QBUS6810: Statistical Learning and Data Mining

Lecture 11: Tree-Based Methods I

Semester 1, 2019

Discipline of Business Analytics, The University of Sydney Business School

Lecture 11: Tree-Based Methods I

1. Regression trees
2. Classification trees
3. Discussion

Classification and regression trees

- Tree-based methods for regression and classification partition the predictor space (all possible x values) into a set of regions (rectangles), and fit a very simple model in each region.
- For example, in the regression case we can predict Y in each region as the average y value of the training points falling in the region.
- In classification, we can predict Y in each region as the most frequent class of the training points falling in the region.

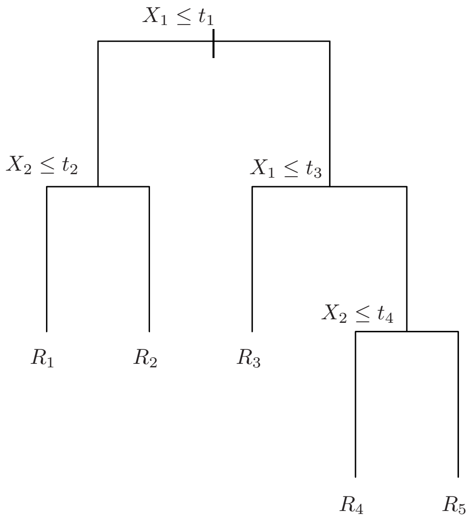
Classification and regression trees

- The partitions are based on successive splits of the predictor space ($\boldsymbol{x}$ values) into two non-overlapping subsets, enabling an interpretable visualisation of the prediction rule as a decision tree.

- We begin at the top of the tree with all $\boldsymbol{x}$ belonging to a single region. Each split is done by separating the $\boldsymbol{x}$ with smaller values of one of the predictors (say X_1) from the $\boldsymbol{x}$ with larger values of the same predictor. The split is indicated via two new branches further down the tree.

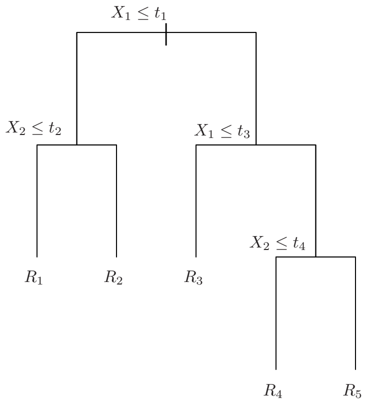
- The next few slides illustrate a tree constructed for two predictors, X_1 and X_2.

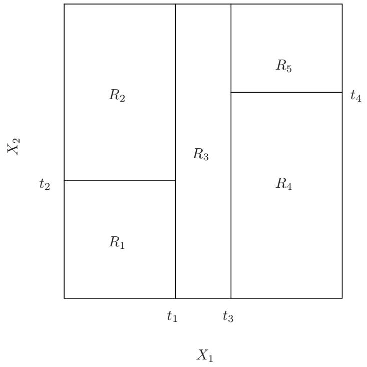
Example



regions $R_1, R_2, \ldots, R_5$ are known as **terminal nodes** or *leaves* of the tree

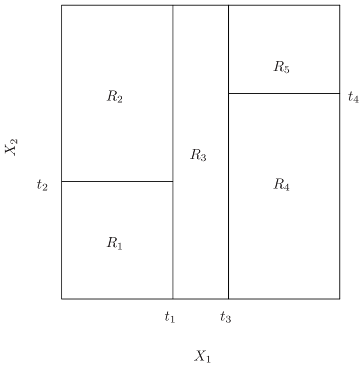
Example

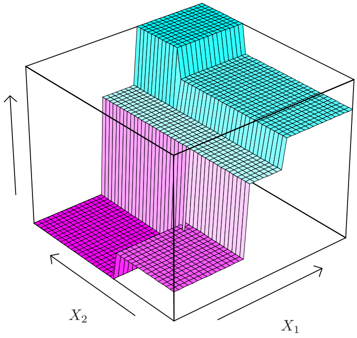




Left: a tree Right: corresponding partition

Example (regression tree)

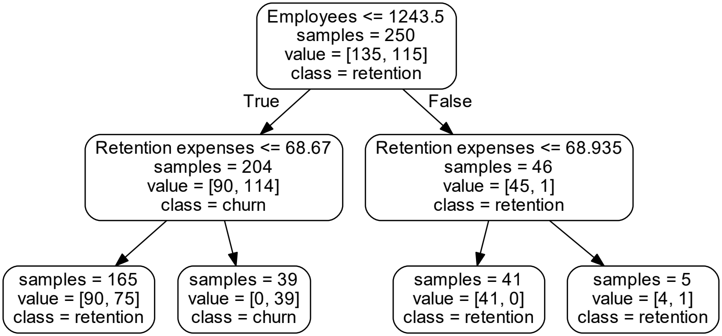




Left: partition of the space

Right: an example of the corresponding estimated regression function (vertical axis)

Example (classification tree): Customer Churn



In each node:

- *samples* refers to the total number of observations (customers)
- *value* refers to the numbers in class *retention* and class *churn*, respectively
- *class* refers to the predicted class

Regression trees

Regression trees

We predict with a **regression tree** as follows:

1. Divide the predictor space (all possible $\boldsymbol{x}$ values) via a sequence of successive splits into M distinct and non-overlapping regions, $R_1, R_2, \ldots, R_M$.

2. Model the response Y as a constant c_m in each region R_m, i.e. make the same prediction c_m for each observation that falls in the region:

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} c_m I(\boldsymbol{x} \in R_m).$$

Regression trees

Regression tree:

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} c_m I(\boldsymbol{x} \in R_m).$$

To fit a regression tree, we therefore need to construct the regions $R_1, R_2, \ldots, R_M$ and choose the corresponding constants $c_1, c_2, \ldots, c_M$.

Fitting a regression tree

If we adopt the residual sum of squares (RSS) as the criterion for fitting the model, then we estimate constant c_m as the sample average of the response values y_i corresponding to the region R_m (i.e. those y_i whose $\boldsymbol{x}_i$ fall in the region R_m):

$$\widehat{c}_m = \frac{1}{n_m} \sum_{i:\, \boldsymbol{x}_i \in R_m} y_i$$

where n_m in the number of observations in region m,

$$n_m = \sum_{i=1}^{n} I(\boldsymbol{x}_i \in R_m).$$

Fitting a regression tree

To find the best rectangular partition of the predictor space, our goal is to select regions $R_1, R_2, \ldots, R_M$ that minimise the RSS, which is given by

$$\mathsf{RSS} = \sum_{i=1}^{n} \left(y_i - \sum_{m=1}^{M} \widehat{c}_m \, I(\boldsymbol{x}_i \in R_m) \right)^2$$

$$= \sum_{m=1}^{M} \sum_{i:\, \boldsymbol{x}_i \in R_m} (y_i - \widehat{c}_m)^2 \,.$$

Fitting a regression tree

We would like to minimise the RSS over the choice of the regions $R_1,...R_M$:

$$\min_{R_1,\ldots,R_M} \sum_{m=1}^{M} \sum_{i:\,\boldsymbol{x}_i \in R_m} (y_i - \widehat{c}_m)^2$$

Solving this problem is generally computationally infeasible. We proceed with a "greedy" algorithm known as **recursive binary splitting**.

Growing a tree by binary splitting

Starting with all of the data, consider a splitting variable X_j (j-th predictor), a split point s, and two regions that define a partition of the predictor space:

$$R_1(j,s) = \{X|X_j \leq s\} \quad \text{and} \quad R_2(j,s) = \{X|X_j > s\}$$

$R_1(j,s)$ contains the predictor values $\boldsymbol{x}$ for which the j-th coordinate is $\leq s$

$R_2(j,s)$ contains the predictor values $\boldsymbol{x}$ for which the j-th coordinate is $> s$

Growing a tree by binary splitting

$$R_1(j,s) = \{X|X_j \le s\} \quad \text{and} \quad R_2(j,s) = \{X|X_j > s\}$$

We seek an index j and a split point s that solve

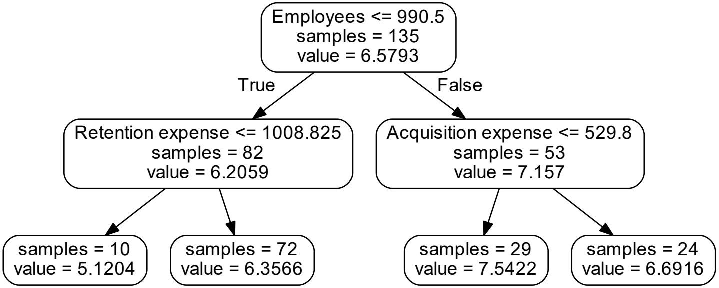
$$\min_{j,s} \left\{ \sum_{i:\, \boldsymbol{x}_i \in R_1(j,s)} (y_i - \widehat{c}_1)^2 + \sum_{i:\, \boldsymbol{x}_i \in R_2(j,s)} (y_i - \widehat{c}_2)^2 \right\}$$

Growing a tree by binary splitting

$$\min_{j,s} \left\{ \sum_{i:\, \boldsymbol{x}_i \in R_1(j,s)} (y_i - \widehat{c}_1)^2 + \sum_{i:\, \boldsymbol{x}_i \in R_2(j,s)} (y_i - \widehat{c}_2)^2 \right\}$$

- We can find j and s quickly by scanning through all the predictors, especially if p is not large.
- Having found the best split, we partition the data into the two resulting regions, and repeat the splitting process on each of them.
- We repeat this process until we reach a minimum node size (say, 5).

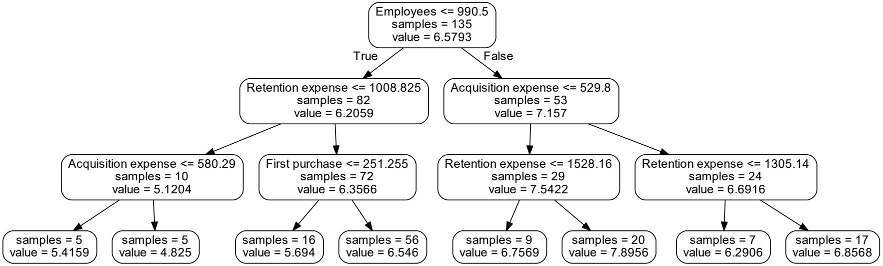
Example: Customer Lifetime Value



In each node:

- *samples* refers to the total number of observations (customers)
- *value* refers to the predicted customer lifetime value

Example: Customer Lifetime Value



Growing a tree by binary splitting

- The algorithm is called *greedy* because at each step of the tree-building process we make the best split for this particular step, without taking into account the consequences further down the tree.

- We also say that recursive binary splitting is *top down*, since it begins at the top of the tree and then successively splits the predictor space.

Tree size

How large should we grow the tree?

- A large tree will clearly overfit the data, with the small number of observations in each region leading to high variance estimators of the constants c_m.
- A small tree may fail to capture important structure in the data.
- The tree size is therefore a tuning parameter governing the complexity of the model, and we should select it by using the data.

Tree pruning

We adopt the following strategy to select the tree size:

1. Grow a large tree T_0, stopping the splitting process only when it reaches the minimum node size (i.e. when splitting the nodes further would result in too few observations in the new regions).

2. Use **cost-complexity pruning** to reduce the tree back to an optimal size.

Cost-complexity pruning

Let $T \subseteq T_0$ be a subtree with $|T|$ terminal nodes. We define the cost-complexity criterion:

$$C_\alpha(T) = \sum_{m=1}^{|T|} \sum_{i:\, \boldsymbol{x}_i \in R_m} (y_i - \widehat{c}_m)^2 \; + \; \alpha |T|$$

for a tuning parameter $\alpha \geq 0$ that determines the trade-off between tree size and the RSS.

For a given α, we find the subtree $T_\alpha \subseteq T_0$ to minimise $C_\alpha(T)$.

Weakest link pruning

- To find the subtree T_α that minimises the cost-complexity criterion we use *weakest link pruning*:

  we successively prune (cut away) branches by taking out the split whose removal results in the smallest increase in the RSS; we continue until we end up with a single node.

- Fact: such a sequence of subtrees contains the T_α for every α.

- We use cross validation to select the optimal value of α.

Summary: building a regression tree

Algorithm Building a regression tree

```
1: Use recursive binary splitting to grow a large tree, stopping only when it reaches the minimum terminal node size.
2: Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, corresponding to different values of $\alpha$.
3: Select $\alpha$ with the lowest cross validation mean squared error.
4: Output the subtree that corresponds to the chosen $\alpha$.
```

Classification trees

Classification tree

A **classification tree** is very similar to a regression tree, except that we predict a categorical response instead of a numerical one.

We split the predictor space in non-overlapping regions $R_1, R_2, \ldots, R_M$, and estimate the class probabilities within each region. We then classify the observations according to the estimated probabilities.

Classification trees

Let the response take values in $\{1, \ldots, C\}$. In a node m representing region R_m with n_m observations, we compute

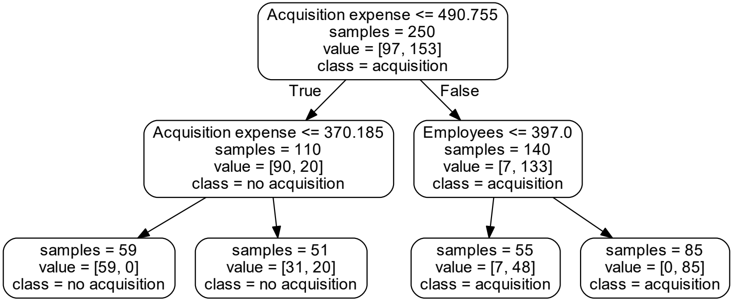
$$\widehat{p}_{my} = \frac{1}{n_m} \sum_{\boldsymbol{x}_i \in R_m} I(y_i = y)$$

the proportion of class y observations in node m.

Under the 0 - 1 loss, we classify to the *majority* class in node m:

$$\widehat{y}_m = \underset{y}{\operatorname{argmax}}\, \widehat{p}_{my}$$

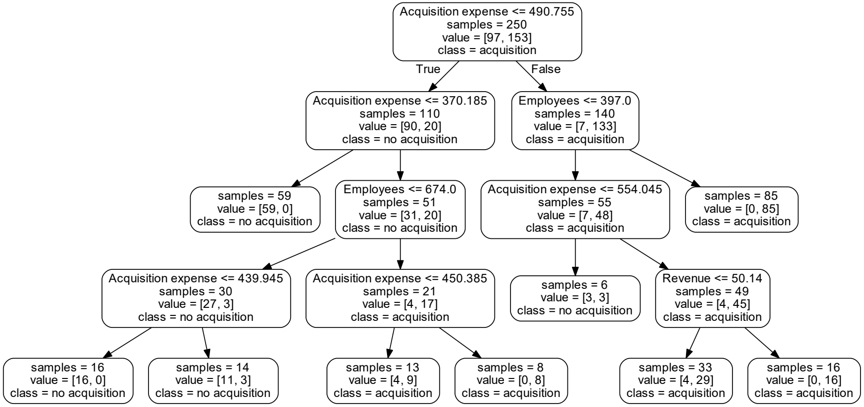
Example: Customer Acquisition



In each node:

- *samples* refers to the total number of observations (customers)
- *value* refers to the numbers in class *no acquisition* and class *acquisition*, respectively
- *class* refers to the predicted class

Example: Customer Acquisition



Growing a classification tree

We need an appropriate criterion for growing a classification tree by binary splitting (the role of the RSS in a regression tree).

We use **node impurity measures** such as misclassification error, Gini index, and cross-entropy.

Node impurity measures

- The **misclassification error** for node m is:

$$1 - \max_y \widehat{p}_{my}$$

- Recall that we classify node m to the class y that maximizes $\widehat{p}_{my}$. Hence, $\max_y \widehat{p}_{my}$ is the proportion of training observations in node m that are correctly classified.

- Thus, misclassification error gives the proportion of misclassifications in node m.

- When the node is *pure*, i.e. all the observations in it belong to the same class, the misclassification error is zero.

Gini index

- The **Gini index** for node m is:

$$\sum_{y=1}^{C} \widehat{p}_{my}(1 - \widehat{p}_{my})$$

- For example, when $C = 2$ the Gini index is $2\,\widehat{p}_m\,(1 - \widehat{p}_m)$, where $\widehat{p}_m$ is the proportion of node m belonging to the second class.

- The Gini index is zero (smallest possible value) when the node is pure.

Cross-entropy

- The **cross-entropy** or **deviance** for node m is:
$$-\sum_{y=1}^{C} \widehat{p}_{my} \log(\widehat{p}_{my})$$
which corresponds to negative log-likelihood for the training data in node m.

- When $C = 2$ the cross-entropy simplifies to $-\widehat{p}_m \log(\widehat{p}_m) - (1 - \widehat{p}_m) \log(1 - \widehat{p}_m)$, where $\widehat{p}_m$ is the proportion of node m belonging to the second class.

- Like the Gini index, the cross-entropy is the smallest when the node is pure.

Node impurity measures

Node impurity measures for $C = 2$ as functions of the proportion of the node belonging to the second class (cross-entropy is scaled for visualisation):

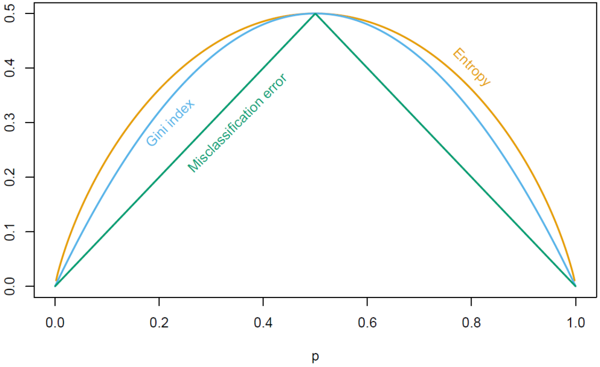


Figure from ESL

Node impurity measures

- When evaluating a split we weight the node impurity measures for the two child nodes by the number of observations in each node. E.g. for misclassification error, we look at the total number of misclassifications.
- Gini index and cross entropy are better at encouraging pure nodes than the misclassification error, and, thus, are generally the preferred measures when growing the tree.
- Gini index and the cross entropy also have the advantage of being differentiable functions of the proportions, which makes them more amenable to numerical optimisation.
- We can use any of the measures (including the misclassification error) to prune the tree.

Discussion

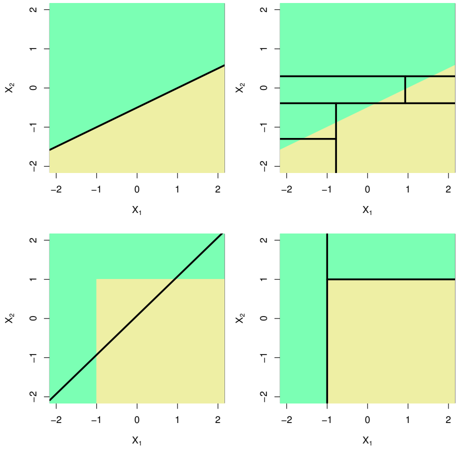
Categorical predictors

- Trees can easily handle categorical predictors without creating dummy variables: a split on such a variable comes down to assigning some of the categories to one branch, and assigning the remaining categories to the other branch.
- Moreover, the case of ordinal categorial variables (i.e. those whose categories can be ordered) is simpler, because we can perform the usual binary splitting by separating the larger and the smaller categories.
- Thus, trees handle ordinal variables in a more natural way than OLS (where a numerical score is assigned to each category).

Categorical predictors

- In the binary classification setting, the following convenient computational simplification is available for working with milti-class predictors:
- We can order the predictor classes according to the proportion of observations with $Y = 1$ in each class, and then treat the predictor as ordinal.
- A similar computational simplification is available in the regression case, but the ordering is done using the mean Y value in each class.
- Note that the partitioning algorithm will tend to favour categorical predictors with many levels, leading to potential overfitting if the number of categories is large.

Trees versus linear models, classification example (figure from ISL)



True decision boundary: green vs. yellow

Top: linear decision boundary   Bottom: nonlinear decision boundary

Left: a linear approach   Right: decision tree approach

Advantages of trees

- Trees are simple and are easy to explain.
- Trees lead to highly interpretable prediction rules.
- Trees can easily handle categorical predictors without the need to create dummy variables.
- Trees can approximate complex nonlinearities, including interactions.
- Trees are the basic component of powerful prediction methods such as random forests and boosting.

Disadvantages of trees

- Instability: due to their hierarchical nature, trees are inherently unstable and have high variance. Small changes in the data may lead to a very different sequence of splits, compromising the interpretation of the model.
- Lack of smoothness: trees lead to non-smooth prediction functions and decision boundaries. Especially in the regression setting, this can degrade the performance.
- Therefore, decision trees may have low predictive accuracy.

Review questions

- What is a regression tree and how do we compute predictions based on it?
- Provide a detailed explanation of the algorithm that is used to fit a regression tree.
- What is a classification tree and how do we compute predictions based on it?
- What are the node impurity measures that we use for classification trees?