 10 && nchar(seq) != nchar(consensus_rt)){
      #
      aligned <- msa(c(seq, consensus_rt), type = "dna")
      seq_alig <- as.character(aligned@unmasked[1])
      cons_alig <- as.character(aligned@unmasked[2])
      i_to_rmv <- sapply(1:nchar(cons_alig), (function(i){
        if (substr(cons_alig, i, i) == '-') { i } else { 0 }
      }))
      i_to_rmv <- i_to_rmv[i_to_rmv != 0]
      if (length(i_to_rmv) > 0 ) {
        seq_alig <- paste(unlist(lapply(1:nchar(seq_alig), (function(k){
          if (! (k %in% i_to_rmv)) {
            substr(seq_alig, k,k)
          }
        }))), collapse = '')
      }
      #
      seq_alig
    } else {
      seq
    }
  }))
  names(scaled_seq) <- NULL
  return(scaled_seq)
}
#
extract_seq_features <- function(seq_vector, consensus_seq, final_sites, final_windows, label) {
  v1 <- seq_vector
  result_matrix <- do.call(rbind, lapply(1:length(v1), (function(i){
    if (nchar(v1[i]) > 10) {
      #extract nucleotide at positions
      nucl_at_site <- as.vector(sapply(final_sites, (function(j){
        substr(v1[i],j,j)
      })))
      #compute distance in windows
      dists_at_ws <- sapply(1:(length(final_windows)/2), (function(j){
        k2 <- final_windows[2 * j]
        k1 <- final_windows[(2 * j) -1]
        adist(substr(v1[i], k1, k2), substr(consensus_seq, k1, k2))
      }))
      #compute CG in windows
      gc_at_ws <- sapply(1:(length(final_windows)/2), (function(j){
        k2 <- final_windows[2 * j]
        k1 <- final_windows[(2 * j) -1]
        my_bases <- unlist(strsplit(substr(v1[i], k1, k2), split = ""))
        my_at <- sum(my_bases == "A" | my_bases == "T") + 0.01
        my_gc <- sum(my_bases == "G" | my_bases == "C") + 0.01
        my_gc / (my_gc + my_at)
      }))
      return_vect <- c(nucl_at_site, dists_at_ws, gc_at_ws)
      names(return_vect) <- NULL
      return_vect
    } else {
      rep(NA, sum(length(final_sites), length(final_windows)))
    }
  })))
  part_1 <- paste(label, "_nucl_", 1:length(final_sites), sep = "")
  part_2 <- paste(label, "_wdist_", 1:(length(final_windows)/2), sep = "")
  part_3 <- paste(label, "_wgc_", 1:(length(final_windows)/2), sep = "")
```

```
  colnames(result_matrix) <- c(part_1, part_2, part_3)
  result_matrix
}
#
extract_pr_features <- function(pr_seq_vector, length_breaks) {
  return_mat <- t(sapply(pr_seq_vector, (function(sq){
    if (nchar(sq) > 10) {
      len_sq <- nchar(sq)
      len_batch_sq <- max(which(len_sq > length_breaks))
      bases_sq <- unlist(strsplit(sq, split = ""))
      my_gc <- sum(bases_sq == "G" | bases_sq == "C")
      my_at <- sum(bases_sq == "A" | bases_sq == "T")
      my_other <- len_sq - (my_gc + my_at)
      gc_ratio <- my_gc / (my_gc + my_at)
      other_ratio <- my_other / len_sq
      c(len_sq, len_batch_sq, gc_ratio, other_ratio)
    } else {
      rep(NA, 4)
    }
  })))
  rownames(return_mat) <- NULL
  colnames(return_mat) <- c("rt_len","rt_Len_group","rt_gc_ratio","rt_other_ratio")
  return(return_mat)
}
#
#
###### Script #######
#
# Read Data
setwd("~/uci_soft/pa/tutorial/")
train <- read.csv("training_data.csv")
test <- read.csv("test_data.csv")
#
head(train)
head(test)
#
# Explore data and define data corrections/extractions
train1 <- as.character(train$PR.Seq)
train2 <- as.character(train$RT.Seq)
train3 <- train$VL.t0
train4 <- train$CD4.t0
train_pid <- train$PatientID
train_resp <- train$Resp
#
test1 <- as.character(test$PR.Seq)
test2 <- as.character(test$RT.Seq)
test3 <- test$VL.t0
test4 <- test$CD4.t0
test_pid <- test$PatientID
test_resp <- test$Resp
#
# VL.t0
sum(is.na(train3))
sum(is.na(test3))
sum(train3==0)
sum(test3==0)
par(mfrow = c(1,2))
hist(train3, main = "VL.t0 in train set", col = "limegreen",
     xlab = "Viral Load at t0",
     breaks = seq(2,7, by = 0.25))
hist(test3, main = "VL.t0 in test set", col = "darkcyan",
     xlab = "Viral Load at t0",
     breaks = seq(2,7, by = 0.25))
#
#VL.t0 is just fine
#
#CD4.t0
sum(is.na(train4))
sum(is.na(test4))
is.integer(train4)
#
par(mfrow = c(1,2))
hist(log(train4+0.5, base = 10),
     breaks = seq(-2,4, by = 0.2),
     main = "CD4.t0 in train set", col = "limegreen",
     xlab = "Log ( CD4.t0 count )")
hist(log(test4+0.5, base = 10),
     breaks = seq(-2,4, by = 0.2),
     main = "CD4.t0 in test set", col = "darkcyan",
     xlab = "Log ( CD4.t0 count )")
#
# CD4.t0 looks fine, but is a variable widely distributed
# Let's just take the Log10
# In order to deal with values = 0, add +0.5 to all values
new_cd4_train <- prep_cd4_var(train4)
new_cd4_test <- prep_cd4_var(test4)
#
# Now, let's start processing the sequences: rt first.
#
#some seqs are missing, some are shorter, some may be longer maybe?
# What is the most common sequence length?
my_len <- names(sort(table(nchar(train1)), decreasing = T)[1])
my_len
#
#Are there InDels?
```

```
indels_train_rt <- detect_indels(train1, my_len)
indels_test_rt <- detect_indels(test1, my_len)
#
#retrieve seqs with the most common length
full_len_rts <- train1[nchar(train1) == as.numeric(my_len)]
#
# In order to extract features, we need to define a consensus.
# COnsensus won't change! If we want to modify the consensus,
# we need to reprocess everything
#
# defining a consensus
rt_consensus <- build_consensus(full_len_rts)
#
#
# how far is each sequence from the consensus?
rt_distance_train1 <- as.numeric(adist(train1, rt_consensus))
rt_distance_test1 <- as.numeric(adist(test1, rt_consensus))
#
# Define quantiles
my_quants <- quantile(rt_distance_train1[rt_distance_train1 < 200], probs = c(0.75,0.9,0.95,0.99))
#
rt_dist_quantile_train <- sapply(rt_distance_train1, (function(dd){
  if(sum(dd> my_quants) > 0) {
    max(which( dd> my_quants))
  } else {
    0
  }
}))
rt_dist_quantile_test <- sapply(rt_distance_test1, (function(dd){
  if(sum(dd> my_quants) > 0) {
    max(which( dd> my_quants))
  } else {
    0
  }
}))
#
# where are the differences? Let's explore...
# I am counting how many sequences differ from the consensus at each position
# Limited to the full length sequences only.
diffs <- sapply(1:my_len, (function(i){
  my_letter <- substr(rt_consensus, i, i)
  sum(sapply(1:length(full_len_rts), (function(j){
    my_letter != substr(full_len_rts[j], i, i)
  })))
}))
names(diffs) <- 1:length(diffs)
#top 6 freq mutated sites
freq_mut_sites <- order(diffs, decreasing = TRUE)[1:6]
par(mfrow = c(1,1))
#
# single point mutation
plot(diffs, ylim = c(0,450), type = "n", main = "PR - Most frequently mutated sites",
     xlab = "Nucleotide position", ylab = "Aggr. distance from consensus")
points(names(diffs[-freq_mut_sites]), diffs[-freq_mut_sites], col = "grey85", pch = 16, cex = 0.5)
points(names(diffs[freq_mut_sites]), diffs[freq_mut_sites], col = "red", pch = 16, cex = 1.25)
text(names(diffs[freq_mut_sites]), diffs[freq_mut_sites], names(diffs[freq_mut_sites]),
     cex = 0.8, font = 2, pos = 3, col = "red")
final_sites <- names(diffs[freq_mut_sites])
#
# Cumulative mutations in windows defined by mutation-refractory boundaries
windows <- seq(0, length(diffs), by = 5)
if (! length(diffs) %in% windows) { windows <- c(windows, length(diffs)) }
w_diffs <- sapply(2: length(windows), (function(i){
  sum(diffs[as.numeric(windows[i-1]+1) : as.numeric(windows[i])])
}))
clor <- rep("gray85", length(w_diffs))
clor[order(w_diffs)[1:20]] <- "black"
clor[order(w_diffs, decreasing = TRUE)[1:20]] <- "red"
barplot(w_diffs, col = clor, main = "PR - Frequently mutated/conserved regions",
        ylab = "Mutation Number", xlab = "5bp DNA Sequence Windows")
#
clor <- rep("gray85", length(w_diffs))
clor[c(1,3,4,13,16,17,19,27,29,30,31,36,39,47,48,51,52,55,59,60)] <- "black"
clor[c(2,5,6,7,8,9,10,11,12,20,21,22,23,24,25,26,28,37,38,40,41,42,43,44,45,46,49,50,53,54,56,57,58)] <- "red"
barplot(w_diffs, col = clor, main = "PR - Frequently mutated/conserved regions",
        ylab = "Mutation Number", xlab = "5bp DNA Sequence Windows")
#
final_windows <- c(6,10,21,60,96,130,136,140,181,190,196,230,241,250,261,270,276,290)
#
# Before checking which nucleotide is present at a frequently mutatedp position
# we need to re-realign the sequence
#
#
#
adj_train_seqs <- scale_seqs(train1, rt_consensus)
adj_test_seqs <- scale_seqs(test1, rt_consensus)
#
new_rt_train_features <- extract_seq_features (adj_train_seqs, rt_consensus, final_sites, final_windows, "pr")
head(new_rt_train_features)
new_rt_test_features <- extract_seq_features (adj_test_seqs, rt_consensus, final_sites, final_windows, "pr")
head(new_rt_test_features)
#
#
# Same feature extraction for RT gene
#some seqs are missing, some are shorter, some may be longer maybe?
```

```
# What is the most common sequence length?
hist(nchar(train2),
     breaks = seq(500, 1600, by = 20), col = "grey75",
     main= "HIV RT gene", xlab = "Gene Length (bp)")
#
pr_batch1 <- train2[nchar(train2)<820]
pr_batch2 <- train2[nchar(train2)>=820 & nchar(train2)<925]
pr_batch3 <- train2[nchar(train2)>=925 & nchar(train2)<1200]
pr_batch4 <- train2[nchar(train2)>=1200]
#
pr_len_1 <- names(sort(table(nchar(pr_batch1)), decreasing = T)[1])
pr_full1 <- pr_batch1[nchar(pr_batch1) == pr_len_1]
pr_consensus_b1 <- build_consensus(pr_full1)
mut_vector_1 <- sapply(1:nchar(pr_full1[1]), (function(i){
  sum(sapply(1:length(pr_full1), (function(j){
    substr(pr_full1[j], i, i) == substr(pr_consensus_b1, i, i)
  })))
}))
#
plot(1:nchar(pr_consensus_b1), mut_vector_1, pch = 16, cex = 0.75,
     main = "Batch #01 RT seqs", xlab = "Nucleotide Position",
     ylab = "Aggr. identity to consensus",
     ylim = c(0,100))
#
pr_len_2 <- names(sort(table(nchar(pr_batch2)), decreasing = T)[1])
pr_full2 <- pr_batch2[nchar(pr_batch2) == pr_len_2]
pr_consensus_b2 <- build_consensus(pr_full2)
mut_vector_2 <- sapply(1:nchar(pr_full2[1]), (function(i){
  sum(sapply(1:length(pr_full2), (function(j){
    substr(pr_full2[j], i, i) == substr(pr_consensus_b2, i, i)
  })))
}))
plot(1:nchar(pr_consensus_b2), mut_vector_2, pch = 16, cex = 0.75,
     main = "Batch #02 RT seqs", xlab = "Nucleotide Position",
     ylab = "Aggr. identity to consensus",
     ylim = c(0,200))
#
pr_len_3 <- names(sort(table(nchar(pr_batch3)), decreasing = T)[1])
pr_full3 <- pr_batch3[nchar(pr_batch3) == pr_len_3]
pr_consensus_b3 <- build_consensus(pr_full3)
mut_vector_3 <- sapply(1:nchar(pr_full3[1]), (function(i){
  sum(sapply(1:length(pr_full3), (function(j){
    substr(pr_full3[j], i, i) == substr(pr_consensus_b3, i, i)
  })))
}))
plot(1:nchar(pr_consensus_b3), mut_vector_3, pch = 16, cex = 0.75,
     main = "Batch #03 RT seqs", xlab = "Nucleotide Position",
     ylab = "Aggr. identity to consensus",
     ylim = c(0,100))
#
pr_len_4 <- names(sort(table(nchar(pr_batch4)), decreasing = T)[1])
pr_full4 <- pr_batch4[nchar(pr_batch4) == pr_len_4]
pr_consensus_b4 <- build_consensus(pr_full4)
mut_vector_4 <- sapply(1:nchar(pr_full4[1]), (function(i){
  sum(sapply(1:length(pr_full4), (function(j){
    substr(pr_full4[j], i, i) == substr(pr_consensus_b4, i, i)
  })))
}))
plot(1:nchar(pr_consensus_b4), mut_vector_4, pch = 16, cex = 0.75,
     main = "Batch #04 RT seqs", xlab = "Nucleotide Position",
     ylab = "Aggr. identity to consensus",
     ylim = c(0,40))
#
# No way we can analyze the sequence this way
# We determine the following parameters:
# len, len group, GC% and % of non ATGC letters (frequency) length and length group
#
pr_train_feats <- extract_pr_features(train2,length_breaks = c(0,800,925,1200))
pr_test_feats <- extract_pr_features(test2,length_breaks = c(0,800,925,1200))
#
#
###### Wrapping up, build the transformed dataset ##################
#
prep_train <- data.frame(cbind(train_pid, train_resp, train3, new_cd4_train,
                               indels_train_rt, rt_distance_train1, rt_dist_quantile_train,
                               new_rt_train_features, pr_train_feats),
                         stringsAsFactors = F)
colnames(prep_train) <- c("patient_id","response","VL.t0", "CD4.t0", "pr_indel", "pr_cons_dist", "pr_d_quant",
                          colnames(new_rt_train_features), colnames(pr_train_feats))
write.csv(prep_train, file = "f1_train_data.csv")
head(prep_train)
#
prep_test <- data.frame(cbind(test_pid, test_resp,test3, new_cd4_test,
                              indels_test_rt, rt_distance_test1, rt_dist_quantile_test,
                              new_rt_test_features, pr_test_feats),
                        stringsAsFactors = F)
colnames(prep_test) <- c("patient_id","response","VL.t0", "CD4.t0", "pr_indel", "pr_cons_dist", "pr_d_quant",
                         colnames(new_rt_train_features), colnames(pr_train_feats))
write.csv(prep_test, file = "f1_test_data.csv")
head(prep_test)
#
```

Appendix II – R code – Data cleaning

```
## Data clean
## Basically, NA removal and some corrections
#
# Functions
reduce_bases <- function(bases_vector, erase_id = NULL) {
  bases <- as.character(bases_vector)
  if (is.null(erase_id)) {
    bases[is.na(bases)] <- "X"
    bases[bases != "A" & bases != "C" & bases != "G" & bases != "T"] <- "X"
  } else {
    if (sum(is.na(bases[-erase_id])) > 0) {
      bases[-erase_id][is.na(bases[-erase_id])] <- "X"
    }
    condition_id <- bases[-erase_id] != "A" & bases[-erase_id] != "C" & bases[-erase_id] != "G" & bases[-erase_id] != "T"
    if (sum(condition_id) > 0 ) {
      bases[-erase_id][condition_id] <- "X"
    }
    bases[erase_id] <- names(sort(table(as.character(bases[-erase_id])), decreasing = T)[1])
  }
  return(bases)
}
#
adjust_distances <- function(dist_vector, log_transform = FALSE, erase_id = NULL) {
  dt <- dist_vector
  dt[is.na(dt)] <- median(dt[!is.na(dt)])
  if (log_transform == FALSE ) {
    return(dt)
  } else {
    return(log(dt + 1, base = 10))
  }
}
#
# Script
setwd("/Users/dami/Documents/uci_soft/pa/tutorial/")
train <- read.csv("f1_train_data.csv")[,-1]
#
for (i in 3: ncol(train)) {
  my_vec <- train[!is.na(train[,i]),i]
  if(is.integer(my_vec)) {
    barplot(table(my_vec), main = colnames(train)[i], col = "grey60")
  } else if(is.numeric(my_vec)) {
    hist(my_vec, main = colnames(train)[i], col = "grey60")
  } else {
    barplot(table(train[!is.na(train[,i]),i]), main = colnames(train)[i], col = "grey60")
  }
}
# Some of the pr sequences were missing? Why? I think that the virus has mutated in the
# regions were the primers for the sequencing anneal, so no amplification was obtained.
# Failed annealing = no sequence, but the seq is still there. Therefore, I assign
# most to these rows a median value for everything
# Maybe the distance from consensus is linked
#
# Running a DM Recipe
# Didn't work
#
tiny_train <- train[train$pr_cons_dist < 200, ]
sp_rt_ln <- split(tiny_train$rt_len, as.factor(tiny_train$pr_d_quant))
barplot(unlist(lapply(sp_rt_ln, mean)), main = "RT length (bp)", ylim = c(0,1200))
sp_rt_ln <- split(tiny_train$rt_gc_ratio, as.factor(tiny_train$pr_d_quant))
barplot(unlist(lapply(sp_rt_ln, mean)), main = "RT %GC content")
sp_rt_ln <- split(tiny_train$rt_other_ratio, as.factor(tiny_train$pr_d_quant))
barplot(unlist(lapply(sp_rt_ln, median)), main = "RT %mismatches")
#
#
my_fact <- which(train$pr_cons_dist>200)
#
train$pr_indel[my_fact] <-names(sort(table(train$pr_indel[-my_fact])))[1]
train$pr_cons_dist[my_fact] <- median(train$pr_cons_dist[-my_fact])
train$pr_d_quant[my_fact] <- median(train$pr_d_quant[-my_fact])
#
#
train$pr_nucl_1 <- reduce_bases(train$pr_nucl_1, erase_id = my_fact)
train$pr_nucl_2 <- reduce_bases(train$pr_nucl_2, erase_id = my_fact)
train$pr_nucl_3 <- reduce_bases(train$pr_nucl_3, erase_id = my_fact)
train$pr_nucl_4 <- reduce_bases(train$pr_nucl_4, erase_id = my_fact)
train$pr_nucl_5 <- reduce_bases(train$pr_nucl_5, erase_id = my_fact)
train$pr_nucl_6 <- reduce_bases(train$pr_nucl_6, erase_id = my_fact)
#
#adjusting wdist depends on how sparse the values are
#after looking through the values, I decided to keep as is (integer)
# all the fields but wdist_2. There I apply a log transform
train$pr_wdist_1 <- adjust_distances(train$pr_wdist_1)
train$pr_wdist_2 <- adjust_distances(train$pr_wdist_2, log_transform = TRUE)
train$pr_wdist_3 <- adjust_distances(train$pr_wdist_3)
train$pr_wdist_4 <- adjust_distances(train$pr_wdist_4)
train$pr_wdist_5 <- adjust_distances(train$pr_wdist_5)
train$pr_wdist_6 <- adjust_distances(train$pr_wdist_6)
train$pr_wdist_7 <- adjust_distances(train$pr_wdist_7)
train$pr_wdist_8 <- adjust_distances(train$pr_wdist_8)
train$pr_wdist_9 <- adjust_distances(train$pr_wdist_9)
```

```
#
#Most fields look fine, we just replace NAs with median values
fields <- c("pr_wgc_1", "pr_wgc_2", "pr_wgc_3", "pr_wgc_4", "pr_wgc_5", "pr_wgc_6",
"pr_wgc_7", "pr_wgc_8", "pr_wgc_9", "rt_len", "rt_Len_group", "rt_gc_ratio")
for (clnm in fields) {
  tmp_v <- train[,clnm]
  tmp_v[is.na(tmp_v)] <- median(tmp_v[!is.na(tmp_v)])
  train[,clnm] <- tmp_v
}
#
#Take care of rt_other_ratio field
nu_rt_other_ratio <- train$rt_other_ratio
nu_rt_other_ratio[is.na(nu_rt_other_ratio)] <- median(nu_rt_other_ratio[!is.na(nu_rt_other_ratio)])
nu_rt_other_ratio <- (-1) * log(nu_rt_other_ratio + 0.001, base = 10)
train$rt_other_ratio <- nu_rt_other_ratio
#
#
#Final Check!
for (i in 3: ncol(train)) {
  my_vec <- train[,i]
  if(is.integer(my_vec)) {
    barplot(table(my_vec), main = colnames(train)[i], col = "orange")
  } else if(is.numeric(my_vec)) {
    hist(my_vec, main = colnames(train)[i], col = "orange")
  } else {
    barplot(table(my_vec), main = colnames(train)[i], col = "orange")
  }
}
#
#
#Do exactly the same for the test dataset, for deployment

test <- read.csv("f1_test_data.csv")[,-1]
#
test$pr_nucl_1 <- reduce_bases(test$pr_nucl_1)
test$pr_nucl_2 <- reduce_bases(test$pr_nucl_2)
test$pr_nucl_3 <- reduce_bases(test$pr_nucl_3)
test$pr_nucl_4 <- reduce_bases(test$pr_nucl_4)
test$pr_nucl_5 <- reduce_bases(test$pr_nucl_5)
test$pr_nucl_6 <- reduce_bases(test$pr_nucl_6)
#
#adjusting wdist depends on how sparse the values are
#if max(value) < 10 , just remove NAs
#otherwise, log transform
test$pr_wdist_1 <- adjust_distances(test$pr_wdist_1)
test$pr_wdist_2 <- adjust_distances(test$pr_wdist_2, log_transform = TRUE)
test$pr_wdist_3 <- adjust_distances(test$pr_wdist_3)
test$pr_wdist_4 <- adjust_distances(test$pr_wdist_4)
test$pr_wdist_5 <- adjust_distances(test$pr_wdist_5)
test$pr_wdist_6 <- adjust_distances(test$pr_wdist_6)
test$pr_wdist_7 <- adjust_distances(test$pr_wdist_7)
test$pr_wdist_8 <- adjust_distances(test$pr_wdist_8)
test$pr_wdist_9 <- adjust_distances(test$pr_wdist_9)
#
#Most fields look fine, we just replace NAs with median values
fields <- c("pr_wgc_1", "pr_wgc_2", "pr_wgc_3", "pr_wgc_4", "pr_wgc_5", "pr_wgc_6",
            "pr_wgc_7", "pr_wgc_8", "pr_wgc_9", "rt_len", "rt_Len_group", "rt_gc_ratio")
for (clnm in fields) {
  tmp_v <- test[,clnm]
  tmp_v[is.na(tmp_v)] <- median(tmp_v[!is.na(tmp_v)])
  test[,clnm] <- tmp_v
}
#
#Take care of rt_other_ratio field
nu_rt_other_ratio <- test$rt_other_ratio
nu_rt_other_ratio[is.na(nu_rt_other_ratio)] <- median(nu_rt_other_ratio[!is.na(nu_rt_other_ratio)])
nu_rt_other_ratio <- (-1) * log(nu_rt_other_ratio + 0.001, base = 10)
test$rt_other_ratio <- nu_rt_other_ratio
#
#
#Final Check!
for (i in 3: ncol(test)) {
  my_vec <- test[,i]
  if(is.integer(my_vec)) {
    barplot(table(my_vec), main = colnames(test)[i], col = "orange")
  } else if(is.numeric(my_vec)) {
    hist(my_vec, main = colnames(test)[i], col = "orange")
  } else {
    barplot(table(my_vec), main = colnames(test)[i], col = "orange")
  }
}
#
#SAve
write.csv(train, "f2_hiv_train_data.csv")
write.csv(test, "f2_hiv_test_data.csv")
```