Statistical Learning, Homework #2

Jacopo Manenti, id: 247279

2024-05-04

1 Introduction and Data Exploration

In this study, we investigate the prediction of prostate-specific antigen (PSA) value, measured in ng/ml on a logarithmic scale (lpsa), a protein primarily produced by cells within the prostate gland. Elevated levels of PSA may indicate the presence of cancer or other prostate-related conditions. To predict it, we will assess and fit three distinct models: pruned decision tree, random forest, and boosted decision tree. Subsequently, we will evaulate their performance in the final stage and opt for the model exhibiting the lowest test error.

This research is conducted using clinical measurements collected from 97 men scheduled to undergo radical prostatectomy, with the dataset stored in "prostate.csv". We commence by identifying and addressing any missing data and factorizing **svi** and **gleason** predictors. It's important to note that variables such as **lcavol**, **lweight**, and **lbph** are logarithmically transformed.

```
[1] "Number of na valuse:  0"
```

	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	lpsa
1	-0.5798185	2.769459	50	-1.386294	0	-1.386294	6	0	-0.4307829
2	-0.9942523	3.319626	58	-1.386294	0	-1.386294	6	0	-0.1625189
3	-0.5108256	2.691243	74	-1.386294	0	-1.386294	7	20	-0.1625189
4	-1.2039728	3.282789	58	-1.386294	0	-1.386294	6	0	-0.1625189

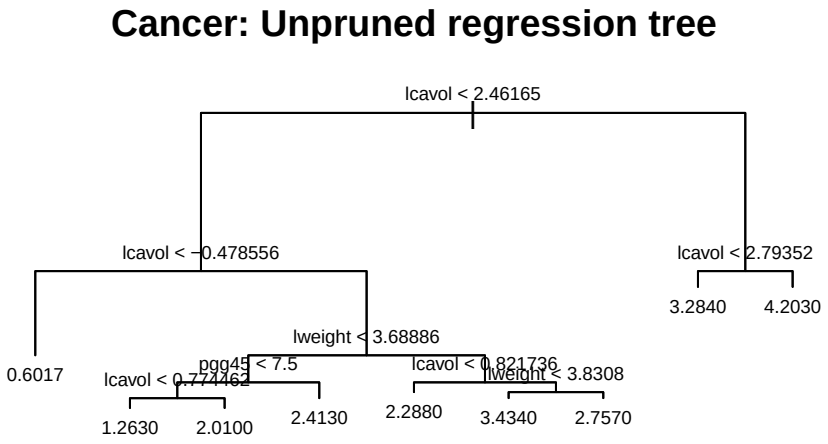
2 Fit a decision tree

The first model we are going to see is a fully grown decision tree using the entire dataset. These models are straightforward and valuable for interpretation. As illustrated in the figure below, they comprise a series of splitting rules starting from the top of the tree and progressing to the bottom. By visualizing the summary, we can extract useful insights.

```r
# fit a regression tree
tree.df <- tree(lpsa ~ ., data = cancer)
summary(tree.df)
```

```
Regression tree:
tree(formula = lpsa ~ ., data = cancer)
Variables actually used in tree construction:
[1] "lcavol"  "lweight" "pgg45"
Number of terminal nodes:  9
Residual mean deviance:  0.4119 = 36.24 / 88
Distribution of residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-1.499000 -0.488000  0.003621  0.000000  0.481200  1.380000
```

The summary outlines that the tree has 9 terminal nodes where predictions are taken, and it employed just 3 out of 8 variables: **lcavol** is the cancer's volume (cm3), **lweight** is the prostate weight (in g), **pgg45** is the percentage of Gleason score 4 or 5 over the entire patient's clinical history. Additionally, the summary provide us with the residual mean deviance, a gauge of model fitting, which stands at 0.41. For a better understanding, we can plot the tree and visualize it.



Decision tree indicates higher prostate antigen levels for larger cancer volumes ($lcavol >= e^{2.46}$), higher Gleason scores ($pgg45 >= 7.5$), and lower prostate weights ($lweight < e^{3.83}$). For example, for a patient with a cancer volume between $e^{-0.47}$ and $e^{2.46}$ and a prostate weight $>= e^{3.83}$, the predicted antigen level is $e^{2.75}$, or 15.64 ng/ml. Generally, PSA levels below 4.0 ng/ml are considered normal for men. In this case, the test would suggest the presence of prostate condition or cancer.

However, decision trees suffer from high variance, particularly with frequent splits like ours (9 splits), which can result in overfitting and poor test set performance. Pruning the tree after it has fully grown is a common solution to determine the optimal complexity. We use cross-validation to find the best number of terminal nodes, selecting the one with the lowest CV error. Additionally, we visualize this process by plotting the CV error against `size`.

```
# Set seed for reproducible results
set.seed(1)
# performing cv on the full grown tree
cv.cancer <- cv.tree(object=tree.df)
# retrive the best size
opt.size <- cv.cancer$size[which.min(cv.cancer$dev)]
# prune the tree
prune_cancer <- prune.tree(tree.df, best=opt.size)
summary(prune_cancer)
```

```
Regression tree:
snip.tree(tree = tree.df, nodes = c(11L, 3L, 10L))
Variables actually used in tree construction:
[1] "lcavol"  "lweight"
Number of terminal nodes:  4
Residual mean deviance:  0.5625 = 52.31 / 93
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.66200 -0.54020 -0.01154  0.00000  0.44560  1.81700
```

Cross-validation favored a simpler tree, with 4 terminal nodes. We also observe a rise in residual mean deviance from 0.41 to 0.56, suggesting a worsened model fit. This is due to raw decision trees' tendency to overfit the dataset. Pruning aims to balance bias and variance, potentially enhancing model performance. Lastly, only *lcavol* and *lweight* were significant this time. This leads to an easier intrepretation of the model.

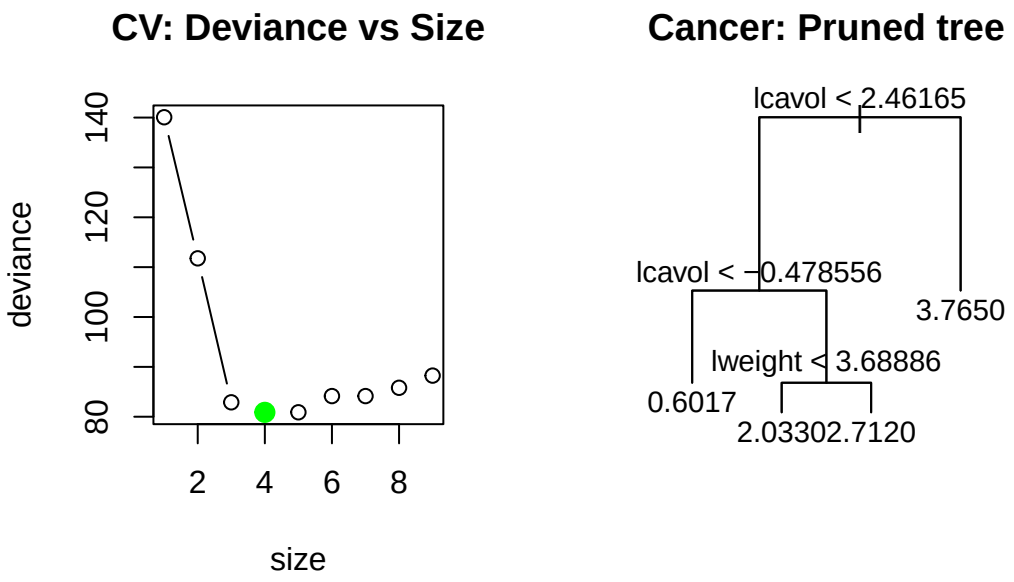


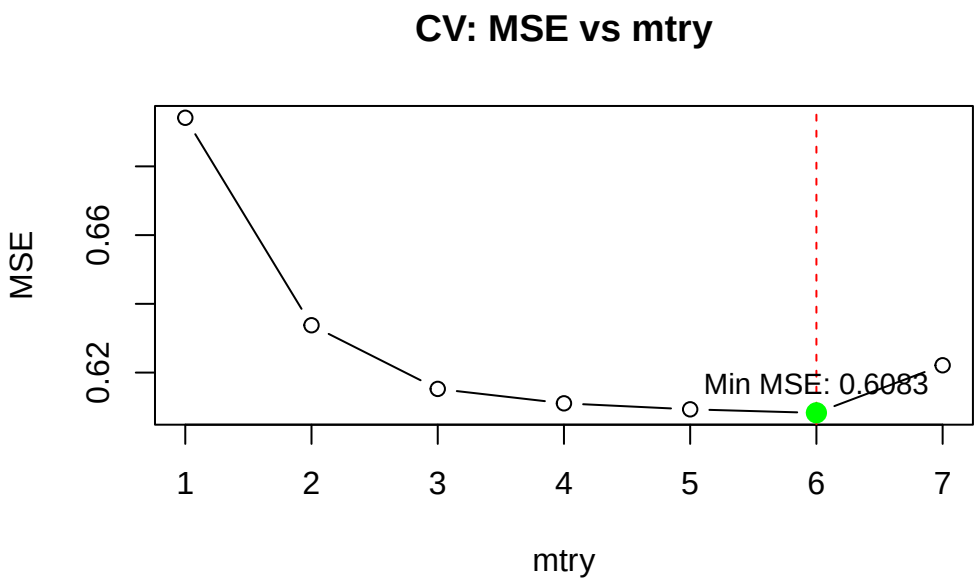
Figure 1: Left: cross-validation for selecting the size; Right: pruned tree distinguishes patients by cancer volume: above $e^{2.46}$ cm^3, below $e^{-0.47}$ cm^3, and within them. The latter group, by futher splitted according to prostate weight: above $e^{3.68}$ and below. Predicted PSA levels for these groups are $e^{3.7650} = 43.16$, $e^{0.6017} = 1.82$, $e^{2.7120} = 15.02$, and $e^{2.0330} = 7.63$.

3 Fit a random forest

The second model we fit is random forest, an ensemble method that leverages the concept of averaging **decorrelated** predictions from diverse decision trees to reduce variance and enhance prediction accuracy. Decorrelation implies carefully selecting the number of predictors to consider at each split. In our scenario, this ranges from 1 to 7, as using all 8 predictors would essentially revert to bagging. The rule of thumb in regression is to use $p/3$ variables, which in our case would be ≈ 2. However, a better approach involves treating it (**mtry**) as a tuning parameter and finding its optimal value through cross-validation, selecting the one associated with the lowest CV error. The algorithm uses 5 folds and 500 trees.

```
set.seed(1)
# create 5 folds
rf_folds = createFolds(cancer$lpsa, k = 5, list = FALSE)
# initialize hyperparameter sequence and mse matrix
```

```r
v.mrty <- 1:7
rf.mse <- matrix(0, nrow = 5, ncol = 7)
# loop
for(i in 1:5){
  # split dataset into training set and test set
  cancer_idx = which(rf_folds == i)
  cancer_trn = cancer[-cancer_idx,]
  cancer_tst = cancer[cancer_idx,]
  # loop over the hyperparameter values for each fold
  for(j in v.mrty){
    rf.cancer <- randomForest(lpsa ~ ., data=cancer_trn, mtry=j,
 ↪  importance=TRUE)
    pred <- predict(rf.cancer, newdata=cancer_tst)
    rf.mse[i, j]<- compute_mse(pred, cancer_tst$lpsa)
  }
}
# compute for each hyperparameter value the cv error
cv.mse.rf <- apply(rf.mse, FUN=mean, MARGIN = 2)
# return the optimal value of the hyperparameter and the lowest cv-error
opt.mrty <-which.min(cv.mse.rf)
min_val_rf <- round(min(cv.mse.rf),4)
```



The figure shows the CV error based on the number of variables used for each split, with

a vertical line indicating the optimal **mtry** value. These results suggest selecting a random forest generated by utilizing 500 trees and 6 variables for each split.

After optimizing the model, we refit it to analyze each predictor's importance. However, the interpretability is limited compared to a single decision tree, as visualizing the entire forest is impractical.

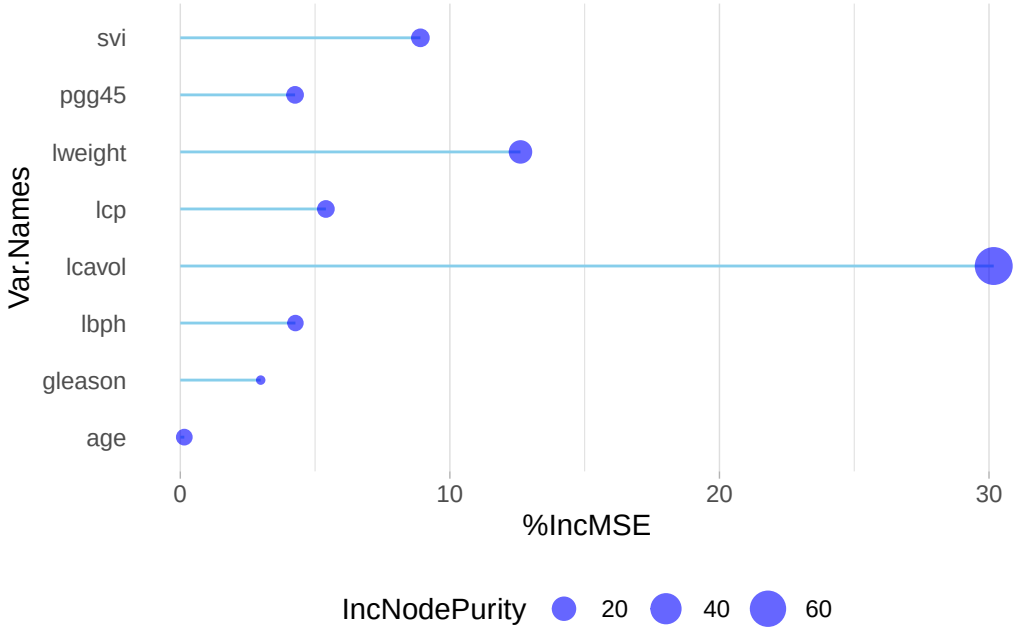


Figure 2: The straight line represents the increase in prediction error (MSE), while the circle indicates the node purity (both when the predictor is included and excluded). The results align with previous decision trees: both **lcavol** and **lweight** are identified as significant predictors of the protein's level. However, an additional predictor, previously overlooked, such as **svi** (seminal vesicle invasion), has now emerged as influential. For instance, removing svi leads to an approximate 9% increase in MSE and a 10% increase in node impurity.

The findings in Figure 2 are logical, as these factors significantly influence PSA production and are crucial indicators of prostatic conditions, including cancer. Thus, including them in the PSA prediction model enhances accuracy.

4 Fit a boosted regression tree

Finally, we apply boosting regression trees, which construct trees sequentially using information from previous trees. Specifically, we use regression gradient boosting (**BRT**). Similar to random forest, BRT requires preliminary hyperparameter tuning, including selecting optimal values for the learning rate, tree complexity, and number of trees. We determine the optimal number of trees through cross-validation [1], keeping the first parameter as default and aiming for simplicity with only a few splits, typically resulting in three or four regions.

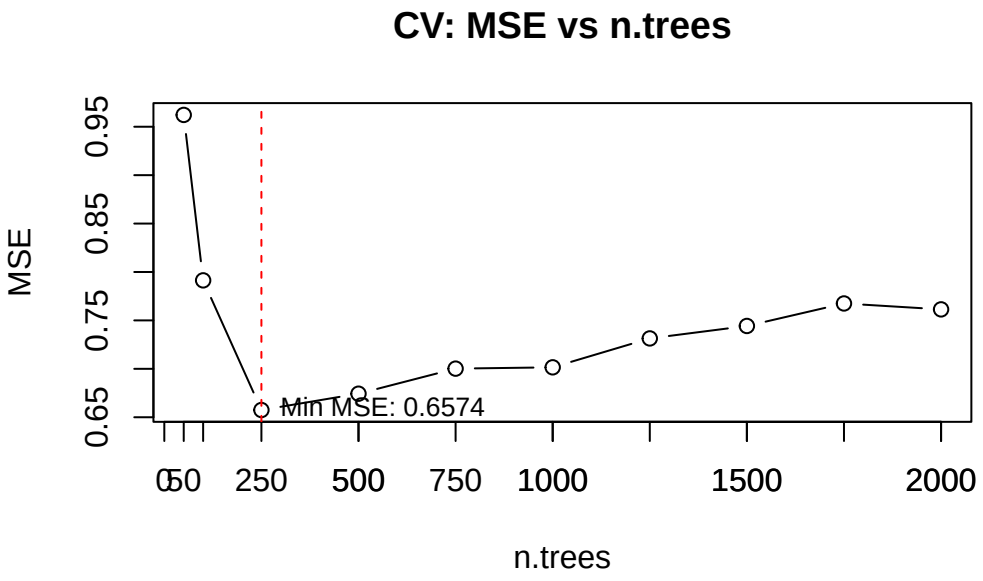


Figure 3: shows the CV error based on the number of trees used, with a vertical line indicating the optimal **ntrees** value. These results suggest selecting a boosting decision tree utilizing 250 trees, learning rate of 0.01, complexity of 3.

As done with random forest, we refit the model with the optimal configuration and analyze each predictor's importance.

```
set.seed(1)
# fit the optimal boosting model
boosted <- gbm( lpsa~ ., data=cancer,
 ↪  distribution="gaussian",n.trees=opt.ntrees, interaction.depth=4)
knitr::kable(summary(boosted, plotit = FALSE, order = TRUE))
```

[1] No boosting cross-validation code reported, as requires little changes from loop in Section 3 in the fitting function (gbm) and hyperparameter sequence from 50 to 2000. See GitHub for more details (lines 250-284).

Table 1: Relative influence for a boostied decision tree with n.trees = 250

	var	rel.inf
lcavol	lcavol	29.725393
lweight	lweight	24.562420
lcp	lcp	11.988218
age	age	9.098943
pgg45	pgg45	8.699344
lbph	lbph	8.185179
svi	svi	6.003152
gleason	gleason	1.737351

The table ranks variables by their *relative influence*, indicating their importance in the model. Both **lcavol** and **lweight** are key features, contributing 29% and 24% to the loss function reduction, respectively. These results align with previous findings from random forest. However, **age** is found to be more important than **pgg45** and **svi**. This conclusion seems reasonable, as age influences the avarege level of PSA production in men, which typically decreases with age. Therefore, if two individuals have the same level of PSA but different ages, the older individual may face a higher risk.

5 Performance comparison

Once we have assessed the optimal configuration for each of our three models, we may consider proceeding with selecting the best model in terms of prediction's performance. However, these models have been optimized using the entire data set. Therefore, model selection would use the same data to tune model parameters and evaluate model performance, yielding an overly-optimistic score. To mitigate this issue, we employ nested cross-validation (CV), which involves a series of train/validation/test set partitions. This approach an inner loop for model tuning and outer loop for computing the generalization error.

In our scenario, both the inner and outer loops utilize 5-fold cross-validation. We will optimize the model based on specific hyperparameters: **mtry** for random forests, **n.trees** for boosting (while retaining default values for other tuning parameters), and **k** as the complexity parameter for pruning. [2]

[2]We have reported nested-cv only for random forest, due to space constraints. The procedure utilized for boosting and pruning was the same. "inner_loop" function performs the inner loop of nested-cv. See GitHub for mode details (lines 306-381).

```r
inner_loop <- function(method, cancer_trn, cancer_tst, tune_grid, hyper,
 ↪  matr, i){
  model <- train(lpsa ~ ., data = cancer_trn, method = method, trControl =
 ↪  cv_5,
                 tuneGrid = tune_grid)
  pred <- predict(model, newdata = cancer_tst)
  error <- mean((cancer_tst$lpsa - pred)^2)
  return(list(model = model, error = error))
}
```

```r
set.seed(1)
# split the data into 5 folds
cancer_folds = createFolds(cancer$lpsa, k = 5, list = FALSE)
# set the traincontrol
cv_5 = trainControl(method = "cv", number = 5,search = "grid")
# collect the predict score got from CV.
predicted_cv <-  matrix(0, nrow=5, ncol = 3,dimnames = list(NULL, c("rf",
 ↪  "gbm", "prun")))
hyperparam <- matrix(0, nrow=5, ncol = 3,dimnames = list(NULL, c("mtry",
 ↪  "n.trees", "k")))
# set the grid for hyperparamenters' values
tune_grid_rf <- expand.grid(mtry = 1:7)

# outer loop
for (i in 1:5){
  cancer_idx <- which(cancer_folds == i)
  cancer_trn <- cancer[-cancer_idx,]
  cancer_tst <- cancer[cancer_idx,]
  # Random Forest
  result <- inner_loop("rf", cancer_trn, cancer_tst, tune_grid_rf)
  hyperparam[i,1] <- result$model$bestTune$mtry
  predicted_cv[i,1] <- result$error
}
```

At the conclusion, we have derived from Table 2 the following results for model evaluation:

- Random Forest: a random forest employing utilizing 500 trees and having a cv-test error of 0.6485
- Boosting: a boosted tree trained with a learning rate of 0.01, complexity of 3, having a cv-test error of 0.6517

Table 2: Performance evaluation

(a) best mtry at iteration i with the corresponding cv error.

mtry	kth-cv-error
5	0.604
2	0.921
5	0.561
4	0.487
7	0.670

(b) best n.trees at iteration i with the corresponding cv error.

n.trees	kth-cv-error
367	0.631
472	0.677
367	0.655
367	0.672
367	0.624

(c) best k at iteration i with the corresponding cv error

k	kth-cv-error
10	1.136
10	1.136
10	1.136
10	1.136
10	1.136

(d) mean of cv-error for each method

	cv_error
rf	0.648
gbm	0.652
prun	1.136

- Cost-Complexity Decision Tree: utilizing a complexity parameter set at 10, which produces a pruned tree with 3 regions that utilizes only lcavol predictor. The cv-test error is 1.1355.

6 Conclusions

In this study, we examined three methods for predicting prostate antigen values using clinical measurements. We started with a decision tree, then transitioned to a pruned tree to address overfitting. Additionally, we explored two ensemble methods to reduce the variance of basic decision trees. Due to the small dataset size, nested cross-validation helped select the best hyperparameters and provided an unbiased estimate of model performance. Comparing test errors, random forest showed the lowest, leading to its selection for better generalization performance. Although it may be the optimal choice, its interpretability is lower than a single decision tree, which could be critical in medical applications. Furthermore, another limitation of the analysis is related to the fact that we only tuned specific parameters of random forest and boosting, leaving some as default (e.g., learning rate in boosting), which could have influenced the model selection differently.