

Decision Trees

Joaquin Vanschoren, Mykola Pechenizkiy, Anne Driemel

Course overview: <http://openml.github.io/course>

```
In [20]: options(warn=-1)
library(mlr) # ML library
library(OpenML) # Data import and sharing results
library(ggplot2) # Plotting
library(cowplot) # Plot styling
library(rattle) # Plotting trees

In [21]: # Helper function for visualizing the hyperrectangles

library("grid")
library("coin")
Colors <- colorspace::rainbow_hcl(3)
Colors_trans <- apply(rbind(col2rgb(colorspace::rainbow_hcl(3)), alpha = 100, maxColorValue = 255),
                      function(x) do.call("rgb", as.list(x)))
plot_rectangles <- function(obj, x, y, class, depth) {
  xname <- paste(deparse(substitute(x)), 500), collapse = "\n")
  yname <- paste(deparse(substitute(y)), 500), collapse = "\n")
  grid.newpage()
  pushViewport(plotViewport(c(5, 4, 2, 2)))
  pushViewport(dataViewport(x,
                           y,
                           name="plotRegion"))
  grid.points(x, y, pch = 19,
             gp=gpar(cex=0.5, col = Colors[class]))
  grid.rect()
  grid.xaxis()
  grid.yaxis()
  grid.text(xname, y=unit(-3, "lines"))
  grid.text(yname, x=unit(-3, "lines"), rot=90)
  seekViewport("plotRegion")
  plot_rect(obj@tree, xname = xname, depth)
  grid.points(x, y, pch = 19,
             gp=gpar(cex=0.5, col = Colors[class]))
}

plot_rect <- function(obj, xname, depth) {
  if (!missing(depth)) {
    if (obj$nodeID >= depth) return()
  }
  if (obj$psplit$variableName == xname) {
    x <- unit(rep(obj$psplit$splitpoint, 2), "native")
    y <- unit(c(0, 1), "npc")
  } else {
    x <- unit(c(0, 1), "npc")
  }
}
```

```

    y <- unit(rep(obj$psplit$splitpoint, 2), "native")
  }
  grid.lines(x, y)
  if (obj$psplit$variableName == xname) {
    pushViewport(viewport(x = unit(current.viewport()$xscale[1], "native"),
      width = x[1] - unit(current.viewport()$xscale[1], "native"),
      xscale = c(unit(current.viewport()$xscale[1], "native"), x[1]),
      yscale = current.viewport()$yscale,
      just = c("left", "center")))
  } else {
    pushViewport(viewport(y = unit(current.viewport()$yscale[1], "native"),
      height = y[1] - unit(current.viewport()$yscale[1], "native"),
      xscale = current.viewport()$xscale,
      yscale = c(unit(current.viewport()$yscale[1], "native"), y[1]),
      just = c("center", "bottom")))
  }
  pred <- ifelse(length(obj$left$prediction) == 1, as.integer(obj$left$prediction > 0.5) + 1, 0)
  grid.rect(gp = gpar(fill = "white"))
  grid.rect(gp = gpar(fill = Colors_trans[pred]))
  if (!is(obj$left, "TerminalNode")) {
    plot_rect(obj$left, xname, depth)
  }
  popViewport()
  if (obj$psplit$variableName == xname) {
    pushViewport(viewport(x = unit(x[1], "native"),
      width = unit(current.viewport()$xscale[2], "native")-x[1],
      xscale = c(x[1], unit(current.viewport()$xscale[2], "native")),
      yscale = current.viewport()$yscale,
      just = c("left", "center")))
  } else {
    pushViewport(viewport(y = unit(y[1], "native"),
      height = unit(current.viewport()$yscale[2], "native")-y[1],
      xscale = current.viewport()$xscale,
      yscale = c(y[1], unit(current.viewport()$yscale[2], "native")),
      just = c("center", "bottom")))
  }
  pred <- ifelse(length(obj$right$prediction) == 1, as.integer(obj$right$prediction > 0.5) + 1, 0)
  grid.rect(gp = gpar(fill = "white"))
  grid.rect(gp = gpar(fill = Colors_trans[pred]))
  if (!is(obj$right, "TerminalNode")) {
    plot_rect(obj$right, xname, depth)
  }
  popViewport()
}

```

1 Decision trees

- Most widely-used machine learning algorithm
 - Fairly easy to understand and implement
 - Output (model) also easy to understand
 - Easy to use: you don't have to tweak many hyperparameters before it returns something useful
 - Pretty scalable (in number of features and instances)
 - Not top performance, but good place to start

- Depending on the type of the target feature
 - Classification tree (predicts class)
 - Regression tree (predicts numeric value)
- Predictive features can be discrete (categorical) or continuous (numeric), or mixed

1.1 Classification

Data comes as a finite learning set $\mathcal{L} = (X, y)$ where * Input samples are given as an array X of shape `n_samples × n_features`, taking their values in \mathcal{X} ; * Output values are given as an array y , taking symbolic values in \mathcal{Y} .

The goal of supervised classification is to build an estimator $\varphi_{\mathcal{L}} : \mathcal{X} \mapsto \mathcal{Y}$ minimizing

$$Err(\varphi_{\mathcal{L}}) = \mathbb{E}_{X,Y} \{L(Y, \varphi_{\mathcal{L}}(X))\}$$

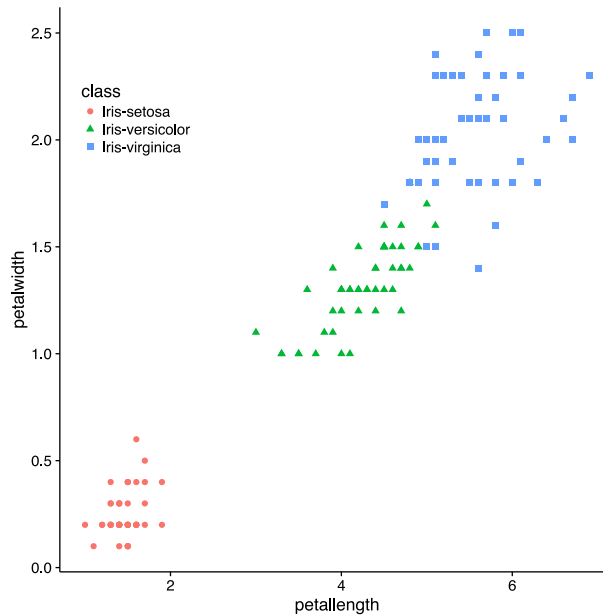
where L is a loss function, e.g., the zero-one loss for classification $L_{01}(Y, \hat{Y}) = 1(Y \neq \hat{Y})$.

1.2 Classification example

```
In [22]: iris = getOMLDataSet(did = 61, verbosity=0)$data
         iristask = makeClassifTask(data = iris, target = "class")
         head(iris)
```

	sepalength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa

```
In [39]: ggplot(data=iris, aes(x=petallength, y=petalwidth, color=class, shape=class)) + geom_point(size=)
```



1.3 Learning algorithms (recap)

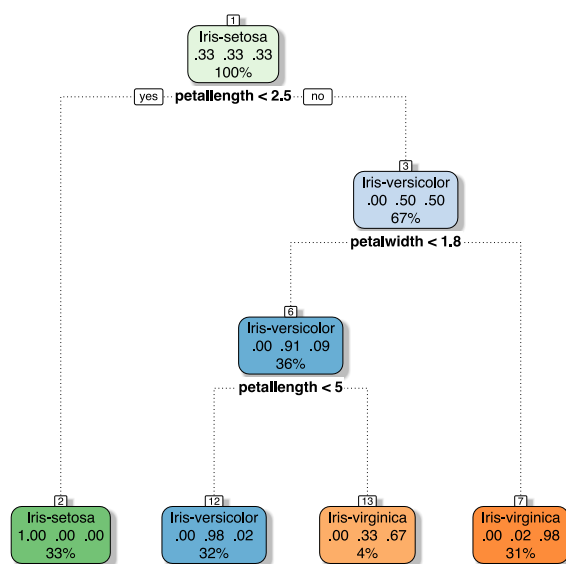
All learners consist of three main components:

- **Representation:** A model must be represented in a formal language that the computer can handle.
 - Defines the concepts it can learn: the hypothesis space
- **Evaluation function:** How to choose one hypothesis over the other?
 - Used internally, can differ from external measure
- **Optimization:** How do we search the hypothesis space?
 - Key to the efficiency of the learner

2 Decision trees: Representation

- **Internal nodes:** test value of one particular feature, divide all instances into branches according to the outcome
- **Leaf nodes:** make predictions based on remaining subset

```
In [24]: rpart = makeLearner("classif.rpart")
rpart = setHyperPars(rpart, cp = 0, minbucket=4)
model = train(rpart, iristask)
fancyRpartPlot(model$learner.model, sub="")
```



Trees divide the feature space into (hyper)rectangular regions (leaves) and fit a simple model (or a constant) in each leaf:

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

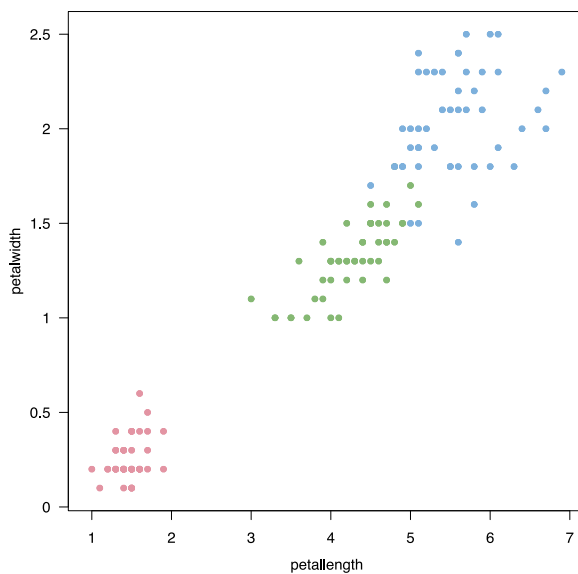
where: M rectangulars R_m are used, and c_m can be: - the most frequent class - the fitted class distribution (probabilistic classifier) - a simple classification model (linear model, naïve Bayes, ...) - a fitted (e.g. average) value (regression) - a fitted regression model (linear, ...)

Variants exist that create non-rectangular regions (e.g. oblique trees)

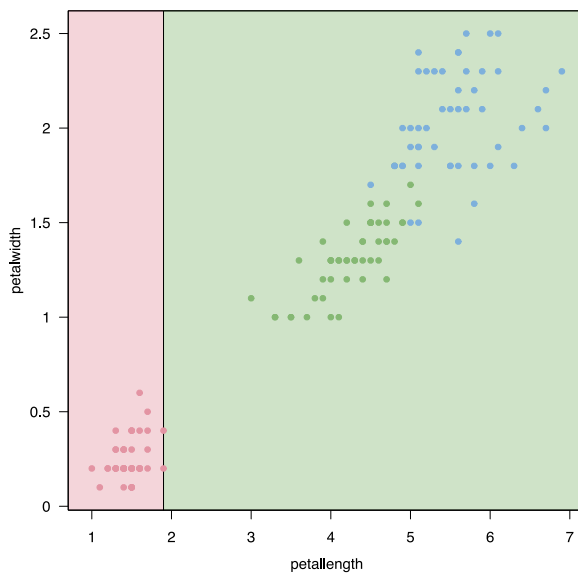
2.1 Can you predict the decision boundaries?

```
In [25]: library(rpart)
library(party)
tree_iris <- ctree(class ~ ., data = iris)
```

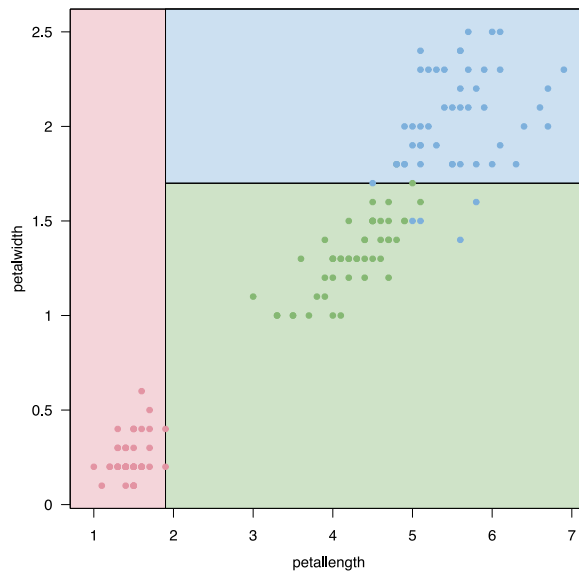
```
In [26]: with(iris, plot_rectangles(tree_iris, petallength, petalwidth, class, 0))
```



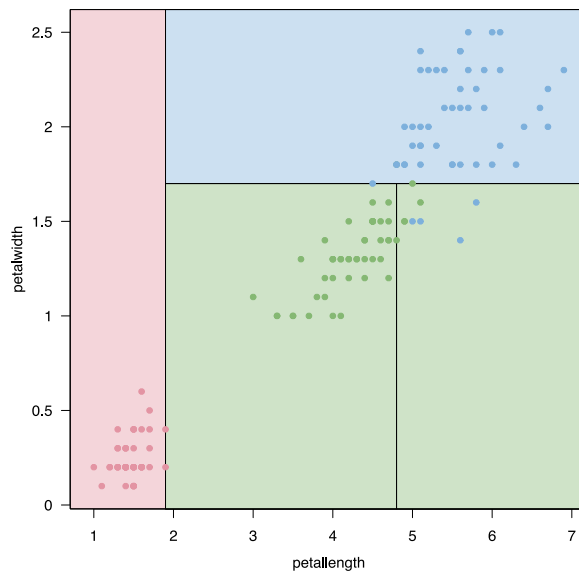
```
In [27]: with(iris, plot_rectangles(tree_iris, petallength, petalwidth, class, 2))
```



```
In [28]: with(iris, plot_rectangles(tree_iris, petallength, petalwidth, class, 4))
```



```
In [29]: with(iris, plot_rectangles(tree_iris, petallength, petalwidth, class))
```



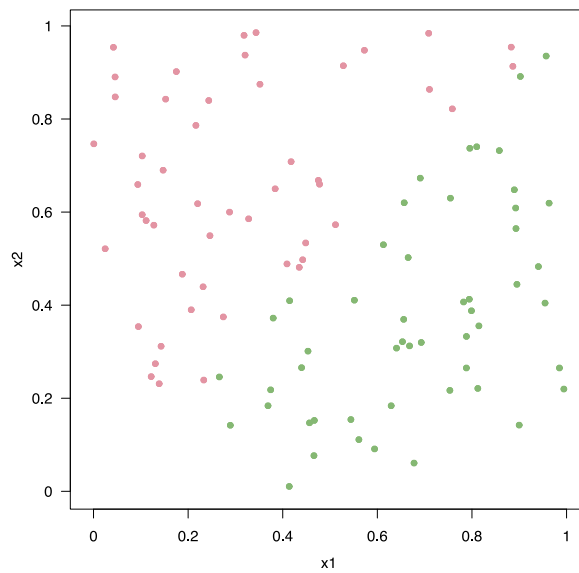
2.2 Representation: limitations

- High instability (variance) of the trees: small changes in data can lead to completely different splits
- Prediction function is not smooth (step function is fitted)

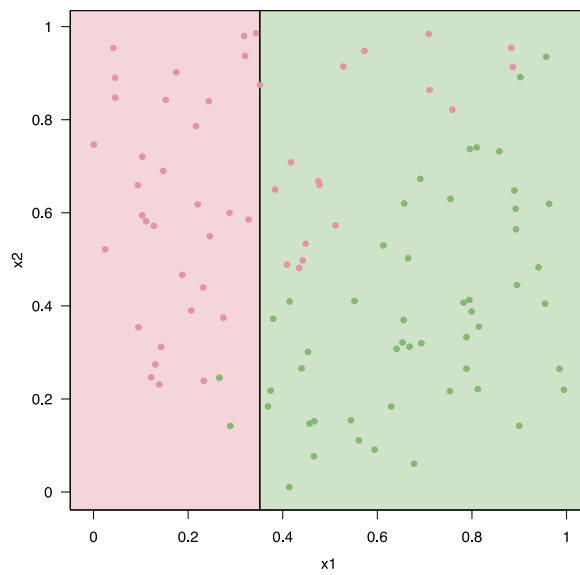
- Cannot learn (only approximate) linear dependencies
 - Still works well in many cases (in high dimensions a good splitting can be found)
- Linear dependencies must be modeled over several splits
 - Simple linear correlations translate into complex trees
- Worst case: parity problem (exponential number of leaves)

```
In [30]: set.seed(123)
         n <- 100
         linear <- data.frame(x1 = runif(n),
                              x2 = runif(n))
         linear$y <- with(linear, as.integer(x1 > x2))
         tree_example <- ctree(y ~ x1 + x2, data = linear)
```

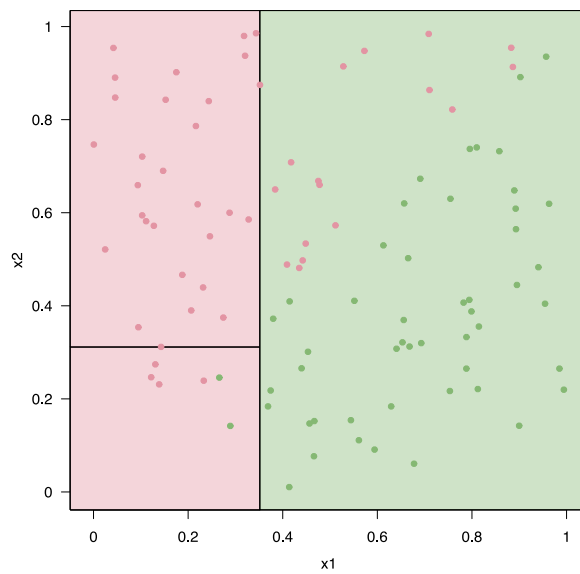
```
In [31]: with(linear, plot_rectangles(tree_example, x1, x2, y + 1, 0))
```



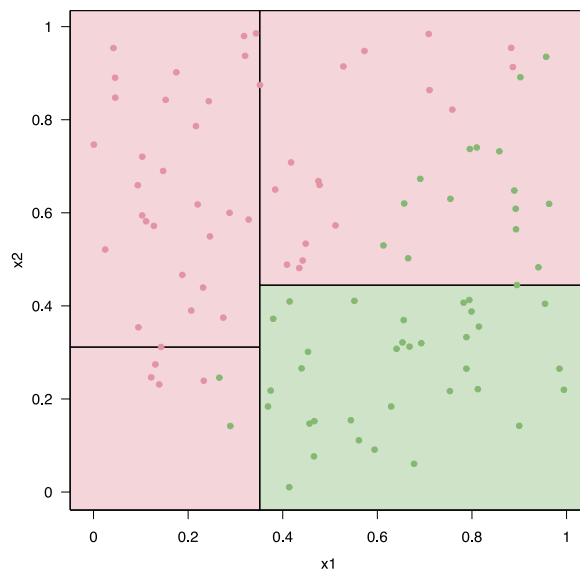
```
In [32]: with(linear, plot_rectangles(tree_example, x1, x2, y + 1, 2))
```



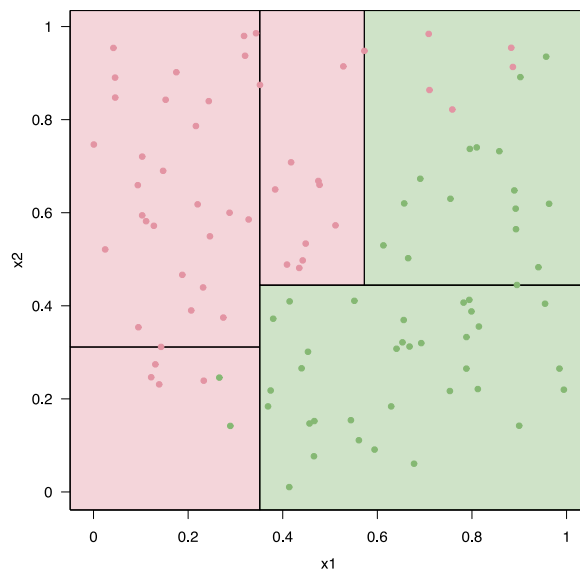
```
In [33]: with(linear, plot_rectangles(tree_example, x1, x2, y + 1, 3))
```



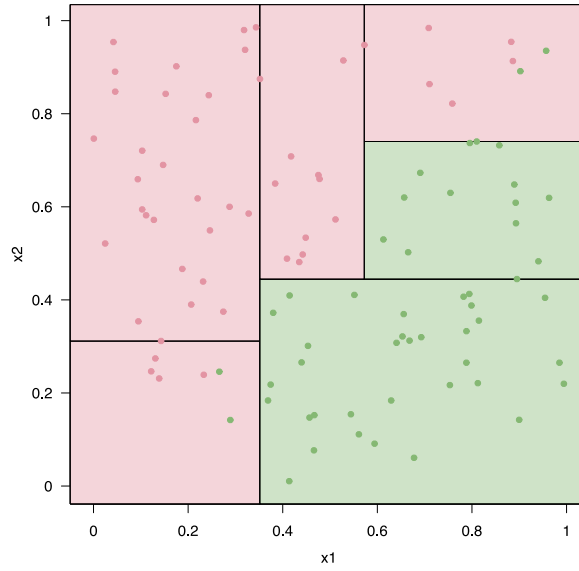
```
In [34]: with(linear, plot_rectangles(tree_example, x1, x2, y + 1, 6))
```

```
In [35]: with(linear, plot_rectangles(tree_example, x1, x2, y + 1, 8))
```



```
In [36]: with(linear, plot_rectangles(tree_example, x1, x2, y + 1, 10))
```



3 Evaluation: which split to choose?

3.1 Classification

For classification ($X_i \rightarrow class_k$): Impurity measures:

- Misclassification Error (leads to larger trees):

$$1 - \operatorname{argmax}_k \hat{p}_k$$

- Gini-Index (probabilistic predictions):

$$\sum_{k \neq k'} \hat{p}_k \hat{p}_{k'} = \sum_{k=1}^K \hat{p}_k (1 - \hat{p}_k)$$

with \hat{p}_k = the relative frequency of class k in the leaf node

- Entropy (of the class attribute) measures unpredictability of the data:
 - How likely will random example have class k ?

$$E(X) = - \sum_{k=1}^K \hat{p}_k \log_2 \hat{p}_k$$

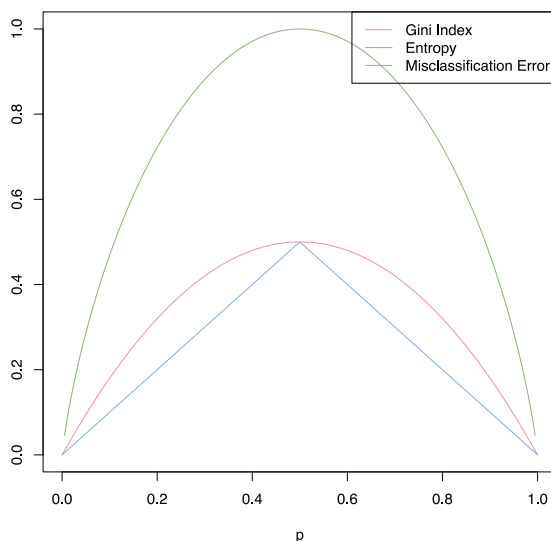
- Information Gain (a.k.a. Kullback–Leibler divergence) for choosing attribute X_i to split the data:

$$G(X, X_i) = E(X) - \sum_{v=1}^V \frac{|X_{i=v}|}{|X_i|} E(X_{i=v})$$

with \hat{p}_k = the relative frequency of class k in the leaf node, X = the training set, containing i features (variables) X_i , v a specific value for X_i , $X_{i=v}$ is the set of examples having value v for feature X_i : $\{x \in X | X_i = v\}$

For binary target (K=2):

```
In [37]: p <- seq(0, 1, length.out = 200)
entropy <- function(p) -((p * log2(p) + (1-p) * log2(1-p)))
gini <- function(p) 2 * p * (1-p)
misclassification <- function(p) (1 - max(p, 1-p))
plot(p, entropy(p), type = "l", col = Colors[2], ylab = "", ylim = c(0, 1))
lines(p, gini(p), col = Colors[1])
lines(p, sapply(p, misclassification), col = Colors[3])
legend("topright", c("Gini Index", "Entropy", "Misclassification Error"),
      col = Colors[1:3], lty = 1)
```



3.1.1 Example

Ex.	1	2	3	4	5	6
a1	T	T	T	F	F	F
a2	T	T	F	F	T	T
class	+	+	-	+	-	-

$E(X)$?

$G(X, X_{a2})$?

$G(X, X_{a1})$?

$E(X) = -(\frac{1}{2} * \log_2(\frac{1}{2}) + \frac{1}{2} * \log_2(\frac{1}{2})) = 1$ (classes have equal probabilities)

$G(X, X_{a2}) = 0$ (after split, classes still have equal probabilities, entropy stays 1)

Ex.	1	2	3	4	5	6
a1	T	T	T	F	F	F
a2	T	T	F	F	T	T
class	+	+	-	+	-	-

$$E(X) = - \sum_{k=1}^K \hat{p}_k \log \hat{p}_k \quad , \quad G(X, X_i) = E(X) - \sum_{v=1}^V \frac{|X_{i=v}|}{|X_i|} E(X_{i=v})$$

$$E(X_{a1=T}) = -\frac{2}{3} \log_2\left(\frac{2}{3}\right) - \frac{1}{3} \log_2\left(\frac{1}{3}\right) = 0.9183 \quad (= E(X_{a1=F}))$$

$$G(X, X_{a1}) = 1 - \frac{1}{2} 0.9183 - \frac{1}{2} 0.9183 = 0.0817$$

hence we split on a1

3.2 Regression

For regression ($x_i \rightarrow y_i \in \mathbb{R}$): Minimal quadratic distance

Dividing the data on split variable X_j at splitpoint s leads to the following half-spaces:

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

The best split variable and the corresponding splitpoint, with predicted value c_i and actual value Y_i :

$$\min_{j,s} \left(\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right)$$

Assuming that the tree predicts y_i as the average of all x_i in the leaf:

$$\hat{c}_1 = \text{avg}(y_i | x_i \in R_1(j, s)) \quad \text{and} \quad \hat{c}_2 = \text{avg}(y_i | x_i \in R_2(j, s))$$

with x_i being the i -th example in the data, with target value y_i

4 Optimization: how to grow the tree?

- Constructive search algorithm (like LEGO's): Start with one piece, add more pieces to build complex structure
- Greedy algorithm: Pick best features and splitpoint, from all features and all possible splitpoints

Options (depends on implementation): - Binary splits of multi-way splits - Criteria for the selection of a variable and its splitpoint(s) - Handling missing value - Stopping Criteria - Pruning (trimming leaves)

4.1 Binary or Multi-way splits

- Binary splits (e.g. CART algorithm)
 - No need to mind the penalization of multi-way splits when choosing the split criteria
 - Interpretability of the tree might suffer: multiple splits of the same variable in different levels

4.2 Selection of variable and splitpoints

- Choice of evaluation function (Information gain, Gini-index, ...)
- Which splits to consider (for numeric features)?
 - Only split between data points of different classes
 - Split at the point, or halfway between points of different classes

4.3 Handling missing values

Strategies:

- Treat ‘missing’ as its own (new) category
- Missing value imputation (e.g. matrix decomposition)
- When considering feature for a split, ignore data points with missing value
 - To pass examples down the tree (while learning or predicting), find surrogate feature producing similar splits

4.4 Handling many-valued features

What happens when a feature has (almost) as many values as examples? - Information Gain will select it
One approach: use Gain Ratio instead:

$$GainRatio(X, X_i) = \frac{Gain(X, X_i)}{SplitInfo(X, X_i)}$$

$$SplitInfo(X, X_i) = - \sum_{v=1}^V \frac{|X_{i=v}|}{|X|} \log_2 \frac{|X_{i=v}|}{|X|}$$

where $X_{i=v}$ is the subset of examples for which feature X_i has value v .

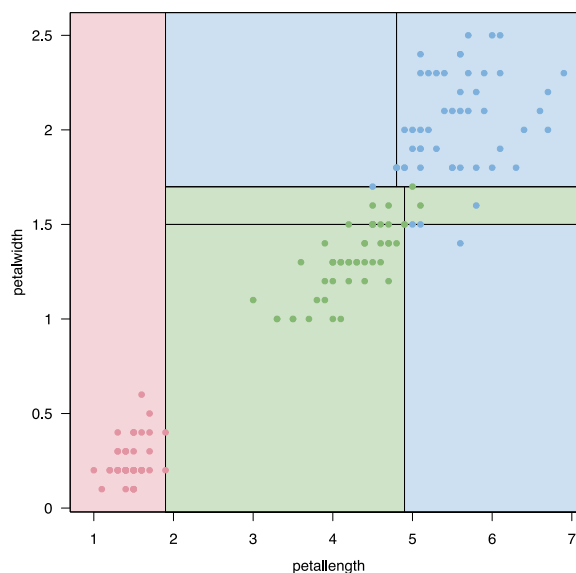
SplitInfo will be big if X_i fragments the data into many small subsets, resulting in a smaller Gain Ratio.

4.5 Avoiding overfitting

Complexity (and chance of overfitting) increase with size of the tree

- Controlled by regularization hyperparameters
 - Stopping criteria
 - Pruning

In [38]: `tree_iris <- ctree(class ~ ., data = iris, controls = ctree_control(mincriterion = 0, minsplitsplit
with(iris, plot_rectangles(tree_iris, petallength, petalwidth, class)))`



4.6 Stopping criteria

Choose:

- A minimal number of examples for node before splitting
- A minimal number of examples that must be in each leaf
- A minimal increase in goodness of fit that must be reached before splitting (confidence)
- A maximum depth (number of levels) in the tree

4.7 Pruning

- Pruning optimizes trade-off between goodness of fit and complexity by cutting leaves
- Different strategies. Can you think of one?

4.7.1 Reduced error pruning

- Post-pruning, simple and fast
- Split (training) data into training and validation set
- Starting at leaves, prune: replace each node with most popular class
- Greedily remove the one that most improves performance on validation set

4.7.2 Cost Complexity Pruning

Cost function R_α = training error + complexity term

$$R_\alpha = R(T) + \alpha \cdot \# \text{leaves}$$

where $R(T)$ represents the error of tree T on the training set

- Generates series of trees
- For every α there is a distinctly defined smallest sub-tree of the original tree, which minimizes the cost function
- $\hat{\alpha}$ can be assessed with cross-validation
- Final tree is fitted on the whole data, where $\hat{\alpha}$ is used to find the optimal size of the tree

4.8 Questions?

4.9 Acknowledgements

Thanks to Bernd Bischl for help with many of the code examples.