Predicting m6A RNA sites

HarvardX Data Science Professional Certificate: PH125.9x Capstone IDV Learners

Oliver Artz

2/23/2022

Contents

1 Introduction

N6-methyladenosine (m6A) is the most abundant mRNA modification in eukaryotes. It is highly conserved among species and can be found in mammals, fish, yeast, and plants. m6A is essential for the proper development of an organism. A disrupted m6A machinery and hence mis-regulated m6A deposition on mRNA is correlated with a plethora of diseases such as cancer or obesity. Various methods have been developed to identify methylated transcriptomic sites. One of the first methods that allows for identification of m6A sites at single base pair resolution was developed by Meyer (2019, https://doi.org/10.1038/s41592-019-0570-0) and is called DART (deamination adjacent to RNA modification targets)-seq.

Experimental identification of m6A sites is resource intensive with respect to labor, money, and time. Hence, researchers have tried to accurately predict m6A sites *in silico*. The *in silico* prediction of m6A sites is an active field of research and experts are using sophisticated machine learning algorithms and deep neural networks on large data sets to solve this problem. While this project will unlikely push the boundaries of our current understanding of m6A biology, the data set generated by Meyer (2019) provides a rich playground to test and apply machine learning concepts.

In this project, I will use m6A sites discovered by Meyer (2019), engineer features that might be helpful for the prediction of m6A sites and train machine learning algorithms to build a model of m6A deposition in the transcriptome of HEKT293T cells, a prominent model for human biology. Lastly, the models will be tested on an entirely different data set take from Linder et al. (2015, https://doi.org/10.1038/nmeth.3453).

2 Methods/Analysis

2.1 Loading libraries and data

Load / install libraries

```
if (!require("tidyverse", quietly = TRUE))
  install.packages('tidyverse')
library(tidyverse)

if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
library(BiocManager)

if (!require("BSgenome.Hsapiens.UCSC.hg19", quietly = TRUE))
  install.packages("BSgenome.Hsapiens.UCSC.hg19")
library(BSgenome.Hsapiens.UCSC.hg19)

if (!require("ChIPseeker", quietly = TRUE))
  install.packages("ChIPseeker")
library(ChIPseeker)

if (!require("TxDb.Hsapiens.UCSC.hg19.knownGene", quietly = TRUE))
  install.packages("TxDb.Hsapiens.UCSC.hg19.knownGene")
library(TxDb.Hsapiens.UCSC.hg19.knownGene)

if (!require("bedr", quietly = TRUE))
  install.packages('bedr')
```

```
library(bedr)

if (!require("caret", quietly = TRUE))
  install.packages('caret')
library(caret)

if (!require("doParallel", quietly = TRUE))
  install.packages('doParallel')
library(doParallel)

if (!require("bookdown", quietly = TRUE))
  install.packages('bookdown')
library(bookdown)

if (!require("formatR", quietly = TRUE))
  install.packages('formatR')
library(formatR)
```

Load data

```
# fetching m6A site data from github repo
url <- "https://raw.githubusercontent.com/oliverartz/HarvardX_PH125.9x_DataScienceCapstone_m6A_pred/mai
df <- read_csv(url)

# cleanup
rm(url)
```

2.2 Feature engineering

To enable the use of supervised ML algorithms a null model of randomly chosen adenines are generated. For the purpose of this exercise, we assume that they are not methylated, although we have not proven this experimentally.

```
# add column to specify methylated adenines
data <- df %>% dplyr::select(-`U/C`)
data$meth <- "methylated"

# generate approximately as many random adenines as in the data set
n_regions <- nrow(data) * 4

set.seed(2021, sample.kind = "Rounding")
random_regions <- get.random.regions(n = n_regions,
                    species = "human",
                    build = "hg19",
                    size.mean = 1,
                    size.sd = 0)

# add random strand information (with same distribution as in data set)
prob_plus <- table(data$Strand)[2] / (table(data$Strand)[1]+table(data$Strand)[2])
random_regions$strand <- sample(c("+","-"),
                                n_regions,
                                prob = c(prob_plus, 1-prob_plus),
```

```
                                  replace = TRUE)
random_regions$meth <- "not_methylated"

# filter for adenines
random_regions <- random_regions %>%
  mutate(base = getSeq(Hsapiens,
                       chr,
                       start = start,
                       end = start,
                       strand = strand,
                       as.character = TRUE)) %>%
  filter(base == "A") %>%
  dplyr::select(-base)

# add non-methylated regions to data frame
colnames(random_regions) <- colnames(data)
data <- rbind(data, random_regions)

# check for duplicates
# to avoid labeling methylated adenines as unmethylated from the random set of loci
dupl <- data %>%
  group_by(`C2U start`,Chr) %>%
  summarize(n = n()) %>%
  filter(n>1) %>%
  pull(`C2U start`)

# remove duplicate
data <- data %>% filter(`C2U start` != dupl)

#cleanup
rm(random_regions, prob_plus,n_regions, dupl)
```

To be able to successfully predict m6A sites, we need more than just their genomic locus. In this section potentially biologically meaningful features will be added to the data frame that will help to build the model later. We will add the sequence context and GC content to the data frame. Five bp up- and downstream of the methylated adenines will be considered. Moreover we will add the GC content of proximal (50 bp up- or downstream) or distal (200 bp up- or downstream) regions. Lastly, we will annotate all sites with respect to their functional regions such as 5'UTR, exon, intron, 3'UTR, promoter, intergenic, etc.

```
# get genomic sequence adjacent to methylated adenine
data <- data %>%
  mutate(seq_plus1 = getSeq(Hsapiens,
                            Chr,
                            start = `C2U start` + 1,
                            end = `C2U start` + 1,
                            strand = Strand,
                            as.character = TRUE),
         seq_plus2 = getSeq(Hsapiens,
                            Chr, start = `C2U start` + 2,
                            end = `C2U start` + 2,
                            strand = Strand,
                            as.character = TRUE),
         seq_plus3 = getSeq(Hsapiens,
```

```
                            Chr,
                            start = `C2U start` + 3,
                            end = `C2U start` + 3,
                            strand = Strand,
                            as.character = TRUE),
         seq_plus4 = getSeq(Hsapiens,
                            Chr,
                            start = `C2U start` + 4,
                            end = `C2U start` + 4,
                            strand = Strand,
                            as.character = TRUE),
         seq_plus5 = getSeq(Hsapiens,
                            Chr,
                            start = `C2U start` + 5,
                            end = `C2U start` + 5,
                            strand = Strand,
                            as.character = TRUE),
         seq_minus1 = getSeq(Hsapiens,
                             Chr,
                             start = `C2U start` - 1,
                             end = `C2U start` - 1,
                             strand = Strand,
                             as.character = TRUE),
         seq_minus2 = getSeq(Hsapiens,
                             Chr,
                             start = `C2U start` - 2,
                             end = `C2U start` - 2,
                             strand = Strand,
                             as.character = TRUE),
         seq_minus3 = getSeq(Hsapiens,
                             Chr,
                             start = `C2U start` - 3,
                             end = `C2U start` - 3,
                             strand = Strand,
                             as.character = TRUE),
         seq_minus4 = getSeq(Hsapiens,
                             Chr,
                             start = `C2U start` - 4,
                             end = `C2U start` - 4,
                             strand = Strand,
                             as.character = TRUE),
         seq_minus5 = getSeq(Hsapiens,
                             Chr,
                             start = `C2U start` - 5,
                             end = `C2U start` - 5,
                             strand = Strand,
                             as.character = TRUE))

# get GC content of adjacent genomic sequence (proximal and distal)
proximal <- 50
distal <- 200

data <- data %>%
```

```
  mutate(proximal_GC_3prime = letterFrequency(getSeq(Hsapiens,
                                                     Chr,
                                                     start = `C2U start` + 1,
                                                     end = `C2U start` + proximal,
                                                     strand = Strand),
                                              letters = "GC",
                                              as.prob = TRUE),
         proximal_GC_5prime = letterFrequency(getSeq(Hsapiens,
                                                     Chr,
                                                     start = `C2U start` - proximal,
                                                     end = `C2U start` - 1,
                                                     strand = Strand),
                                              letters = "GC",
                                              as.prob = TRUE),
         distal_GC_3prime = letterFrequency(getSeq(Hsapiens,
                                                   Chr,
                                                   start = `C2U start` + 1,
                                                   end = `C2U start` + distal,
                                                   strand = Strand),
                                            letters = "GC",
                                            as.prob = TRUE),
         distal_GC_5prime = letterFrequency(getSeq(Hsapiens,
                                                   Chr, start = `C2U start` - distal,
                                                   end = `C2U start` - 1,
                                                   strand = Strand),
                                            letters = "GC",
                                            as.prob = TRUE))

# get feature annotation for m6A sites
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

peaks <- data %>% dplyr::select(1:4)
colnames(peaks) <- c("chr","start","end","strand")
peaks.gr <- makeGRangesFromDataFrame(peaks)

peak_annotation <- annotatePeak(peaks.gr, TxDb = txdb, verbose = FALSE)

peak_annotation <- peak_annotation %>%
  as.data.frame() %>%
  dplyr::select("annotation")

# clean up annotation (making it more general)
peak_annotation$annotation <- gsub("^Exon.*","Exon", peak_annotation$annotation)
peak_annotation$annotation <- gsub("^Promoter.*","Promoter", peak_annotation$annotation)
peak_annotation$annotation <- gsub("^Intron.*","Intron", peak_annotation$annotation)

# merge peak annotations with data frame
data <- cbind(data, peak_annotation)

# remove genomic locus from data frame
data <- data %>%  dplyr::select(-c(1:4))

# encode non-numeric columns as factors (necessary for some ML algorithms)
```

```r
columns_to_factor <- c("seq_plus1",
                       "seq_plus2",
                       "seq_plus3",
                       "seq_plus4",
                       "seq_plus5",
                       "seq_minus1",
                       "seq_minus2",
                       "seq_minus3",
                       "seq_minus4",
                       "seq_minus5",
                       "annotation",
                       "meth")

data[columns_to_factor] <- lapply(data[columns_to_factor], factor)

# cleanup
rm(proximal, distal, txdb, peaks, peaks.gr, peak_annotation, columns_to_factor, ChIPseekerEnv)
```

2.3 Data exploration

```r
# dimensions of data frame
dim <- dim(data)

# check for NAs
sum_NA <- sum(is.na(data))

# summary of data
summary_df <- summary(data)
```

The data set contains 84217 genomic loci. The original data set from Meyer (2019) provides 40262 methylated sites. 43955 not methylated random genomic sites centering around adenines were added to the data set.

2.4 Model performance measures

To develop predictive algorithms for m6A deposition on transcripts, features mainly with respect to the sequence context of methylated adenines were used as independent variables. Model performance was assessed mainly based on two parameters. Accuracy (Equation 1) takes the sensitivity (or true positive rate) and specificity (or true negative rate) into account and can be calculated using the following formula:

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}} \tag{1}$$

The F1 (Equation 2) score is generally a better measure of model performance when classes are imbalanced. Since this data set provides an almost 50/50 split between classes of the dependent variable, it should be very close to the accuracy score. The F1 score is calculated using the following formula:

$$\text{F1 score} = \frac{2(\text{True Positive})}{2(\text{True Positive}) + \text{False Positive} + \text{False Negative}} \tag{2}$$

Cross validation is important to improve model performance, avoid over-training, and tune model parameters. Here we use k-fold cross validation during model fitting. While a larger number of folds and repeats would

be advantageous, increasing those values comes at a computational cost. Values of 5 or 10 are widely used. In this project, we will define the number of folds and repeats as 5.

2.5 Splitting the data set

Before analyzing and exploring the data, they were split into train and test set (80%/20%). Defining a test data set is essential to develop models. This step helps making the model more generalizable and avoid overfitting, thus being able to employ the model to categorize sequences that have not been presented to the model before. Model performance was assessed using the accuracy and F1 metric. Since compute time is an important consideration when training models, we will also show the wall time for each model generation process.

```
# initialize data frame for model performance
results_model_performance <- data.frame(model = character(),
                                        accuracy = numeric(),
                                        F1 = numeric(),
                                        wall_time_min = numeric())
```

```
set.seed(2022, sample.kind = "Rounding")
test_index <- createDataPartition(data$meth, times = 1, p = 0.2, list = FALSE)
test_set <- data %>% dplyr::slice(test_index)
train_set <- data %>% dplyr::slice(-test_index)

# cleanup
rm(test_index)
```

2.6 Exploring training and test set

```
#proportion of methylated adenines in training
mean_a_train <- (mean(train_set$meth == "methylated") * 100) %>% round(digits = 3)

#proportion of methylated adenines in test
mean_a_test <- (mean(test_set$meth == "methylated") * 100) %>% round(digits = 3)
```

The training and test data sets consist of about 48% methylated adenines (47.807%, 47.809%). Hence, they exhibit the same slight bias towards containing less methylated loci. The split is balanced and can be used for model training and validation.

To evaluate models, cross validation can be used. The caret package allows for easy implementation of cross validation for many different algorithms. The parameters used in this project are defined below. When applicable, we will use 5-fold cross-validation repeated 5 times.

```
# cross validation parameters
fitControl <- trainControl(method = "repeatedcv",
                           number = 5, # n-fold cross-validation
                           repeats = 5, # repeat each cross-validation n times
                           classProbs = TRUE, # compute class probabilities
                           returnResamp = "final", # save resampled summary metrics
                           savePredictions = "final") # save the final predictions
```

Building models is a computationally intensive task. Often we build several models subsequently to compare their performance and choose appropriate tuning parameters. On machines with multiple processors or cores, these models can be generated in parallel instead of subsequently, thus leading to much shorter overall wall times for processing. Here, we make use of multiple cores for model generation.

```
# define number of cores to use
cl <- makePSOCKcluster(5)

# register parallel backend
registerDoParallel(cl)
```

2.7 Modeling with logistic regression

Together with linear regression, logistic regression is among the most widespread modeling techniques. Logistic regression assumes that the dependent and independent variable follow a normal distribution. Logistic regression is particularly useful when we are investigating at a categorical dependent variable.

```
# logistic regression model

# set seed to make results reproducible
set.seed(2022, sample.kind = "Rounding")

# model fitting
t_start <- Sys.time()
fit_glm <- train(meth ~ .,
                 method = "glm",
                 data = train_set,
                 trControl = fitControl)

t_end <- Sys.time()

# model validation
y_hat_glm <- predict(fit_glm, test_set)

# add results to performance data frame
cm <- confusionMatrix(y_hat_glm, mode = "everything", reference = test_set$meth)

results_model_performance[1, ] <- c(model = "logistic regression",
                                    accuracy = cm$overall[1] %>% round(digits = 3),
                                    F1 = cm$byClass[7] %>% round(digits = 3),
                                    wall_time_min = difftime(t_end, t_start, units='min') %>%
                                      as.numeric() %>%
                                      round(digits = 2))

# cleanup
rm(cm, t_start, t_end)
```

2.8 Modeling with k-nearest neighbor (with k optimization)

In the k-nearest neighbor model, the Euclidian distance between neighboring data points is calculated. Assuming that proximity correlates with similarity, we can thus categorize values into different bins according to their nearest neighbors. Here, k denotes the number of neighboring data points that are being considered.

k is a commonly used tuning parameter. In the following we will build models with ten different values for k (even numbers from 2 to 20) and choose the best model for our data set.

```
# k-nearest neighbor model

# set seed to make results reproducible
set.seed(2022, sample.kind = "Rounding")

# define ks for optimization
k = data.frame(k = seq(2,20,2))

# model fitting
t_start <- Sys.time()
fit_knn <- train(meth ~ .,
                 method = "knn",
                 data = train_set,
                 tuneGrid = k,
                 trControl = fitControl)
t_end <- Sys.time()

# validate model
y_hat_knn <- predict(fit_knn, test_set)

# add results to performance data frame
cm <- confusionMatrix(y_hat_knn, mode = "everything", reference = test_set$meth)

results_model_performance[2, ] <- c(model = "k-nearest neighbor",
                                    accuracy = cm$overall[1] %>% round(digits = 3),
                                    F1 = cm$byClass[7] %>% round(digits = 3),
                                    wall_time_min = difftime(t_end, t_start, units='min') %>%
                                      as.numeric() %>%
                                      round(digits = 2))

# cleanup
rm(cm, t_start, t_end, k)
```

2.9 Modeling with Random Forest (with mtry optimization)

Random Forests generate a number of decision trees and generate an average prediction based on all generated trees. Each tree is build by regressing on a subset of parameters from the original data set (mtry value). Here, we tune mtry by picking 15 different values and assessing the final model performance. To make parameter tuning more computationally efficient, we reduce the number of trees to be generated for the forest (ntree) from 500 to 5. The number of trees to be used for the Random Forest can also be used as a tuning parameters, but the effect of generating more trees on model performance or a possible interaction between mtry and ntree will not be explored.

```
# random forest model with mtry optimization

# set seed to make results reproducible
set.seed(2022, sample.kind = "Rounding")

# use random search to find best mtry for random forest
t_start <- Sys.time()
```

```
# reduce number of trees to be produced to speed up optimization
ntree <- 5

# start fitting
rf_random <- train(meth ~ .,
                   data = train_set,
                   method = 'rf',
                   metric = 'Accuracy',
                   tuneLength  = 15, # number of randomly generated mtry values
                   trControl = fitControl,
                   ntree = ntree)

# pull best mtry value for subsequent model fitting
mtry <- data.frame(mtry = rf_random$bestTune %>% pull(mtry))

# set seed to make results reproducible
set.seed(2022, sample.kind = "Rounding")

# model fitting
fit_rf <- train(meth ~ .,
                method = "rf",
                data = train_set,
                tuneGrid = mtry,
                importance = TRUE,
                trControl = fitControl)
t_end <- Sys.time()

# validate model
y_hat_rf <- predict(fit_rf, test_set)

# add results to performance data frame
cm <- confusionMatrix(y_hat_rf, mode = "everything", reference = test_set$meth)

results_model_performance[3, ] <- c(model = "random forest",
                                    accuracy = cm$overall[1] %>% round(digits = 3),
                                    F1 = cm$byClass[7] %>% round(digits = 3),
                                    wall_time_min = difftime(t_end, t_start, units='min') %>%
                                      as.numeric() %>%
                                      round(digits = 2))

# cleanup
rm(mtry, cm, t_start, t_end, ntree)
```

2.10 Modeling with Linear Discriminant Analysis (LDA)

LDA reduces the dimensionality of the data set by assuming that the correlation structure for all classes is identical, and allows to capture linear relationships. It is very similar to logistic regression.

```
# linear discriminant analysis (LDA)

# set seed to make results reproducible
set.seed(2022, sample.kind = "Rounding")
```

```r
# model fitting
t_start <- Sys.time()
fit_lda <- train(meth ~ .,
                 method = "lda",
                 data = train_set,
                 trControl = fitControl)
t_end <- Sys.time()

# model validation
y_hat_lda <- predict(fit_lda, test_set)

# add results to performance data frame
cm <- confusionMatrix(y_hat_lda, mode = "everything", reference = test_set$meth)

results_model_performance[4, ] <- c(model = "linear discriminant analysis",
                                    accuracy = cm$overall[1] %>% round(digits = 3),
                                    F1 = cm$byClass[7] %>% round(digits = 3),
                                    wall_time_min = difftime(t_end, t_start, units='min') %>%
                                      as.numeric() %>%
                                      round(digits = 2))


# cleanup
rm(cm, t_start, t_end)
```

2.11 Model with (shallow) Neural Network

Neural networks have become very popular in the machine learning and data science community, particularly because of their ability to capture non-linear relationships of high dimensional data. Depending on the number of layers in a network, they are termed either 'shallow' or 'deep'. Here, we employ a shallow neural network consisting of three layers, the minimum depth needed to model non-linear relationships. Parameter tuning was not explored.

```r
# neural network model

# set seed to make results reproducible
set.seed(2022, sample.kind = "Rounding")

# model fitting
t_start <- Sys.time()
fit_nn <- train(meth ~ .,
                method = "nnet",
                data = train_set,
                trControl = fitControl,
                trace = FALSE)
t_end <- Sys.time()

# model validation
y_hat_nn <- predict(fit_nn, test_set)

# add results to performance data frame
cm <- confusionMatrix(y_hat_nn, mode = "everything", reference = test_set$meth)
```

```r
results_model_performance[5, ] <- c(model = "neural network",
                                    accuracy = cm$overall[1] %>% round(digits = 3),
                                    F1 = cm$byClass[7] %>% round(digits = 3),
                                    wall_time_min = difftime(t_end, t_start, units='min') %>%
                                      as.numeric() %>%
                                      round(digits = 2))

# cleanup
rm(cm, t_start, t_end)
```

2.12 Using ensemble prediction to make majority prediction

Combining several models to make a majority vote on the prediction can improve the accuracy and stability of the models and outperform each of the individual models. Here, we used all trained models for this project. While it is tempting to only use the models that showed the highest performance on the test data in the ensemble, incorporating all models might make the ensemble more stable when challenged with completely new data sets and avoid certain biases or over-fitting.

```r
#accuracy of the ensemble model
prediction_matrix <- cbind(y_hat_glm, y_hat_knn, y_hat_lda, y_hat_rf,y_hat_nn)

votes <- rowMeans(prediction_matrix == 1)
y_hat_vote <- ifelse(votes > 0.5, 1, 0)
ensemble_accuracy <-(1 - mean(y_hat_vote == as.numeric(test_set$meth)-1)) %>%
  round(digits = 3)

# re-factor votes from numeric to "methylated"/"not_methylated"
y_hat_vote <- y_hat_vote %>%
  as.factor() %>%
  recode_factor("1" = "methylated", "0" = "not_methylated")

# add results to performance data frame
cm <- confusionMatrix(y_hat_vote, mode = "everything", reference = test_set$meth)

results_model_performance[6, ] <- c(model = "ensemble",
                                    accuracy = cm$overall[1] %>% round(digits = 3),
                                    F1 = cm$byClass[7] %>% round(digits = 3),
                                    wall_time_min = NA)


#which is the best model?
accuracy <- colMeans(prediction_matrix == as.numeric(test_set$meth))


# cleanup
rm(accuracy, prediction_matrix, votes, y_hat_vote, ensemble_accuracy)
```

2.13 Validation with external data set

Machine learning algorithms are often generated to predict dependent variables in unseen data sets. Hence, an important step in assessing the validity of a trained model is to challenge it with novel data, ideally from

an independent experiment. That way, we will be able to better estimate the general applicability of the model for our research question or the scientific community at large.

Here, we will investigate how well the models predict methylation in a data set generated by Linder et al. (2015). In this study, the authors used a different method called cross-linking–induced mutation sites-based m6A individual-nucleotide-resolution cross-linking and immunoprecipitation (CIMS miCLIP) to identify 9536 putative m6A sites in the transcriptome of the same HEK237 cells.

2.13.1 Loading data and feature engineering as performed for Meyer (2019) data.

```
# fetching m6A site data from github repo
url <- "https://raw.githubusercontent.com/oliverartz/HarvardX_PH125.9x_DataScienceCapstone_m6A_pred/mai
df2 <- read_csv(url)

# cleanup
rm(url)

# select relevant columns
data2 <- df2 %>% dplyr::select(2,3,4,6,5)

# copy column names from other data frame to make later steps easier
colnames(data2) <- colnames(df)

# add column to specify methylated adenines
data2$meth <- "methylated"

# generate approximately as many random adenines as in the data set
n_regions <- nrow(data2) * 4

set.seed(2021, sample.kind = "Rounding")
random_regions <- get.random.regions(n = n_regions,
                    species = "human",
                    build = "hg19",
                    size.mean = 1,
                    size.sd = 0,
                    )

# add random strand information (with same distribution as in data set)
prob_plus <- table(data2$Strand)[2] / (table(data2$Strand)[1]+table(data2$Strand)[2])
random_regions$strand <- sample(c("+","-"),
                                n_regions,
                                prob = c(prob_plus, 1-prob_plus),
                                replace = TRUE)
random_regions$meth <- "not_methylated"

# filter for adenines
random_regions <- random_regions %>%
  mutate(base = getSeq(Hsapiens,
                       chr,
                       start = start,
                       end = start,
                       strand = strand,
                       as.character = TRUE)) %>%
```

```
  filter(base == "A") %>%
  dplyr::select(-base)

random_regions <- random_regions %>%
  mutate(fill = 1) %>%
  dplyr::select(1,2,3,6,4,5)

# add non-methylated regions to data frame
colnames(random_regions) <- colnames(data2)
data2 <- rbind(data2, random_regions)

# check for duplicates
# to avoid labeling methylated adenines as unmethylated from the random set of loci
dupl <- data2 %>%
  group_by(`C2U start`,Chr) %>%
  summarize(n = n()) %>%
  filter(n>1) %>%
  pull(`C2U start`)

# remove duplicate
#data2 <- data2 %>% filter(`C2U start` != dupl)

#cleanup
rm(random_regions, prob_plus,n_regions, dupl)

# get genomic sequence adjacent to methylated adenine
data2 <- data2 %>%
  mutate(seq_plus1 = getSeq(Hsapiens,
                            Chr,
                            start = `C2U start` + 1,
                            end = `C2U start` + 1,
                            strand = Strand,
                            as.character = TRUE),
         seq_plus2 = getSeq(Hsapiens,
                            Chr,
                            start = `C2U start` + 2,
                            end = `C2U start` + 2,
                            strand = Strand,
                            as.character = TRUE),
         seq_plus3 = getSeq(Hsapiens,
                            Chr,
                            start = `C2U start` + 3,
                            end = `C2U start` + 3,
                            strand = Strand,
                            as.character = TRUE),
         seq_plus4 = getSeq(Hsapiens,
                            Chr,
                            start = `C2U start` + 4,
                            end = `C2U start` + 4,
                            strand = Strand,
                            as.character = TRUE),
         seq_plus5 = getSeq(Hsapiens,
                            Chr,
```

```
                            start = `C2U start` + 5,
                            end = `C2U start` + 5,
                            strand = Strand,
                            as.character = TRUE),
         seq_minus1 = getSeq(Hsapiens,
                             Chr,
                             start = `C2U start` - 1,
                             end = `C2U start` - 1,
                             strand = Strand,
                             as.character = TRUE),
         seq_minus2 = getSeq(Hsapiens,
                             Chr,
                             start = `C2U start` - 2,
                             end = `C2U start` - 2,
                             strand = Strand,
                             as.character = TRUE),
         seq_minus3 = getSeq(Hsapiens,
                             Chr,
                             start = `C2U start` - 3,
                             end = `C2U start` - 3,
                             strand = Strand,
                             as.character = TRUE),
         seq_minus4 = getSeq(Hsapiens,
                             Chr,
                             start = `C2U start` - 4,
                             end = `C2U start` - 4,
                             strand = Strand,
                             as.character = TRUE),
         seq_minus5 = getSeq(Hsapiens,
                             Chr,
                             start = `C2U start` - 5,
                             end = `C2U start` - 5,
                             strand = Strand,
                             as.character = TRUE))

# get GC content of adjacent genomic sequence (proximal and distal)
proximal <- 50
distal <- 200

data2 <- data2 %>%
  mutate(proximal_GC_3prime = letterFrequency(getSeq(Hsapiens,
                                                     Chr,
                                                     start = `C2U start` + 1,
                                                     end = `C2U start` + proximal,
                                                     strand = Strand),
                                              letters = "GC",
                                              as.prob = TRUE),
         proximal_GC_5prime = letterFrequency(getSeq(Hsapiens,
                                                     Chr,
                                                     start = `C2U start` - proximal,
                                                     end = `C2U start` - 1,
                                                     strand = Strand),
                                              letters = "GC",
```

```
                                                as.prob = TRUE),
         distal_GC_3prime = letterFrequency(getSeq(Hsapiens,
                                                   Chr,
                                                   start = `C2U start` + 1,
                                                   end = `C2U start` + distal,
                                                   strand = Strand),
                                            letters = "GC",
                                            as.prob = TRUE),
         distal_GC_5prime = letterFrequency(getSeq(Hsapiens,
                                                   Chr,
                                                   start = `C2U start` - distal,
                                                   end = `C2U start` - 1,
                                                   strand = Strand),
                                            letters = "GC",
                                            as.prob = TRUE))

# get feature annotation for m6A sites
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

peaks <- data2 %>% dplyr::select(1,2,3,5)
colnames(peaks) <- c("chr","start","end","strand")
peaks.gr <- makeGRangesFromDataFrame(peaks)

peak_annotation <- annotatePeak(peaks.gr, TxDb = txdb, verbose = FALSE)

peak_annotation <- peak_annotation %>%
  as.data.frame() %>%
  dplyr::select(1,2,3,5,6)
colnames(peak_annotation) <- c("Chr",
                               "C2U start",
                               "C2U end",
                               "Strand",
                               "annotation")

# clean up annotation (making it more general)
peak_annotation$annotation <- gsub("^Exon.*","Exon", peak_annotation$annotation)
peak_annotation$annotation <- gsub("^Promoter.*","Promoter", peak_annotation$annotation)
peak_annotation$annotation <- gsub("^Intron.*","Intron", peak_annotation$annotation)

# merge peak annotations with data frame
data2 <- left_join(data2,
                   peak_annotation,
                   by = c("Chr", "C2U start", "C2U end", "Strand"))

# annotating mitochondrial genes leads to incomplete cases
# filter incomplete cases / mitochondrial genes
data2 <- data2 %>% na.omit

# remove genomic locus from data frame
data2 <- data2 %>% dplyr::select(-c(1:5))

# encode non-numeric columns as factors (necessary for some ML algorithms)
columns_to_factor <- c("seq_plus1",
```

```
                        "seq_plus2",
                        "seq_plus3",
                        "seq_plus4",
                        "seq_plus5",
                        "seq_minus1",
                        "seq_minus2",
                        "seq_minus3",
                        "seq_minus4",
                        "seq_minus5",
                        "annotation",
                        "meth")

data2[columns_to_factor] <- lapply(data2[columns_to_factor], factor)


# cleanup
rm(distal, proximal, txdb, peak_annotation, peaks.gr)
```

2.13.2 Testing models on Linder (2015) data

```
# applying fits to new data
y_hat_glm_Linder <- predict(fit_glm, data2)
y_hat_knn_Linder <- predict(fit_knn, data2)
y_hat_lda_Linder <- predict(fit_lda, data2)
y_hat_rf_Linder <- predict(fit_rf, data2)
y_hat_nn_Linder <- predict(fit_nn, data2)

# accuracy of the ensemble model
prediction_matrix <- cbind(y_hat_glm_Linder,
                           y_hat_knn_Linder,
                           y_hat_rf_Linder,
                           y_hat_lda_Linder,
                           y_hat_nn_Linder)

votes <- rowMeans(prediction_matrix == 1)
y_hat_vote <- ifelse(votes > 0.5, 1, 0)

ensemble_accuracy <- (1 - mean(y_hat_vote == as.numeric(data2$meth)-1)) %>%
  round(digits = 3)

# re-factor votes from numeric to "methylated"/"not_methylated"
y_hat_vote <- y_hat_vote %>%
  as.factor() %>%
  recode_factor("1" = "methylated", "0" = "not_methylated")


# add accuracy on Linder to results data frame
cm <- confusionMatrix(y_hat_vote, mode = "everything", reference = data2$meth)

results_model_performance_Linder <- data_frame(model = "ensemble",
                                    accuracy = cm$overall[1] %>% round(digits = 3),
                                    F1 = cm$byClass[7] %>% round(digits = 3))
```

```
# cleanup
rm(accuracy_Lin, prediction_matrix, votes, y_hat_vote, ensemble_accuracy)
```

3 Results

3.1 Model performance on Meyer (2019) data set

The performance of all models generated in this project are depicted in Table 1. All models performed well with an average accuracy of 0.964 and an average F1 score of 0.963. The ensemble model had the highest accuracy (0.971). The k-nearest neighbor model had the lowest accuracy (0.942) despite taking a relatively long time to train (346.78 min). For future model generation and protoyping, we could consider omitting the knn model to increase computational efficiency without sacrificing overall model performance. For our use case, simple methods such as logistic regression and LDA exhibited amongst the best model performances (accuracy 0.97, 0.962) while needing very little time to train (2.28 min, 0.52 min). Please note that parameter tuning for the knn and Random Forest models were considered part of the wall time. Hence, reducing the search space for parameters for those models or including more extensive parameter tuning in models such as the neural network will have a substantial effect on the depicted wall times. Moreover, models were not trained on a dedicated machine. The computer used for model training was also used for other activities during the training process, which might influence available computational resources and therefore training time. A standardized approach and more replications on a dedicated machines will be necessary to obtain more objective wall times.

```
results_model_performance %>%
  rename("model" = "model",
         "accuracy" = "accuracy",
         "F1" = "F1",
         "wall_time_min" = "wall time (min)") %>%
  knitr::kable(caption = "Model performance on Meyer data",
               align = "lrrr")
```

Table 1: Model performance on Meyer data

model	accuracy	F1	wall time (min)
logistic regression	0.97	0.969	2.28
k-nearest neighbor	0.942	0.942	346.78
random forest	0.97	0.968	120.29
linear discriminant analysis	0.962	0.96	0.52
neural network	0.97	0.968	28.77
ensemble	0.971	0.969	NA

3.2 Parameter tuning

Machine learning algorithms are highly adaptable to different data sets, which makes it necessary to tune particular parameters depending on specific requirements. As described above, in this project we explored parameter tuning for various models. Figure 1 depicts the accuracy of the model as a function of k, the number of neighbors. We chose to test even values from two to 20 and found k = 8 to be ideal parameter under our conditions.

```
plot(fit_knn)
```

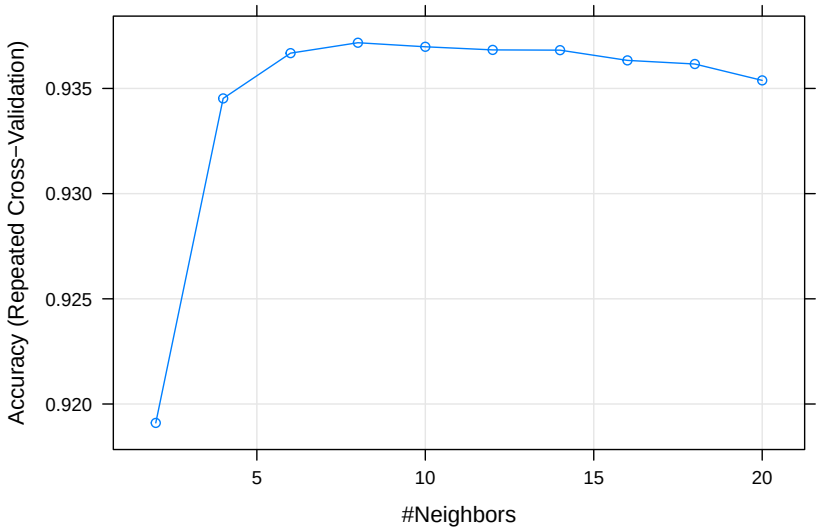

Figure 1: Tuning number of neighbors (k) for knn model

The Random Forest model was optimized by exploring the mtry parameter which represents the number of factors that are being pulled to make a decision at each step (Figure 2). Fifteen numbers of randomly selected predictors were assesed with respect to their influence on model performance. For our data set mtry = 12 was identified to lead to the best model performance. Note that for tuning mtry, the number of trees per random forest was reduced to make the process more time efficient.

```
plot(rf_random)
```

3.3 Variable importance

When computing models, different features of the data set have a different weight. In the following we will explore which features harbor the biggest weight for the trained models. The top 10 variables with their corresponding importance for each model are shown in Figure 3. Importance has been scaled to a value between 0 and 100.

```
# number of variables to plot
n_vars <- 10

# make plots
plot(varImp(fit_glm),
     top = n_vars,
     adj = 0,
     main = "(A) Linear regression")

plot(varImp(fit_knn),
     top = n_vars,
     adj = 0,
```

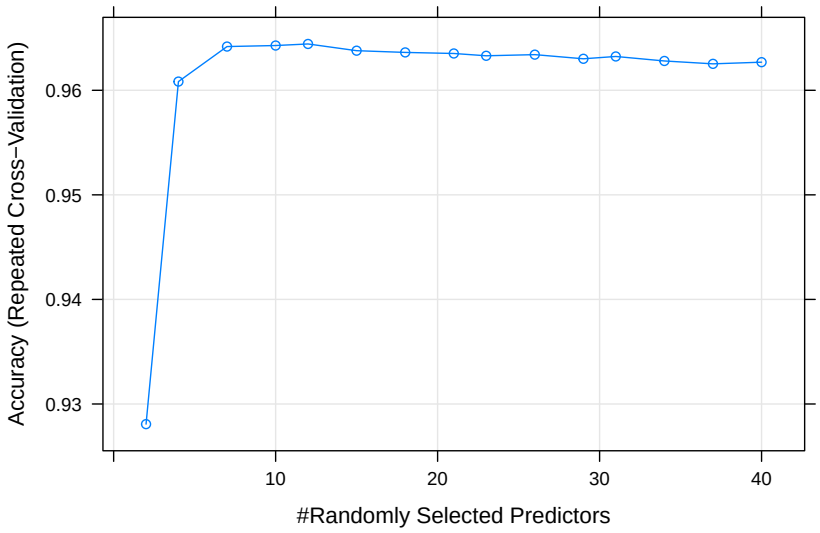

Figure 2: Tuning number of considered factors (mtry) for Random Forest model

```
     main = "(B) K-nearest neigbors")

plot(varImp(fit_nn),
     top = n_vars,
     adj = 0,
     main = "(C) Neural network")

plot(varImp(fit_rf),
     top = n_vars,
     adj = 0,
     main = "(D) Random forest")

plot(varImp(fit_lda),
     top = n_vars,
     adj = 0,
     main = "(E) Linear discriminant analysis")

# cleanup
rm(n_vars)
```

In general, the annotation of the methylated site plays an important role. From other experiments we know that m6A is deposited on transcripts mostly close to the transcriptional end site (TES). Since most introns are spliced out of transcripts we would expect intronic regions to be strong predictors for not methylated regions. Due to the way DART-seq works, another expected predictor of methylated sites is the presence of a C in the plus 1 position. In this technique, C following a methylated A is modified and detected, which allows for indirect detection of methylated As. Since this is an inherent bias of the method and thus should be strongly correlated with methylated sites, it is noteworthy that only models based on a neural network and random forest show this variable having the highest importance. It has also been described, that transcripts are predominantly methylated in the context of a DRACH motif (where A denotes the methylatable adenosine, D denotes A, G or T, R denotes A and G, and H denotes A, C or T). Finding proximal bases to be important for prediction of m6A sites is therefore not surprising.

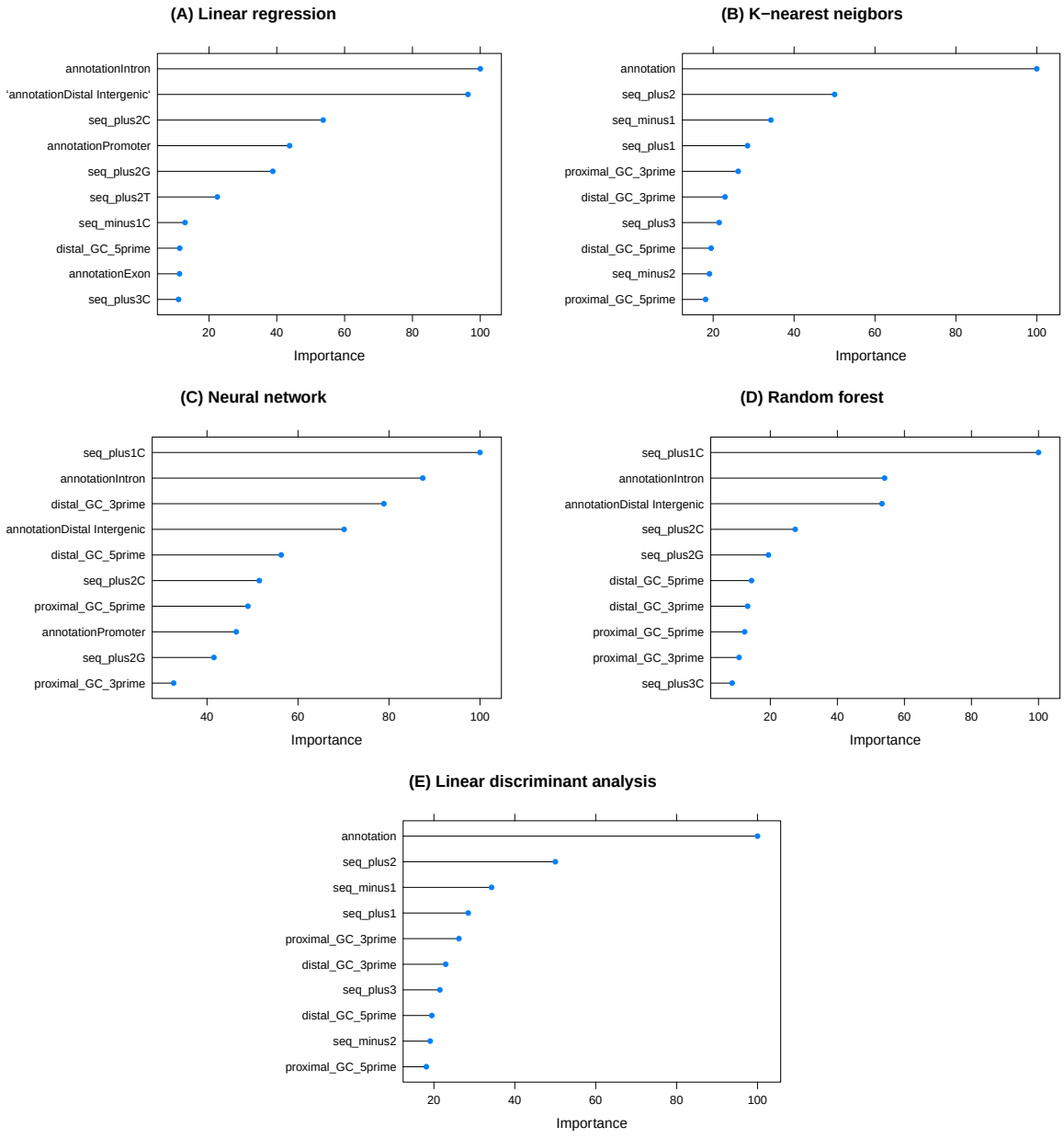

Figure 3: Variable importance

3.4 Model performance on Linder (2015) data set

The ensemble model had an accuracy of 0.753 and an F1 of 0.661 when tested on the Linder (2015) data set, which is substantially lower than for the Meyer (2019) data set.

4 Conclusions

All trained models in this project were able to predict methylation of adenines as a dependent variable better than expected from random sampling. The ensemble model combining all individual models was the best model for prediction. Upon challenging the ensemble model with a new data set of methylated/non methylated adenines the accuracy of the model was strongly reduced. Various reasons could lead to the observed decrease in performance compared to the Meyer (2019) data set. On the one hand, the results could reflect biological variability. The deposition of m6A is highly dynamic and in many studies even the overlap of identified m6A sites between biological replicates has been reported to be far from perfect. On the other hand the difference could be explained by technical issues: Different methods to detect methylated sites were employed by the two different groups. Each method comes with its own set of biases. We know, for instance, that DART-seq is dependent on a C after the methylated A. While the consensus motif for m6A has been described to be DRACH, other motifs such as UGUA have been described in the literature. Hence Meyer et al. might have missed a specific population of methylated adenines. Another possible problem could be the engineered features. While it has been described that m6A is preferentially found in certain regions of transcripts and within specific motifs, it is very likely that m6A deposition depends on much more than pure sequence context. Transcript methylation is a tightly controlled gene regulatory process and other aspects such as RNA secondary structure or the regulation of proteins involved in m6A deposition (readers, writers, erasers) will have a significant effect. We cannot exclude that the generated models were overfit to the Meyer (2019) data set, which could be another way to explain the discrepancy in model performance. Additional parameter tuning for models such as the neural network might help improve the predictive capabilities of the ensemble model too.

In the future it would be interesting to overcome the shortcomings of this project to build a powerful model to predict m6A sites in silico. Ideally we would use a richer and more diverse data set with m6A sites from various different experiments. Engineering more features, thus generating a higher-dimensional data set might greatly improve predictions. Since we know that m6A can have an effect on various aspects of gene expression, other data sets such as RNA-sequencing data might be useful. It is advisable to invest more resources in parameter tuning. Since the relationships within high dimensional data are likely to be non-linear and complex, more sophisticated modeling strategies such as deep neural networks will be instrumental in uncovering the hidden rules of m6A deposition.

5 Appendix

The following R environment was used

```
sessionInfo()
```

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
##
```

```
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] formatR_1.11
##  [2] bookdown_0.24
##  [3] doParallel_1.0.17
##  [4] iterators_1.0.14
##  [5] foreach_1.5.2
##  [6] caret_6.0-90
##  [7] lattice_0.20-45
##  [8] bedr_1.0.7
##  [9] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
## [10] GenomicFeatures_1.44.2
## [11] AnnotationDbi_1.54.1
## [12] Biobase_2.52.0
## [13] ChIPseeker_1.28.3
## [14] BSgenome.Hsapiens.UCSC.hg19_1.4.3
## [15] BSgenome_1.60.0
## [16] rtracklayer_1.52.1
## [17] Biostrings_2.60.2
## [18] XVector_0.32.0
## [19] GenomicRanges_1.44.0
## [20] GenomeInfoDb_1.28.4
## [21] IRanges_2.26.0
## [22] S4Vectors_0.30.2
## [23] BiocGenerics_0.38.0
## [24] BiocManager_1.30.16
## [25] forcats_0.5.1
## [26] stringr_1.4.0
## [27] purrr_0.3.4
## [28] readr_2.1.2
## [29] tidyr_1.2.0
## [30] tibble_3.1.6
## [31] ggplot2_3.3.5
## [32] tidyverse_1.3.1
## [33] dplyr_1.0.8
##
## loaded via a namespace (and not attached):
##   [1] utf8_1.2.2                  R.utils_2.11.0
##   [3] tidyselect_1.1.2            RSQLite_2.2.10
##   [5] grid_4.1.1                  BiocParallel_1.26.2
##   [7] scatterpie_0.1.7            pROC_1.18.0
##   [9] munsell_0.5.0               codetools_0.2-18
##  [11] future_1.24.0               withr_2.5.0
```

```
##  [13] colorspace_2.0-3            GOSemSim_2.18.1
##  [15] filelock_1.0.2              knitr_1.37
##  [17] rstudioapi_0.13             DOSE_3.18.3
##  [19] listenv_0.8.0               MatrixGenerics_1.4.3
##  [21] GenomeInfoDbData_1.2.6      polyclip_1.10-0
##  [23] bit64_4.0.5                 farver_2.1.0
##  [25] treeio_1.16.2               parallelly_1.30.0
##  [27] vctrs_0.3.8                 generics_0.1.2
##  [29] lambda.r_1.2.4              ipred_0.9-12
##  [31] xfun_0.30                   BiocFileCache_2.0.0
##  [33] R6_2.5.1                    graphlayouts_0.8.0
##  [35] bitops_1.0-7                cachem_1.0.6
##  [37] fgsea_1.18.0                gridGraphics_0.5-1
##  [39] DelayedArray_0.18.0         assertthat_0.2.1
##  [41] BiocIO_1.2.0                scales_1.1.1
##  [43] ggraph_2.0.5                nnet_7.3-17
##  [45] enrichplot_1.12.3           gtable_0.3.0
##  [47] globals_0.14.0              tidygraph_1.2.0
##  [49] timeDate_3043.102           rlang_1.0.1
##  [51] splines_4.1.1               lazyeval_0.2.2
##  [53] ModelMetrics_1.2.2.2        broom_0.7.12
##  [55] yaml_2.3.5                  reshape2_1.4.4
##  [57] modelr_0.1.8                backports_1.4.1
##  [59] qvalue_2.24.0               tools_4.1.1
##  [61] lava_1.6.10                 ggplotify_0.1.0
##  [63] gplots_3.1.1                ellipsis_0.3.2
##  [65] RColorBrewer_1.1-2          Rcpp_1.0.8
##  [67] plyr_1.8.6                  progress_1.2.2
##  [69] zlibbioc_1.38.0             RCurl_1.98-1.6
##  [71] prettyunits_1.1.1           rpart_4.1.16
##  [73] viridis_0.6.2               cowplot_1.1.1
##  [75] SummarizedExperiment_1.22.0 haven_2.4.3
##  [77] ggrepel_0.9.1               fs_1.5.2
##  [79] magrittr_2.0.2              futile.options_1.0.1
##  [81] data.table_1.14.2           DO.db_2.9
##  [83] reprex_2.0.1                matrixStats_0.61.0
##  [85] hms_1.1.1                   patchwork_1.1.1
##  [87] evaluate_0.15               XML_3.99-0.9
##  [89] VennDiagram_1.7.1           readxl_1.3.1
##  [91] gridExtra_2.3               testthat_3.1.2
##  [93] compiler_4.1.1              biomaRt_2.48.3
##  [95] KernSmooth_2.23-20          shadowtext_0.1.1
##  [97] crayon_1.5.0                R.oo_1.24.0
##  [99] htmltools_0.5.2             ggfun_0.0.5
## [101] tzdb_0.2.0                  aplot_0.1.2
## [103] lubridate_1.8.0             DBI_1.1.2
## [105] tweenr_1.0.2                dbplyr_2.1.1
## [107] MASS_7.3-55                 rappdirs_0.3.3
## [109] boot_1.3-28                 Matrix_1.4-0
## [111] brio_1.1.3                  cli_3.2.0
## [113] R.methodsS3_1.8.1           gower_1.0.0
## [115] igraph_1.2.11               pkgconfig_2.0.3
## [117] GenomicAlignments_1.28.0    recipes_0.2.0
## [119] xml2_1.3.3                  ggtree_3.0.4
```

```
## [121] hardhat_0.2.0              prodlim_2019.11.13
## [123] rvest_1.0.2                yulab.utils_0.0.4
## [125] digest_0.6.29              rmarkdown_2.12
## [127] cellranger_1.1.0           fastmatch_1.1-3
## [129] tidytree_0.3.9             restfulr_0.0.13
## [131] curl_4.3.2                 gtools_3.9.2
## [133] Rsamtools_2.8.0            rjson_0.2.21
## [135] lifecycle_1.0.1            nlme_3.1-155
## [137] jsonlite_1.8.0             futile.logger_1.4.3
## [139] viridisLite_0.4.0          fansi_1.0.2
## [141] pillar_1.7.0               plotrix_3.8-2
## [143] KEGGREST_1.32.0            fastmap_1.1.0
## [145] httr_1.4.2                 survival_3.2-13
## [147] GO.db_3.13.0               glue_1.6.2
## [149] png_0.1-7                  bit_4.0.4
## [151] ggforce_0.3.3              class_7.3-20
## [153] stringi_1.7.6              blob_1.2.2
## [155] caTools_1.18.2             memoise_2.0.1
## [157] future.apply_1.8.1         ape_5.6-2
```