Homework2: tree based methods

Camilla Bonomo matr. 255138

2025-04-22

In this analysis, I investigate how a set of clinical measurements relates to the progression of diabetes one year after baseline. The data set serves as the foundation for three tree-based predictive methods: regression tree, random forest, and gradient-boosted machines. My goals are to build each model, tune its complexity, interpret its structure or variable importance, and finally compare their predictive accuracy via cross-validation (link to repository: https://github.com/camillabonomo02/Homework_2.git).

Data exploration and cleaning

For this analysis I used a data set - defined here as *df_diab* - of diabetes data to investigate the association between disease progression after 1 year from a baseline (*progr*; the higher the value, the worse the progression) and a number of clinical predictors, measured in 442 diabetic patients (e.g. ages, sex, body mass index, average blood pressure,…).

In particular, the ***progr*** variable is the **numerical response** variable that returns an integer that represents the value of the progression. The predictors are the other 10 variables included in the data set, which are all numerical variables but one (*sex*). Having said that, it is important to factorize the latter .

By further exploration, it turned out that the data set does not contain missing values.

Single decision tree

First, I fit a regression tree on the full data set. A regression tree recursively partitions the predictor space into regions that are increasingly homogeneous with respect to the outcome.

```
set.seed(1)
tree.full <- tree(progr ~ ., data = df_diab)
```

By inspecting the summary of the model it turns out that although I supplied nine candidate predictors, the tree chose only five to form its splits. The fact that age, sex, total cholesterol, LDL and the TC:HDL ratio never appear tells us they added no further reduction in squared error once the other five were accounted for. With 12 terminal nodes, the tree is moderately complex. The MSE of this model fitted on the entire data set is 2674 and the residual distribution is centered around 0, implying that the tree is unbiased overall.

```
Regression tree:
tree(formula = progr ~ ., data = df_diab)
Variables actually used in tree construction:
[1] "TG"  "BMI" "HDL" "GC"  "BP"
Number of terminal nodes:  12
Residual mean deviance:  2674 = 1150000 / 430
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-140.900  -35.830   -4.805    0.000   33.540  154.100
```

It is possible to view the actual tree:

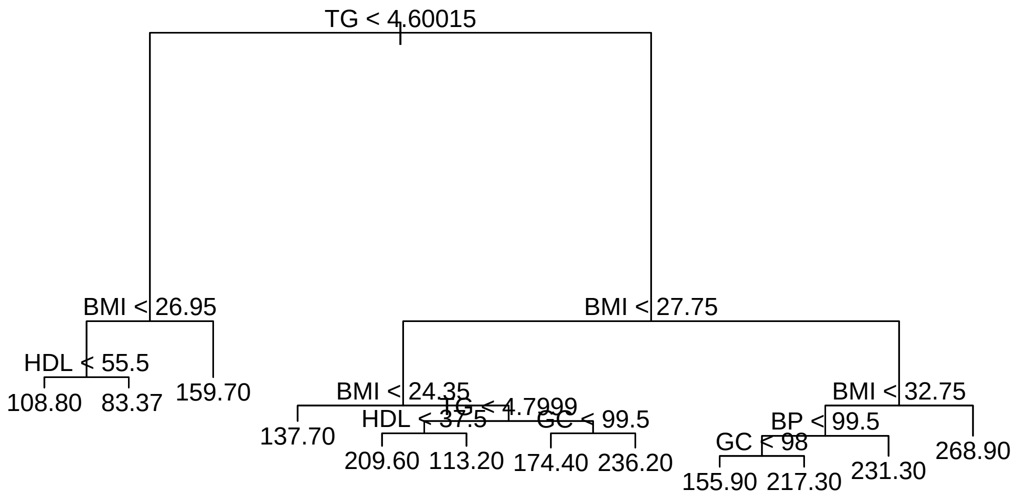

Figure 1: Unpruned Regression Tree

This balanced view sets the stage for both pruning considerations and comparison to more powerful ensemble methods.

Choosing complexity via cross-validation (pruning)

After fitting the full tree, I use cross-validation (via *cv.tree*) to measure how predictive deviance changes as I prune back to smaller trees. The optimal tree size minimizes the cross-validated deviance, balancing bias against variance. I then prune to that size and display the final tree structure.

```r
cv.out <- cv.tree(tree.full, FUN = prune.tree)
```

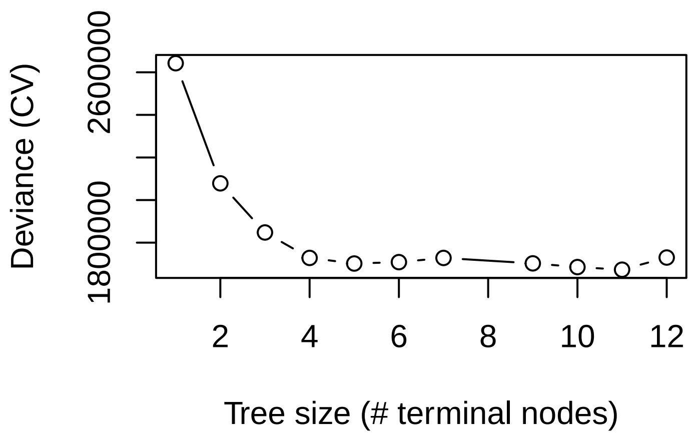

Figure 2: Number of nodes vs deviance

After looking at the curve we can think of pruning the tree to less than 12 terminal nodes. Let's see what is the actual tree size "k" that minimizes the cross-validation deviance.

```r
best.size <- cv.out$size[which.min(cv.out$dev)]
```

Optimal k
11

The tree can be pruned to 11 terminal nodes as such:

```r
tree.pruned <- prune.tree(tree.full, best = best.size)
```

And the following tree is now generated:

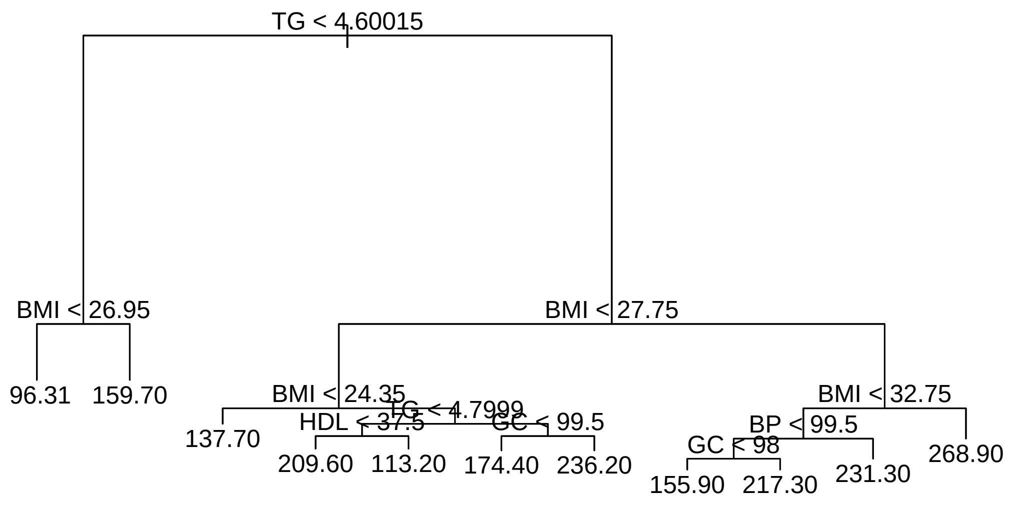

Figure 3: Pruned Tree (size=11)

The pruned tree removes one split from the full version (specifically, the leftmost HDL-based branch under the "TG < 4.6" and "BMI < 26.95" path). Basically, while both trees rely heavily on TG, BMI, GC, HDL, and BP for splits, the pruned tree makes the decision path more streamlined and interpretable. It still captures the key interactions but with one fewer terminal region to explain.

Random forest

Random forests grow an ensemble of decision trees on bootstrapped samples, and at each split only consider a random subset of m predictors. This de-correlates the trees and dramatically reduces variance relative to a single tree or bagging. For consistency,even this model is fitted in the entire data set at first. This full data fit is not used for model validation but it serves as a way to understand the internal mechanics of each model under identical conditions.

I try out a few **m** values and record the correspondent **OOB error**.

```r
set.seed(1)

oob.err <- numeric(10)
for (mtry in 1:10) {
    rf.mod <- randomForest(progr ~ ., data = df_diab,
        mtry = mtry, ntree = 500, importance = TRUE)
    oob.err[mtry] <- rf.mod$mse[500]
}
```

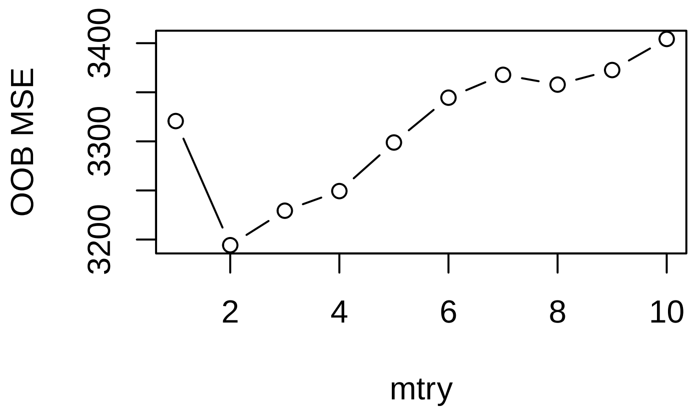

Figure 4: m vs OOB MSE

From the plot I can see that the best number of variables **m** to consider at each split is:

optimal m
2

I then fit the optimal forest with best m value (m=2) and evaluate the predictors.

```r
set.seed(1)
rf.opt <- randomForest(progr ~ ., data = df_diab, mtry = best.m,
    ntree = 1000, importance = TRUE)
print(importance(rf.opt))
```

```
      %IncMSE IncNodePurity
age  3.980547     152346.04
sex  8.051857      35264.45
BMI 43.160538     506491.56
BP  24.988656     298208.88
TC   9.939241     159197.99
LDL 10.999665     173171.77
HDL 17.202367     229690.74
TCH 18.496721     185076.88
TG  42.502145     495755.00
GC   9.438693     222953.87
```

The *importance()* measures (%IncMSE and IncNodePurity) rank predictors by how much they reduce error when included in splits. We find that BMI and TG are most influential, suggesting these clinical factors are the strongest drivers of one-year progression in this cohort. I can visually show the content of the table with the *varImpPlot* function:

```
varImpPlot(rf.opt)
```

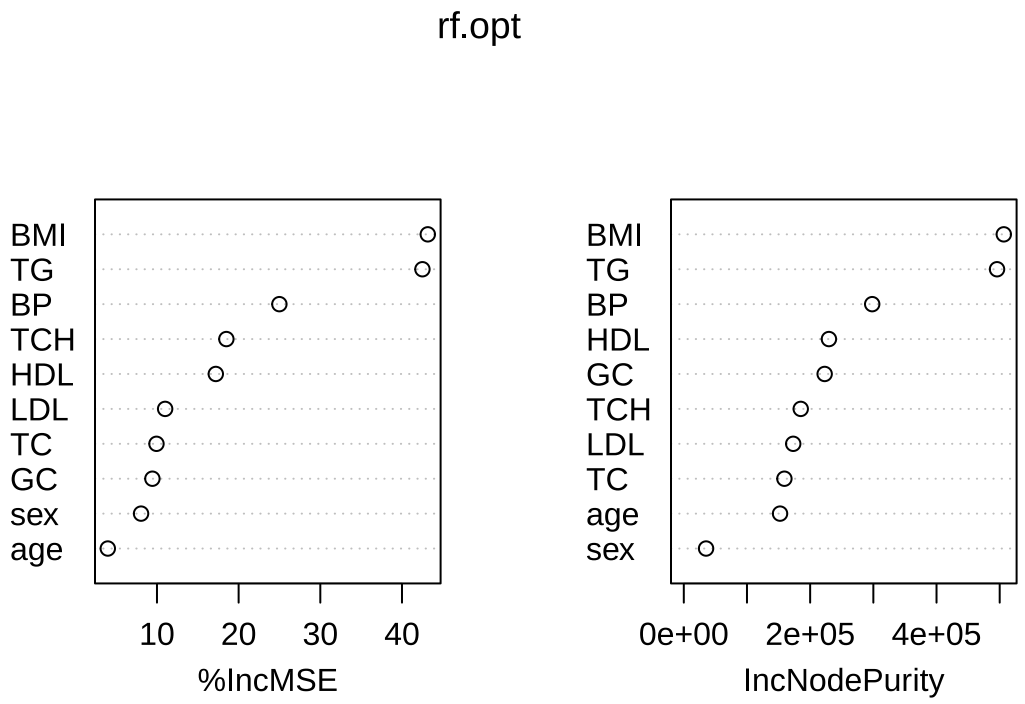

Figure 5: Predictors' Importance - Random Forest

Boosted regression trees

Boosting fits trees sequentially, each one correcting the residuals of the previous ensemble. Key parameters are the number of trees (n.trees), tree depth (interaction.depth), and learning rate (shrinkage). I fit 5,000 trees of depth 3 with a learning rate of 0.01 so that the initial fit is done with a large n.trees and modest shrinkage. Here, for the reasons highlighted before, I still fit the model on the entire data set.

```r
set.seed(1)
boost.mod <- gbm(progr ~ ., data = df_diab, distribution = "gaussian",
    n.trees = 5000, interaction.depth = 3, shrinkage = 0.01,
    cv.folds = 5, verbose = FALSE)
```

Then I use 5-fold cross-validation within the *gbm* call to find the optimal number of trees (best.iter).

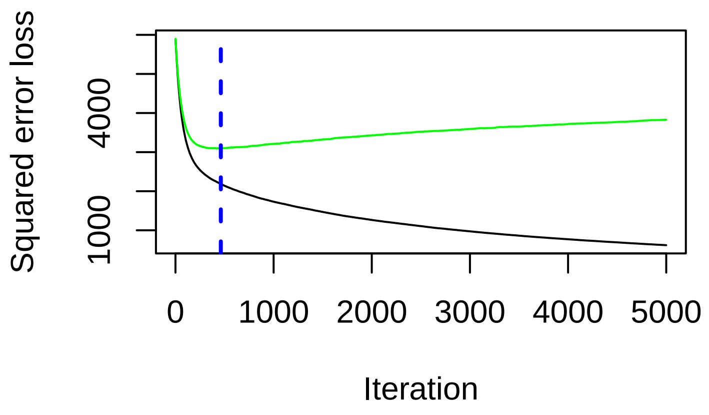

Figure 6: Number of Iterations vs Squared Error Loss

As the plot shows, the optimal ensemble size is **462** trees (blue dashed line), beyond which further trees offer diminishing returns. As a matter of fact, the black line represents the "training loss" (that in our case is the entire data set) that decreases monotonically (as expected) as the number of iterations increases. The green line is the cross-validation error, averaged over the *cv.folds* splits. It provides an estimate of the model's generalization error. Initially, it decreases as more trees are added, but after a certain point, it starts to increase (classic sign of overfitting).

I then extract the relative influence of each predictor that can be represented as follows:

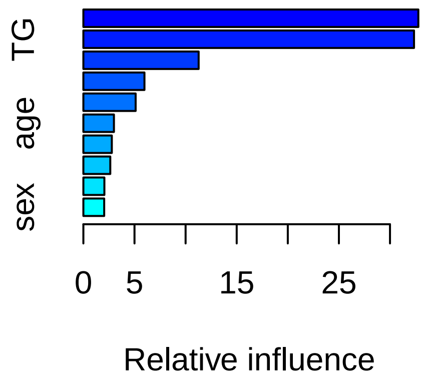

Figure 7: Predicotors' Importnace - Boosting

```
    var   rel.inf
BMI BMI 32.774276
TG   TG 32.351409
BP   BP 11.279589
GC   GC  5.977057
HDL HDL  5.113823
age age  2.990358
TC   TC  2.785554
LDL LDL  2.633750
TCH TCH  2.055551
sex sex  2.038633
```

The relative-influence plot highlights BMI and TG, largely in agreement with the random forest.

Cross-validated comparison of MSEs

So far I explore the 3 models using the full data set in order to understand how predictions are being made. Now, to compare methods fairly on their predictive performance, I run a 10-fold cross-validation loop, re-tuning the tree size and boosting iterations within each training fold. The cross-validated models will allow me to assess how well they generalize.

At first I create the folds (k=10) and 3 empty arrays for storing the MSE values for each model.

```r
set.seed(1)
folds <- createFolds(df_diab$progr, k = 10)

mse.tree <- mse.rf <- mse.boost <- numeric(length(folds))
```

Then, for each fold, I compute the train/test split and then on the training set: - repeat the CV-prune for the regression tree; - redo the tuning of the random forest hyperparameters; - redo the cross-validation during boosting for finding the optimal number of trees.

```r
for (i in seq_along(folds)) {
    test.idx <- folds[[i]]
    train.idx <- setdiff(seq_len(nrow(df_diab)), test.idx)

    train <- df_diab[train.idx, ]
    test <- df_diab[test.idx, ]

    # a) tree: repeat CV-prune on train
    tr <- tree(progr ~ ., data = train)
    cv.tr <- cv.tree(tr, FUN = prune.tree)
    sz <- cv.tr$size[which.min(cv.tr$dev)]
    tr.pr <- prune.tree(tr, best = sz)
    pred.tr <- predict(tr.pr, newdata = test)
    mse.tree[i] <- mean((pred.tr - test$progr)^2)

    # b) random forest: redo tuning on train
    rf <- randomForest(progr ~ ., data = train, mtry = best.m,
        ntree = 500)
    pred.rf <- predict(rf, newdata = test)
    mse.rf[i] <- mean((pred.rf - test$progr)^2)

    # c) boosting: redo CV on train
    bm <- gbm(progr ~ ., data = train, distribution = "gaussian",
        n.trees = 5000, interaction.depth = 3, shrinkage = 0.01,
        cv.folds = 5, verbose = FALSE)
    bi <- gbm.perf(bm, method = "cv", plot.it = FALSE)
    pred.bo <- predict(bm, newdata = test, n.trees = bi)
    mse.boost[i] <- mean((pred.bo - test$progr)^2)
}
```

I then collect the cross-validation results divided for the three models for the performance evaluation as such:

```
        Method   CV_MSE
1         Tree 4152.010
2 RandomForest 3192.186
3     Boosting 3091.543
```

The table shows that:

- **Single regression tree** shows the highest average MSE, reflecting its high variance;
- **Random forest** reduces MSE markedly, thanks to variance stabilization from ensembling;
- **Boosting** typically matches or slightly outperforms the forest, capitalizing on bias reduction via sequential fitting.

Thus, while all three methods are flexible, boosting provides the best predictive accuracy on this diabetes progression task.

Conculsions

In this project, I applied and compared three tree-based methods to model diabetes progression: a pruned regression tree, random forest, and gradient boosting. The pruned tree provided simple and interpretable decision rules but showed limited predictive accuracy. Random forest improved performance by reducing variance through ensembling, while gradient boosting outperformed both by sequentially minimizing error and capturing complex interactions. Cross-validation confirmed that boosting yielded the lowest prediction error, making it the most accurate model overall. Nonetheless, each method offers value—boosting for performance, and pruned trees for interpretability in clinical contexts.