HOMEWORK 2

Luisa Porzio (ID: 255069)

2025-03-22

1 Introduction

The following analysis is based on diabetes data with the aim of investigate the association between different clinical factors correlated with diabetes progression after 1 year from a baseline, identifying the main predictors among them. The data were collected from 442 diabetic patients.

The main explanatory variables considered are: **progr** (the target variable measuring disease progression. The higher the value, the worse the progression), **age**, **sex**, **BMI** (body mass index), **BP** (average blood pressure, in mm Hg), **TC** (total cholesterol, mg/dl), **LDL** (low-density lipoproteins, mg/dl), **HDL** (high-density lipoproteins, mg/dl), **TCH** (ratio between total cholesterol and HDL), **TG** (triglycerides level, mg/dl, log-scaled), **GC** (blood glucose, mg/dl).

The full code to replicate the analysis is available on GitHub.

2 Exploratory Analysis

The first step of the analysis is conducting a data pre-processing to check the structure of the data-set. In this instance, there are no missing values.

The only categorical variable is sex, and in order to proceed it is necessary to transform the nature of the variable from `numeric` to `factor`. It is not possible to assign a label to the values of the variable (1,2) given that there are no informations on which value corresponds to male and which to female.

```
data <- data %>%
    mutate(sex = as.factor(sex))
```

3 Model fitting

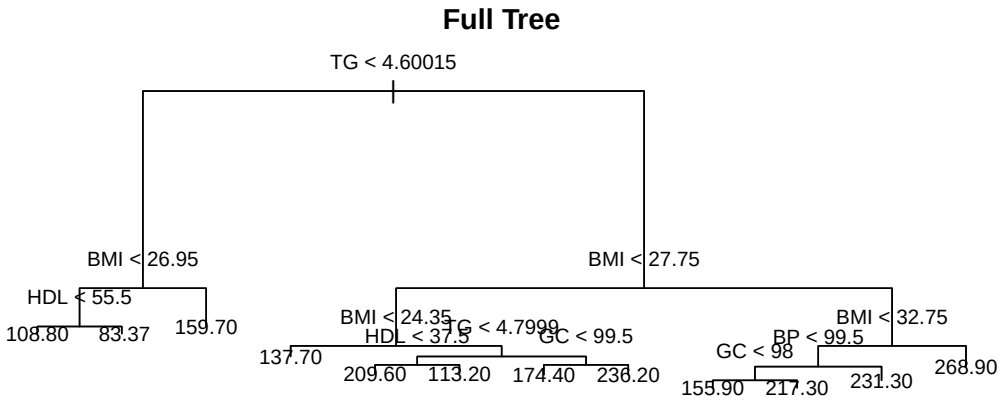
In the following section the different requested models will be fitted, starting from the Tree Model, followed by the Random Forest and the Boosted Forest. Each model will be deeply and carefully analyzed in order to then draw the conclusions on which model best fits the data which will be available in the last section of this report.

3.1 Tree

In order to fit the tree model it is necessary to tune the necessary hyper-parameters. In this case it is not recomanded to adopt a train/test split given that the used dataset only counts 442 observations. Employing such method would most likely lead to a bad performance of the model that might not accurately learn due to the training set being too small. To avoid this, it is best to fit the model on the whole dataset.

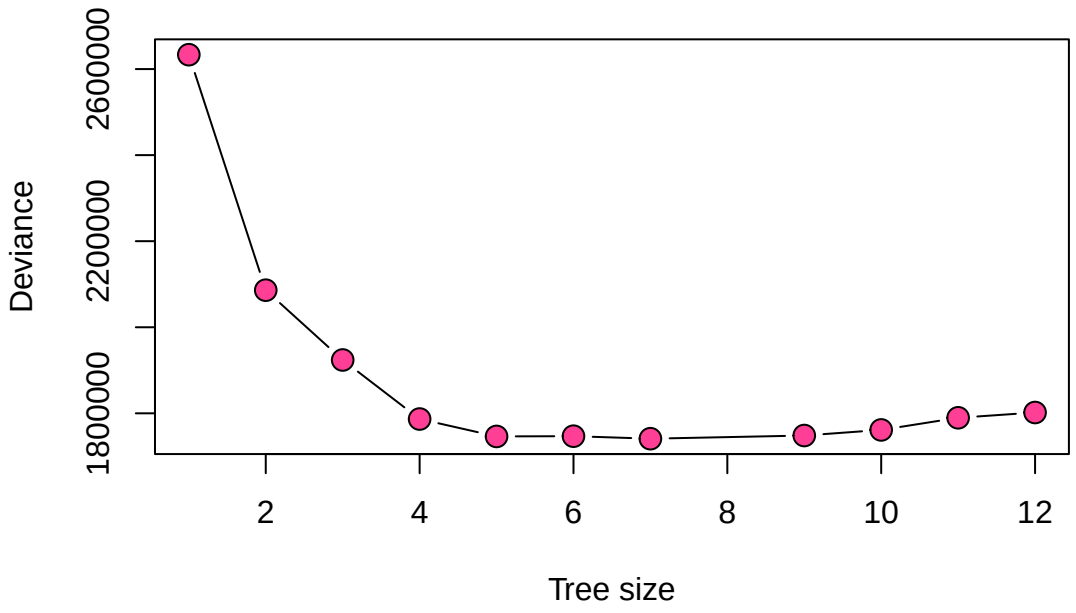
3.1.1 Full Tree

```
full_tree <- tree(progr ~ ., data = data)
```



However, in order to choose accurately the hyper-parameters it is possible to conduct a cross-validation that helps simulate multiple train/test splits. To do so, we plot the deviance which measures the error and the tree size.

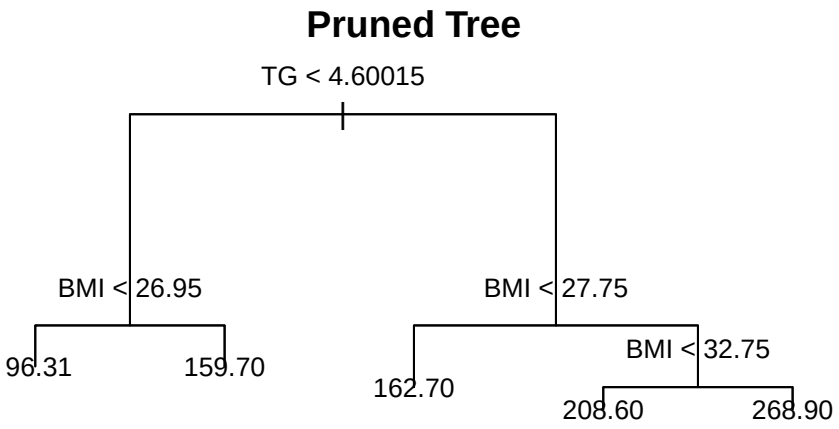
```
cv_tree <- cv.tree(full_tree)
```



From this is is possible to observe that the best size is 5 as it minimizes the deviance in a 10-fold CV setting. Therefore the initial Full Tree in Section 3.1.1 can be pruned on the base of this parameter.

3.1.2 Pruned Tree

```
pruned_tree <- prune.tree(full_tree, best = 5)
```



We can see from the fitted model and its representation that the tree has 5 terminal nodes, as specified by the pruning condition after we found the optimal size. Importantly, we notice that the model only uses two variables to partition the prediction space into five. The two variables are TG and BMI which have been selected internally by the model since they were minimizing the error.

3.2 Random Forest

If we wanted to add more variance to our model and prevent it from always selecting TG and BMI to perform all the initial splits we can use a Random Forest (RF) model. An RF model is an ensemble model that allows different trees to be fitted on a specific subset of features that is randomly picked by each of them. The number of features that each tree has to select is, therefore, the main hyperparameter to tune for this model. This can be done by exploring different strategies based on multiple re-sampling techniques, but there is one specific strategy that is peculiar to RF models and can be used to save some computational power. Indeed, with RF models each tree is fitted only on some specific observations and, therefore, this characteristic can be used to ask a tree to predict an observation from the dataset that it has not encountered during training. This solves the problem of data leakage as those predictions will be comparable to predictions made on a held-out test set. RF models refer to this error as the Out-of-bag (OOB) error. Using this error to tune the hyper-parameters of the model saves time and assures performances equivalent to performing a k-fold CV.

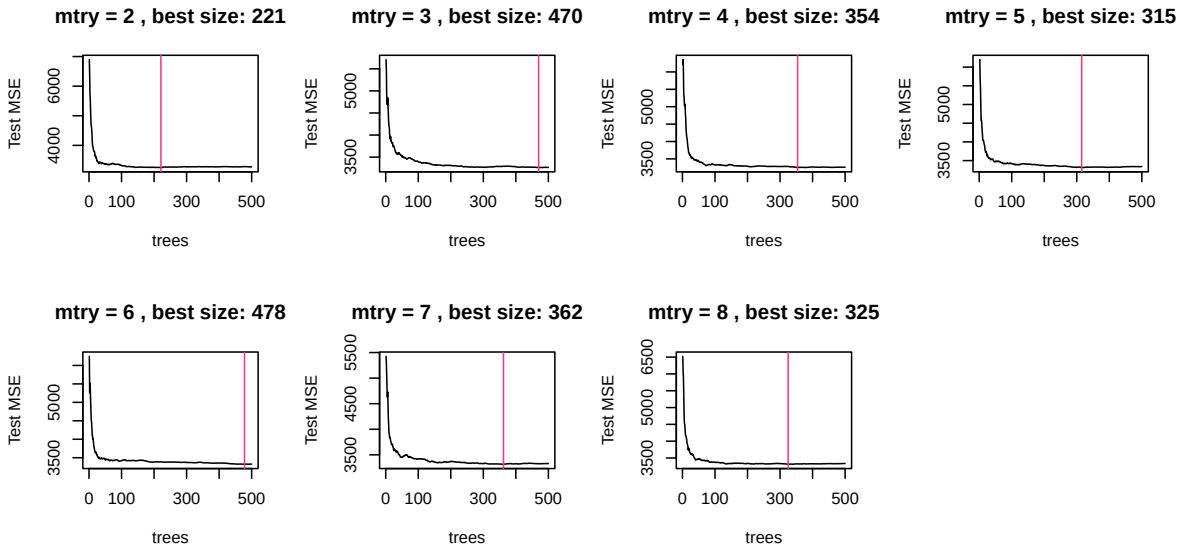
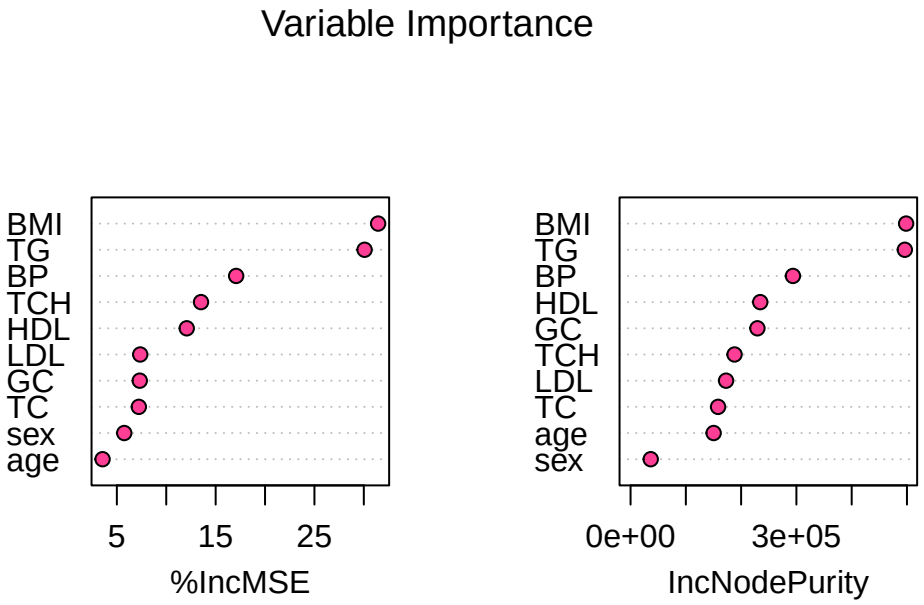


Table 1: Evaluating the optimal mtry hyperparameter

n_vars	mse
4	3251.173
2	3251.985
3	3264.465
8	3314.111
5	3314.326
7	3315.775
6	3322.831

From the table above it is possible to see that the best number of variables to be considered for each split is 4. However, that is achieved with a RF model with a size of 354. By looking at the plots, however, it is possible to see that the lowest error when mtry is set to 2 is very close in terms of performance to the MSE when mtry is set to 4. Moreover, simplifying to a model having 2 variables for each tree is also the fastest way to achieve the lowest MSE as the model reaches that level of performance after fitting 221 trees out of the total 500 trees.

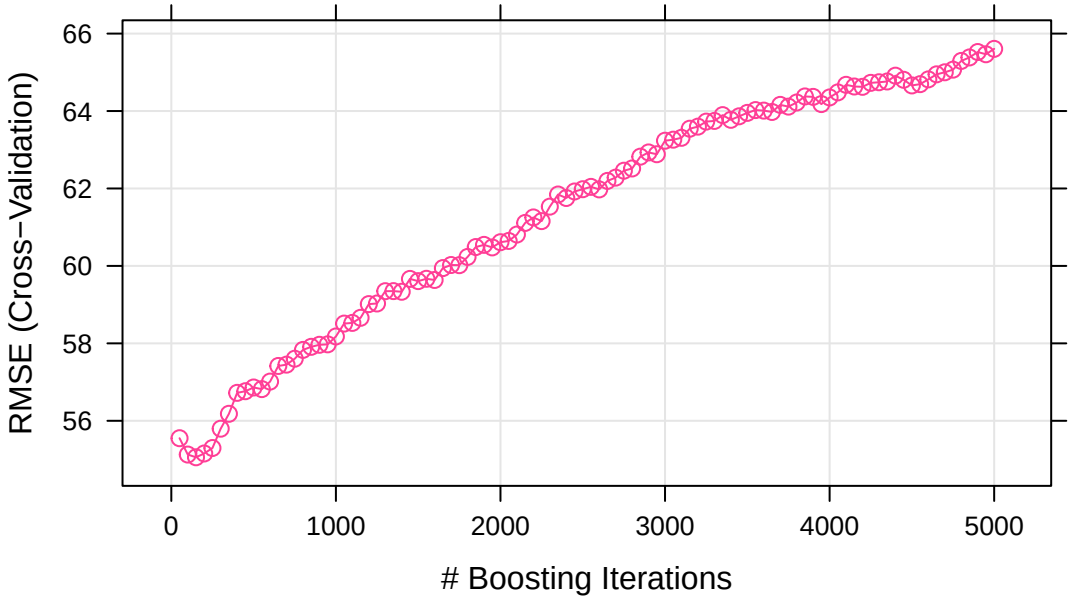


The importance of each variable can be observed from the plots above. In the plot on the left we can see that the increase in MSE in case the variables BMI and TG were to be substitued by another predictor is very high (around 60% when combined). This means that these variables are truly important for the model and that it relies on them for making the best predictions. In the plot on the right, this idea is confirmed since the increase in node purity when these values are added to a tree is very high compared to all the others.

3.3 Boosted Models

3.3.1 Alternative 1

Another model that can be fitted to provide an alternative solution to this regression task is a **boosted regression tree model**. This model follows an approach that is different from a RF. In this case the model is built sequentially and each tree has a chance to learn from the errors (residuals) made by its predecessors. This allows the model to become very efficient at recognizing specific patterns while retaining some flexibility to review its optimization process along the way. In this case, the number of iterations, and therefore the number of trees that are built sequentially, is the most important hyperparameter to tune. For this study, a sequence of numbers from 50 to 5000 in steps of 50 (e.g. 50, 100, 150, …) were tried as possible candidates.

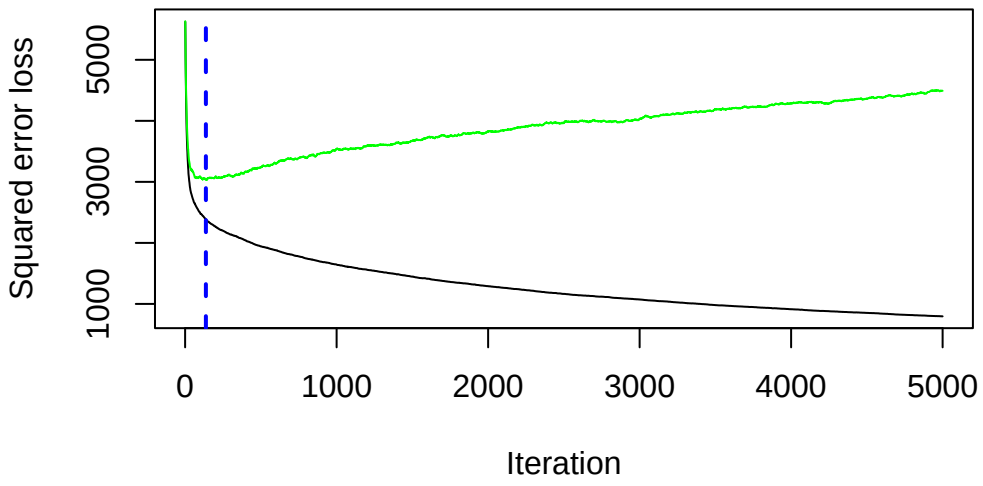


Based on the root mean squared error from CV we can say that the best choice for the `n.trees` parameter among those tried is 150. This is the value that minimizes the CV error measured through RMSE. Importantly, we also notice that after the optimal number of iterations the RMSE of the model starts to increase, thus suggesting that the model is now fitting noise that is lowering the quality of its predictions.

3.3.2 Alternative 2

In alternative to using a random sequence of numbers to tune the number of iterations, it is possible to directly use the `gbm` function that uses in this case a 10-fold CV. This should give a more specific estimate for the hyper-parameter.

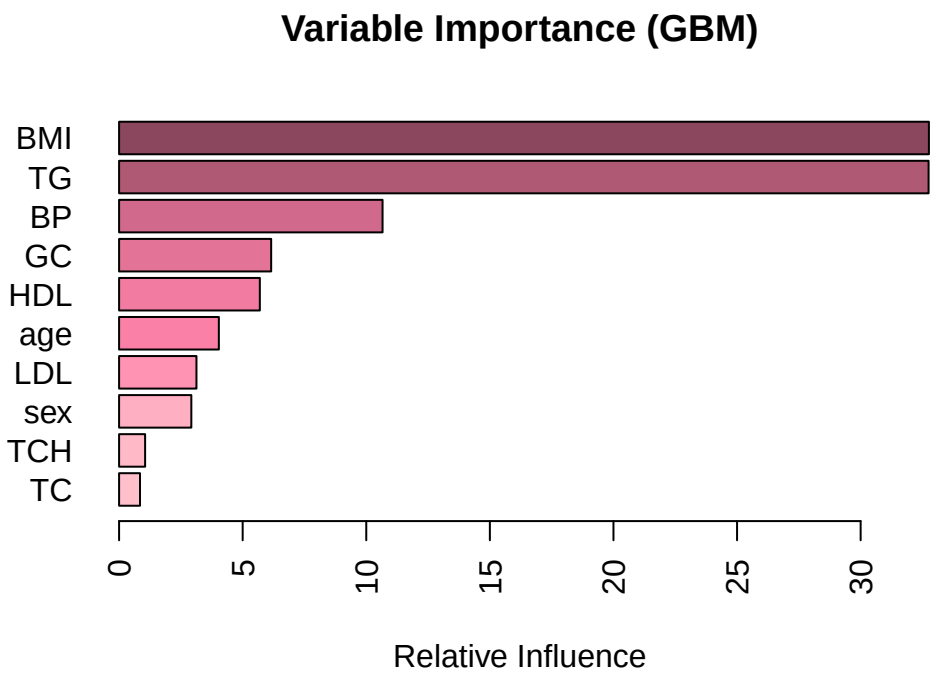
```
set.seed(1)
boosted <- gbm(progr ~ ., distribution = "gaussian", data = data,
    n.trees = 5000, cv.folds = 10)
```



From the plot above it seems that after performing a 10-fold CV the best number of iterations is 137, which corroborates the result from Section 3.3.1 , as 150 was the closest number to 137 among those used. This is due to the fact that the model appears to reach its best performance after a relatively low number of iterations. Interestingly, increasing the number of iterations seems to be adding noise to the model whose error (as measured by the squared error loss) starts increasing shortly after the optimal number of iterations found.

```
boosted_tuned <- gbm(progr ~ ., distribution = "gaussian", data = data,
    n.trees = best_iter_num, cv.folds = 10)
```

3.4 Variables Importance

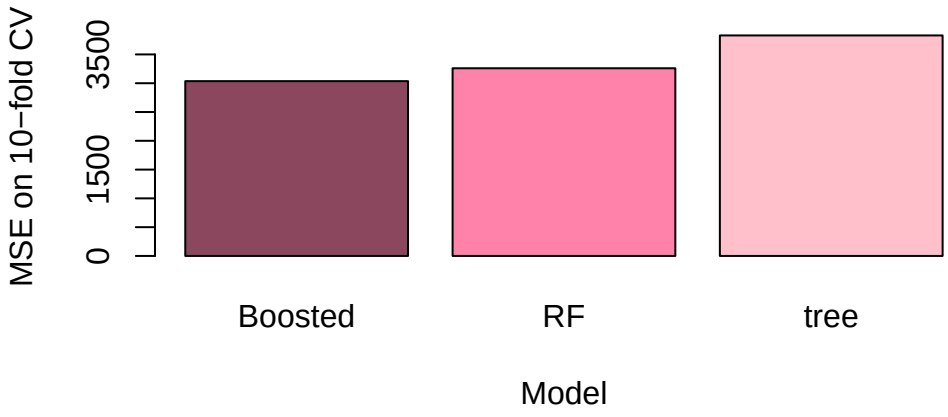


From the relative influence plot above we can form an idea as to what are the main predictors that influence the outputs of the model. Unsurprisingly, we see that also in this model the BMI and TG variables cover a predominant role compared to all other variables available in the dataset.

4 Model selection

In this section, we compare all the models fitted so far and we perform further experiments to select the best model for this specific task. The main procedure that will be followed in this section is the following:

1. perform a 10-fold cross validation
2. in each iteration the model is trained on the train data and the hyper-parameters are adjusted based solely on those
3. the MSE is computed for each iteration on the unseen test fold of the dataset
4. at the end the average MSE is computed for each model and the model with the lowest average MSE is picked



After performing a 10-fold CV the boosted model seems to be performing better compared to the simple tree model and the RF model. Indeed, the results indicate that the **boosted model** outperforms both the **single decision tree** and the **random forest**, demonstrating its superior predictive performance. Additionally, the **random forest** shows improved performance over the single tree model, highlighting its greater flexibility and effectiveness when properly tuned.

5 Conclusion

The main conclusions drawn from this analysis will be now discussed. First, few features stand out as particularly important for understanding the progression of diabetes over time. As discussed in Section 3.4, **Body Mass Index** (**BMI**) and **triglycerides** (**TG**) consistently outperformed other predictors. Therefore, it is possible to consider these two clinical factors as the most important ones, to which diabetes patients should pay more attention compared to other factors that such as sex that has a lesser impact on the development of the disease.

Moreover, this task favors models that learn sequentially (incrementally correcting previous errors) over more flexible, high-variance models like random forests. This is likely due to the relatively low number of dominant predictors, which aligns well with the strengths of boosting algorithms.

However, it's important to note that boosting is also more computationally intensive. In larger datasets, random forests may be preferred despite slightly lower predictive performance, as their parallelizable structure makes them more efficient to train at scale.