Tutorial for R package eClock

Yu Liu

3/11/2022

Introduction

DNA methylation (DNAm) has a close relationship to senescence, so it has been used to develop statistical models, or called clock models, to predict chronological lifespan ages or gestational ages accurately *[1-5]*.

A clock model is essentially a penalized regression model using squared error loss function to calculate empirical risk and using norm of the regression coefficients to calculate structural risk. Because it is learned from data, its performance is heavily influenced by sample age distribution, and if that of the training data is biased, the model performance on samples with a small distribution density will become weak. To solve this problem, we developed the R package *eClock (ensemble-based clock)*. It combines bagging and SMOTE (Synthetic Minority Over-sampling Technique) methods to adjust the biased distribution and predict DNAm age with an ensemble model, which can improve the model performance on rare samples significantly. In addition, it also provides other functions to train the normal clock model, display features, convert methylation probe/gene/DMR (DNA methylation region) values, etc.

Package installation

Code and documentation of *eClock* are freely available at https://github.com/yuabrahamliu/eClock.

The following commands can be used to install this R package.

```
library(devtools)

install_github('yuabrahamliu/eClock')
```

Data preparation

To demonstrate the functions of *eClock*, this tutorial uses a dataset that accompany with the package. It includes a matrix with 359 rows and 18626 columns. Each row represents a sample and each column represents a DNA methylation probe from Infinium 27K BeadChip platform. The values recorded in this matrix are the DNA methylation beta values detected from the samples, which are placenta tissues from donors. Among the 359 samples, 258 of them are from healthy or uncomplicated donors (control samples), while the remaining 101 ones are from patients with preeclampsia (preeclampsia samples), which is a pregnancy complication coupled with accelerated placental aging *[5]*. The row names of the matrix are the sample IDs while the column names are the DNA methylation probe IDs.

In addition to this matrix, a data.frame is also with this dataset, which records the meta data for the 359 samples, including the sample IDs (column "sampleid" in this data.frame, corresponding to the row names of the beta value matrix), their control/preeclampsia status (column "condition"), and the gestational ages of the donors (column "Response"). Actually, the 359 samples are combined from different GEO (Gene

Expression Omnibus) datasets, and so the original GEO dataset IDs of the samples are shown by the "dataset" column.

Now, attach *eClock* to the R session and get the example data.

```
library(eClock)

betas <- system.file('extdata', 'placentabetas.rds', package = 'eClock')
betas <- readRDS(betas)

pds <- system.file('extdata', 'placentapds.rds', package = 'eClock')
pds <- readRDS(pds)
```

The beginning parts of the beta value matrix and the meta data.frame are shown below.

```
betas[1:6, 1:6]
#>           cg00000292 cg00002426 cg00003994 cg00007981 cg00008493 cg00008713
#> GSM788417  0.5131720  0.4642324  0.4638180  0.4025065  0.5440042  0.3299099
#> GSM788419  0.5524222  0.4729627  0.4338205  0.3452889  0.5286614  0.2623551
#> GSM788420  0.5071099  0.4681941  0.4386223  0.3688171  0.5474948  0.2875639
#> GSM788421  0.5159766  0.4755099  0.4427750  0.3296488  0.5773932  0.2529015
#> GSM788414  0.5345167  0.4710890  0.4334234  0.3299371  0.6096593  0.1925150
#> GSM788415  0.5486957  0.4648642  0.4439560  0.3295007  0.6011283  0.2008011
```

```
head(pds)
#>    sampleid  dataset condition Gestwk Response
#> 1 GSM788417 GSE31781   Control      8        8
#> 2 GSM788419 GSE31781   Control      8        8
#> 3 GSM788420 GSE31781   Control      8        8
#> 4 GSM788421 GSE31781   Control      9        9
#> 5 GSM788414 GSE31781   Control     12       12
#> 6 GSM788415 GSE31781   Control     12       12
```

Clock model training

To train the clock model, we only use the control samples in the dataset and the preeclampsia samples will be used later.

```
ctrlpd <- subset(pds, condition == 'Control')
ctrlbetas <- betas[ctrlpd$sampleid,]
```

Then, we will train the bagging-SMOTE based clock model (called as balanced model hereafter), but before that, we first use the function **resamplebin** in the package to show the original sample gestational age distribution and the adjustment effect of this function. Several parameters should be set to run this task. We transfer the data.frame **ctrlpd** to the parameter **resrange**, which needs a data.frame with at least two columns, one is the column "sampleid" while the other is the one named "Response" recording the response variable (gestational age here). Besides, we transfer the matrix **ctrlbetas** to the parameter **samplevar**, which requires a matrix recording the feature values (methylation probe beta values here) for all the samples with rows representing samples and columns representing features.

This function will first equally divide the gestational age range into several bins, and count the sample numbers within each bin. If any of them has a sample number less than 2, the original bin width will be increased by a step size and the age range will be re-divided using this wider width to include more samples

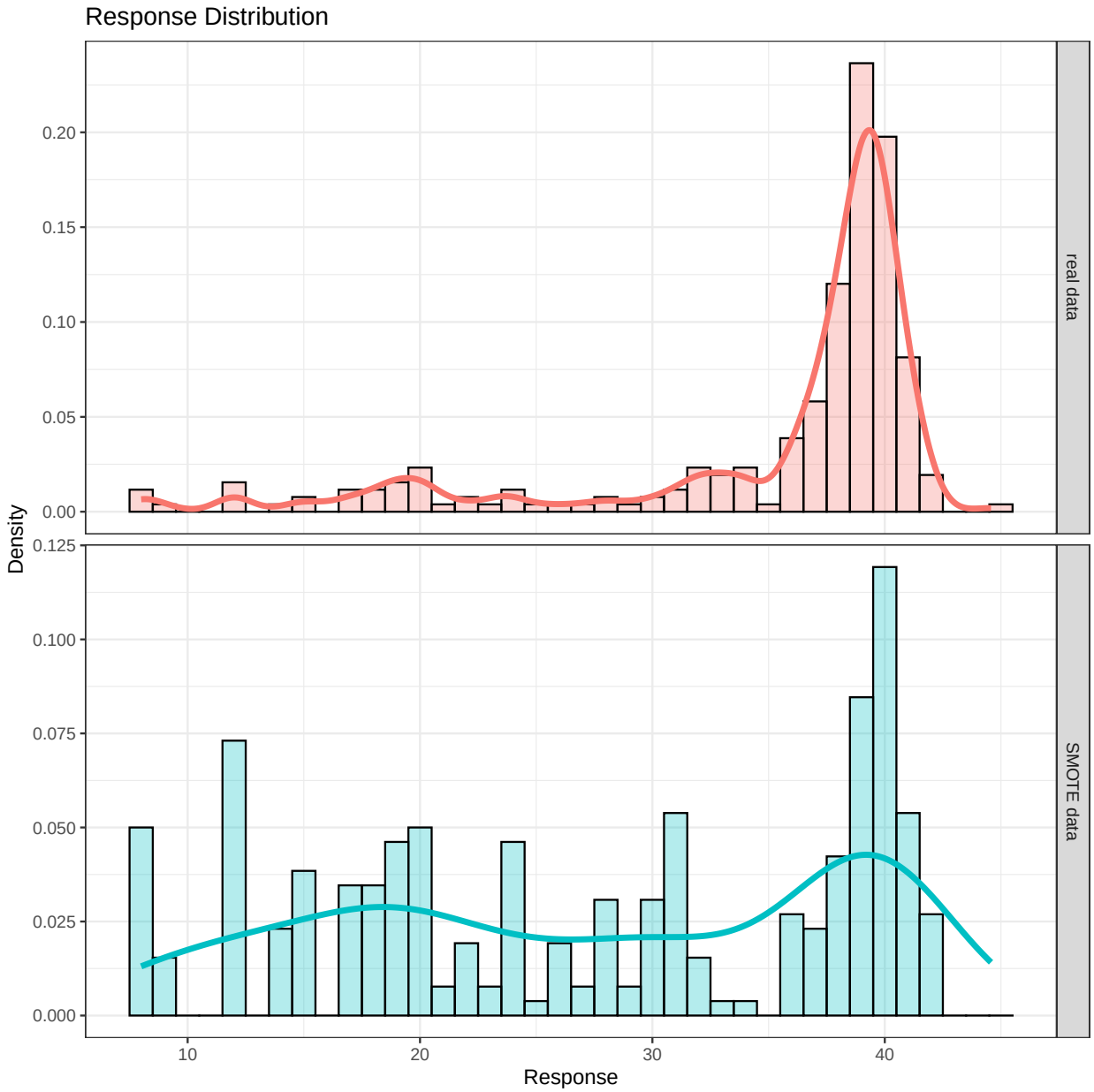
for each bin. This recursion process will be continued until all the bins have no less than 2 samples. As to the function parameters, `binwith` is used to set the initial bin width, while `minwith` is the increased step size to be added to each bin.

After this recursion, for each bin, under or over-sampling will be used on the samples there, until the final sample number reaches the ceiling of *total sample number/bin number*. The under-sampling will be done via bootstrapping while the over-sampling will be performed with SMOTE. Both of them will be conducted based on the same random seed and it is set using the parameter `sampleseed`.

There is another parameter named `balanceidx`. It is used to calculate a metric named "balance index" to quantify the imbalance of the original sample age distribution. A value less than 4 indicates a serious biased distribution and the smaller this value is, the more biased the distribution is. If this parameter is set as TRUE, the balance index of the original distribution will be computed.

Finally, the `plotting` parameter should be set as TRUE so that the histograms of the age distribution before and after the adjustment can be shown directly.

```
ctrladjust <- resamplebin(resrange = ctrlpd,
                          samplevar = ctrlbetas,
                          binwith = 1,
                          minwith = 1,
                          sampleseed = 2022,
                          balanceidx = TRUE,
                          plotting = TRUE)
```



The whole dataset has a gestational age range from 8 weeks to 45 weeks, while from the upper histogram, it can be seen that the original distribution has a serious bias toward samples with a gestational age between 36 weeks and 42 weeks, corresponding to a balanced index as small as 2.66, which is recorded in the `balanceidx` slot of the result returned. While after the adjustment via bootstrapping and SMOTE, the new samples have a more balanced age distribution, as shown by the lower histogram.

In addition to `balanceidx`, the returned result `ctrladjust` also contains a slot named "resampleresponse", which is a data.frame recording the response variable (gestational age) values of the new samples generated by bootstrapping-SMOTE. While another slot contained in `ctrladjust` is `resamplevars`. It is a matrix with methylation probe values for the new samples, which is similar to the original sample feature matrix `ctrlbetas`. Alougth the samples undergoing bootstrapping during the adjustment still keep the same probe values and age values, which can be found from the orignal meta data and feature matrix, the SMOTE method used by this function synthesizes some new samples via interpolation and cannot be found in the original matrix, so it is necessary to return the new feature matrix with the result.

Then, the slots `resamplevars` and `resampleresponse` can be used to construct a clock model. However, because the bins undergoing under-sampling during the adjustment have some samples discarded, to rescue them, a bagging framework is introduced to the balance model, so that the original dataset will be sampled for several times to generate several different adjusted subsets, and although each subset losses some sample information due to the under-sampling, for the whole of them, all the samples are used. Then, for each subset, a base learner will be trained and finally they will be ensembled together to get an ensemble clock model.

The function to train such a balanced ensemble model is `singlebalance`. It has integrated the function `resamplebin` in it so actually we don't need to run `resamplebin` separately as above and can start the task from the orignal data `ctrlbetas` and `ctrlpd`.

```
balanceres <- singlebalance(orivar = ctrlbetas,
                            oripd = ctrlpd,
                            seednum = 2022,
                            fold = 0.75,
                            balancing = TRUE,
                            binwith = 1,
                            samplenum = 10,
                            alphas = 0.5,
                            errortype = '1ses',
                            prescreen = FALSE,
                            cores = 1,
                            plotting = FALSE)
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.327
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.434
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.389
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.553
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.308
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.281
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.332
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.446
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.339
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.318
```

The beta value matrix `ctrlbetas` and the meta data are transferred to the parameters `orivar` and `oripd` respectively. This function first divides the samples into a training set and a testing set randomly. Its random seed is set by the parameter `seednum`, and the proportion of the training samples in the whole samples can be defined by the parameter `fold`. Here, we set it as 0.75, meaning 75% (194 samples) of the whole samples will be used to train the ensemble model. While the remaining 64 samples will be assigned to the testing set and will only used to test the model performance without any participation in the model training.

To do the distribution adjustment to train the model, the parameter `balancing` need to be set as TRUE, and it should be noted that the function `resamplebin` integrated in `singlebalance` will only adjust the training

samples, although in the example above we used it to adjust all the samples. The parameter `binwith` defines both the initial bin width and the step size of the adjustment process.

The parameter `samplenum` is the number of base learners will be trained for the ensemble model, we set it as 10 here. For each base learner, it first does a feature selection to find the probes most related to the age. This is achieved via the elastic net regularization and the alpha value controlling the balance of L1 and L2 penalties can be defined and transferred using the parameter `alphas`. It is set as 0.5 here. On the other hand, the regularization constant lambda is chosen during 10-fold cross validation, and the best alpha-lambda combination will be selected. When the function is running, the best combination for each base learner will be printed. The choice of the best one is controlled by the parameter `errortype`. If it is set as "1ses", the combination giving a cross-validation error within one standard error of the minimum one will be chosen, while if it is "min", that giving the minimum will be used. Before this elastic net selection, a preliminary feature selection can also be performed if the parameter `prescreen` is TRUE. It calculates the Pearson correlation coefficients between age and the probes and only the ones with a correlation coefficient p-value less than 0.05 will be kept and transferred to the model training step.

This function also supports parallelization and the parameter `cores` is used to set the threads number.

The returned result `balanceres` contains several slots. Among them, `baselearners` and `baseweights` are the 10 base learners trained here and their weights used to ensemble the whole model. Then, `traincomp` and `testcomp` are two matrices recording the true gestational ages and the ensemble model predicted ones for the training samples and the testing samples. For the features selected by this model, the slot `modelscores` records them and their scores, which reflects their importance. The larger the score is, the more important a feature is. While `modelcoeffs` records the regression coefficients of these features in the 10 base learners. It is a matrix and each row is one feature, each column is one base learner.

Actually, `singlebalance` can also train another ensemble based clock model, but it only uses bagging without SMOTE and doesn't do distribution adjustment for the base learners. Just set the parameter `balancing` as FALSE, and keep other parameters same as above, then this model (called as bootstrapped model hereafter) can be trained.

```
bootres <- singlebalance(orivar = ctrlbetas,
                         oripd = ctrlpd,
                         seednum = 2022,
                         fold = 0.75,
                         balancing = FALSE,
                         binwith = 1,
                         samplenum = 10,
                         alphas = 0.5,
                         errortype = '1ses',
                         prescreen = FALSE,
                         cores = 1,
                         plotting = FALSE)
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.231
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.73
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.613
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.461
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.293
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.539
#> Choose the elastic net mixing parameter (alpha) as 0.5
```

```
#> Choose the regularization constant (lambda) as 0.184
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.35
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.257
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.361
```

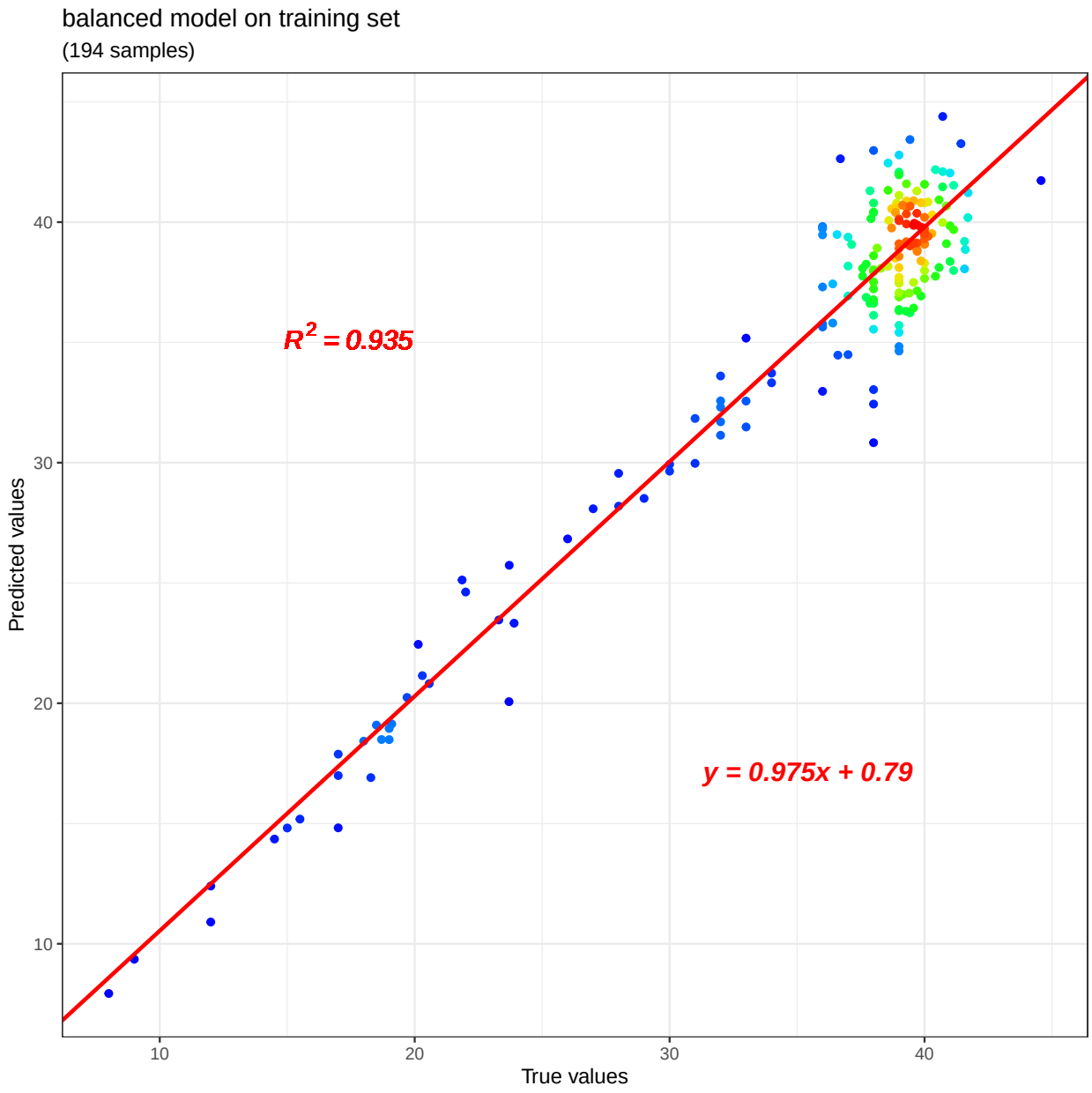
In addition to these ensemble based models, *eClock* also provides a function to construct a normal clock model that is a single elastic net model without distribution adjustment (called as normal model hereafter). This can be fulfilled by the function `singleselection` and the usage of its parameters are the same as `singlebalance`.

```
singleres <- singleselection(orivar = ctrlbetas,
                             oripd = ctrlpd,
                             seednum = 2022,
                             fold = 0.75,
                             alphas = 0.5,
                             errortype = '1ses',
                             prescreen = FALSE,
                             cores = 1,
                             plotting = FALSE)
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 1.01
```

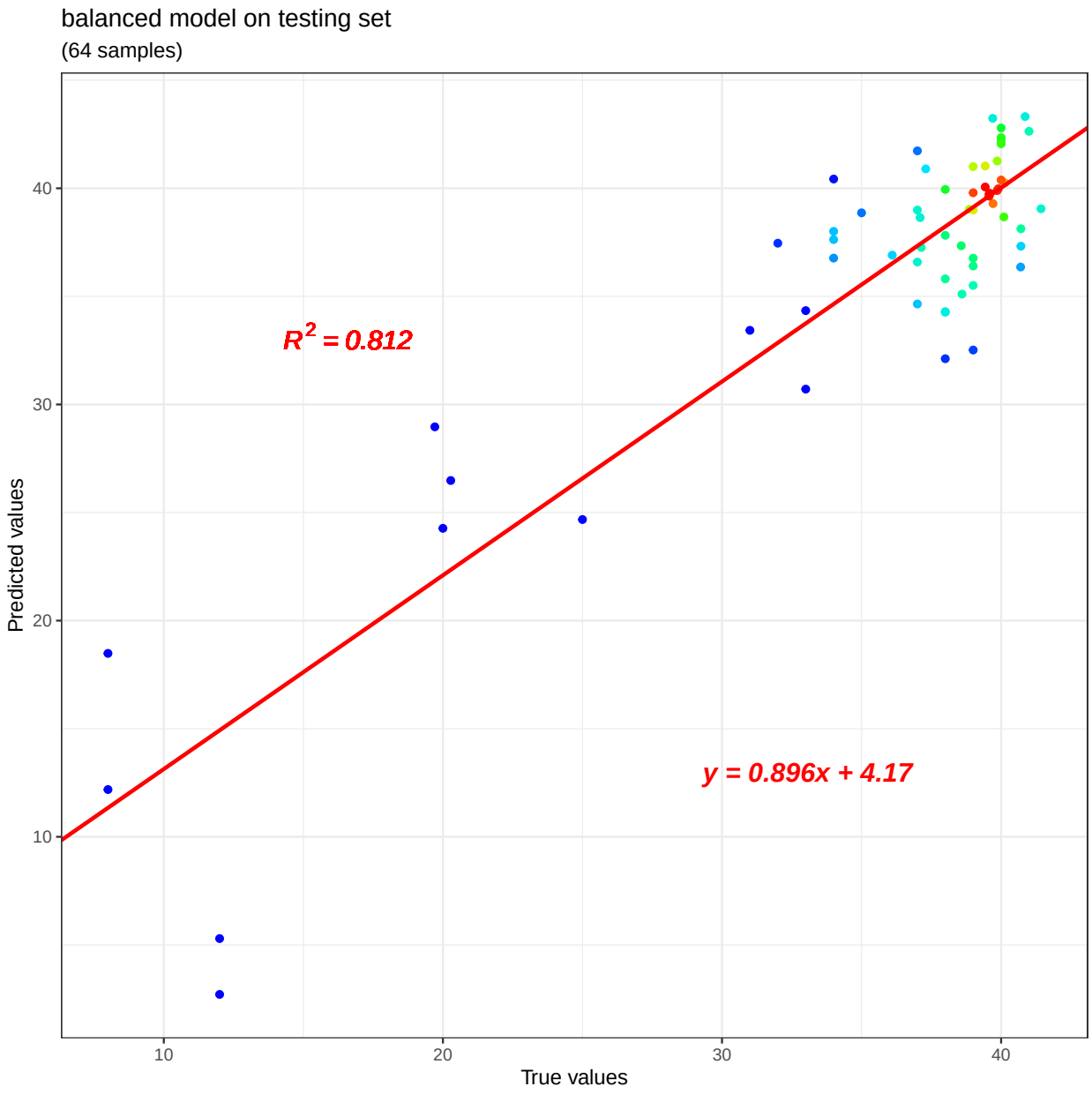
Model evaluation

The package also provides some functions to facilitate plotting and evaluation of the model performance. For example, to show the scatter plot comparing the true gestational ages and the model predicted ones, the function `scatterplot` can be used. For the balanced model, its scatter plots for the training and testing datasets can be generated by setting the parameter `comptab` as the `traincomp` and the `testcomp` slots of `balanceres` respectively. This parameter needs a matrix or a table with the columns named "Prediction" and "True" recording the model predicted and the true response values. Another parameter named "colorful" is used to define whether the dots in the scatter plot should be colored using gradually changed colors to reflect the density of the dots in the plot, and if it is set as TRUE, the density color will be used.

```
scatterplot(comptab = balanceres$traincomp,
            title = 'balanced model on training set',
            colorful = TRUE)
```




```
scatterplot(comptab = balanceres$testcomp,
            title = 'balanced model on testing set',
            colorful = TRUE)
```



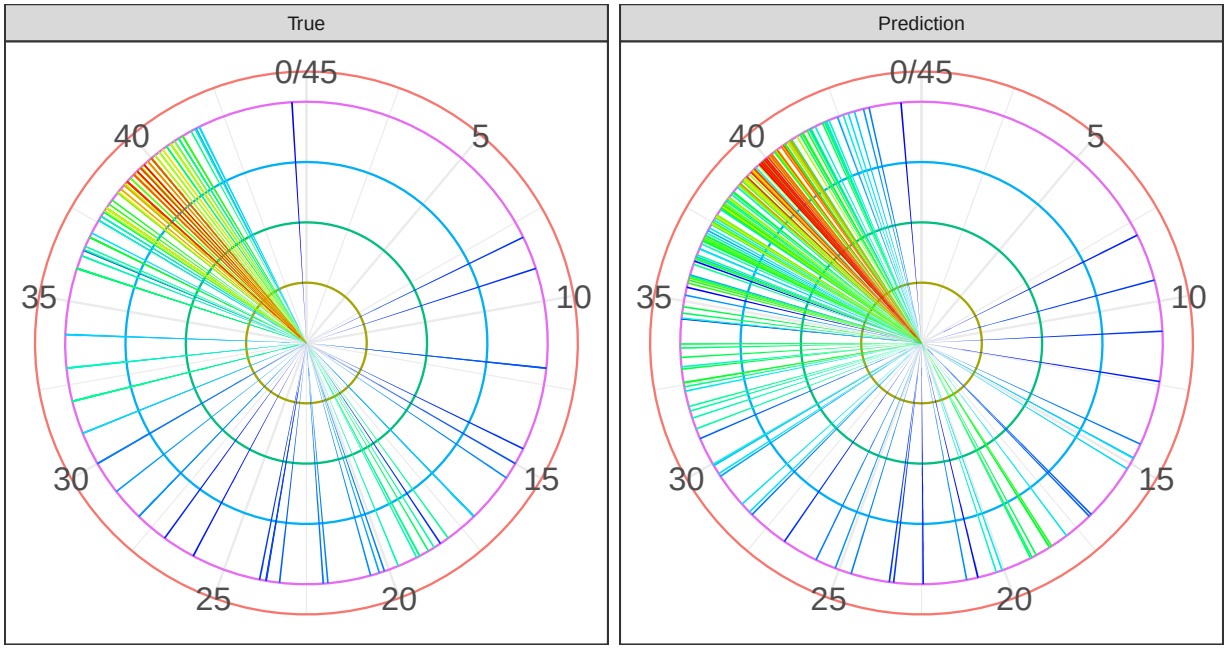
From the plots generated, it can be seen that the balanced model achieves an R square of 0.935 on the 194 training samples, while for the 64 testing samples that never participate in the model training, its R square on them is 0.812. The color gradient of the dots clearly reflects that the original samples have a distribution bias toward gestational ages greater than 36 weeks.

In addition to scatter plot, clock plot can also be used to compare the predicted and true ages, and the function `clockplot` can be used with the same parameters.

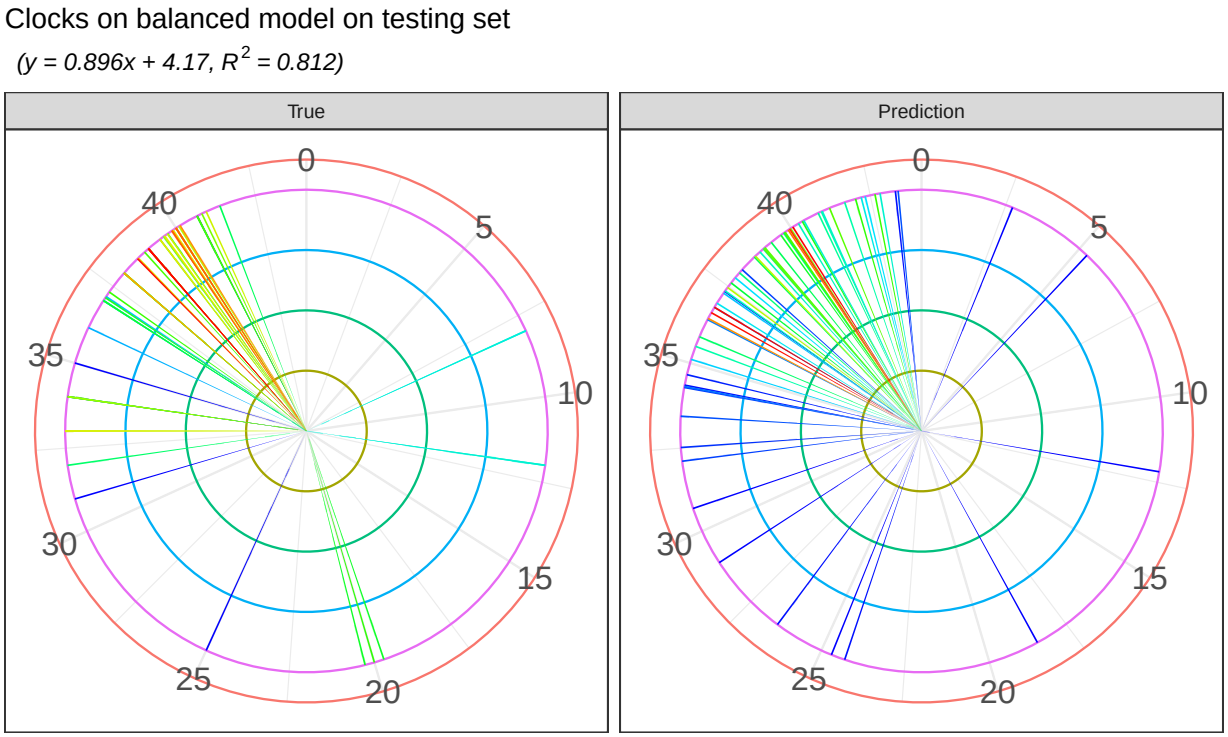
```
clockplot(comptab = balanceres$traincomp,
          title = 'balanced model on training set',
          colorful = TRUE)
```

Clocks on balanced model on training set

(y = 0.975x + 0.79, R^2 = 0.935)




```
clockplot(comptab = balanceres$testcomp,
          title = 'balanced model on testing set',
          colorful = TRUE)
```



The scale around the dial indicates the gestational age and each pointer represents one sample. The color gradients of the pointers indicate the density of the samples. These clock plots are provided according to the name of "clock models".

For the bootstrapped model and the normal model, their results can also be plotted like this, and the R squares will be shown on the plots. More directly, the training set R square of the bootstrapped model is,

```
cor(bootres$traincomp[,1], bootres$traincomp[,2])^2
#> [1] 0.9558295
```

And that of the normal model is,

```
cor(singleres$traincomp[,1], singleres$traincomp[,2])^2
#> [1] 0.9292126
```

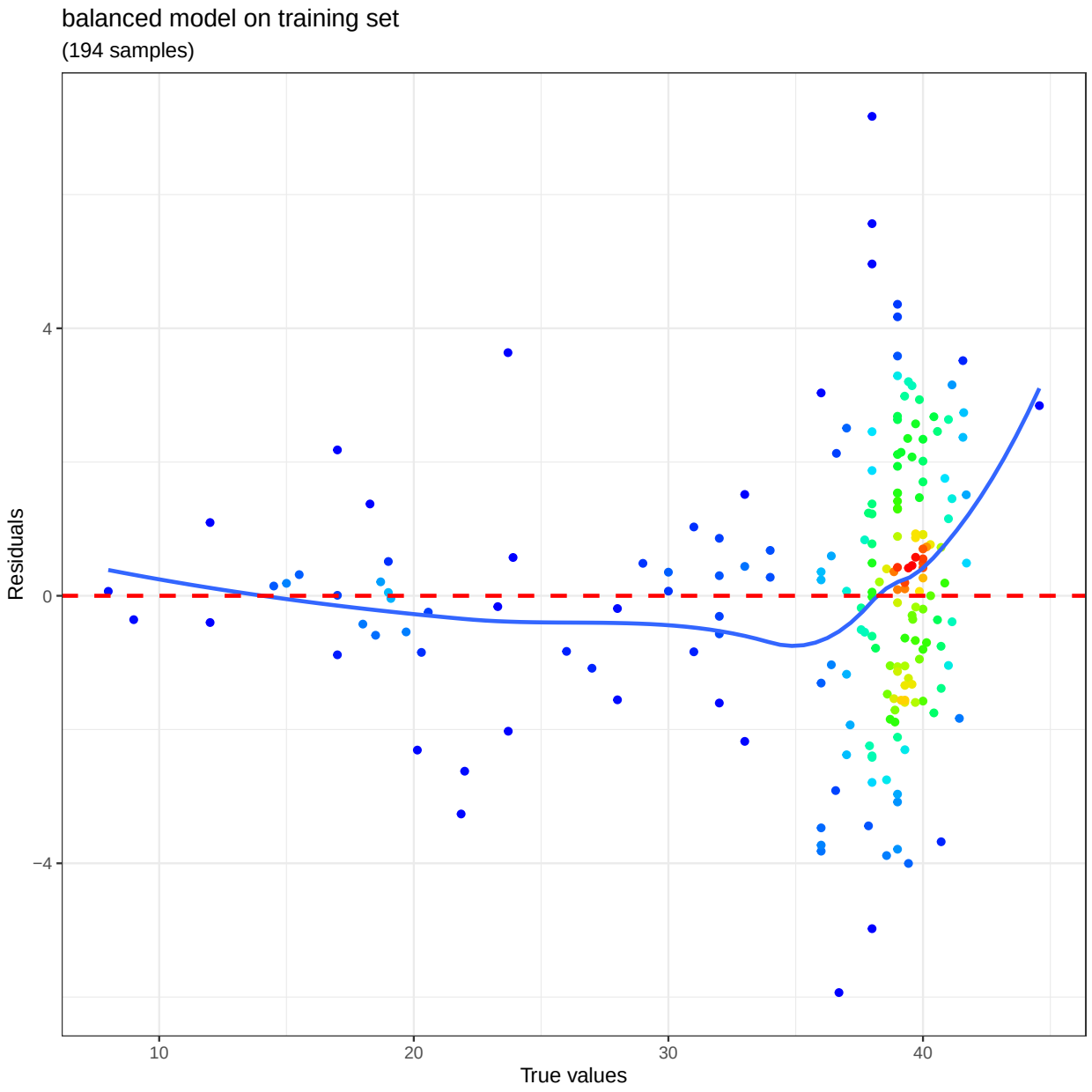
While for the testing set, their R squares are,

```
cor(bootres$testcomp[,1], bootres$testcomp[,2])^2
#> [1] 0.8070754
```

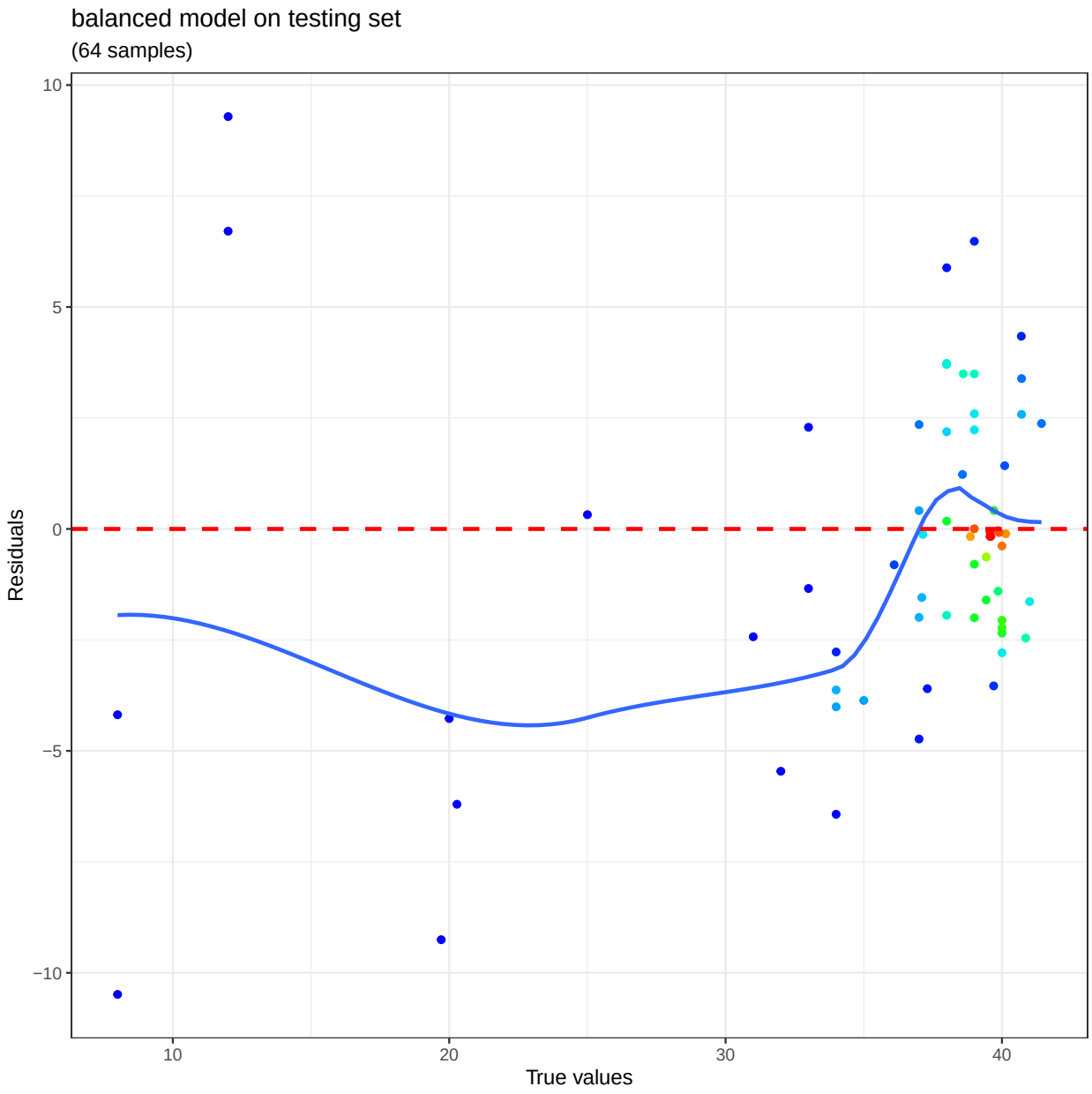
```
cor(singleres$testcomp[,1], singleres$testcomp[,2])^2
#> [1] 0.8228099
```

Hence, the balanced model, bootstrapped model, and normal model achieve an R square of 0.935, 0.956, and 0.929 on the training set, while that on the testing set are 0.812, 0.807 and 0.823. These values are very similar, and the normal model is actually a little better, in view of its best R square on the testing set. However, R square only reflects the overall consistency between all the predicted and true ages, it is also important to look into the residuals for samples with different gestational ages, and the function `residualplot` is provided for this purpose. It shows the sample residuals along the age range, and for the balanced model, the plot can be made similarly as the scatter plot and clock plot.

```
residualplot(comptab = balanceres$traincomp,
             title = 'balanced model on training set',
             colorful = TRUE)
```

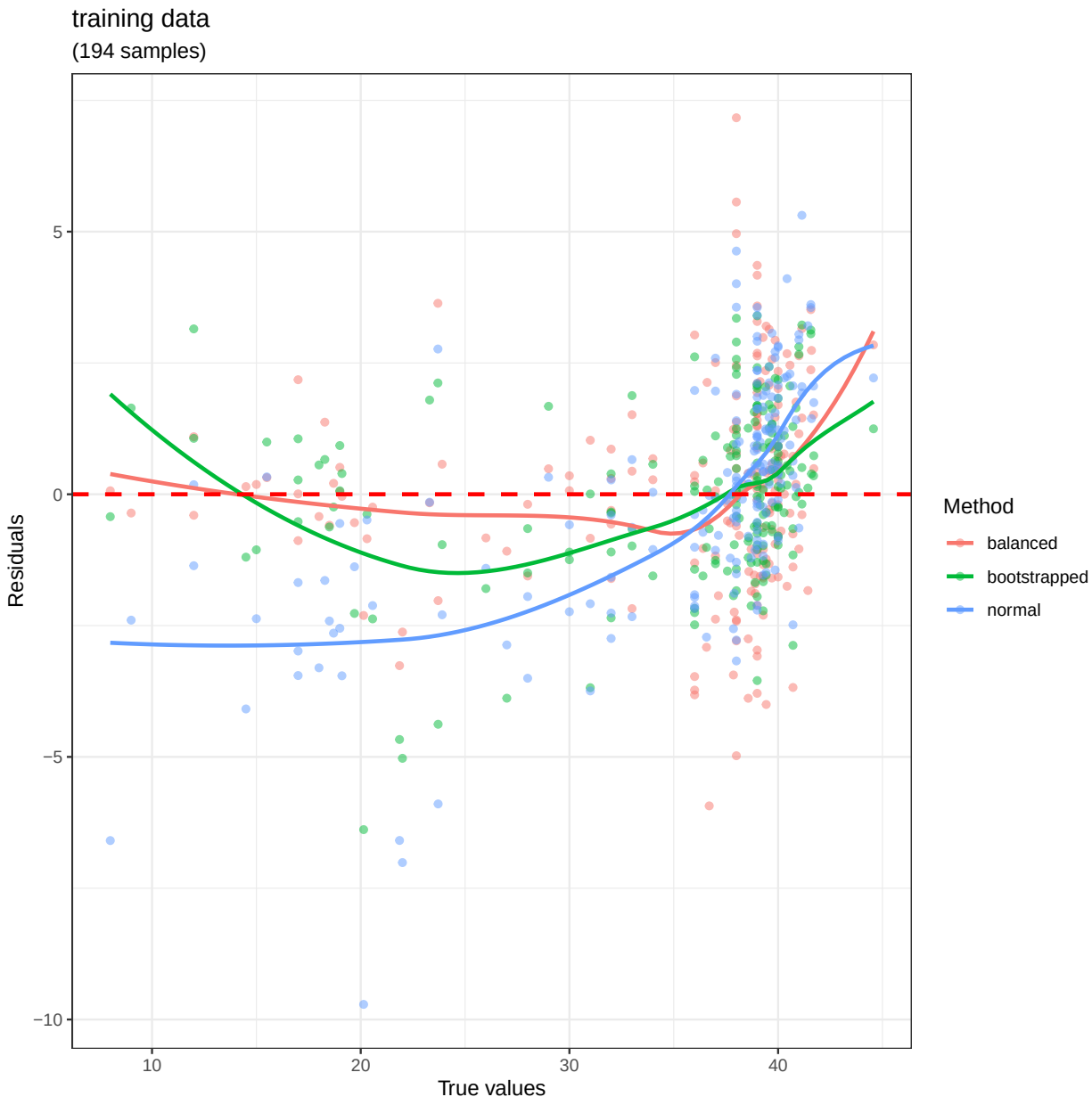



```
residualplot(comptab = balanceres$testcomp,
             title = 'balanced model on testing set',
             colorful = TRUE)
```

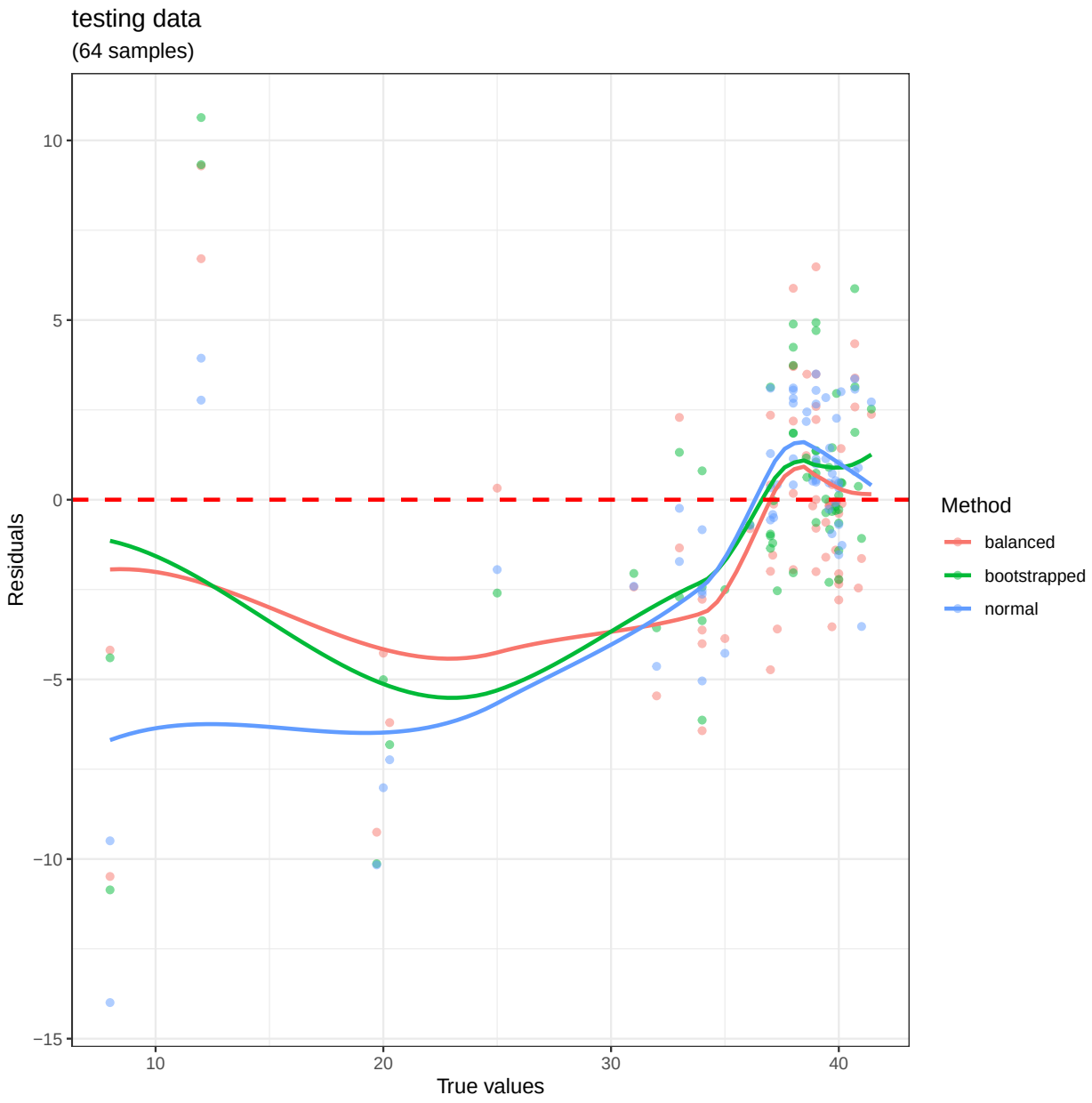


For the bootstrapped model and the normal model, their residual plots can also be generated separately using this function. If want to compare the residuals of the three models directly, the function `residualcomp` can be used. It needs a parameter named "comptabs", which is actually a list consisting of the prediction-true value matrices of the models and correspondingly, a parameter named "methodnames" is required to illustrate the model names for each matrix in `comptabs`.

```
residualcomp(comptabs = list(balanceres$traincomp,
                             bootres$traincomp,
                             singleres$traincomp),
             title = 'training data',
             methodnames = c('balanced', 'bootstrapped', 'normal'))
```




```
residualcomp(comptabs = list(balanceres$testcomp,
                             bootres$testcomp,
                             singleres$testcomp),
             title = 'testing data',
             methodnames = c('balanced', 'bootstrapped', 'normal'))
```



From these residual plots, it can be seen that for the wide gestational age range from 8 weeks to 35 weeks, but with a small sample number, the balanced model fitted much better than the other two, indicating the advantage of the bagging-SMOTE strategy.

The model performance on the 64 testing samples never participate in the model training process have been shown above. Actually, the predicted ages on them were obtained via the function `ensemblepredict` that is integrated in and used by the functions `singlebalance` and `singleselection`. If want to use the models on other datasets, such as the 101 preeclampsia samples above, this function can be used separately. Just set up the preeclampsia samples and transfer them to the parameter `vardat`, together with the model trained, `balanceres`, `bootres`, or `singleres`.

```
preepd <- subset(pds, condition == 'Preeclampsia')
preebetas <- betas[preepd$sampleid,]
```

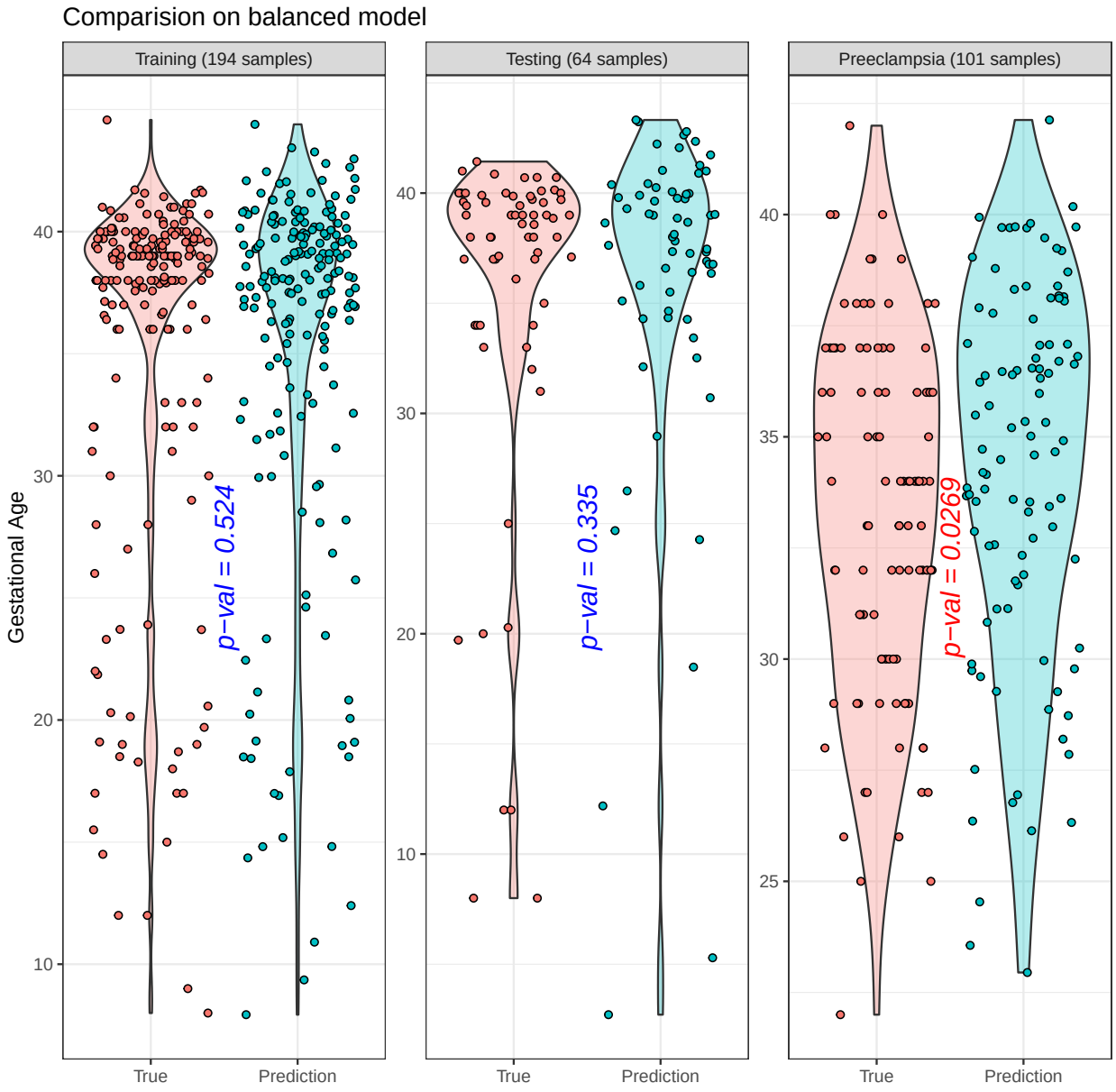
```
balancepredict <- ensemblepredict(balanceres = balanceres,
                                  vardat = preebetas)
```

The result `balancepredict` is a matrix with one column named "Prediction" and several rows with the sample IDs as row names, which records the predicted DNAm ages on the preeclampsia samples. Then, combine it with the true age values and do some comparison on them. Because preeclampsia is a pregnancy complication coupled with accelerated placental aging as reported before *[5]*, it can be expected the DNAm ages should be greater than the chronological ages (true ages).

```
balancepredict <- balancepredict[preepd$sampleid, , drop = FALSE]
balancepredictcomp <- data.frame(Prediction = balancepredict,
                                 True = preepd$Response,
                                 stringsAsFactors = FALSE)
row.names(balancepredictcomp) <- row.names(balancepredict)

biologicalviolin <- function(comptablist,
                             title,
                             groupnames,
                             ylabname,
                             testdir){

  if(length(comptablist) <= 1){
    return(NULL)
  }

  for(i in 1:length(comptablist)){

    comptab <- comptablist[[i]]
    comptab <- as.data.frame(comptab, stringsAsFactors = FALSE)
    comptab$Group <- groupnames[i]

    if(i == 1){
      dats <- comptab
    }else{
      dats <- rbind(dats, comptab)
    }
  }

  predictpart <- dats[,c(1, 3)]
  truepart <- dats[,c(2, 3)]
  predictpart$Type <- "Prediction"
  truepart$Type <- "True"
  predictpart$Samples <- row.names(predictpart)
  truepart$Samples <- row.names(truepart)
  row.names(predictpart) <- 1:nrow(predictpart)
  row.names(truepart) <- 1:nrow(truepart)

  names(predictpart) <- names(truepart) <-
    c('Value', 'Group', 'Type', 'Samples')
  dats <- rbind(predictpart, truepart)
  row.names(dats) <- 1:nrow(dats)

  dats$Type <- factor(x = dats$Type, levels = c('True', 'Prediction'),
```

```
                    ordered = TRUE)

for(j in 1:length(unique(groupnames))){

  groupname <- unique(groupnames)[j]
  groupset <- subset(dats, Group == groupname)
  groupnum <- 0.5*nrow(groupset)

  groupset$Group <- paste0(groupname, ' (', groupnum, ' samples)')

  truevals <- subset(groupset, Type == 'True')$Value
  predictvals <- subset(groupset, Type == 'Prediction')$Value

  wilcoxp <- wilcox.test(predictvals, truevals,
                         alternative = testdir)$p.value
  wilcoxp <- signif(wilcoxp, 3)

  ymed <- mean(c(min(c(truevals, predictvals)),
                 max(c(truevals, predictvals))))

  if(j == 1){
    groupsets <- groupset
    wilcoxps <- wilcoxp
    ymeds <- ymed
  }else{
    groupsets <- rbind(groupsets, groupset)
    wilcoxps <- c(wilcoxps, wilcoxp)
    ymeds <- c(ymeds, ymed)
  }

}

dats <- groupsets
rm(groupsets)

dats$Group <- factor(x = dats$Group, levels = unique(dats$Group),
                     ordered = TRUE)

wilcoxpdat <- data.frame(Label = wilcoxps,
                         Group = unique(dats$Group),
                         Ycord = ymeds,
                         stringsAsFactors = FALSE)
wilcoxpdat$Group <- factor(x = wilcoxpdat$Group,
                           levels = unique(wilcoxpdat$Group),
                           ordered = TRUE)
wilcoxpdat$Label <- paste0('p-val = ', wilcoxpdat$Label)


p <- ggplot2::ggplot(dats)
print(
  p + ggplot2::geom_violin(ggplot2::aes(x = Type, y = Value,
                                        fill = Type), alpha = 0.3) +
    ggplot2::geom_jitter(ggplot2::aes(x = Type, y = Value,
```

```
                                        fill = Type), shape = 21) +
      ggplot2::xlab('') + ggplot2::ylab(ylabname) +
      ggplot2::ggtitle(paste0('Comparision on ', title)) +
      ggplot2::scale_fill_discrete(guide = FALSE) +
      ggplot2::facet_wrap(c('Group'), nrow = 1, scales = 'free') +
      ggplot2::geom_text(data = wilcoxpdat,
                         ggplot2::aes(x = 1.5, y = Ycord,
                                      group = Group, label = Label),
                         parse = FALSE, fontface = 'italic', size = 5,
                         col = rep(c('blue', 'red'), nrow(wilcoxpdat))[c(rbind(wilcoxps >= 0.05,
                                                                               wilcoxps < 0.05))],
                         angle = 90) +
      ggplot2::theme_bw()

  )

}

biologicalviolin(comptablist = list(balanceres$traincomp,
                                    balanceres$testcomp,
                                    balancepredictcomp),
                 title = 'balanced model',
                 groupnames = c('Training', 'Testing', 'Preeclampsia'),
                 ylabname = 'Gestational Age',
                 testdir = 'greater')
```



From the plots, it can be seen that, although both the testing samples and the preeclampsia samples never participate in the model training, the testing samples with a control status have a predicted age similar to their true age, but the preeclampisa samples have a predicted age significantly greater than their true one, indicating their accelerated aging status.

Feature annotation

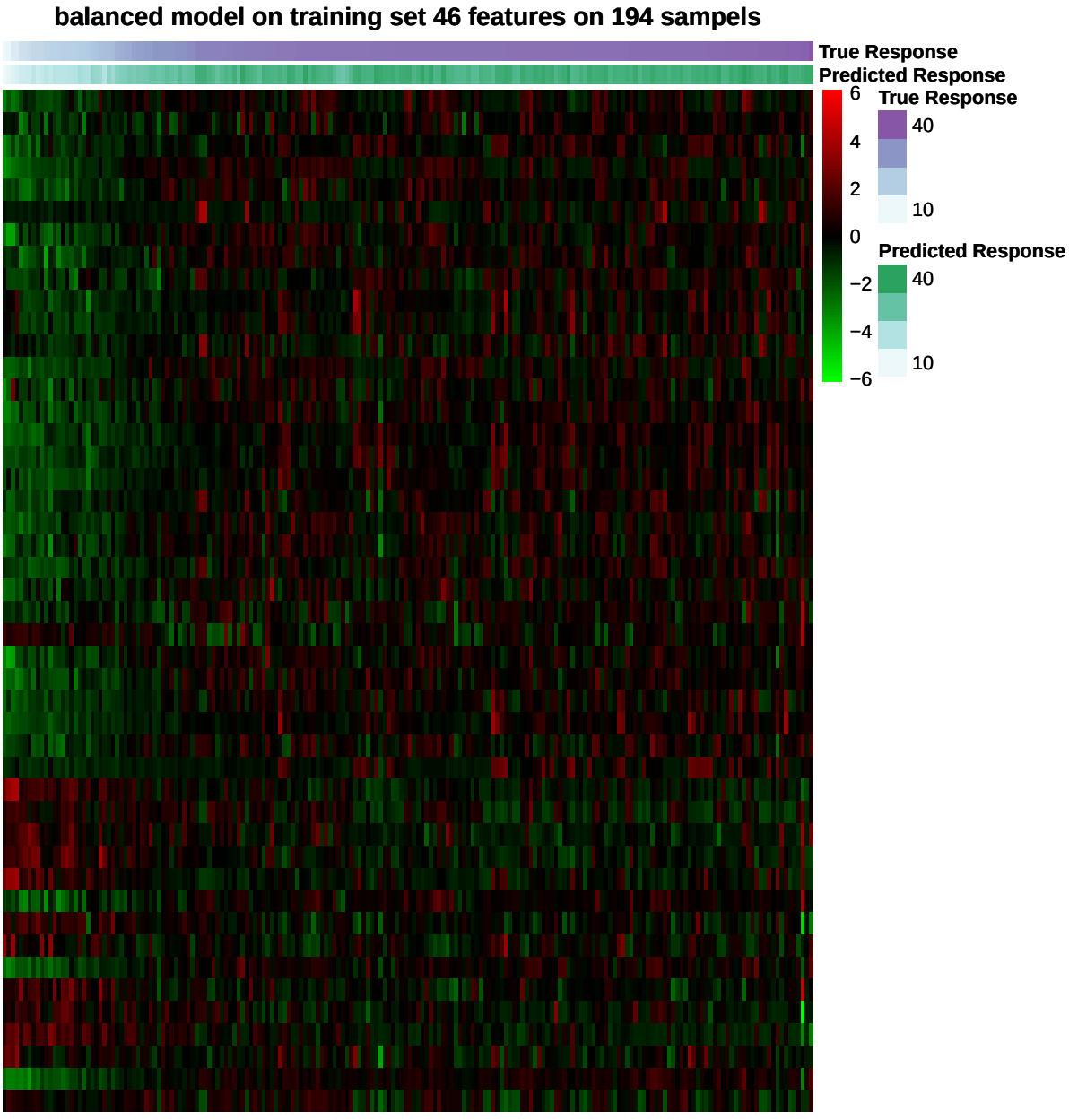
In addition to model performance evaluation, it is also important to have an understanding in the features selected by the models. The function `extractprobes` can be used to extract the features (here is the methylation probes) from the model result.

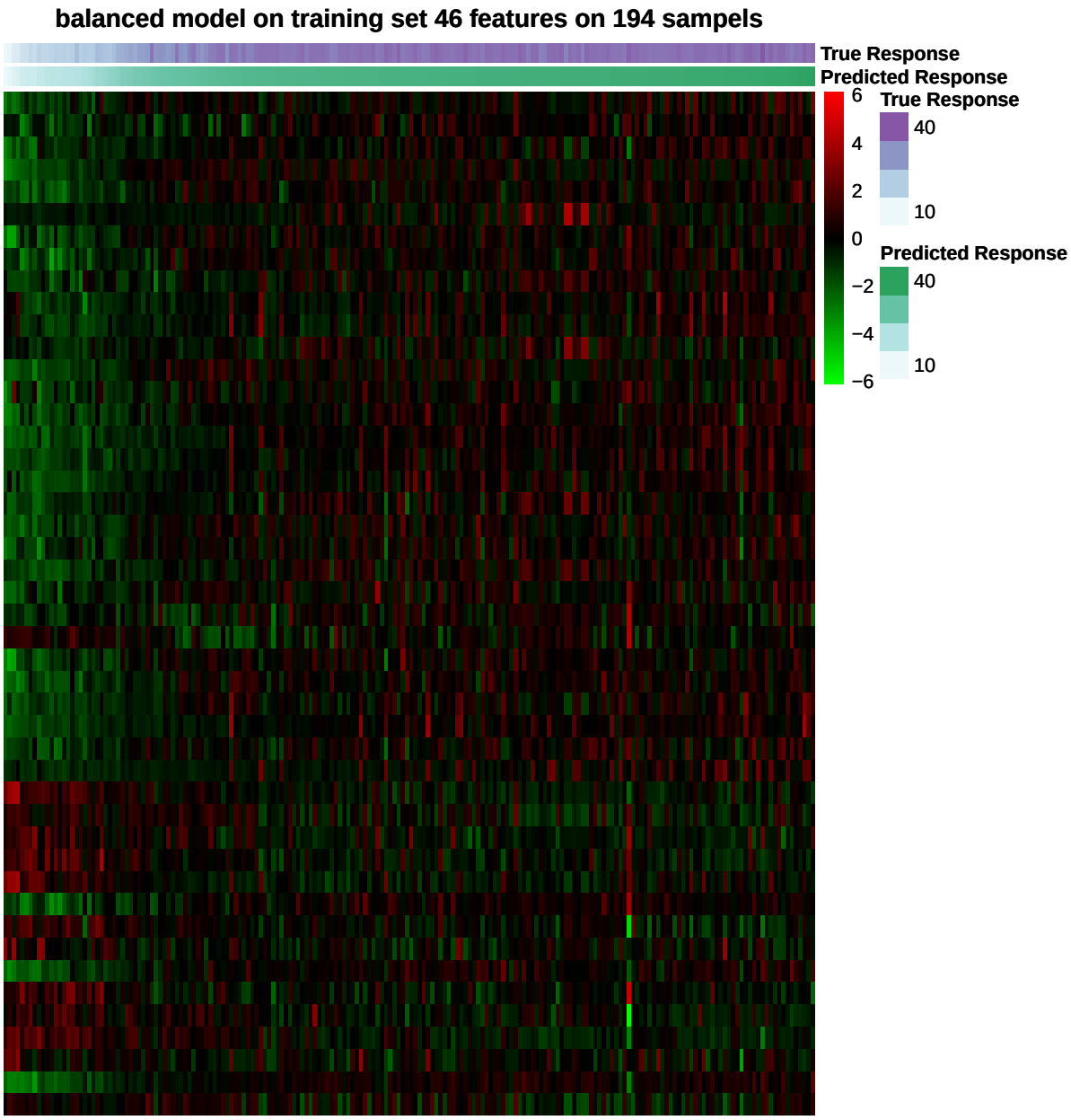
```
features <- extractprobes(res = balanceres)
```

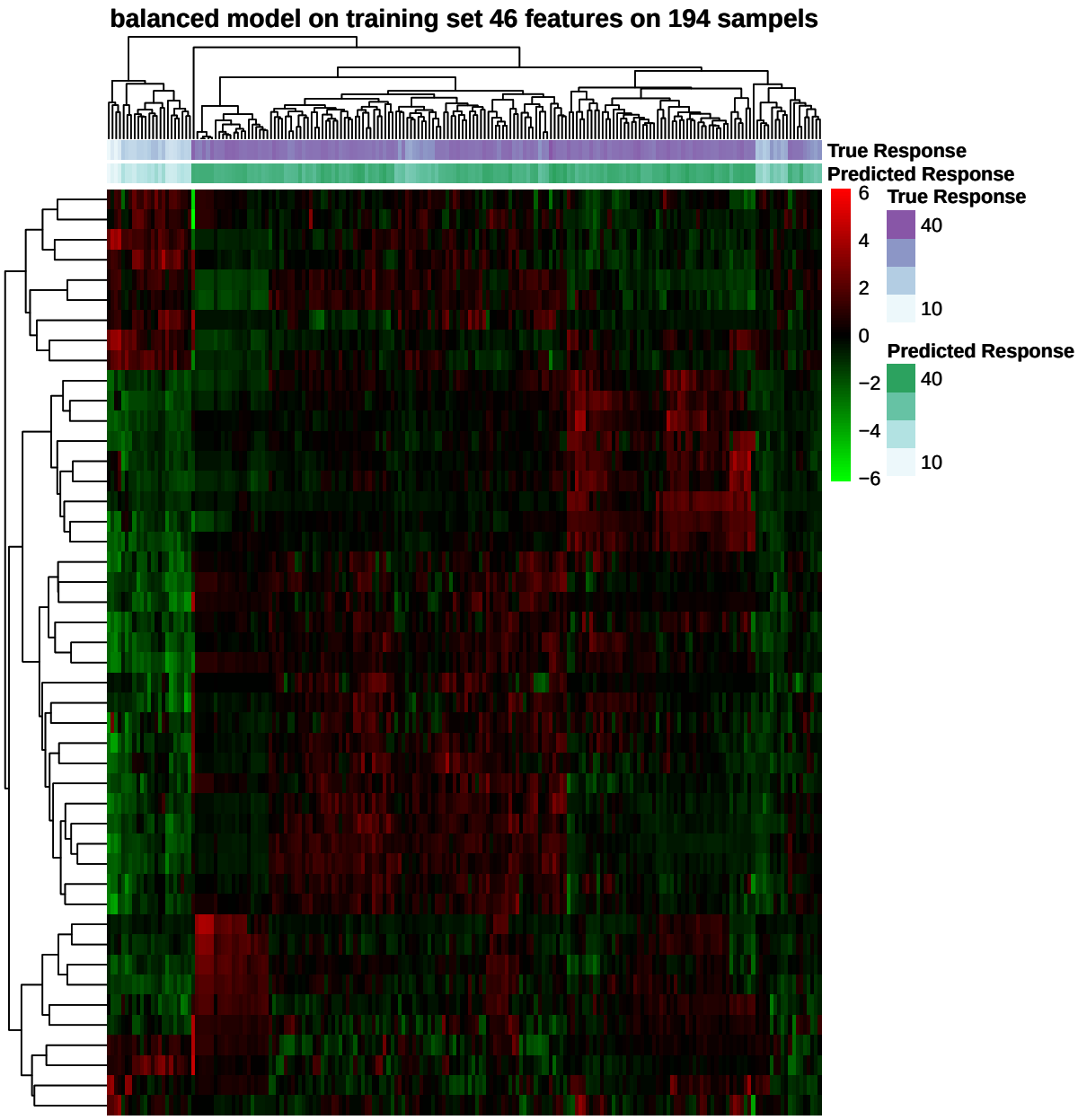
The result `features` contains 46 rows representing the totally 46 probes selected by the balanced model, as well as 2 columns, with one named "Probe" indicating the probe IDs, and the other named "Coeffsum" that are the sum of the feature coefficients from all the base learners, and the 46 features are ordered by this column with values from large to small.

Then, the function `heatmapplot` can be used to show these features.

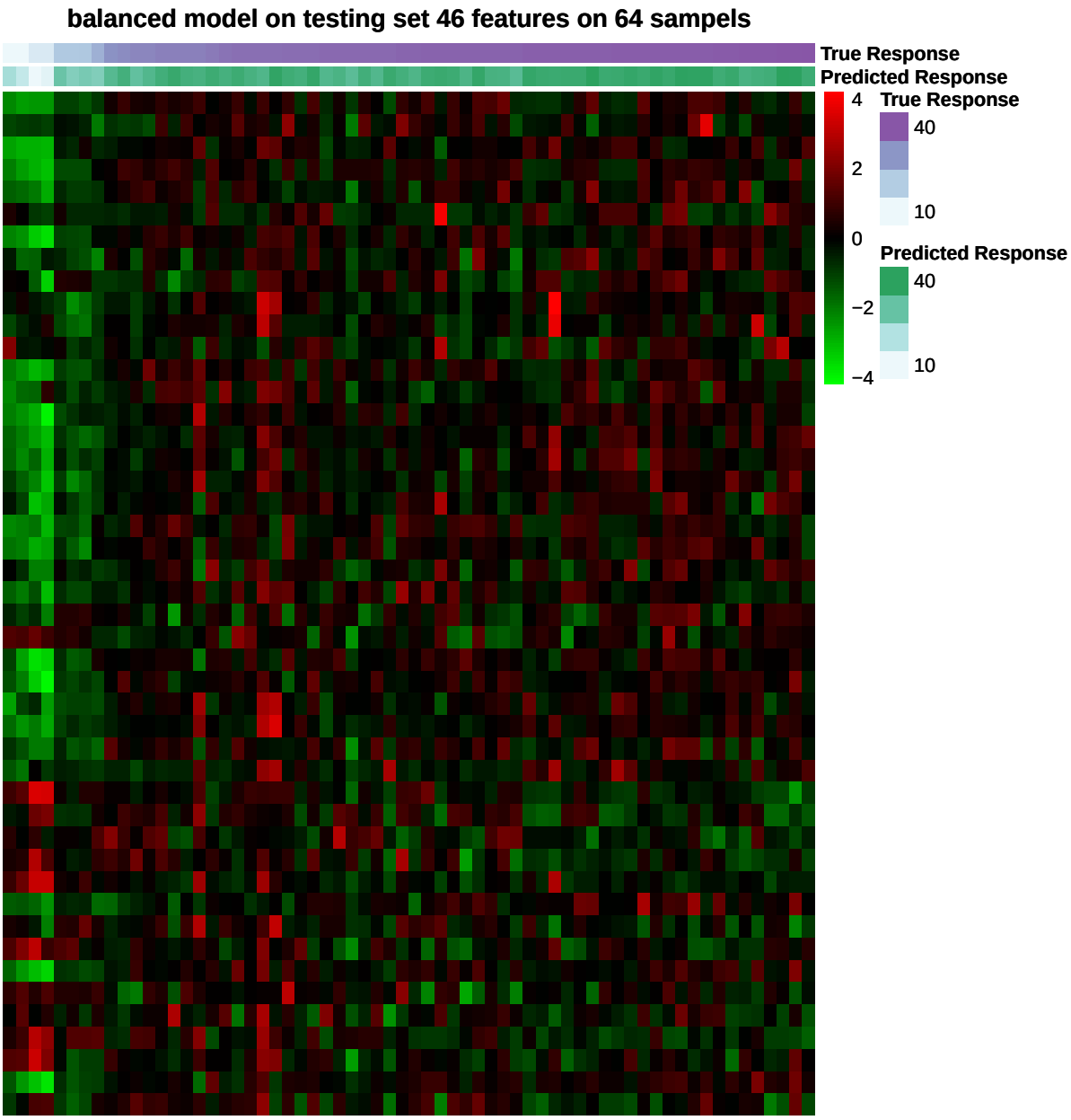
```
heatmapplot(orivar = ctrlbetas,
            comptab = balanceres$traincomp,
            featurenames = features$Probe,
            title = 'balanced model on training set')
```

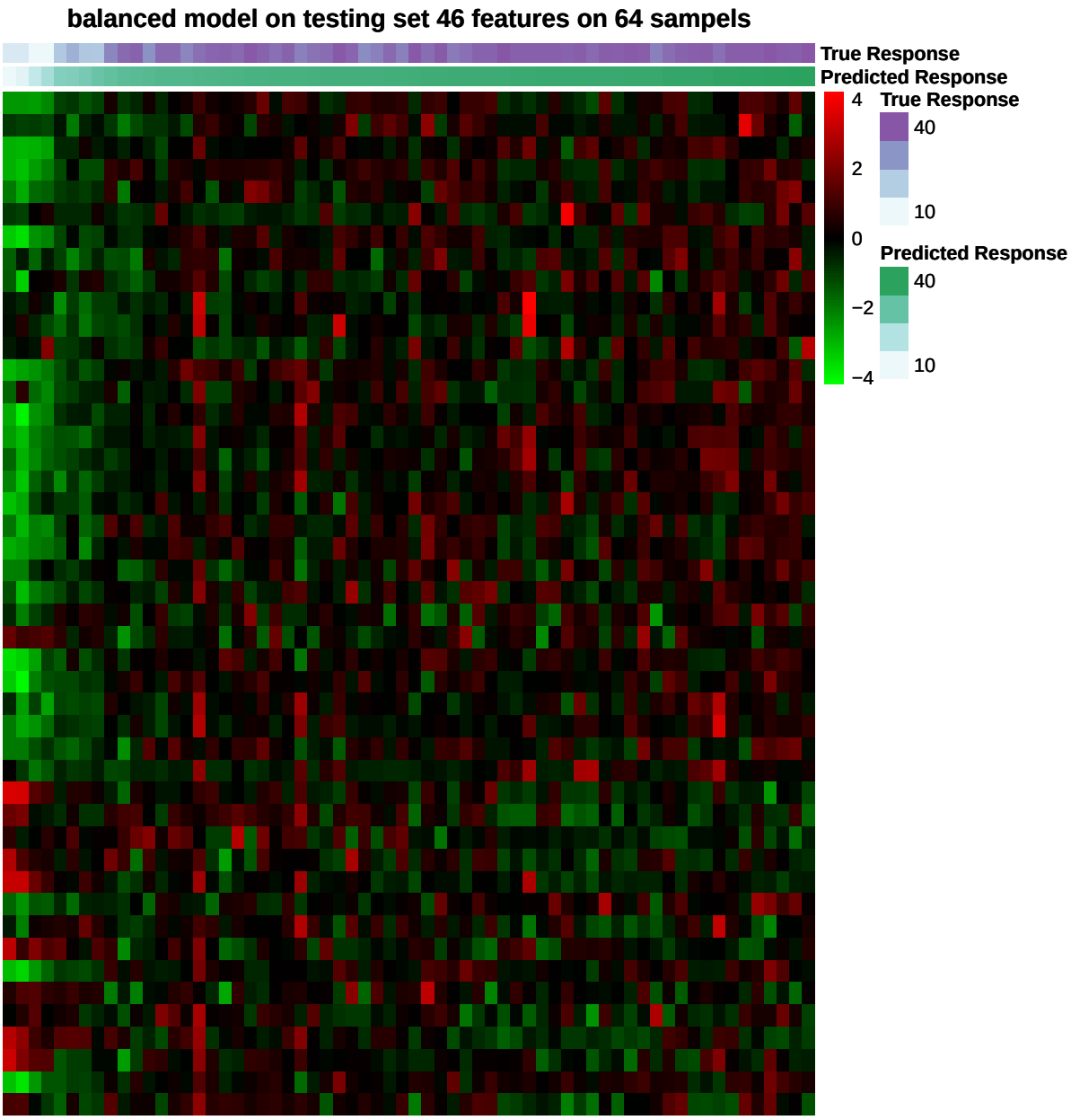


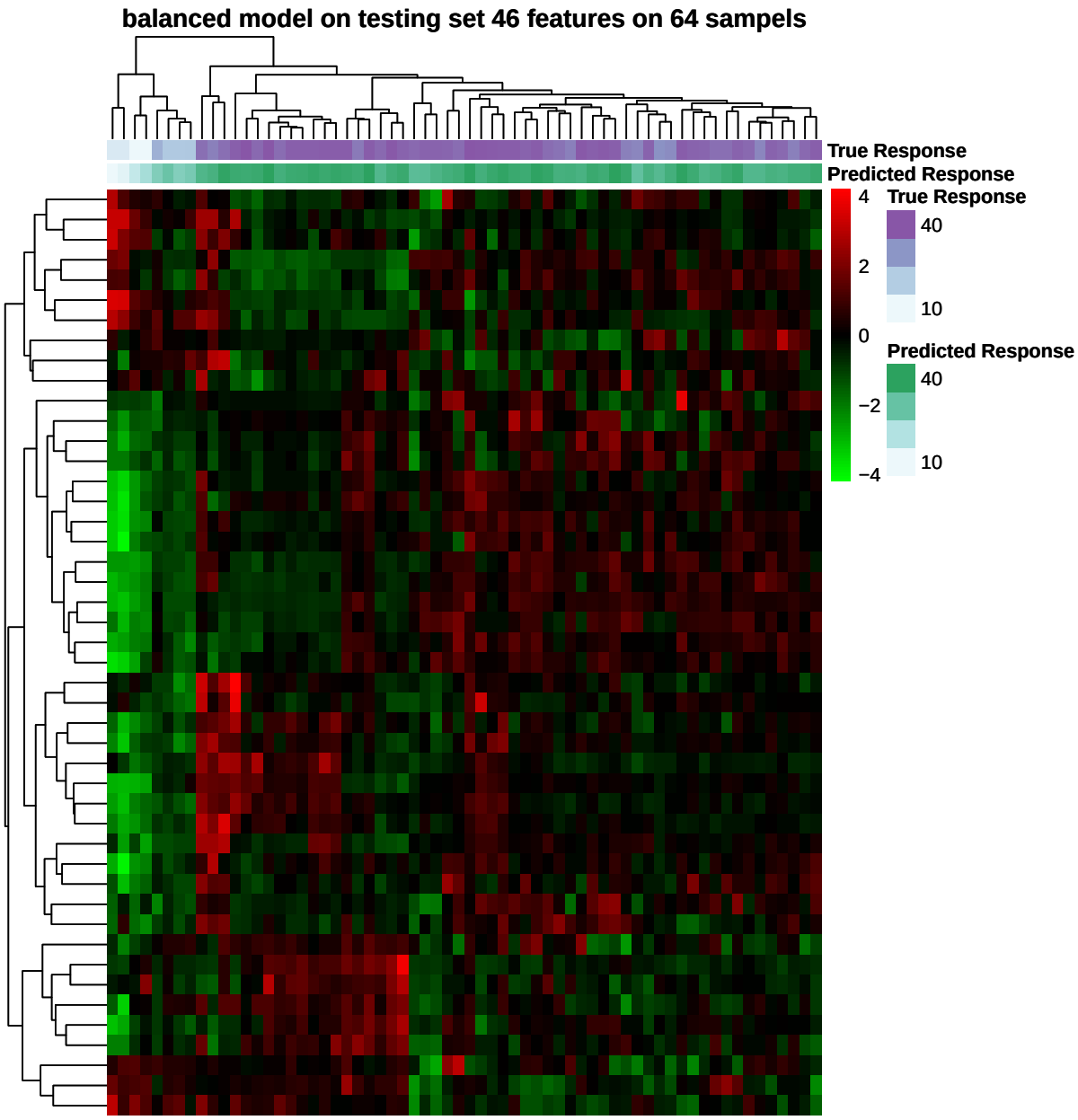

balanced model on training set 46 features on 194 sampels
True Response
Predicted Response
6
4
2
0
−2
−4
−6
True Response
40
10
Predicted Response
40
10




```
heatmapplot(orivar = ctrlbetas,
            comptab = balanceres$testcomp,
            featurenames = features$Probe,
            title = 'balanced model on testing set')
```


balanced model on testing set 46 features on 64 sampels
True Response
Predicted Response
4
2
0
−2
−4
True Response
40
10
Predicted Response
40
10


balanced model on testing set 46 features on 64 sampels
True Response
Predicted Response
4
2
0
−2
−4
True Response
40
10
Predicted Response
40
10



Each row of a heatmap is a probe, each column is a sample, and each entry is the beta value of the probe scaled on the row direction. Every time `heatmapplot` gives three heatmaps, for the first two, their rows are ordered according to the feature orders in the function parameter `featurenames`, which is actually decided by the "Coeffsum" column in the data.frame `features` above, meaning the probes are ordered by their coefficient sums from high to low. For the columns in the first heatmap, they are ordered according to the true sample ages, while the ones in the second heatmap are ordered by the predicted sample ages. For the third heatmap, both its rows and columns are ordered according to hierarchical clustering.

It should be clarified that for most plots demonstrated above, actually the corresponding functions such as `scatterplot`, `residualplot`, and `heatmapplot`, have been integrated into the functions `singlebalance` and `singleselection`, so as long as the parameter `plotting` is set as TRUE, the above plots can be generated automatically with no need to make them specially.

For the probes shown in the heatmaps, to have a further understanding in their biological function, `probeannotation` can be used. It can return the genomic locations of the probes, as well as their relevant

genes.

```
probeanno <- probeannotation(finalprobes = features$Probe)
```

```
unique(probeanno$Relation_to_Island)
#> [1] "OpenSea" "Island"  "N_Shore" "S_Shore" "N_Shelf"
```

```
unique(probeanno$UCSC_RefGene_Name)
#>  [1] "C14orf115"    "ST6GALNAC1"   "ASAM"         "APOBEC2"      "VTN"
#>  [6] "CDKN2BAS"     "CDKN2A"       "INHBE"        "TOP1MT"       "DIRAS3"
#> [11] "TMEM119"      "GPD1"         "PAX6"         "LOC100128788" "SRRM2"
#> [16] "C2orf66"      "NUP62"        "ATF5"         "IL4I1"        "DLX5"
#> [21] "TULP2"        "PLCD1"        "SERINC4"      "C15orf63"     "SNORD24"
#> [26] "SNORD36B"     "MED22"        "RPL7A"        "SNORD36A"     "MPST"
#> [31] "TST"          "CLDN6"        "SGK2"         ""             "MAK16"
#> [36] "RTEL1"        "ITGAD"        "AQP2"         "SLC13A2"      "ACTN3"
#> [41] "MFNG"         "HPS1"         "OR3A3"        "GPR162"       "LEPREL2"
#> [46] "GATA5"        "GK2"          "TNFRSF18"     "IL27RA"       "TMEM129"
#> [51] "TACC3"        "GJB5"         "BANK1"        "TMEM184A"     "CD248"
#> [56] "COL16A1"      "MALL"
```

From the result returned, it can be seen that the probes locate in various methylation island regions, including Opensea, Island, N_Shore, S_Shore and N_Shelf, and they have totally 57 relevant genes. This gene number is greater than the probe number of 46 because of some probes related to multiple genes, such as the probe "cg01605984", which has 5 relevant genes including 3 snoRNA genes (SNORD24, SNORD36A and SNORD36B) and 2 protein coding genes (MED22 and RPL7A), indicating the close association between this probe and the basic protein translation process.

The 57 genes can be transferred to some function enrichment tools such as *EnrichR (https://maayanlab.cloud/Enrichr/)* to get their enriched biological functions *[6, 7]*.

Feature conversion

The package can also transform methylation probe beta values to gene beta values and DMR beta values, so that in addition to probes, the candidate features can also be genes and DMRs.

To convert probes to genes, the function `probetogene` can be used. Its parameter `betadat` needs a matrix recording the beta values of methylation probes for samples, with each column representing one sample and each row representing one probe. The previous matrix `ctrlbetas` uses samples as rows and probes as columns, so it should be transposed first and then transferred to this parameter.

For a specific gene, this function attributes probes in defined gene regions to it and then averages these probe beta values to get the gene beta value, and the gene regions are set by the parameter `group450k850k`. Its default value is a vector `c("TSS200", "TSS1500", "1stExon")`, which means probes within these three regions will be used to calculate the gene beta value.

Another parameter is `includemultimatch` and it is used to deal with the probes that can be attributed to more than one gene. If it is set as TRUE, these probes will be involved into the beta value calculation for all their related genes. Otherwise, they will be discarded to make all the genes are averaged only from their uniquely related probes.

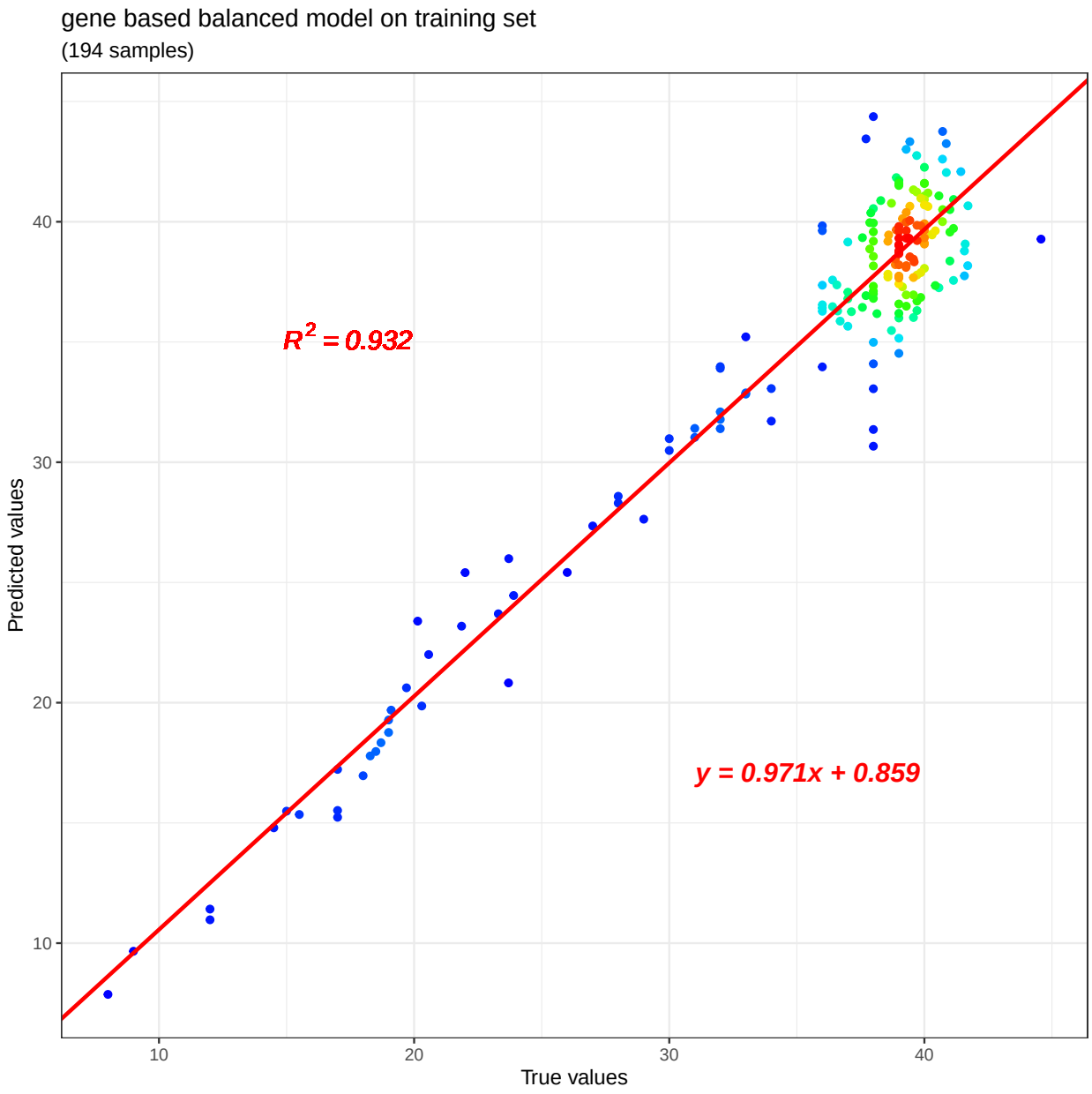
```
ctrlgenebetas <- probetogene(betadat = t(ctrlbetas),
                             group450k850k = c('TSS200', 'TSS1500', '1stExon'),
                             includemultimatch = FALSE)
```

The returned matrix `ctrlgenebetas` contains 9633 rows (genes) and 258 columns (samples), so the original 18626 probe features have been converted to these 9633 gene features successfully. Then, transpose and transfer this matrix to `singlebalance` to get a balanced model using genes as features.

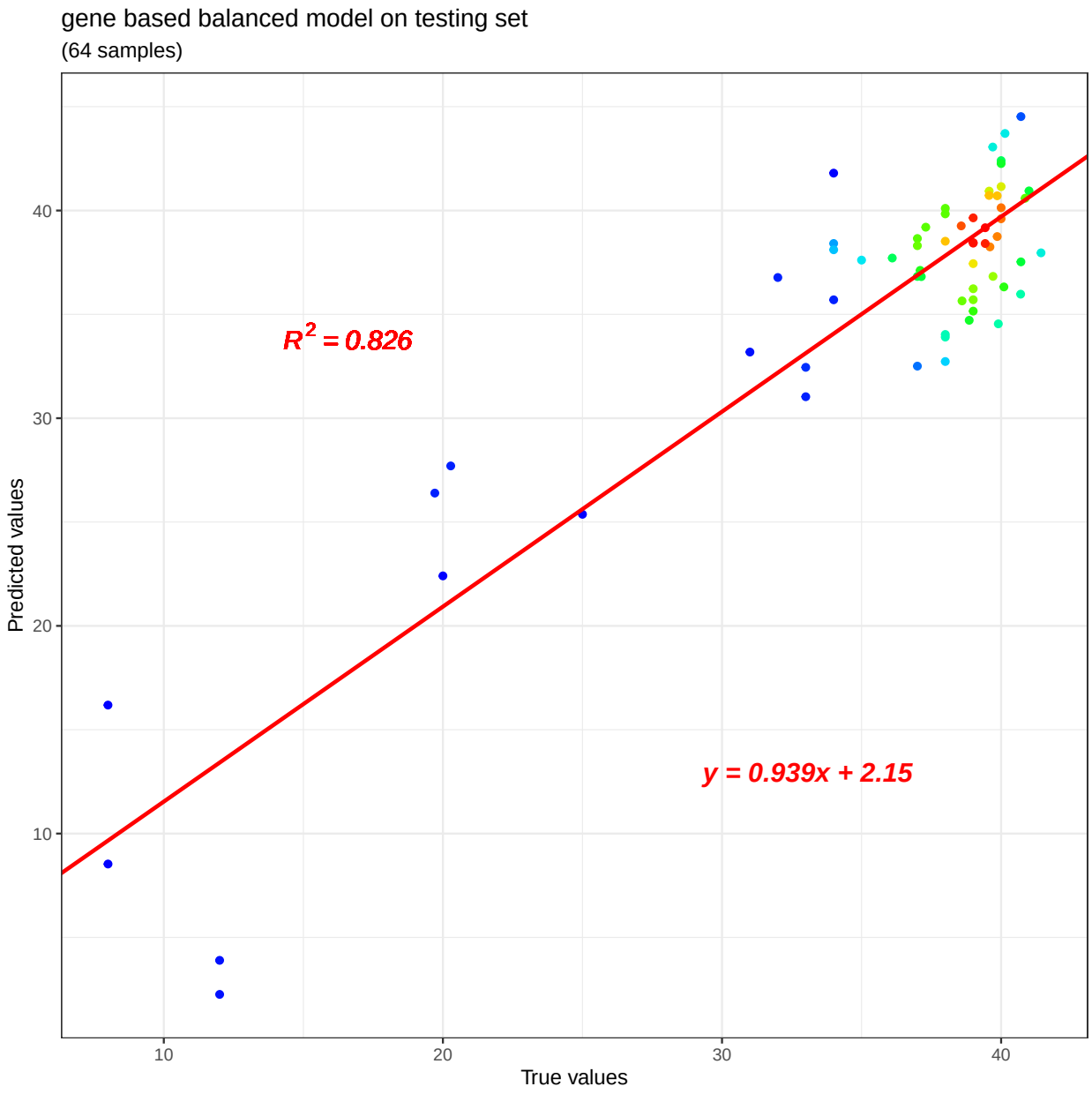
```
balancegeneres <- singlebalance(orivar = t(ctrlgenebetas),
                                oripd = ctrlpd,
                                seednum = 2022,
                                fold = 0.75,
                                balancing = TRUE,
                                binwith = 1,
                                samplenum = 10,
                                alphas = 0.5,
                                errortype = '1ses',
                                prescreen = FALSE,
                                cores = 1,
                                plotting = FALSE)
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.41
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.328
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.34
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.288
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.513
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.649
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.44
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.491
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.329
#> Choose the elastic net mixing parameter (alpha) as 0.5
#> Choose the regularization constant (lambda) as 0.367
```

The various plots for this gene based model can be generated as before, such as the scatter plots.

```
scatterplot(comptab = balancegeneres$traincomp,
            title = 'gene based balanced model on training set',
            colorful = TRUE)
```




```
scatterplot(comptab = balancegeneres$testcomp,
            title = 'gene based balanced model on testing set',
            colorful = TRUE)
```



For the gene features selected, also use the function `extractprobes` to get them from the model.

```
genefeatures <- extractprobes(res = balancegeneres)
```

From the result returned, it can be seen that this model totally selects 44 gene features, including MEN1::4221, RBM18::92400, etc. If both the symbol and the ENTREZ ID of a gene can be found, this gene will be expressed as "symbol::ENTREZ", such as "MEN1::4221" and "RBM18::92400". Although these are actually IDs for genes, they are still recorded in the column named "Probe".

To get their biological function, `geneannotation` can be used, either the gene symbols or the ENTREZ IDs can be transferred to it.

```
genefeaturesymbols <- gsub(pattern = ':::*$',
                           replacement = '',
```

```
                            x = genefeatures$Probe)
genefeaturesymbols <- unique(genefeaturesymbols)

geneanno <- geneannotation(genesymbols = genefeaturesymbols)
```

The result `geneanno` is a data.frame including various gene location and function information. Each gene accounts for one row and among the columns, one named "annotation" contains detailed gene function description. For example, for the gene MEN1, its function recorded here is "FUNCTION: Essential component of a MLL/SET1 histone methyltransferase (HMT) complex, a complex that specifically methylates 'Lys-4' of histone H3 (H3K4). Functions as a transcriptional regulator. Binds to the TERT promoter and represses telomerase expression. Plays a role in TGFB1-mediated inhibition of cell-proliferation, possibly regulating SMAD3 transcriptional activity. Represses JUND-mediated transcriptional activation on AP1 sites, as well as that mediated by NFKB subunit RELA. Positively regulates HOXC8 and HOXC6 gene expression. May be involved in normal hematopoiesis through the activation of HOXA9 expression (By similarity). May be involved in DNA repair. {ECO:0000250|UniProtKB:O88559, ECO:0000269|PubMed:11274402, ECO:0000269|PubMed:11526476, ECO:0000269|PubMed:12837246, ECO:0000269|PubMed:12874027, ECO:0000269|PubMed:14992727, ECO:0000269|PubMed:22327296}".

References

1. Horvath S. DNA methylation age of human tissues and cell types, Genome Biology 2013;14:3156.
2. Hannum G, Guinney J, Zhao L et al. Genome-wide methylation profiles reveal quantitative views of human aging rates, Mol Cell 2013;49:359-367.
3. Bocklandt S, Lin W, Sehl ME et al. Epigenetic Predictor of Age, PLOS ONE 2011;6:e14821.
4. Weidner CI, Lin Q, Koch CM et al. Aging of blood can be tracked by DNA methylation changes at just three CpG sites, Genome Biology 2014;15:R24.
5. Mayne BT, Leemaqz SY, Smith AK et al. Accelerated placental aging in early onset preeclampsia pregnancies identified by DNA methylation, Epigenomics 2016;9:279-289.
6. Chen EY, Tan CM, Kou Y et al. Enrichr: interactive and collaborative HTML5 gene list enrichment analysis tool, BMC Bioinformatics 2013;14:128.
7. Kuleshov MV, Jones MR, Rouillard AD et al. Enrichr: a comprehensive gene set enrichment analysis web server 2016 update, Nucleic Acids Res 2016;44:W90-97.