SML PUBH 6886 Final Project

Mahdi Baghbanzadeh

2023-05-03

Introduction

Pregnancy is a critical period for both the mother and child, involving significant physiological changes and metabolic adaptations throughout each week(Alkema et al. 2016; Say et al. 2014). Any deviations from the norm during this time can harm the pregnancy at various stages. For instance, about 20% of pregnancies end in miscarriage before 20 weeks, while 10% result in preterm birth before 37 weeks(Blencowe et al. 2013; Wang et al. 2003). Preterm birth is the primary cause of neonatal morbidity and mortality worldwide(Blencowe et al. 2013). The global incidence of maternal and perinatal deaths associated with pregnancy and childbirth stands at 300,000 and 7 million, respectively, out of 200 million pregnancies annually (Aboyans, Death Collaborators, et al. 2015; Sedgh, Singh, and Hussain 2014). However, by gaining a better understanding of pregnancy regulation, slight improvements in obstetric healthcare can have a positive impact on the well-being of women and children. The placenta connects the maternal and fetal circulatory systems, transporting various bioactive molecules and biomarkers such as steroid hormones, micronutrients, and circulating nucleic acids, whose levels vary throughout gestation (Liang et al. 2020). Studies have shown that maternal blood markers, specifically cell-free RNA, can be used to estimate gestational age. Still, the accuracy is currently not ideal and sequencing is time consuming and expensive (Ngo et al. 2018). Researchers are exploring alternative methods to estimate gestational age to improve accuracy and affordability, such as analyzing blood metabolites. The purpose of this study (Liang et al. 2020) is to methodically analyze blood metabolites during pregnancy by using weekly collected maternal blood samples. There are a total of 784 longitudinal samples from 30 subjects throughout their pregnancy and the postpartum period.

Methods

Study Design and Data

The data that I am working on in this study can be accessed freely through this link. In this study (Liang et al. 2020), there are a total of 30 women with weekly blood samples were assigned to a discovery (N = 21) and a validation (test set 1, N = 9) cohort. The final data set has 784 samples with information on 264 metabolites and the associated gestational age. The samples are also labeled with a validation/discovery set. Each sample has the actual gestational age at the time of birth and the type of labor being natural or advanced. A description of the variables in the data set is given in Table 1.

Table 1: Description of the dataset.

Variable.type	Variable.name	Description	Values
Primary response	Gestational age (GA)/weeks	It is a continuous variable showing the gestational age at sampling time	Values range between 4.5 to 46.2
Predictors	There are 264 predictors in this dataset	The value in each column shows the Log2(intensity) of the metabolite	log2(intensity) of metabolites

Predictive Models

The regression question at hand is to model gestational age based on metabolite intensity values. The data set contains 784 samples and 264 features. To accomplish this task, I have selected several machine learning (ML) algorithms to compare their performance in predicting gestational age. The algorithms are: 1) Linear regression, 2) Decision tree, 3) Decision tree with bagging, 4) Random Forest, 5) Boosting tree, 6) LASSO, 7) Ridge, 8) Forward selection, 9) Backward selection, 10) Partial least squares regression, 11) Principal component regression. Linear regression is a widely used algorithm that is straightforward and easy to interpret. It assumes a linear relationship between the dependent variable and the independent variables, which may not be the case in all situations. Decision trees are versatile and easy to interpret but can be prone to overfitting. Decision tree with bagging and random forest are ensemble methods that combine multiple decision trees to improve accuracy and reduce overfitting. Boosting tree is another ensemble method that focuses on improving the performance of weak learners. Lasso and ridge regression are regularized linear regression methods that can handle a large number of features and prevent overfitting. Forward and backward selection are feature selection methods that choose a subset of features to use in the model. Partial least squares regression and principal component regression are dimensionality reduction techniques that can help reduce the number of features in the model. In this case, the large number of features in the data set could pose a challenge for some of the algorithms. Linear regression, decision tree, and decision tree with bagging may struggle with handling the high dimensionality of the data set. Random forest, boosting tree, lasso, ridge, forward selection, backward selection, partial least squares regression, and principal component regression are better suited to handle this type of data. Random forest and boosting tree are particularly well-suited for handling high-dimensional data but may require more computational resources than some of the other algorithms. Lasso and ridge regression can handle a large number of features and prevent overfitting but may not perform as well if there are many irrelevant features in the data set. Forward and backward selection can be computationally expensive if the data set has a large number of features. Partial least squares regression and principal component regression are useful for reducing the number of features in the data set but may not perform as well if there are nonlinear relationships between the features and the dependent variable. The choice of algorithm depends on the specific characteristics of the data set and the goals of the analysis. In this case, I am going to try the algorithms mentioned above and compare their performance using cross-validation in order to select the best one for the task at hand.

Subject-based Cross-validation with Fixed Indexes for Model Comparison

In this study, cross-validation was performed to assess the performance of the predictive model. To prevent person-specific information cross-over between the training folds and the test fold, samples were distributed into folds by subject instead of by individual samples. Specifically, each fold contained all samples from a given subject, which allowed for a more accurate assessment of the model's generalization performance. To ensure a fair comparison between different models, a fixed set of indexes was used for cross-validation across all models.

Evaluating the Predictive Models

For this study, some common performance metrics that can be used are Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R-squared (R^2), and Mean Absolute Percentage Error (MAPE). In this study, as the focus is on minimizing the absolute difference between the predicted and true gestational age, I am going to consider the 10-fold cross validation MAE and RMSE to compare the models.

To implement all of these, I am using `R` (R Core Team 2022), `R-studio` (Posit team 2022) and the following `R` packages: `readxl` (Wickham and Bryan 2022), `caret` (Kuhn 2022), `glmnet` (Friedman, Hastie, and Tibshirani 2010), `rpart` (Therneau and Atkinson 2022), `randomForest` (Liaw and Wiener 2002), and `gbm` (Greenwell et al. 2022). For writing the report, I am using the `knitr` package (Xie 2022).

Implementation and Results

```
# loading required libraries
library(tidyverse)
library(cowplot)
library(readxl)
library(caret)
library(glmnet)
library(rpart)
library(randomForest)
library(gbm)
```

Loading data

```
# loading data
# link to download the data:
# https://www.cell.com/cms/10.1016/j.cell.2020.05.002/attachment/444d8f28-3bd3-46b8-b2e2-3a038a3d61b4/m
df <- read_excel('data/data.xlsx') %>%
  data.frame()
colnames(df) = make.names(colnames(df)) # cleaning names
df <- df %>%
  filter(Gestational.age..GA..weeks < Birth.GA.weeks) # only samples before birth
# dividing df into train and test
train <- df %>%
  filter(Gestational.age..GA..weeks < Birth.GA.weeks, # only samples before birth
         startsWith(Cohort, 'Discovery')) # only discovery group
test <- df %>%
  filter(Gestational.age..GA..weeks < Birth.GA.weeks, # only samples before birth
         startsWith(Cohort, 'Valida')) # only validation group
# selecting the metabolite columns
metabolite_cols = colnames(df)[6:ncol(df)]
```

Group Cross Validation

First, I use the following code to create 10 stratified folds for cross-validation by subject ID. The process involves randomly sampling 80% of the unique subject IDs as a training set for each fold, while maintaining balance across the subject IDs. The code starts by setting the seed for reproducibility and obtaining a vector of unique subject IDs. It then creates two empty lists, `folds` and `folds_info`, to store the indices and information about each fold. The `check` vector is initialized to keep track of the subject IDs included in the training sets across all folds. The `prob_vec` vector is initialized with equal probabilities for all subject IDs. In the loop, the subject IDs are randomly sampled based on the probabilities in `prob_vec` with the probabilities for each subject ID adjusted to ensure balance across folds. The indices corresponding to the sampled subject IDs are added to the `folds` list, and the sampled subject IDs are added to the `folds_info` list. The resulting `folds` object is a list of indices for each fold, which can be used for cross-validation.

```
# set seed for reproducibility
set.seed(112)
# obtain unique subject IDs
groups = unique(train$Subject.ID)
```

```r
# initialize lists to store fold information
folds = list()
folds_info = list()
# initialize variables for balance checking
check = c()
prob_vec = rep(1, length(groups))
# number of folds
k = 10
# loop through 10 folds
for(i in 1:k){
  # randomly sample subject IDs for training set
  sample_group = sample(groups, size = length(groups)*0.85, prob = prob_vec)
  # add sampled subject IDs to check vector
  check  = c(check , sample_group)
  # adjust probabilities for subject IDs to ensure balance
  tb = table(check)
  for (j in 1:length(groups)){
    if(groups[j] %in% names(tb)){
      prob_vec[j] = max(0.1, 1 - 0.2 * tb[groups[j]])
    }
  }
  # create fold name based on fold number
  if(i < 10 ){
    name = paste0('Fold0', i)
  }else{
    name = paste0('Fold', i)
  }
  # store fold information in lists
  folds[[name]] = which(train$Subject.ID %in% sample_group)
  folds_info[[name]] = sample_group
}
```

In order to see which subject IDs are included in each fold I check the IDs in each fold and the results are shown in Fig. 1. For example, in `Fold01`, Subject IDs 8, 15, 17, 19, 20 are used as validation.

```r
# obtain unique subject IDs
sub_ids = unique(train$Subject.ID)
# initialize matrix to store holdout information
mat = matrix(NA, ncol = length(sub_ids), nrow = k)
# assign names to rows and columns of matrix
dimnames(mat) = list(names(folds), sub_ids)
# loop through fold names and fill in matrix with holdout information
for(name in names(folds)){
  # determine which subject IDs are not in the current fold
  mat[name, ] = !sub_ids %in% folds_info[[name]]
}
# convert matrix to data frame and reshape for easier analysis
mat = data.frame(mat) %>%
  rownames_to_column(var = 'Fold') %>%
  gather(subject, holdout, -Fold)
```

```r
# Create the plot
design_plot = ggplot(mat, aes(x = Fold, y = subject, fill = holdout)) +
  geom_tile() +
  scale_fill_manual(values = c("#4e79a8", "#f28e2b")) +
  xlab("Folds") +
  ylab("Subject ID") +
  ggtitle("Group 10-fold Cross-validation") +
  theme_classic() +
  theme(plot.title = element_text(hjust = 0.5),
        axis.text.x = element_text(size = 8),
        axis.text.y = element_text(size = 8))
ggsave(filename = 'design.pdf',plot = design_plot,
       width = 7.2, height = 4, dpi = 300)
design_plot
```

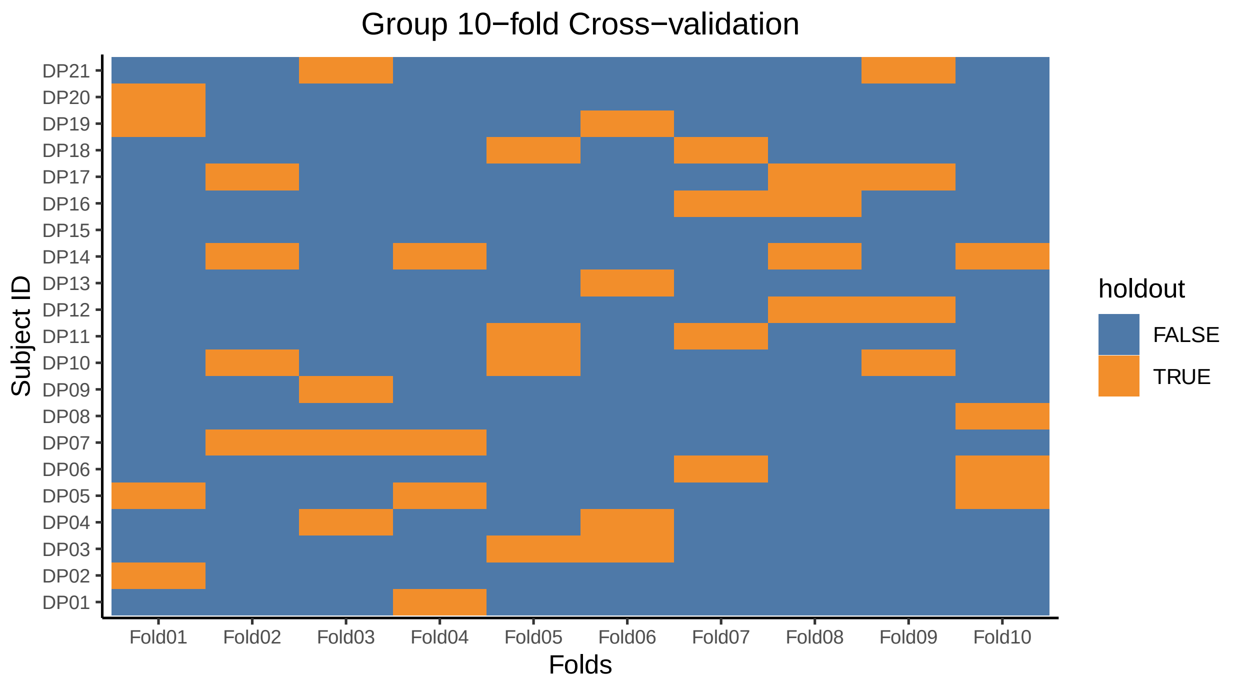

Figure 1: Group 10-fold cross validation design.

As I am going to fit multiple ML models, and I want to compare their performances, I am writing the following function, to keep track of the performance of the models:

```
update_report <- function(caret_obj, report_df = NULL, model_name) {
  # check if report_df is null
  if (is.null(report_df)) {
    # if it is, create a new data frame with the best RMSE, MAE, and Rsquared values
    report_df = data.frame(caret_obj$results[which.min(caret_obj$results$RMSE),
                                              c("RMSE", "MAE", "Rsquared")])
    # set the row name to the model name
    rownames(report_df) = model_name
  } else {
    # if it's not null, add a new row to the data frame
    # with the best RMSE, MAE, and Rsquared values
    n = nrow(report_df)
    report_df[n + 1,] = caret_obj$results[which.min(caret_obj$results$RMSE),
                                          c("RMSE", "MAE", "Rsquared")]
    # set the row name to the model name
    rownames(report_df)[n + 1] = model_name
  }
  # return the updated report_df
  return(report_df)
}
```

Simple Linear Regression (SLR)

Here, I am fitting a simple linear regression model to predict Gestational age (GA) in weeks using the metabolite columns from my dataset. To do this, I set the seed to ensure reproducibility and used the `train()` function from the caret package with method = "lm" argument to specify a linear regression model. I also used preProcess = c("center", "scale") argument to center and scale my data, which is a common preprocessing step in linear regression. Additionally, I specified trControl = trainControl(method = "cv", number = 10, index = folds) to perform 10-fold cross-validation and use the indexes generated by the folds object.

```
# set seed (for reproducibility) and obtain 10-fold CV error for the linear model
set.seed(1234)
lm_10cv <- train(
  x = train[, metabolite_cols],
  y = train$Gestational.age..GA..weeks,
  method = "lm",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", number = 10, index = folds)
)
```

Based on the results, for the simple linear regression, the 10-fold cross validation `RMSE` is 4.5646 and R^2 is 0.7864.

```
# updating the report_df
report_df = update_report(caret_obj = lm_10cv, report_df = NULL,
                         model_name = 'Simple Linear Regression')
```

Forward Selection

Forward selection regression is a feature selection technique that involves starting with no features in the model and adding the best-performing features one at a time until a stopping criterion is met. One of the main benefits of using forward selection regression is that it can help simplify the model by selecting only the most important features, which can improve the model's *interpretability* and reduce *overfitting*. Additionally, forward selection can be computationally efficient, as it only involves fitting a subset of possible models, rather than all possible combinations of features. Another advantage is that it can be used with any regression model, including linear and nonlinear models. Forward selection can help improve the accuracy and generalizability of a regression model by selecting the most important features and avoiding overfitting. Here, I am fitting a forward selection regression using the `leapForward` method with 10-fold cross-validation. The predictor variables are the metabolite columns in the training dataset and the response variable is the Gestational age (GA) in weeks. I am also centering and scaling the data as a pre-processing step. The `tuneGrid` parameter specifies the maximum number of variables to include in the model, and it is set to the length of the metabolite columns. The results from this model will help me identify which metabolites are most strongly associated with GA.

```r
# forward selection, running the 10-fold CV
set.seed(1234)
forward_train <- train(
  x = train[, metabolite_cols],
  y = train$Gestational.age..GA..weeks,
  method = "leapForward",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", number = 10, index = folds),
  tuneGrid = data.frame(nvmax = 1:length(metabolite_cols))
)
```

```r
# Get the best tune from the forward selection model
bt = forward_train$bestTune
# Get the results for the best tune of the forward selection model
best_result = forward_train$results[which(forward_train$results$nvmax == bt$nvmax), ]

# Update the model comparison dataframe with the results of
# the forward selection model
report_df = update_report(caret_obj = forward_train,
                          report_df = report_df,
                          model_name = 'Forward Selection')
```

```r
# Get the variable table from the final model of forward selection
forward_var_table = summary(forward_train$finalModel)$outmat
# Get the variables in the model with the minimum RMSE from the variable table
min_rmse_model_vars = forward_var_table[bt$nvmax[1],]
# Get the names of the variables in the model with
# the minimum RMSE from the forward selection
min_rmse_model_vars_forward = names(which(min_rmse_model_vars == "*"))
```

Based on the results, for the best fitted forward selection model, the 10-fold cross validation `RMSE` is 3.244 and R^2 is 0.8716. The number of variables selected are 6 and the selected metabolites are X.R..2.Hydroxycaprylic.acid, Androsterone.sulfate, Estriol.16.Glucuronide, Indole.3.carboxaldehyde, THDOC, Progesterone.

Backward Selection

Backward selection regression is a feature selection method that starts with a model containing all the available features and then iteratively removes the least significant feature(s) until the stopping criterion is reached. The benefit of using backward selection regression is that it can save time and resources by eliminating irrelevant or redundant features, leading to a simpler and more interpretable model. Additionally, it can improve the model's predictive power by reducing the effects of *overfitting* and *multicollinearity*. Moreover, it can be used to determine the most important features in the model, which can provide insights into the underlying relationships between the predictor variables and the response variable. Backward selection regression is a useful technique for building more efficient and accurate predictive models.
Here, I am fitting a backward selection regression using the `leapBackward` method with 10-fold cross-validation. The predictor variables are the metabolite columns in the training dataset and the response variable is the Gestational age (GA) in weeks. I am also centering and scaling the data as a pre-processing step. The `tuneGrid` parameter specifies the maximum number of variables to include in the model, and it is set to the length of the metabolite columns. The results from this model will help me identify which metabolites are most strongly associated with GA.

```r
# backward selection, running the 10-fold CV
set.seed(1234)
backward_train <- train(
  x = train[, metabolite_cols],
  y = train$Gestational.age..GA..weeks,
  method = "leapBackward",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", number = 10, index = folds),
  tuneGrid = data.frame(nvmax = 1:length(metabolite_cols))
)
```

```r
# Get the best tune from the backward selection model
bt = backward_train$bestTune
# Get the results for the best tune of the backward selection model
best_result = backward_train$results[which(backward_train$results$nvmax == bt$nvmax), ]

# Update the model comparison dataframe with the results of
# the backward selection model
report_df = update_report(caret_obj = backward_train,
                          report_df = report_df,
                          model_name = 'Backward Selection')
```

```r
# Get the variable table from the final model of backward selection
backward_var_table = summary(backward_train$finalModel)$outmat
# Get the variables in the model with the minimum RMSE from the variable table
min_rmse_model_vars = backward_var_table[bt$nvmax[1],]
# Get the names of the variables in the model with
# the minimum RMSE from the backward selection
min_rmse_model_vars_backward = names(which(min_rmse_model_vars == "*"))
```

Based on the results, for the best fitted backward selection model, the 10-fold cross validation `RMSE` is 3.5928 and R^2 is 0.8552. The number of variables selected are 23 and the selected metabolites are X.R..2.Hydroxycaprylic.acid, X12.13.Dihydroxy.9Z.octadecenoic.acid, X7.Methylguanine, X7.alpha..Hydroxy.3.oxo.4.cholestenoic.ac. . . , X7alpha.24.Dihydroxy.4.cholesten.3.one, Androsterone.glucuronide, C10.0.AC..Decanoylcarnitine., C12.0.AC..Dodecanoylcarnitine., C14.1.AC..Tetradecenoylcarnitine., C14.2.AC..Tetradecadiencarnitine., Corticosterone, Erucic.acid, Estriol.16.Glucuronide, Hippuric.acid, Isobutyryl.L.carnitine, PE.P.16.0e.0.0., LPE.20.1., LysoPE.18.0.0.0., MG.20.0., MG.24.0., THDOC, Tetracosahexaenoic.acid, Theobromine.

Principal Component Regression (PCR)

Here, I am fitting a Principal Component Regression (PCR) using the `pcr` method with 10-fold cross-validation. The predictor variables are the metabolite columns in the training dataset and the response variable is the Gestational age (GA) in weeks. I am also centering and scaling the data as a pre-processing step. The `tuneGrid` parameter specifies the range of principal components to be considered during the model selection process.
The main benefit of using PCR is that it can handle multicollinearity among predictor variables. PCR reduces the dimensionality of the predictor variables by transforming them into a smaller set of uncorrelated variables, called principal components. These principal components are then used as predictors in the regression model, which can help to improve the model's predictive accuracy and reduce the risk of overfitting. Additionally, PCR can also help to identify which predictor variables are most important for predicting the outcome variable.

```
# pcr, running the 10-fold CV
pcr_train <- train(
  x = train[, metabolite_cols],
  y = train$Gestational.age..GA..weeks,
  method = "pcr",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", number = 10, index = folds),
  tuneGrid = data.frame(ncomp = 1:length(metabolite_cols))
)
```

```
# Get the best tune from the pcr model
bt = pcr_train$bestTune
# Get the results for the best tune of the pcr selection model
best_result = pcr_train$results[which(pcr_train$results$ncomp == bt$ncomp), ]

# Update the model comparison dataframe with the results of
# the pcr selection model
report_df = update_report(caret_obj = pcr_train,
                          report_df = report_df,
                          model_name = 'PCR')
```

Based on the results, for the best fitted PCR model, the 10-fold cross validation `RMSE` is 3.2368 and R^2 is 0.875. The number of components selected is 64.

Partial Least Squares Regression (PLS)

Here, I am fitting a Partial Least Squares Regression (PLS) using the `pls` method with 10-fold cross-validation. The predictor variables are the metabolite columns in the training dataset and the response variable is the Gestational age (GA) in weeks. PLS is a technique that finds linear combinations of the predictors that explain the maximum covariance between the predictors and the response variable. It is similar to PCR, but instead of using the predictors themselves, it uses the linear combinations of the predictors. The benefit of using PLS is that it can handle situations where there are many predictors and they are highly correlated, which can cause problems with other regression techniques. By using PLS, I hope to improve the accuracy of my model and identify the most important predictors for predicting gestational age. I am running a 10-fold cross-validation to assess the performance of the model.

```
# pls, running the 10-fold CV
pls_train <- train(
  x = train[, metabolite_cols],
  y = train$Gestational.age..GA..weeks,
  method = "pls",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", number = 10, index = folds),
  tuneGrid = data.frame(ncomp = 1:length(metabolite_cols))
)
```

```
# Get the best tune from the pls model
bt = pls_train$bestTune
# Get the results for the best tune of the pls selection model
best_result = pls_train$results[which(pls_train$results$ncomp == bt$ncomp), ]

# Update the model comparison dataframe with the results of
# the pls selection model
report_df = update_report(caret_obj = pls_train,
                          report_df = report_df,
                          model_name = 'PLS')
```

Based on the results, for the best fitted PLS model, the 10-fold cross validation `RMSE` is 3.2018 and R^2 is 0.8777. The number of components selected is 7.

Ridge

Here, I am fitting a ridge regression model to the data. First, I use the `glmnet()` function to create a ridge regression model with an alpha value of 0, which specifies a pure ridge regression. I set the seed to ensure reproducibility. Next, I create a `tuneGrid` object with the lambda values obtained from the `glmnet` model. I then use the `train()` function to run 10-fold cross-validation on the data and search for the optimal `lambda` value. The model is trained using `glmnet` method, and the pre-processing steps involve centering and scaling the data.
Ridge regression has several benefits. First, it can help to prevent overfitting in models that have a large number of predictors, by reducing the impact of any individual predictor that may have a large coefficient. This is achieved by adding a penalty term to the model's cost function, which shrinks the magnitude of the regression coefficients. Second, it can improve the stability and accuracy of the model's predictions, particularly in cases where there is multicollinearity among the predictor variables. Third, it can help to identify which predictors are most important in the model, by shrinking the coefficients of less important variables towards zero.

```
# ridge regression
set.seed(1234)
ridge_glmnet <- glmnet(x = as.matrix(train[, metabolite_cols]),
                       y = train$Gestational.age..GA..weeks,
                       alpha = 0.0)

# set up a ridge regression tuneGrid object
tg_ridge <- data.frame(alpha = 0, lambda = ridge_glmnet$lambda)

ridge_train <- train(
  x = train[, metabolite_cols],
  y = train$Gestational.age..GA..weeks,
  method = "glmnet",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", index = folds),
  tuneGrid = tg_ridge
)
```

```
# Get the best tune from the ridge model
bt = ridge_train$bestTune
# Get the results for the best tune of the ridge selection model
best_result = ridge_train$results[which(ridge_train$results$lambda == bt$lambda), ]

# Update the model comparison dataframe with the results of
# the ridge selection model
report_df = update_report(caret_obj = ridge_train,
                          report_df = report_df,
                          model_name = 'Ridge')
```

Based on the results, for the best fitted Ridge model, the 10-fold cross validation `RMSE` is 3.0841 and R^2 is 0.8875. The lambda for the best model is 4.2248826.

LASSO

Here, I am fitting a LASSO model to the data. First, I use the `glmnet()` function to create a ridge regression model with an alpha value of 1, which specifies a LASSO regression. I set the seed to ensure reproducibility. Next, I create a `tuneGrid` object with the lambda values obtained from the `glmnet` model. I then use the `train()` function to run 10-fold cross-validation on the data and search for the optimal `lambda` value. The model is trained using `glmnet` method, and the pre-processing steps involve centering and scaling the data. Lasso regression has several benefits. First, it can perform variable selection by shrinking some coefficients to exactly zero. This helps to identify the most important predictors and to simplify the model. Second, it can help to prevent overfitting by adding a penalty term to the regression objective function. The penalty term encourages the coefficients to be small, which can help to reduce variance and improve model performance. Finally, lasso regression can handle multicollinearity between predictors, as it can choose one variable over another that has similar information, leading to more robust models.

```r
# lasso regression
set.seed(1234)
lasso_glmnet <- glmnet(x = as.matrix(train[, metabolite_cols]),
                       y = train$Gestational.age..GA..weeks,
                       alpha = 1)

# set up a lasso regression tuneGrid object
tg_lasso <- data.frame(alpha = 1, lambda = lasso_glmnet$lambda)

lasso_train <- train(
  x = train[, metabolite_cols],
  y = train$Gestational.age..GA..weeks,
  method = "glmnet",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", index = folds),
  tuneGrid = tg_lasso
)
```

```r
# Get the best tune from the lasso model
bt = lasso_train$bestTune
# Get the results for the best tune of the lasso selection model
best_result = lasso_train$results[which(lasso_train$results$lambda == bt$lambda), ]

# Update the model comparison dataframe with the results of
# the lasso selection model
report_df = update_report(caret_obj = lasso_train,
                          report_df = report_df,
                          model_name = 'LASSO')
```

Based on the results, for the best fitted lasso model, the 10-fold cross validation `RMSE` is 3.0011 and R^2 is 0.8906. The lambda for the best model is 0.2307748.

Decision Tree

Here, I am fitting a decision tree. Decision trees are a non-parametric technique that can model complex relationships between predictors and response variables. The `rpart` function recursively partitions the data based on the predictor variables, creating a tree structure that can be used for prediction. In this case, I am using the `anova` method to build the tree for predicting GA in weeks using all the predictors in the `new_train` dataset. To tune the *hyperparameters* of the tree, I am setting up a cp (complexity parameter) grid using the `cptable` from `rpart` result, and using cross-validation to evaluate the model's performance. The benefit of using decision tree is that it is easy to interpret and visualize, and it can capture non-linear relationships between predictors and the response variable. It also allows for interactive data exploration and can handle missing values.

```r
# fit a regression tree via recursive partitioning for
# GA on all predictors using training data
new_train = train[, metabolite_cols]
new_train$Gestational.age..GA..weeks = train$Gestational.age..GA..weeks

# Set the random seed for reproducibility and fit a decision tree
set.seed(1234)
rtree_ga <- rpart(Gestational.age..GA..weeks ~ .,
                  data = new_train, method = "anova",
                  control = rpart.control(minsplit = 20,
                                          minbucket = 5,
                                          cp = 0, xval = 10))

# look at the alpha (cp) values from the fitted tree and set up values to check
tg_regtr <- data.frame(cp = rtree_ga$cptable[,1])

# using train function to run cross validation
set.seed(1234)
tree_train <- train(x = train[, metabolite_cols],
                     y = train$Gestational.age..GA..weeks,
                     method = "rpart",
                     control = rpart.control(minsplit = 20, minbucket = 5),
                     tuneGrid = tg_regtr,
                     trControl = trainControl(method = "cv", index = folds))
```

```r
# Get the best tune from the decision tree
bt = tree_train$bestTune
# Get the results for the best tune of the decision tree model
best_result = tree_train$results[which(tree_train$results$cp == bt$cp), ]

# Update the model comparison dataframe with the results of
# the decision tree selection model
report_df = update_report(caret_obj = tree_train,
                          report_df = report_df,
                          model_name = 'Decision Tree')
```

Based on the results, for the best fitted decision tree, the 10-fold cross validation `RMSE` is 3.2869 and R^2 is 0.866. The *cp* (complexity parameter) for the best model is 0.0101466.

Bagged Tree

Here, I am fitting a bagged tree. Bagging stands for Bootstrap Aggregating, which is a technique used to reduce the variance of a decision tree model. In bagging, multiple trees are constructed using bootstrap samples from the original dataset, and then the results are combined using an average or majority vote. This technique helps reduce the correlation between the individual trees and further improves the performance of the model. In my code, I am using the `rf` method in the `train` function to fit a bagged tree model with 500 trees. I also set `mtry` to be equal to the total number of the available variables.

```r
### bagging
set.seed(1234)
bagg_train <- train(x = train[, metabolite_cols],
                    y = train$Gestational.age..GA..weeks,
                    method = "rf", ntree = 500,
                    tuneGrid = data.frame(mtry = length(metabolite_cols)),
                    trControl = trainControl(method = "cv", index = folds))
```

```r
# Get the best tune from the bagged tree
bt = bagg_train$bestTune
# Get the results for the best tune of the bagged tree model
best_result = bagg_train$results[which(bagg_train$results$mtry == bt$mtry), ]
# Update the model comparison dataframe with the results of
# the bagged tree selection model
report_df = update_report(caret_obj = bagg_train,
                          report_df = report_df,
                          model_name = 'Bagg Tree')
```

Based on the results, for the best fitted bagged tree, the 10-fold cross validation `RMSE` is 2.735 and R^2 is 0.9058. The `mtry` for the best model is 264 which is equal to the size of metabolite variables.

Random Forest

Random forest is an ensemble learning method that builds multiple decision trees and combines their outputs to improve prediction accuracy and reduce overfitting. The benefit of using a random forest model is that it can handle large datasets with high dimensionality and can capture nonlinear relationships between predictors and response variables. It also has a built-in method for feature selection, as it considers only a subset of variables at each split, which can help to reduce the impact of irrelevant variables and improve model performance. Additionally, random forest models are relatively easy to tune and have good out-of-the-box performance.

Here, I am fitting a random forest to predict Gestational age based on metabolite data. To perform the random forest analysis, I used the `train` function from the `caret` package. I specified the method to be `rf` for random forest and set the number of trees to be 500. I also defined the tuneGrid as a data frame with four different values of mtry, which controls the number of variables that are considered at each split in the tree. These values are (n, $\sqrt{n}$, $n/3$, and $\log_2(n)$). I set the `trControl` parameter to perform a 10-fold cross-validation to assess the performance of the model.

```r
### random Forest
set.seed(1234)
rf_train <- train(x = train[, metabolite_cols],
                  y = train$Gestational.age..GA..weeks,
                  method = "rf", ntree = 500,
                  tuneGrid = data.frame(mtry = c(length(metabolite_cols),
                                                 floor(sqrt(length(metabolite_cols))),
                                                 floor(length(metabolite_cols)/3),
                                                 floor(log2(length(metabolite_cols))))),
                  trControl = trainControl(method = "cv", index = folds))
```

```
# Get the best tune from the random forest
bt = rf_train$bestTune
# Get the results for the best tune of the bagged tree model
best_result = rf_train$results[which(rf_train$results$mtry == bt$mtry), ]
# Update the model comparison dataframe with the results of
# the random forest selection model
report_df = update_report(caret_obj = rf_train,
                          report_df = report_df,
                          model_name = 'Random Forest')
```

Based on the results, for the best fitted random forest model, the 10-fold cross validation `RMSE` is 2.6864 and R^2 is 0.9125. The `mtry` for the best model is 88 which is equal to the one-third of the size of metabolite variables.

Boosted Tree

Gradient Boosting Machines (GBMs) can be highly accurate and powerful for predictive modeling, as they can handle all data types (continuous and categorical) and are able to identify non-linear relationships between predictors and the target variable. GBMs work by iteratively adding trees to the model, each of which attempts to improve upon the errors of the previous tree. This process allows the model to continuously improve its performance and accuracy over time. Additionally, GBMs can handle missing data and outliers in a robust way. Overall, the benefit of using GBMs is that they can provide highly accurate predictions with minimal assumptions about the data, making them a useful tool in many real-world applications.
Here, I am fitting a GBM using the `caret` and `gbm` packages. I have set up a tuning grid using the `expand.grid` function to define the combinations of hyperparameters to be tested during the model training process. The hyperparameters include the number of trees (`n.trees`), the maximum depth of each tree (`interaction.depth`), the learning rate (`shrinkage`), and the minimum number of observations in each tree node (`n.minobsinnode`). I have then used the train function to train the GBM model on the training dataset, specifying the method as `gbm`, bag fraction as 0.50, and the tuning grid as `tg_boostreg` (the data frame that has the grid of hyperparameters). I have also set up cross-validation using the `trainControl` function with method as "cv" and specified the number of folds using folds. Additionally, I have set `verbose` to FALSE to avoid output messages during the training process.

```
# set up tuning grid
tg_boostreg <- expand.grid(n.trees = c(200, 400, 600),
                           interaction.depth = c(3:10),
                           shrinkage = c(0.0001, 0.001, 0.01, 0.1),
                           n.minobsinnode = 10)

### gbm
set.seed(1234)
gbm_train <- train(x = train[, metabolite_cols],
                   y = train$Gestational.age..GA..weeks,
                   method = "gbm", bag.fraction = 0.50, tuneGrid = tg_boostreg,
                   trControl = trainControl(method = "cv", index = folds),
                   # to avoid unwanted messages, use verbose = FALSE
                   verbose = FALSE)
```

```
# Get the best tune from the gbm
bt = gbm_train$bestTune
# Get the results for the best tune of the gbm
best_result = gbm_train$results[which.min(gbm_train$results$RMSE), ]
```

```r
# Update the model comparison dataframe with the results of
# the random forest selection model
report_df = update_report(caret_obj = gbm_train,
                          report_df = report_df,
                          model_name = 'GBM')
```

Based on the results, for the best fitted GBM, the 10-fold cross validation `RMSE` is 2.4804 and R^2 is 0.9227. The best combination of parameters is: `shrinkage` = 0.01, `interaction.depth` = 9, `n.minobsinnode` = 10, and `n.trees` = 600.

Summary

I have trained 11 different ML models on the training data and evaluated their performance using 10-fold cross-validation. The results are summarized in Table 2, which shows that GBM had the highest R^2 value and the lowest `RMSE` and `MAE` values among all the models. Based on the `MAE`, GBM model predicts the gestational age with an average error of less than two weeks, making it the most accurate model for this task.

Table 2: Performance of the models on a 10-fold cross vaidation study.

	RMSE	MAE	Rsquared
GBM	2.480	1.938	0.923
Random Forest	2.686	2.052	0.912
Bagg Tree	2.735	2.125	0.906
LASSO	3.001	2.427	0.891
Ridge	3.084	2.498	0.887
PLS	3.202	2.596	0.878
PCR	3.237	2.608	0.875
Forward Selection	3.244	2.552	0.872
Decision Tree	3.287	2.621	0.866
Backward Selection	3.593	2.939	0.855
Simple Linear Regression	4.565	3.575	0.786

Based on the Fig. 2, it is clear that the performance of the 11 models varies significantly. Fig. 2.a shows that GBM has the lowest RMSE and MAE values, followed by Random Forest and Bagged Tree. The order of the top three models remains the same when considering the R^2, with GBM having the highest value, followed by Random Forest and Bagged Tree. The other models show a relatively higher error and lower R^2 compared to these top three models. These results suggest that GBM and Random Forest are the most effective models for predicting gestational age based on the given metabolite values.

```r
# define color palette
cbp = c("#FFC72C", '#008364',"#A75523")
# create first plot
p1 = report_df %>%
  # Add rownames as a new column called 'model'
  rownames_to_column(var = 'model') %>%
  # Select only the 'model', 'RMSE', and 'MAE' columns
  select(model, RMSE, MAE) %>%
  # Gather the 'RMSE' and 'MAE' columns into a new column called 'val',
  # and 'Metric' is a new column name
  gather('Metric', 'val', -model) %>%
  ggplot(aes(x = reorder(model, desc(val)), y = val, group = desc(Metric)))+
  geom_bar(aes(fill = Metric), stat = 'identity', position=position_dodge())+
  ylab('Error')+
  coord_flip()+
  theme_classic()+
  # Manually set the colors of the legend
  scale_fill_manual(values = cbp[1:2])+
  theme(axis.title.y = element_blank(),
        axis.text.x = element_text(size = 6),
        axis.text.y = element_text(size=6),
        axis.title.x = element_text(face="bold", size = 8),
        legend.text = element_text(size = 6),
        legend.title = element_text(size = 8),
        legend.position = c(0.9, 0.9),
        legend.key.size = unit(.3, 'cm'))

# create second plot
p2 = report_df %>%
  # Add rownames as a new column called 'model'
  rownames_to_column(var = 'model') %>%
  # Select only the 'model' and 'Rsquared' columns
  select(model, Rsquared) %>%
  ggplot(aes(x = reorder(model, Rsquared), y = Rsquared))+
  # Create a bar plot with a fixed color
  geom_bar(fill = cbp[3], stat = 'identity', position=position_dodge())+
  ylab(bquote(R^2))+
  scale_y_continuous(limits = c(0, 1), breaks = seq(0,1,.1))+
  coord_flip()+
  theme_classic()+
  theme(panel.grid.major.x = element_line( size=.3, color="gray" ),
        axis.title.y = element_blank(),
        axis.text.x = element_text(size = 6),
        axis.text.y = element_text(size=6),
        axis.title.x = element_text(face="bold", size = 8),
        legend.text = element_text(size = 8),
        legend.title = element_text(size = 10))

# Combine both plots into a single plot
final_plot = ggdraw()+
  draw_plot(p1,
            x = 0, y = 0, width = .49, height = 0.9)+
  draw_plot(p2,
            x = 0.51, y = 0, width = .49, height = 0.9)+
  # Add a title to the plot
  draw_text('Performance of the models on a 10-fold cross vaidation study.', x = .5, y = .95)+
  # Add labels to both subplots
  draw_plot_label(label = c('a', 'b'),
                  size = 10, x = c(0.05, 0.55), y = c(0.9, 0.9))
# save the plot
ggsave(filename = 'figures/performance.pdf', plot = final_plot,
       width = 7.2, height = 2.6, dpi = 300)
final_plot
```

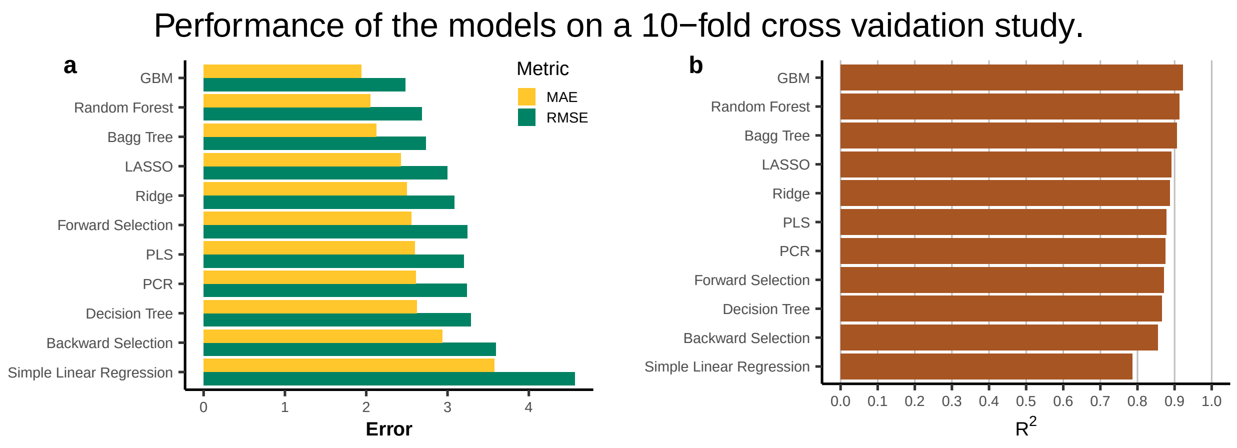

Figure 2: Performance of the models on a 10-fold cross vaidation study. a) RMSE and MAE values, b) Rsquare score.

```r
# Make predictions using the GBM and Random Forest models
gbm_pr = predict(gbm_train, test[, metabolite_cols])
rf_pr = predict(rf_train, test[, metabolite_cols])

# Combine the observed and predicted values into a data frame
preds = data.frame('observed' = test$Gestational.age..GA..weeks,
                   'GBM' = gbm_pr,
                   'rf' = rf_pr) %>%
  gather('Model', 'Predictions', -observed)

# Rename the Random Forest model
preds$Model = ifelse(preds$Model=='GBM', preds$Model, 'Random Forest')

# Calculate the residuals
preds$Residuals = preds$observed - preds$Prediction

# Reshape the data for plotting
preds = preds %>%
  gather('Type', 'val', -c(observed:Model))

# Calculate the mean absolute error for each model
gbm_mae = mean(abs(gbm_pr - test$Gestational.age..GA..weeks))
rf_mae = mean(abs(rf_pr - test$Gestational.age..GA..weeks))

# Create a plot of the predicted vs. observed values
plot_preds <- ggplot(preds, aes(x = observed, y = val, color = Model))+
  geom_point(alpha = 0.8)+
  theme_classic()+
  xlab('Gestational Age (Validation Set)') +
  ylab('Value') +
  scale_color_manual(values = c("#033C5A", "#AA9868"))+
  facet_wrap(~Type, scales = 'free_y')+
  theme(axis.title.y = element_text(face="bold",size=8),
        axis.text.x = element_text(face="bold", size = 8),
        axis.text.y = element_text(face="bold",size=8),
        axis.title.x = element_text(face="bold", size = 10),
        legend.text = element_text(size = 8),
        legend.title = element_text(size = 10),
        strip.background = element_blank())

# Save the plot to a PDF file
ggsave(filename = 'figures/preds.pdf', plot = plot_preds,
       width = 7.2, height = 2.8, dpi = 300)
plot_preds
```

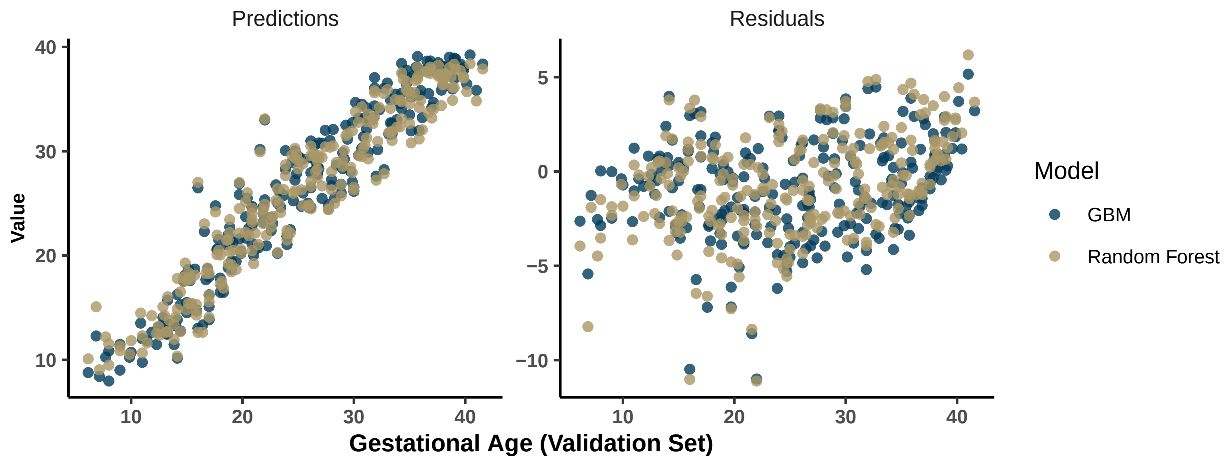

Figure 3: Performance of the top 2 models on test data from the validation set.

Fig. 3 shows the performance of top two models, `GBM` and `Random forest` on the unseen data from the validation set. `MAE` of these two models on this data are 2.13 and 2.22, respectively.

Fig. 4 shows the top 5 important metabolites for predicting gestational age using GBM and Random Forest models. The height of each bar indicates the variable importance measure, with the most important variable at the top. The selected metabolites are ranked based on their importance scores, which were calculated using the mean decrease in accuracy method. These metabolites could potentially be used as biomarkers for gestational age prediction.

```
# relative influence approach
# calculate the variable importance for the GBM and Random Forest models
gbm_imp <-  varImp(gbm_train$finalModel) %>% rownames_to_column(var = 'var')
gbm_imp$Overall <-  gbm_imp$Overall*100/sum(gbm_imp$Overall)
colnames(gbm_imp)[2] <- 'GBM'
rf_imp <-  varImp(rf_train$finalModel) %>% rownames_to_column(var = 'var')
rf_imp$Overall <-  rf_imp$Overall*100/sum(rf_imp$Overall)
colnames(rf_imp)[2] <- 'Random Forest'

# extract the top 5 important metabolites for each model and combine the results
top_gbm <- gbm_imp %>%
  top_n(5, wt = GBM) %>%
  left_join(rf_imp, by = 'var') %>%
  gather('Model', 'imp', -var)

# create a bar plot of the relative importance of the top 5 metabolites for each model
```

```
plot_imp <- ggplot(top_gbm, aes(x = reorder(var, imp), y = imp, group = desc(Model)))+
  geom_bar(aes(fill = Model), stat = 'identity', position=position_dodge())+
  ylab('Relative Importance')+
  ggtitle('Top 5 important metabolites for predicting gestational age',
          subtitle =  'Based GBM and Random Forest models')+
  coord_flip()+
  theme_classic()+
  scale_fill_manual(values = c("#033C5A", "#AA9868"))+
  theme(axis.title.y = element_blank(),
        axis.text.x = element_text(size = 6),
        axis.text.y = element_text(size=6),
        axis.title.x = element_text(face="bold", size = 8),
        legend.text = element_text(size = 6),
        legend.title = element_text(size = 8),
        legend.position = c(0.75, 0.27),
        legend.key.size = unit(.3, 'cm'))

# save the plot to a pdf file
ggsave(filename = 'figures/varimp.pdf',
       plot = plot_imp+ggtitle('', subtitle = ''),
       width = 3.2, height = 2, dpi = 300)
ggsave(filename = 'figures/varimp_with_title.pdf',
       plot = plot_imp,
       width = 7.2, height = 4, dpi = 300)

# display the plot
plot_imp
```

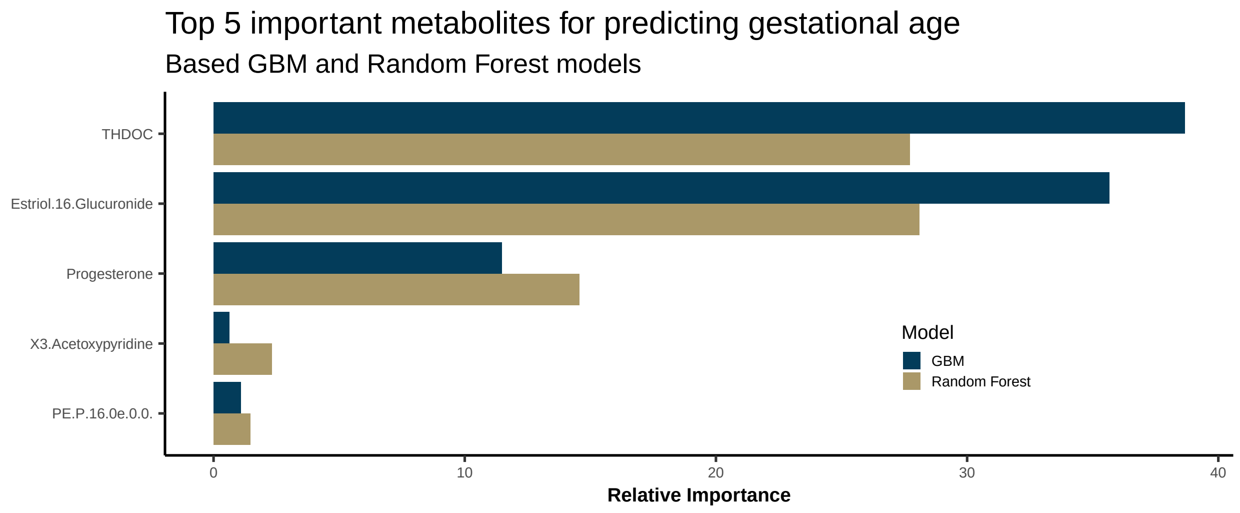

Figure 4: Variable importance plot based on mean decrease in Gini index.

Fig. 5 displays the marginal effect of the top three variables in predicting gestational age using the GBM model. The marginal effect is obtained by integrating out the other variables and examining the relationship between each predictor variable and gestational age. The plot shows how the predicted gestational age changes as the value of each variable changes, while holding all other variables constant at their mean values. Understanding the marginal effect of each variable is important for identifying the most influential predictors of gestational age in the GBM model. It can be seen that `Estriol 16 Glucuronide` starts to increase uniformly from week 20 and continues to increase until the end of the pregnancy period, which implies that it is a good predictor of gestational age. Similarly, `THDOC` starts to increase uniformly from week 22 and continues to increase until the end of the pregnancy period. Lastly, the marginal effect of `Progesterone` starts to increase uniformly from week 23 and reaches its peak around week 27. These findings suggest that the selected variables are important predictors of gestational age and could be potentially useful in clinical practice.

```r
# Select top 3 variables by importance score in the GBM model,
# and extract their names
top_3 <- gbm_imp %>%
  top_n(3, wt = GBM) %>%
  select(var) %>%
  unlist()

# Create an empty data frame to store the marginal effect
# of each top variable on Gestational Age prediction
marginal_df <- data.frame()

# For each of the top 3 variables, calculate its marginal
# effect on the predicted Gestational Age using the GBM model
for(var in top_3){
  # Obtain the partial dependence plot for the variable
  tmp <- plot(gbm_train$finalModel, i = var, return.grid = TRUE)
  # Convert the plot to a data frame and gather the x and y variables
  tmp <- tmp %>% gather('Metabolite','val', -y)
  # Append the data to the marginal effect data frame
  marginal_df <- marginal_df %>% bind_rows(tmp)
}
title = paste0('Marginal Effect of Top Three Variables on ',
               'Gestational Age Prediction in GBM Model')
# Plot the marginal effect of each top variable on
# the predicted Gestational Age using the GBM model
plot_marginal <- ggplot(marginal_df, aes(x = val, y = y, color = Metabolite))+
  geom_line()+
  ylab('Gestational Age')+
  xlab(bquote(Log[2](Intensity))) +
  ggtitle(title)+
  theme_classic()+
  scale_color_manual(values = c("#FFC72C", "#A75523", '#008364'))+
  theme(axis.title.y = element_text(face="bold", size = 8),
        axis.text.x = element_text(size = 6),
        axis.text.y = element_text(size=6),
        axis.title.x = element_text(face="bold", size = 8),
        legend.text = element_text(size = 6),
        legend.title = element_text(size = 8),
        legend.position = c(0.25, 0.8),
        legend.key.size = unit(.3, 'cm'))

# Save the marginal effect plot as a PDF file
ggsave(filename = 'figures/marginal.pdf', plot = plot_marginal+ggtitle(''),
       width = 3.2, height = 2, dpi = 300)

ggsave(filename = 'figures/marginal_with_title.pdf',
       plot = plot_marginal,
       width = 7.2, height = 3.5, dpi = 300)
# Show the marginal effect plot
plot_marginal
```

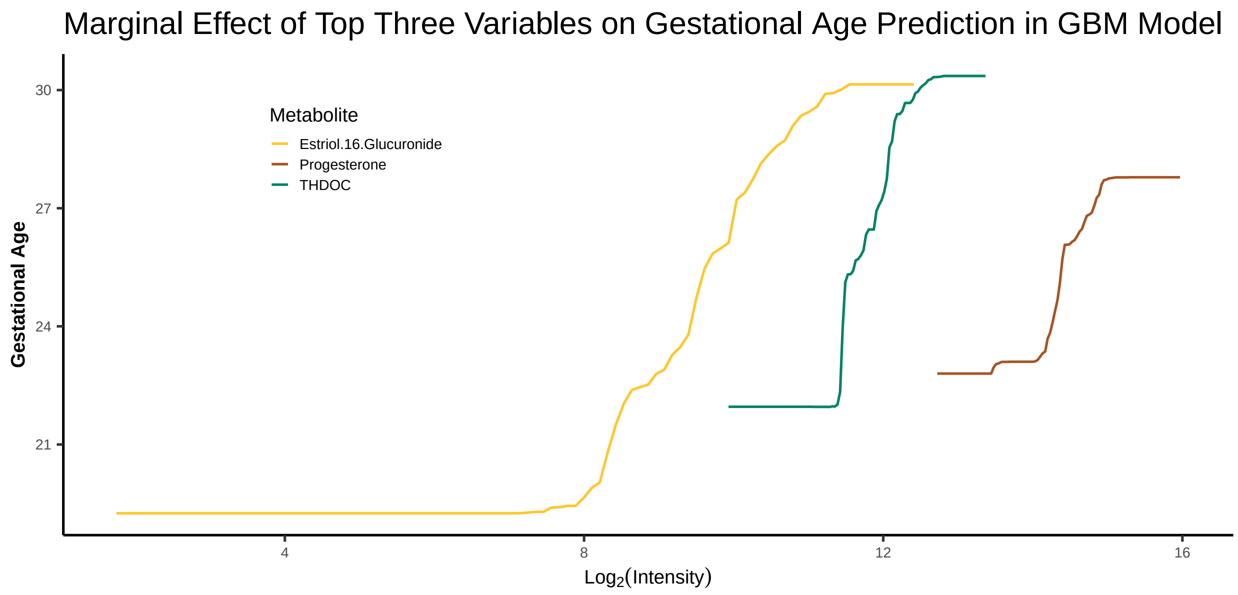

Figure 5: Marginal Effect of Top Three Variables.

Discussion

The gradient boosting machine outperformed all other 11 models in terms of `RMSE`, `MAE`, and R^2. *GBMs* are highly accurate and powerful for predictive modeling, as they can handle various data types and identify non-linear relationships between predictors and the target variable. The iterative tree-building process allows the model to continuously improve its performance, making it a useful tool in many real-world applications. In the original paper (Liang et al. 2020), LASSO was used to fit the data, and the LASSO model developed in this study had comparable performance. However, other models such as random forest, bagged tree, and GBM performed better in this study. One challenge in developing these models was hyperparameter tuning to optimize their performance, which required substantial computational resources and time. Additionally, the sample size of the dataset used in this study was relatively small, limiting the generalizability of the results. Future studies with larger sample sizes could help validate the findings of this study.

References

Aboyans, Victor, Causes of Death Collaborators, et al. 2015. "Global, Regional, and National Age-Sex Specific All-Cause and Cause-Specific Mortality for 240 Causes of Death, 1990-2013: A Systematic Analysis for the Global Burden of Disease Study 2013." *The Lancet (British Edition)* 385 (9963): 117–71.

Alkema, Leontine, Doris Chou, Daniel Hogan, Sanqian Zhang, Ann-Beth Moller, Alison Gemmill, Doris Ma Fat, et al. 2016. "Global, Regional, and National Levels and Trends in Maternal Mortality Between 1990 and 2015, with Scenario-Based Projections to 2030: A Systematic Analysis by the UN Maternal Mortality Estimation Inter-Agency Group." *The Lancet* 387 (10017): 462–74.

Blencowe, Hannah, Simon Cousens, Doris Chou, Mikkel Oestergaard, Lale Say, Ann-Beth Moller, Mary Kinney, and Joy Lawn. 2013. "Born Too Soon: The Global Epidemiology of 15 Million Preterm Births." *Reproductive Health* 10 (1): 1–14.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software* 33 (1): 1–22. https://doi.org/10.18637/jss.v033.i01.

Greenwell, Brandon, Bradley Boehmke, Jay Cunningham, and GBM Developers. 2022. *Gbm: Generalized Boosted Regression Models*. https://CRAN.R-project.org/package=gbm.

Kuhn, Max. 2022. *Caret: Classification and Regression Training*. https://CRAN.R-project.org/package=caret.

Liang, Liang, Marie-Louise Hee Rasmussen, Brian Piening, Xiaotao Shen, Songjie Chen, Hannes Röst, John K Snyder, et al. 2020. "Metabolic Dynamics and Prediction of Gestational Age and Time to Delivery in Pregnant Women." *Cell* 181 (7): 1680–92.

Liaw, Andy, and Matthew Wiener. 2002. "Classification and Regression by randomForest." *R News* 2 (3): 18–22. https://CRAN.R-project.org/doc/Rnews/.

Ngo, Thuy TM, Mira N Moufarrej, Marie-Louise H Rasmussen, Joan Camunas-Soler, Wenying Pan, Jennifer Okamoto, Norma F Neff, et al. 2018. "Noninvasive Blood Tests for Fetal Development Predict Gestational Age and Preterm Delivery." *Science* 360 (6393): 1133–36.

Posit team. 2022. *RStudio: Integrated Development Environment for r*. Boston, MA: Posit Software, PBC. http://www.posit.co/.

R Core Team. 2022. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Say, Lale, Doris Chou, Alison Gemmill, Özge Tunçalp, Ann-Beth Moller, Jane Daniels, A Metin Gülmezoglu, Marleen Temmerman, and Leontine Alkema. 2014. "Global Causes of Maternal Death: A WHO Systematic Analysis." *The Lancet Global Health* 2 (6): e323–33.

Sedgh, Gilda, Susheela Singh, and Rubina Hussain. 2014. "Intended and Unintended Pregnancies Worldwide in 2012 and Recent Trends." *Studies in Family Planning* 45 (3): 301–14.

Therneau, Terry, and Beth Atkinson. 2022. *Rpart: Recursive Partitioning and Regression Trees*. https://CRAN.R-project.org/package=rpart.

Wang, Xiaobin, Changzhong Chen, Lihua Wang, Dafang Chen, Wenwei Guang, Jonathan French, Reproductive Health Study Group, et al. 2003. "Conception, Early Pregnancy Loss, and Time to Clinical Pregnancy: A Population-Based Prospective Study." *Fertility and Sterility* 79 (3): 577–84.

Wickham, Hadley, and Jennifer Bryan. 2022. *Readxl: Read Excel Files*. https://CRAN.R-project.org/package=readxl.

Xie, Yihui. 2022. *Knitr: A General-Purpose Package for Dynamic Report Generation in r*. https://yihui.org/knitr/.