Statistical learning and prediction

Tree and ensemble methods

Tree-growing algorithms
(e.g., CART)

- A decision tree divides $\{X_1, X_2, \ldots, X_P\}$ into a set of distinct regions $\{R_1, R_2, \ldots, R_J\}$, corresponding to the terminal nodes of the tree
- Regions should be as similar (pure) as possible with respect to the outcome Y
  - That is: var(Y) should be as small as possible within each region
- Finding regions $\{R_1, R_2, \ldots, R_J\}$ that provide a global optimum is computationally difficult (or infeasible)
- Most tree-growing algorithms turn it into a feasible task by employing:
  - Rectangular regions (i.e., each split defined by a single variable)
  - Only two-way splits
  - Greedy search (*current* best split is selected in each step, no looking ahead)
- Once tree is constructed, the mean of the outcome variable Y in region R_j gives $\hat{Y}$ for new observations (sometimes, median may be preferred)

Splitting criteria

At each step, we seek variable *j* and value *s* that define
$R_{\text{left}}(j,s) = \{ X \mid X_j < s \}$ and $R_{\text{right}}(j,s) = \{ X \mid X_j \geq s \}$,
which minimize the sum of the **loss function** in both regions

Different loss functions can be used, e.g.,

Regression trees:

Absolute loss: $\sum_{i:x_i \in R_m} | y_i - \hat{y}_{R_m} |$

Squared error loss: $\sum_{i:x_i \in R_m} (y_i - \bar{y}_{R_m})^2$

Classification trees:

Classification error: $E = 1 - \max_k (\hat{p}_{mk})$

Gini index: $G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$

Cross-entropy: $D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$

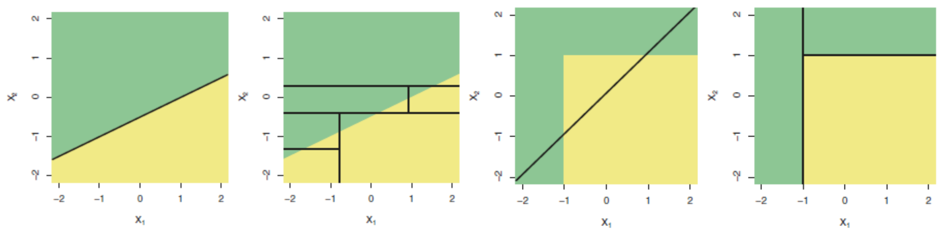
Squared error loss and Gini index are most often used

Both minimize variance among observations within every terminal node

Bias-variance trade-off

Trees do not provide optimal predictive accuracy, because:

- Small trees have low variance, but (likely) high bias
- Large trees have low bias, but high variance



Instability

Trees have high variance, a.k.a. instability:

- A split may come out different due to minor sampling variations. Further splits are conditional on earlier splits -> Minor sampling variations may greatly affect the tree structure.
- Looks very ugly if you look at the tree structure (e.g., add or remove only a few observations to the data and the whole tree may be different).
- But note, very different tree structure may provide very similar predictions, due to correlated covariates. Instability may look worse than it is.

Balancing bias and variance

<u>Solution 1</u>: Pruning

tree should not be too small, nor too large

How?

1. grow full tree untill impurity cannot be further reduced
2. cut of branches by selecting the tree that minimizes the penalized loss function:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_j - \hat{y}_{R_m})^2 + \alpha |T|$$

where T equals the # of terminal nodes; α should be determined by k-fold CV

Variable selection bias

CART performs exhaustive search: for every split, all possible splitting values are considered

-> biased variable selection: given two (or more) variables who are equally predictive of the outcome, variable with largest number of possible splitting values has higher probability of being selected for splitting

Solution: First select splitting variable, then select splitting value

-> unbiased variable selection: variables with equal predictive power have equal probability of being selected

Unbiased recursive partitioning

Conditional inference trees: Select splitting variable based on statistical tests

Separates variable and cutpoint selection

-> no selection bias towards variables with larger number of values

Solution 2 for balancing bias and variance: Yields natural stopping criterion: Significance level

Conditional inference trees

Statistical testing to select splitting variable:

$H0_j$: variable X_j and response Y are independent in the population

HA_j: variable X_j and response Y are dependent in the population

Thus, for each variable X_j a *p* value under H0 can be calculated, based on the observations in the current node

Variable X_j with lowest *p* value is selected for splitting the observations into two daughter nodes

If all *p* values in current node > pre-specified value α, no further splitting is performed

After selecting splitting variable, select cutpoint as with CART (i.e., take splitting value that yields minimum value of sum of loss functions in the two daugthernodes)

Implemented in R in ctree() function of package 'partykit'

Exercise: Variable selection bias

a) Set the random seed and generate 200 observations from independent variables x1, x2 and e (you are free to choose the shape and parameters of the distribution yourself).

b) Create two datasets consisting of x1, x2 and y:
   - one where y = e (the 'independent' dataset), and
   - one where y = x2 + e (the 'dependent' dataset).

c) Using function tree() from the tree library, fit a regression tree using x1 and x2 to predict y, using each dataset.

   Inspect the results using the plot() and text() functions. Which variable is most often selected for splitting? Is that what you would expect?

Exercise: Variable selection bias

d) Find the optimal number of nodes using the cv.tree() function. Plot the results. Prune the trees using function prune.tree(). Are there any splits left?

e) Use the ctree() function from the **partykit** package to fit a conditional inference tree to the independent and dependent data. Also plot the resulting conditional inference tree.

f) Compare the results you obtained in parts a, b, c and d.

Exercise: Dataset

- Download the file MASQ.txt from github.
- Read it into R using: read.table("MASQ.txt", header=TRUE)
- It contains subscale scores of the Mood and Anxiety Symptom Questionnaire:
  - Anhedonic Depression (22 items):
  - Anxious Arousal items (17 items):
  - General Distress Depression (12 items)
  - General Distress Anxiety (11 items):
  - General Distress Mixed (15 items):
- It also contains:
  - D_TOT (the total number of diagnoses present)
  - D_DEPDYS (whether a depressive or dysthymic disorder is present; make sure it is coded as a factor)
  - leeftijd (age)
  - geslacht (gender)
- Separate data into 80% training and 20% test observations

Exercise: Regression tree

a) Fit a tree with function tree() from package **tree**:
   - Predict D_TOT using all other variables as predictors, except D_DEPDYS
   - Add min.dev = .002 as an argument (type ?tree.control to see what this does)
   - Inspect the tree using the print() function; plot the tree using the plot() and text() functions

b) Prune the tree:
   - Set the random seed and apply the cv.tree() function to obtain the optimal number of terminal nodes, inspect the CV results using plot(). What do the values on the x-axis and upper and lower y-axes represent?
   - Prune the tree using the prune.tree() function
   - Inspect the pruned tree using print(), plot() and text()

c) Compare performance:
   - Using the predict() function, compare the training and test MSEs of the unpruned and pruned trees

d) Note that D_TOT is a count variable. If you would model such a response with a linear regression model, which assumption would be violated? How would we fix that? Did we violate any assumptions by fitting a regression tree to a count response?

e) Fit a conditional inference tree to predict D_TOT, using the ctree() function from package **partykit**. Plot the results, and compare with the tree fitted above.

Tree ensembles

Single trees

- Small trees have low variance, but high bias
- Large trees have low bias, but high variance

Tree ensembles

- By combining the fitted (predicted) values from a large number of single trees, variance is reduced
- A price is paid in interpretability: instead of a single tree, we get a large number of trees

Ensemble learning

- Predictions of a large number of *weak learners* (e.g., single trees) are combined to create a *strong learner* (e.g., ensemble of trees)
  - Other learners than trees may be ensembled, too
- A weak learner is a method that has at least some predictive ability, that is at least better than random guessing
- Predictive accuracy of the full ensemble is always equal or better than that of any of its constituent members

15

Ensemble learning

- <u>Ensembling works well for unstable learning algorithms</u>
  (algorithms whose output undergoes major changes in response to small changes in the training data)
  - Unstable learning algorithms (generally, relatively): e.g., decision trees, neural networks, rule learning algorithms, OLS regression
  - Stable learning algorithms (generally, relatively): e.g., penalized linear regression, nearest neighbour, linear threshold algorithms
- Popular tree ensemble methods :
  - Bootstrap aggregation (bagging)
  - Random forests
  - Boosting

16

Bagging

Sampling rows of X decorrelates trees

A bagged tree ensemble is construct y

1. Taking bootstrap samples from the training data
2. Fitting a tree on each of the bootstrap samples
3. Combining the trees into an ensemble

17

Random forests

Sampling rows of X

A random forest is constructed by

1. Taking bootstrap samples from the training dataset
2. Fitting a tree on each of the bootstrap samples, using a random subset of predictor variables for each split
3. Combining the trees into an ensemble

Sampling columns of X

Double decorrelation of trees and reduction of variance

18

Tuning parameters (bagging and random forests)

- ntrees (# of trees / samples)
  Sensible default: 500 (bagged and RF ensembles hardly overfit)
- nodesize & maxnodes (treesize)
  Sensible defaults:
  - nodesize: 1 for classification, 5 for regression
  - maxnodes: no limit (ensembling trees reduces variance, so trees may as well be large, with low bias)
- mtry (# of predictor vars used for split selection)
  Sensible default: mtry = p for bagging, mtry = srtq(p) for RF
- replace (sampling with or without replacement)
  Sensible default: Bootstrap sampling (replace = TRUE)
  Bootrapping increases likelihood of noise variables being selected. Subsampling (replace = FALSE) may perform better in terms of variable inclusion frequencies (De Bin et al., 2014).

Out-of-bag error estimation

When trees are grown on bootstrap or sub samples of the data, we do not need separate test data for error estimation:

- For each training observation *i*, predict *Y* using trees that were grown on samples not including observation *i* (out-of-bag)
- Then calculate as usual: $MSE = \frac{1}{n}\sum_{i=1}(y_i - \hat{y}_i)^2$

Interpretation

- The price for the better predictive accuracy of ensembles, compared to single trees, is interpretability
- Variable importances aid in interpretation of the model
- For bagging and random forests, relative importance of variable X_j is assessed as follows:
  - For every tree in the ensemble:
    1) Predictive accuracy for OOB observations is computed
    2) Predictive accuracy for OOB observations, with values of Xj randomly permuted are computed
  - Difference between accuracies under 1) and 2), averaged over all trees in the ensemble, is used to calculate importance of X_j

Boosting

- The variance of the ensemble is reduced more when trees (and predictions) are less correlated
  - Bagged ensembles decorrelate through sampling of observations
  - Random forests doubly decorrelate through sampling observations as well as variables
  - Boosting decorrelates trees through growing them sequentially: each tree is grown on the residuals of earlier trees

Pseudo response variable in boosting

In every step b of the boosting algorithm, a tree is fitted on pseudo response y_b^*, instead of the original response y:

$$y_b^* = y - \sum_{i=1}^{b-1} \nu\, f_i(\mathrm{x})$$

where ν is the learning rate

Tuning parameters (boosting)

Boosting can overfit, have to determine optimal parameter values by CV!

- interaction.depth (tree depth):
  - Should be relatively low (boosting works best with weak learners)
  - Rule of thumb: interaction.depth = 4
- n.trees (# of trees):
  - Rule of thumb: at least 1 / shrinkage
- shrinkage (learning rate):
  - At least 1 / n.trees
- bag.fraction (fraction of training observations randomly selected for each sample)
  - Bootstrap sampling (bag.fraction < 1) or no sampling (bag.fraction = 1)

24

Trees and ensembles: Concluding remarks

- With a single tree, we trade in some predictive accuracy for interpretability
- Depends on the context of the application if this is worth it, e.g.
  - Simple decision trees are pre-eminently suited as tools for human decision making (i.e., their use requires limited time and computational power)
  - Tree ensembles generally use all of the predictor variables: evaluation of all variables for new cases may be expensive / infeasible
- If we have lots of data and computation power available, tree ensembles provide very good predictive accuracy
- Boosting requires careful parameter tuning to perform well, random forest works well out-of-the-box

25

Exercise: Bagging

- Use the MASQ dataset use the same train and test sets as in the earlier exercises
- Fit a bagged ensemble to the D_TOT outcome using the `randomForest` package and function
- Make sure D_DEPDYS is not included as a predictor
- Make sure to set the random seed first
- For a bagged ensemble, set mtry = p
- Check `?randomForest` to figure out:
  - How many trees are generated by default?
  - Is bootstrap or subsampling employed by default? How can you change this?
  - Which arguments are used to control tree depth and what are the defaults?
- Use `importance()` and `varImpPlot()` to see which variables are most important for predicting the outcome
- Use `partialplot()` to obtain partial dependence plots for AD, GDD and geslacht
- Use `plot()` to see how the OOB error decreases as a function of the number of trees
- Use `predict()` to assess the train and test MSE

Exercise: Random forest

- Fit a random forest to D_TOT using the randomForest function
- For a random forest, set mtry = sqrt(p)
- Use `importance()` and `varImpPlot()` to see which variables are most important for predicting the outcome
- Use `partialplot()` to obtain partial dependence plots for AD, GDD and geslacht
- Use `plot()` to see how the OOB error decreases as a function of the number of trees
- Use `predict()` to assess the test MSE

Exercise: Boosting

- Fit a boosted ensemble to the D_TOT outcome using the gbm package and function
- Make sure to set the random seed first
- Check `?gbm` to figure out:
  - How many trees are generated by default?
  - Which argument is used to control the observation sampling strategy employed for fitting each tree? What is the default of this argument?
  - Which argument is used to control tree depth and what is the default?
  - Which argument is used to control the learning rate or shrinkage?
- Fit a boosted ensemble with a learning rate of 0.01, the default tree depth, and 1,000 trees
- Fit a second boosted ensemble with a learning rate of 0.1, the default tree depth, and 1,000 trees
- Use `gbm.perf()` with argument `oobag.curve=TRUE` to see how OOB error decreases as a function of the number of trees, for the two boosted ensembles
- Use `plot()` with argument `i.var` to obtain partial dependence plots for AD, GDD and geslacht, for the two boosted ensembles
- Use `summary()` to obtain variable importances, for the two boosted ensembles
- Use `predict()` to assess the test MSE, for the two boosted ensembles

Exercise: Parameter tuning through CV

With the following code we can estimate which parameter settings can be expected to yield the best predictive accuracy for the boosted ensemble:

```
library("caret")
grid <- expand.grid(shrinkage = c(.1, .01, .001),
                    n.trees = c(10, 100, 1000),
                    interaction.depth = 1:4,
                    n.minobsinnode = 10)
grid
set.seed(3)
gbmFit <- train(D_TOT ~ ., data = MASQ[train,],
                method = "gbm", tuneGrid = grid)
gbmFit
gbmFit$bestTune
```

If the best parameter values differ from those you used before to fit the boosted ensemble, refit the ensemble using the best parameter values and again assess test MSE

NB1: Parameter values for the `randomForest()` function can also be tuned using method = 'rf' with caret
NB2: See http://topepo.github.io/caret/train-models-by-tag.html# for all models that can be tuned using caret

Suggested further reading

Introduction to trees and random forests:

Strobl, C., Malley, J., & Tutz, G. (2009). An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological Methods*, *14*(4), 323.

Unbiased recursive partitioning:

Hothorn, T., Hornik, K., & Zeileis, A. (2006). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, *15*(3), 651-674.

Boosting and ensemble learning:

Bühlmann, P., & Hothorn, T. (2007). Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science 22*(4), 477-505.

Trees for multilevel data:

Fokkema, M., Smits, N., Zeileis, A. , Hothorn, T. & Kelderman, H. (2018). Detecting treatment-subgroup interactions in clustered data with generalized linear mixed-effects model trees. *Behavior Research Methods*, https://doi.org/10.3758/s13428-017-0971-x

GAMs:

An introduction to GAMs by simon wood (author of mgcv): https://people.maths.bris.ac.uk/~sw15190/talks/snw-Koln.pdf