University of Applied Sciences and Arts Northwestern Switzerland
School of Business

DS-06 Lab: Regression Trees

Dr. Gwendolin Wilke

Task 1: Grow a Regression Tree on the `Hitters` Data to Predict `Salary`

- Load and explore the `Hitters` Data from the `ISLR` Library.
- It holds data from the Major Baseball Leagues from the 1986 and 1987 seasons.
- Fit a regression tree using `tree()` from the library `tree`.

Preliminaries

1. Load the `ISLR` library.
2. Read the data dictionary.

```
?Hitters
```

3. Make a copy of your data so that the original stays unchanged.

```
Ht <- Hitters
```

4. Explore the `Hitters` dataset a bit using the usual suspects, but **do not spend too much time on this step!**

```
View(Ht), str(Ht), summary(Ht), hist(Ht), pairs(Ht), etc.
```

5. Remove the NAs.

```
Ht_nonas <- na.omit(Ht)
```

6. Set a seed and split the data in training and test set using a 30% - 70% split.

```
…
Ht_nonas.train <- …
Ht_nonas.test <- …
```

Grow and Evaluate a Regression Tree in 2 Variables

1. Load the `tree` library.

2. Use `tree()` to predict `Salary` from `Years` and `Hits`.

```
tree.HitsYears <- tree(Salary ~ Years + Hits, data = Ht_nonas.train)
```

3. Plot the tree and inspect the plot. Which of the variables is more important?

```
plot(tree.HitsYears)
text(tree.HitsYears, cex=0.75) # cex: set character size to 0.75
```

4. Compare the plot with the textual description.

```
tree.HitsYears
```

`node):`	node number
`split:`	split criterion, e.g. `League: A`, or `Years < 4.5`
`n:`	number of observations in that branch
`dev:`	**"deviance" (RSS)** of the node (the smaller the better)
`yval:`	prediction for the branch (mean value of all observations in this node)
`*`	indicates a terminal node

5. Plot the input regions with predicted values.

```
plot(Ht_nonas.train$Years, Ht_nonas.train$Hits, col='steelblue', pch=20, xlab="Years",ylab="Hits"))
partition.tree(tree.HitsYears, ordvars=c("Years","Hits"),add=TRUE,cex=1)
```

6. Calculate training and test error.

Use `tree.control()` to Change the Default Stop Criteria

1. Use `tree.control()` to predict `Salary` from `Years` and `Hits`.

```
tree.HitsYears.control = tree.control(nobs = dim(Ht_nonas.train)[1], mincut=5, minsize = 10, mindev = 0.01)
tree.HitsYearsCon <- tree(Salary ~ Years + Hits, data = Ht_nonas.train, control = tree.HitsYears.control)
```

2. Play with the parameters `mincut, minsize, mindev.` Compare the training and test errors.

   `nobs`: The number of observations in the training set.
   `mincut`: The minimum number of observations to include in a child node. **Default: 5.**
   `minsize`: The smallest allowed node size. **Default: 10**.
   `mindev`: The within-node deviance (RSS) must be at least this times that of the root node for the node to be split. **Default: 0.01**.

   Particularly, fit and evaluate a tree that fits the data perfectly (**saturated tree**) by using `mincut = 1, minsize = 2` and `mindev = 0.`

3. Plot and evaluate your trees.

Grow and Evaluate a Regression Tree in *all* Variables

1. Use `tree.control()` to predict `Salary` from `all input varaibles.`

```
tree.all.control = tree.control(nobs = dim(Ht_nonas.train)[1], mincut=5, minsize = 10, mindev = 0.01)
tree.all <- tree(Salary ~ ., data = Ht_nonas.train, control=tree.all.control)
```

2. Plot and evaluate your trees.

Task 2: Grow a Pruned Regression Tree

- Use the function `cv.tree()` and `prune.tree()` from the `tree` library to apply cost complexity pruning.

Grow a Pruned Tree

1. Grow the saturated tree (`mincut=1, minsize = 2, mindev = 0`) using *all* input variables .

```
tree.full <- …
```

2. Use `cv.tree()` to run cross validation to find the best pruning parameter alpha.

```
set.seed (1)
cv.tree.full = cv.tree(tree.full)
              # returns the "deviance" (RSS) as a function of the cost-complexity parameter alpha.
mindev.idx <- which(cv.tree.full$dev == min(cv.tree.full$dev))
              # returns the index that holds smallest alpha
best.size <- min(cv.tree.full$size[mindev.idx])
              # returns the tree size of the tree with smallest alpha
```

3. Prune the tree using `prune.tree()` based on the best alpha.

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

4. Plot your tree and compare it to the corresponding tree in Task 1 (the tree that used all input variables).

5. Evaluate your tree on the training and test set, and compare the results to the corresponding results from Task 1

Task 3: Apply Bagging, Random Forests and Boosting

- Use the function `randomForest()` from the `randomForest` library to grow a bagged ensemble and a random forest.
- Use the `gbm()` from library `gbm` to grow a boosted tree.

We will do this Task together in class.