Statistical Learning and Data Mining

Module 13: Tree-Based Methods

Semester 2, 2017

Discipline of Business Analytics, The University of Sydney Business School

Module 13: Tree-Based Methods

1. Regression trees
2. Classification trees
3. Implementation details and discussion
4. Bagging
5. Random forests

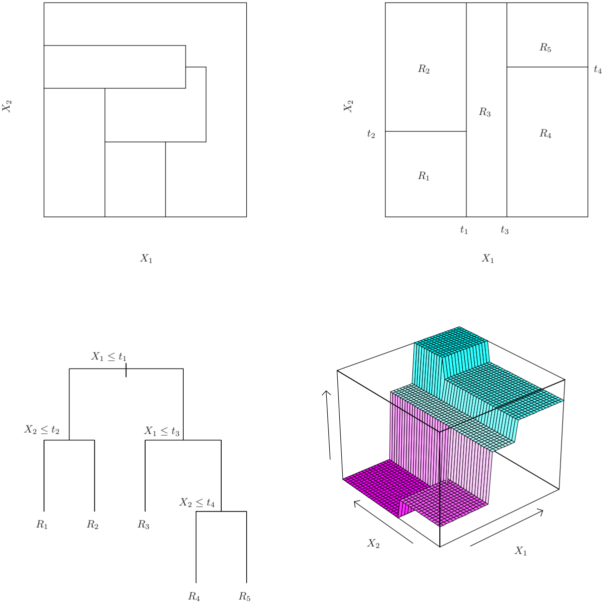
Classification and regression trees

- Tree-based methods for regression and classification partition the predictor space into a set of rectangles, and fit a simple model such as a constant in each one.

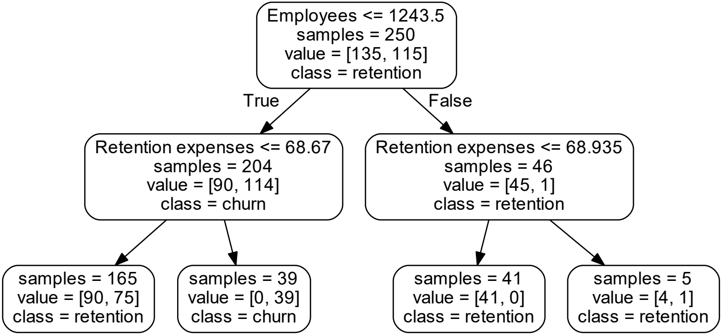
- The partitions are based on *recursive binary splitting*, enabling an interpretable visualisation of the prediction rule as a decision tree.

- The next slide illustrates a regression tree for two predictors X_1 and X_2.

Rectangular partitions and trees



Example (classification tree): Customer Churn



Regression trees

Regression trees (key concept)

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

1. Divide the predictor space – that is, the set of possible values for $X_1, X_2, \ldots, X_p$ – into M distinct and non-overlapping regions, $R_1, R_2, \ldots, R_M$.
2. We then model the response Y as a constant c_m in each region. We 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 (key concept)

Regression tree:

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

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

Fitting a regression tree

If we adopt the sum of squared errors as the criterion for fitting the model, the estimated constant $\widehat{c}_m$ is the sample average of the response values y_i in region 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 samples 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, given by

$$\begin{aligned}\text{RSS} &= \sum_{i=1}^{N} \left(y_i - \sum_{m=1}^{M} \widehat{c}_m \, I(\boldsymbol{x}_i \in R_m) \right)^2 \\ &= \sum_{m=1}^{M} \sum_{i:\, \boldsymbol{x}_i \in R_m} (y_i - \widehat{c}_m)^2 \,.\end{aligned}$$

Fitting a regression tree

Fitting a regression tree:

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

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

Growing a tree by binary splitting (key concept)

Starting with all of the data, consider a splitting variable j and a split point s, and define the regions

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

We then seek a splitting variable j and split point s that solve

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

Growing a tree by binary splitting

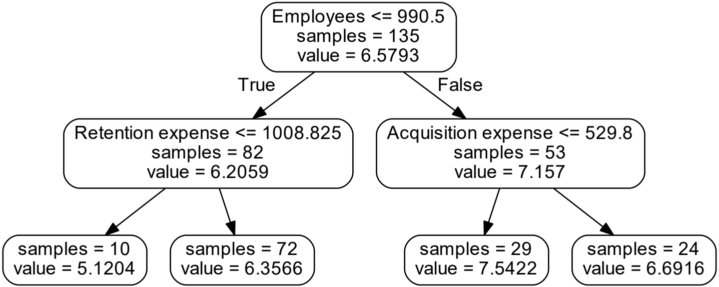
$$\min_{j,s}\left\{\left(\sum_{i:\,\boldsymbol{x}_i\in R_1(j,s)}(y_i-\widehat{c}_1)^2\right)+\left(\sum_{i:\,\boldsymbol{x}_i\in R_2(j,s)}(y_i-\widehat{c}_2)^2\right)\right\}.$$

- We can find j and s very quickly by scanning through all the predictors, especially if p is not large.

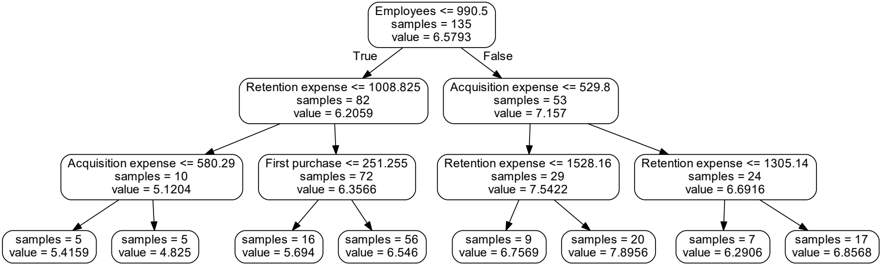
- Having found the best split, we partition the data into the two resulting regions, and repeat the splitting process on each of them.

- We repeat this process until we reach a minimum node size (say, 5).

Example: Customer Lifetime Value



Example: Customer Lifetime Value



Employees <= 990.5
samples = 135
value = 6.5793

True

False

Retention expense <= 1008.825
samples = 82
value = 6.2059

Acquisition expense <= 529.8
samples = 53
value = 7.157

Acquisition expense <= 580.29
samples = 10
value = 5.1204

First purchase <= 251.255
samples = 72
value = 6.3566

Retention expense <= 1528.16
samples = 29
value = 7.5422

Retention expense <= 1305.14
samples = 24
value = 6.6916

samples = 5
value = 5.4159

samples = 5
value = 4.825

samples = 16
value = 5.694

samples = 56
value = 6.546

samples = 9
value = 6.7569

samples = 20
value = 7.8956

samples = 7
value = 6.2906

samples = 17
value = 6.8568

Growing a tree by binary splitting

- The algorithm is greedy because at each step of the tree-building process, we make the best split at a 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 samples in each region leading to high variance estimators of the 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 adaptively from 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 (that is, when splitting the nodes further would result in too few observations in the new regions).

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

Cost complexity pruning

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

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

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

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

Weakest link pruning

- For each α, there is a unique subtree that minimises $C_\alpha(T)$.
- To find T_α, we use *weakest link pruning*: we successively remove the internal node (split) that causes the smallest increase in the RSS (least contributes to fit), and continue until we reach the root tree (one region).
- We can show that this sequence of subtrees must contain T_α.
- We then use cross validation to select the optimal 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 (region) size.
2: Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of different values of $\alpha$.
3: Select $\alpha$ with the lowest cross validation mean squared error (the average MSE across folds).
4: Output the subtree that corresponding to the best $\alpha$.
```

Classification trees

Classification tree

A **classification tree** is very similar to a regression tree, except that we predict a qualitative response instead of quantitative 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 $y \in \{1, \dots, C\}$. In a node m representing region R_m with N_m observations, we compute

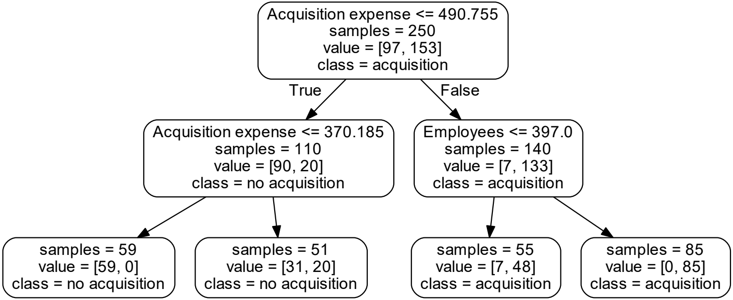
$$\widehat{p}_{mc} = \frac{1}{N_m} \sum_{\boldsymbol{x}_i \in R_M} I(y_i = c)$$

the proportion of class c observations in node m.

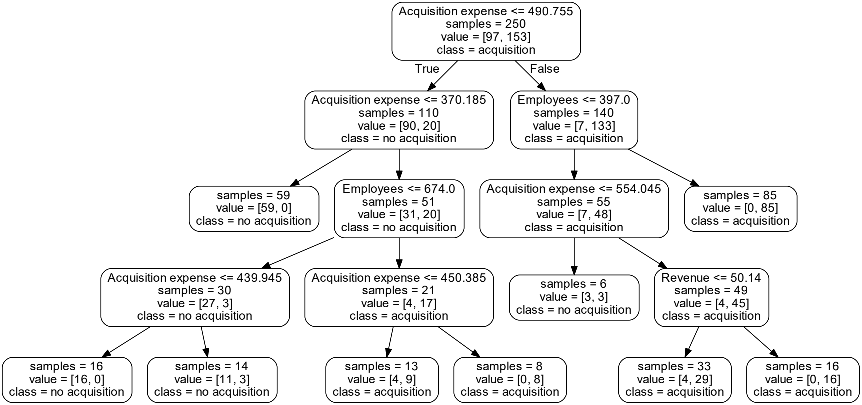
Under the zero-one loss, we classify to the majority class in node m:

$$\widehat{y}_m = \underset{c}{\operatorname{argmax}}\ \widehat{p}_{mc}$$

Example: Customer Acquisition



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).

The **node impurity measures** for classification include the misclassification error, Gini index, cross-entropy.

Node impurity measures

The **misclassification error** is

$$1 - \widehat{p}_{mc'},$$

where $c' = \widehat{y}_m$ and $\widehat{p}_{mc'}$ is the proportion of training observations in node m that are incorrectly classified.

Gini index (key concept)

- The **Gini index** is

$$\sum_{c=1}^{C} \widehat{p}_{mc}(1-\widehat{p}_{mc}) = \sum_{c=1}^{C}\sum_{j\neq c} \widehat{p}_{mc}\,\widehat{p}_{mj},$$

  which measures the total variance across the classes.

- For example, when $C = 2$ the Gini index is $2\,\widehat{p}\,(1-\widehat{p})$, where $\widehat{p}$ is the proportion in the second class.

- The Gini index takes on a small value if all the probabilities are close to zero or one (that is, when the node is pure).

Cross-entropy (key concept)

- The **cross-entropy** or **deviance** is

$$-\sum_{c=1}^{C} \widehat{p}_{mc} \log(\widehat{p}_{mc}),$$

  which corresponds to negative multinoulli log-likelihood for the training observations in node m evaluated at $\widehat{p}_{m1}, \ldots, \widehat{p}_{mC}$.

- When $C = 2$ the cross-entropy is $-\widehat{p}\log(\widehat{p}) - (1-\widehat{p})\log(1-\widehat{p})$, where $\widehat{p}$ is the proportion in the second class.

- Like the Gini index, the cross-entropy takes on a small value if all the probabilities are close to zero or one (that is, when the node is pure).

Node impurity measures

In the two class case (the cross-entropy is scaled for visualisation):

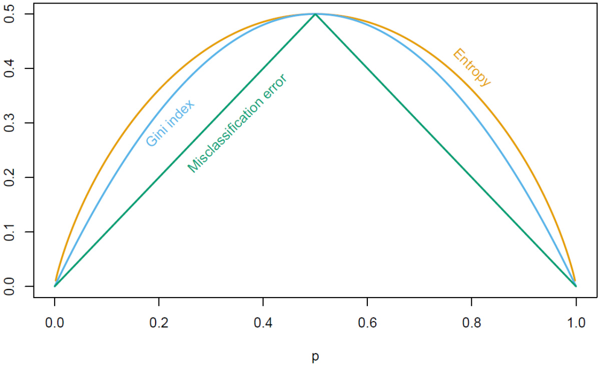


Figure from ESL.

Node impurity measures

- The Gini index and the cross entropy are more sensitive to changes in the node probabilities than the misclassification error, so that we prefer these measures when growing the tree.
- These two measures also have the advantage of being differentiable, which makes them more amenable to numerical optimisation.
- We can use any of the measures (including the misclassification error) to prune the tree.

Implementation details and discussion

Categorical and ordinal variables

- Trees lead to a convenient way to work with categorical predictors with multiple levels. For binary classification and regression, we can order the predictor classes according to their proportion falling in class one average response value respectively.

- We then code the categorical variable as a numerical variable (instead of creating dummy variables), and treat it as such.

- The disadvantage is that the partitioning method will tend to favour categorical predictors with many levels, leading to overfitting if the number of categories is large.

- Trees also handle ordinal variables in a natural way, unlike a linear regression estimated by OLS.

Loss matrix

- To incorporate general loss functions into the modelling process, we can modify the Gini index to

$$\sum_{c=1}^{C} \sum_{j \neq c} L_{cj} \widehat{p}_{mc} \widehat{p}_{mj}$$

  where L_{cj} is the loss incurred for classifying a class c observation as class j.

- Another option is use observation weights, which also applies to the cross-entropy loss.

- In a terminal node, we classify to class

$$\widehat{y}_m = \operatorname*{argmin}_{c} \sum_{\ell} L_{\ell c} \widehat{p}_{mc}.$$

Other tree building procedures

- Our discussion focuses on the CART (classification and regression trees) implementation of trees.

- The other popular methodology is ID3 and its later versions, C4.5 and C5.0, which are also commonly used in the data mining industry

Advantages of trees

- Trees are simple.
- Trees lead to interpretable prediction rules.
- Trees can easily handle categorical and ordinal features 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 can lead to a very different sequence of splits, compromising interpretability if the objective is to describe the population.

- Lack of smoothness: trees lead to non-smooth prediction functions, which can degrade performance specially in regression settings.

- Trees have difficulty in capturing additive structure.

- Therefore, decision trees tend to have low predictive accuracy.

Trees versus linear models

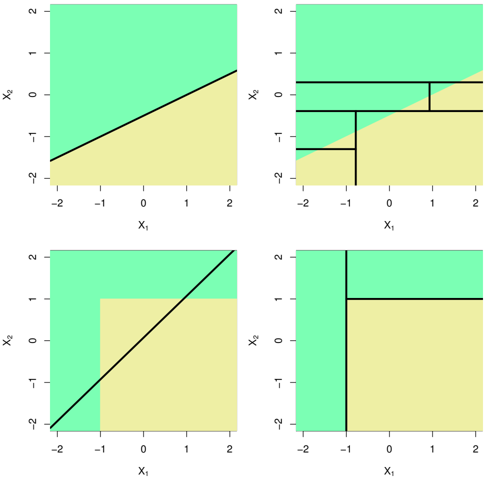


Figure from ISL.

Bagging

Bagging

The idea of **bootstrap aggregation** or **bagging** is to average many noisy but approximately unbiased models, therefore reducing variance.

Decision trees are ideal candidates for bagging, since they capture complex interaction structures in the data, and if grown deeply, have relatively low bias, but high variance.

Bagging (key concept)

Suppose that we fit a regression model to the training data $\mathcal{D}$ to obtain the prediction $\widehat{f}(\boldsymbol{x})$ for an input point $\boldsymbol{x}$. Bagging averages this prediction over a collection of bootstrap samples, thereby reducing its variance.

For each bootstrap sample $\mathcal{D}_b^*$, $b = 1, \ldots, B$, we fit the model to obtain a prediction $\widehat{f}_b^*(\boldsymbol{x})$. The bagging prediction is:

$$\widehat{f}_{\mathsf{bag}}(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} \widehat{f}_b^*(\boldsymbol{x}).$$

Bagging

- The bagged estimate will differ from the original prediction ($B \to \infty$) only when the estimator $\widehat{f}(\cdot)$ is a nonlinear or adaptive function of the data (due to the linearity of expectations).

- Therefore, bagging does not improve the prediction from linear basis expansion models.

- Trees in turn are highly nonlinear estimators, and may benefit greatly from averaging.

- Increasing B does not overfit. The higher the B, the more stable the average.

Bagging classifiers (key concept)

In the classification setting, we obtain a prediction $\widehat{y}_b^*(\boldsymbol{x})$ for each bootstrap $\mathcal{D}_b^*$, $b = 1, \ldots, B$. The bagged classifier selects the class with the most "votes" from the B trees,

$$\widehat{y}_{\mathsf{bag}}(\boldsymbol{x}) = \underset{c}{\operatorname{argmax}} \sum_{b=1}^{B} I(\widehat{y}_b^*(\boldsymbol{x}) = c).$$

If we require a class probability estimate $\widehat{p}(\boldsymbol{x})$, the bagged probability estimate is

$$\widehat{p}_{\mathsf{bag}}(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} \widehat{p}_b^*(\boldsymbol{x}),$$

where $\widehat{p}_b^*(\boldsymbol{x})$ is the estimated probability according to the model fitted to the bootstrap sample b.

Random forests

Random forests (key concept)

Random forests improve over bagging by adding a tweak that decorrelates the trees: before each binary split, we select a *random sample* of only $k \leq p$ predictors as split candidates.

Random forest (key concept)

Algorithm Random forest

```
for $b = 1$ to $B$ do
    Sample $N$ observations with replacement from the training data $\mathcal{D}$
    to obtain the bootstrap sample $\mathcal{D}_b^*$.
    Grow a random forest tree $T_b$ to $\mathcal{D}_b^*$ by repeating the following
    steps for each terminal node of the tree, until the minimum node
    size $n_{\mathsf{min}}$ is reached:
    (i)  Select $k$ variables at random from the $p$ variables.
    (ii) Pick the best variable and split point among the $k$ candidates.
    (iii) Split the node into two daughter nodes.
end for
Output the ensemble of trees $\{T_b\}_1^B$.
```

Random forest: implementation details

The tuning parameters in a random forest are k (the number of split variables) and the minimum node size for each tree.

- For classification the default value for k is $\lfloor \sqrt{p} \rfloor$ and the minimum node size is one.
- For regression, the default value for k is $\lfloor p/3 \rfloor$ and the minimum node size is five.

In practice the best values depend on the problem, and we can use cross validation to select them. Using a small k will typically be helpful when there is a large number of correlated predictors.

Example: Customer Acquisition

Test results

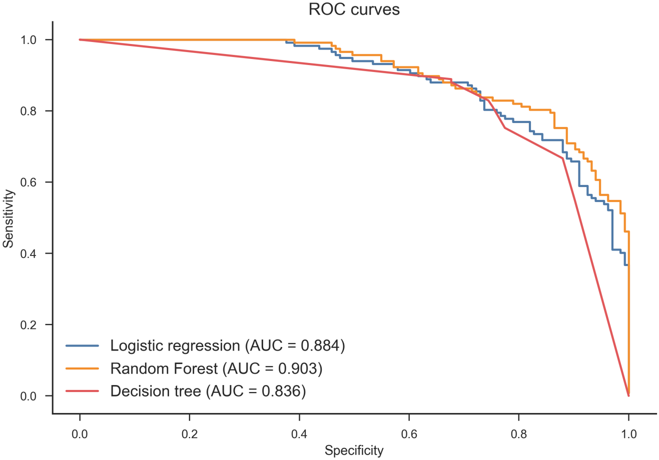
	Error rate	Sensitivity	Specificity	AUC	Precision
Logistic	0.120	0.856	0.910	0.960	0.922
ℓ_2 regularised	0.112	0.878	0.901	0.960	0.917
Decision Tree	0.132	0.906	0.820	0.921	0.863
Bagged trees	0.112	0.906	0.865	0.957	0.894
Random forest	0.100	0.914	0.883	0.967	0.907

Example: Customer Churn

Test results

	Error rate	Sensitivity	Specificity	AUC	Precision
Logistic regression	0.224	0.795	0.759	0.884	0.744
ℓ_2 regularised	0.212	0.795	0.782	0.885	0.762
Decision Tree	0.236	0.752	0.774	0.836	0.746
Bagged trees	0.192	0.778	0.835	0.890	0.805
Random forest	0.180	0.803	0.835	0.903	0.810

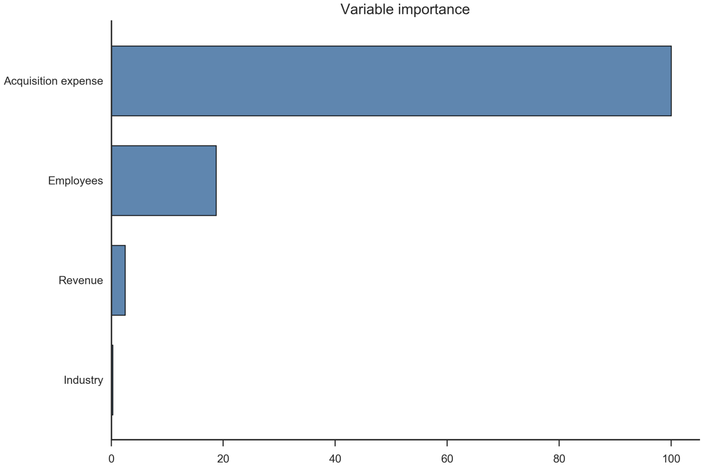
Example: Customer Churn



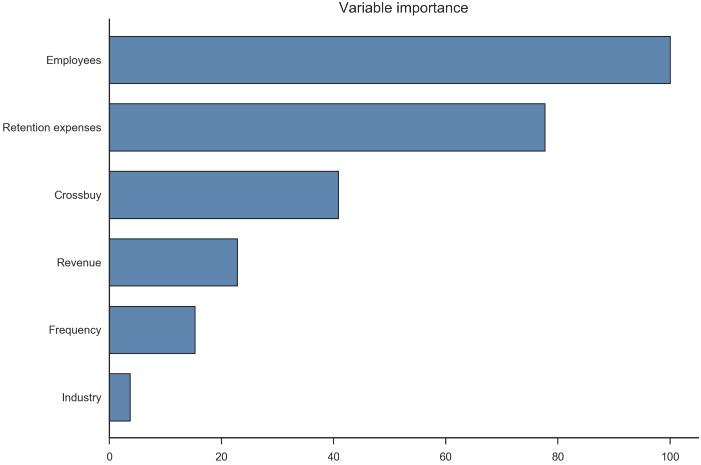
Variable importance measures

- Random forests (including bagging as a special case) improve prediction at the expense of the simple interpretability of the prediction generated by a single tree.
- We can obtain an overall summary of the importance of a predictor in a random forest by using the RSS (regressor) or the Gini index/cross-entropy (classifier).
- We measure the importance of each variable by adding up the total decrease in the split criterion due to splits over a single predictor, averaged over all B trees.

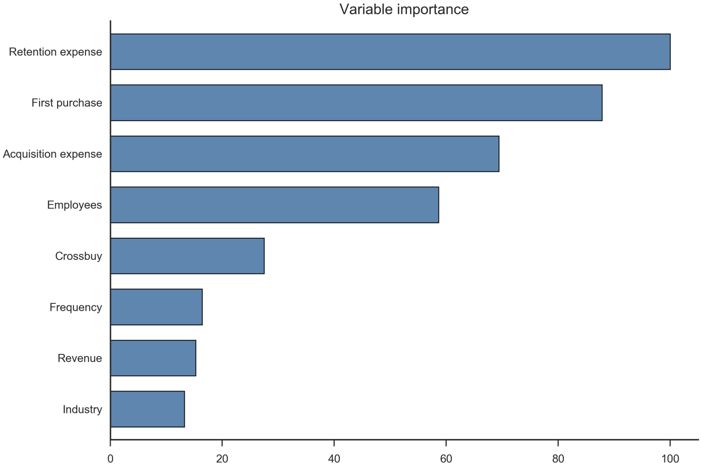
Example: Customer Acquisition



Example: Customer Churn



Example: Customer Lifetime Value



Why do random forests work?

- When there is a small subset of strong predictors, these variables will tend to dominate the top splits of the trees over the bootstrap samples, making the bagged trees quite similar to each other. Since the trees are similar, the bagged trees will be highly correlated.
- Averaging over highly correlated variables does not lead to a large reduction in variance.
- Random forests overcome this problem by giving a higher chance to other predictors, leading to decorrelation (at the expense of some bias).

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?
- Explain the main ideas of bagging and random forests. How do random forests improve over bagging?
- What are the tuning parameters for a random forest?