

Introduction to Machine Learning

NPFL 054

<http://ufal.mff.cuni.cz/course/npfl054>

Barbora Hladká
hladka@ufal.mff.cuni.cz

Martin Holub
holub@ufal.mff.cuni.cz

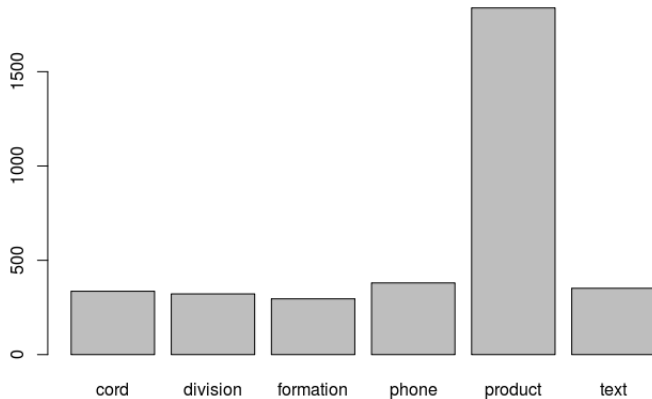
Charles University,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics

Outline

- **Entropy and conditional entropy**
 - definition, calculation, and meaning
 - application for feature selection
- **Generalization error — estimating and minimizing**
- **Decision Trees**
 - Heuristic algorithms for building Decision Trees
- **Random Forests**
 - bootstrapping extension of Decision Trees

WSD task — distribution of target class values

```
> examples <- read.table("wsd.development.csv", header=T)
> plot(examples$SENSE)
>
```



Amount of information contained in a value?

How much information do you gain when you observe a random event?

According to the **Information Theory**, **amount of information** contained in an event is given by

$$I = \log_2 \frac{1}{p} = -\log_2 p$$

where p is probability of the event occurred.

Thus, the lower probability, the more information you get when you observe an event (e.g. a feature value). If an event is certain ($p = 100\%$), then the amount of information is zero.

Amount of information in SENSE values

```
### probability distribution of SENSE
> round(table(examples$SENSE)/nrow(examples), 3)

      cord  division formation      phone  product      text
0.095    0.091    0.084    0.108    0.522    0.100
>

### amount of information contained in SENSE values
> round(-log2(table(examples$SENSE)/nrow(examples)), 3)

      cord  division formation      phone  product      text
3.391    3.452    3.574    3.213    0.939    3.324
>
```

What is the average amount of information that you get when you observe values of the attribute SENSE?

Entropy

The average amount of information that you get when you observe random values is

$$\sum_{\text{value}} \Pr(\text{value}) \cdot \log_2 \frac{1}{\Pr(\text{value})} = - \sum_{\text{value}} \Pr(\text{value}) \cdot \log_2 \Pr(\text{value})$$

This is what information theory calls *entropy*.

- Entropy of a random variable X is denoted by $H(X)$
 - or, $H(p_1, p_2, \dots, p_n)$ where $\sum_{i=1}^n p_i = 1$
- Entropy is a measure of the uncertainty in a random variable
 - or, measure of the uncertainty in a probability distribution
- The unit of entropy is bit; entropy says how many bits *on average* you necessarily need to encode a value of the given random variable

Properties of entropy

Normality

$$H\left(\frac{1}{2}, \frac{1}{2}\right) = 1$$

Continuity

$H(p, 1 - p)$ is a continuous function

Non negativity and maximality

$$0 \leq H(p_1, p_2, \dots, p_n) \leq H\left(\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n}\right)$$

Symmetry

$H(p_1, p_2, \dots, p_n)$ is a symmetric function of its arguments

Recursivity

$$H(p_1, p_2, p_3, \dots, p_n) = H(p_1 + p_2, p_3, \dots, p_n) + (p_1 + p_2)H\left(\frac{p_1}{p_1 + p_2}, \frac{p_2}{p_1 + p_2}\right)$$

Entropy of SENSE

Entropy of SENSE is 2.107129 bits.

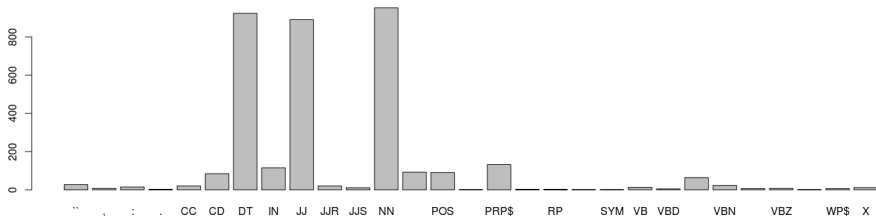
```
### probability distribution of SENSE
> p.sense <- table(examples$SENSE)/nrow(examples)
>
### entropy of SENSE
> H.sense <- - sum( p.sense * log2(p.sense) )
> H.sense
[1] 2.107129
```

The maximum entropy value would be $\log_2(6) = 2.584963$ if and only if the distribution of the 6 senses was uniform.

```
> p.uniform <- rep(1/6, 6)
> p.uniform
[1] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
>
### entropy of uniformly distributed 6 senses
> - sum( p.uniform * log2(p.uniform) )
[1] 2.584963
```

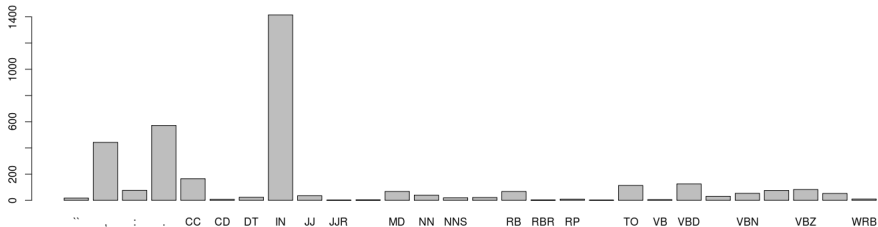

Distribution of feature values – A16

```
> levels(examples$A16)
[1] "" " ," ":" ". ." "CC" "CD" "DT" "IN" "JJ"
[10] "JJR" "JJS" "NN" "NNS" "POS" "PRP" "PRP$" "RB" "RP"
[19] "-RRB-" "SYM" "VB" "VBD" "VBG" "VBN" "VBP" "VBZ" "WDT"
[28] "WP$" "X"
> plot(examples$A16)
>
```



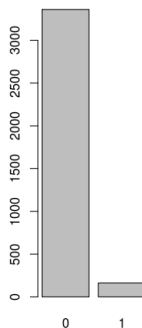
Distribution of feature values – A17

```
> levels(examples$A17)
[1] "``"      ", "      ":"      ". "      "CC"      "CD"      "DT"      "IN"      "JJ"
[10] "JJR"     "-LRB-"   "MD"      "NN"      "NNS"     "PRP"     "RB"      "RBR"     "RP"
[19] "-RRB-"   "TO"      "VB"      "VBD"     "VBG"     "VBN"     "VBP"     "VBZ"     "WDT"
[28] "WRB"
> plot(examples$A17)
>
```



Distribution of feature values – A4

```
> levels(examples$A4)
[1] "0" "1"
>
```



Entropy of features

Entropy of A16 is 2.78 bits.

```
> p.A16 <- table(examples$A16)/nrow(examples)
> H.A16 <- - sum( p.A16 * log2(p.A16) )
> H.A16
[1] 2.777606
```

Entropy of A17 is 3.09 bits.

```
> p.A17 <- table(examples$A17)/nrow(examples)
> H.A17 <- - sum( p.A17 * log2(p.A17) )
> H.A17
[1] 3.093003
```

Entropy of A4 is 0.27 bits.

```
> p.A4 <- table(examples$A4)/nrow(examples)
> H.A4 <- - sum( p.A4 * log2(p.A4) )
> H.A4
[1] 0.270267
```

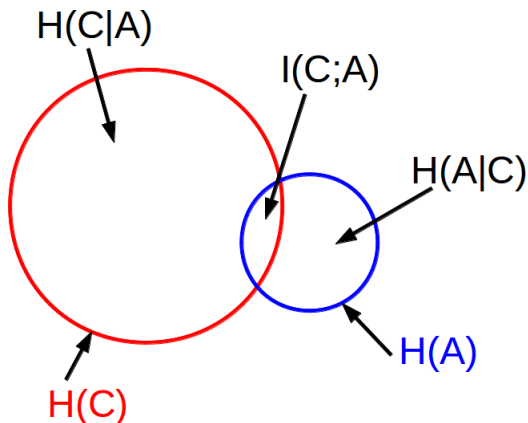
Conditional entropy $H(C | A)$

How much does target class entropy decrease if we have the knowledge of a feature?

The answer is **conditional entropy**:

$$H(C | A) = - \sum_{y \in C, x \in A} \Pr(y, x) \cdot \log_2 \Pr(y | x)$$

Conditional entropy and mutual information



WARNING

There are NO SETS in this picture! Entropy is a quantity, only a number!

Conditional entropy and mutual information

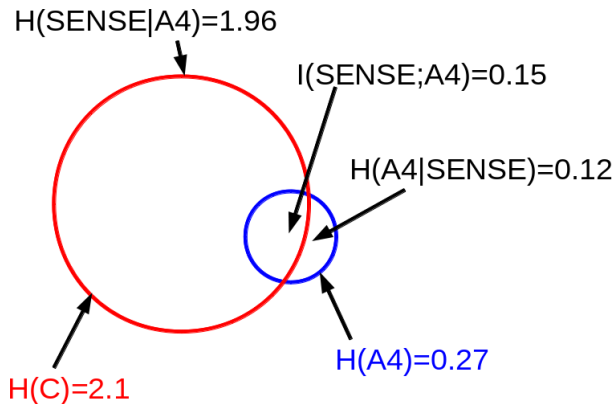
Mutual information measures the amount of information that can be obtained about one random variable by observing another.

Mutual information is a symmetrical quantity.

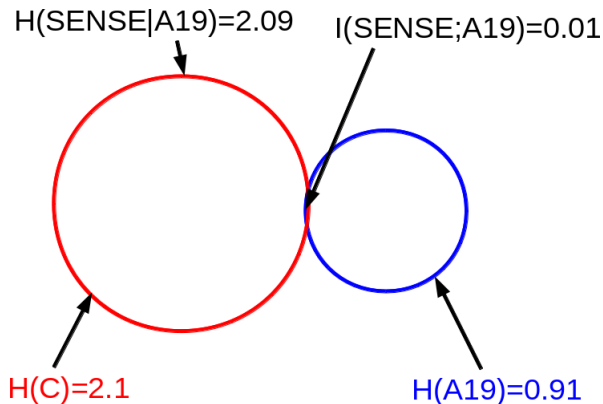
$$H(C) - H(C|A) = I(C;A) = H(A) - H(A|C)$$

Another name for mutual information is **information gain**.

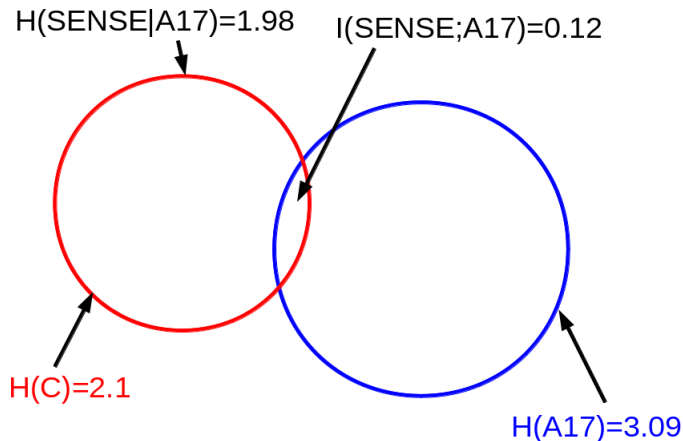
Conditional entropy – feature A4



Conditional entropy – feature A19



Conditional entropy – feature A17



User-defined functions in R

Structure of a user-defined function

```
myfunction <- function(arg1, arg2, ... ){  
  ... statements ...  
  return(object)  
}
```

Objects in a function are local to the function.

Example – a function to calculate entropy

```
> entropy <- function(x){  
+   p <- table(x) / NROW(x)  
+   return( -sum(p * log2(p)) )  
+ }  
>  
  
# invoking the function  
> entropy(examples$SENSE)  
[1] 2.107129
```

Summary

- **Information theory provides a measure** for comparing how the knowledge of features *statistically* contribute to the knowledge about target class.
- The lower conditional entropy $H(C | A)$, the better chance that A is a useful feature.
- However, since features typically interact, conditional entropy $H(C | A)$ should NOT be the only criterion when you do feature selection. You need experiments to see if a feature with high information gain really helps.

Note

Also, decision tree learning algorithm makes use of entropy when it computes purity of training subsets.

You do NOT have to submit it

- Write your own function for computing conditional entropy in R. New function `entropy.cond(x,y)` will take two factors of the same length and will compute $H(x|y)$.

Example use: `entropy.cond(examples$SENSE, examples$A4)`

You should understand and be able to explain and practically use

- entropy
 - motivation
 - definition
 - main properties
 - calculation in R
- conditional entropy
 - definition and meaning
 - relation to mutual information
 - calculation in R
 - information gain – application in feature selection

Sample accuracy and sample error rate

To measure the performance of classification tasks we often use (sample) *accuracy* and (sample) *error rate*

Sample accuracy is the number of correctly predicted examples divided by the number of all examples in the predicted set

Sample error rate is equal to $1 - \text{accuracy}$

Training error rate is the sample error rate measured on the training data set

Test error rate is the sample error rate measured on the test data set

Sample error and generalization error

Sample error of a hypothesis h with respect to a data sample S of the size n is usually measured as follows

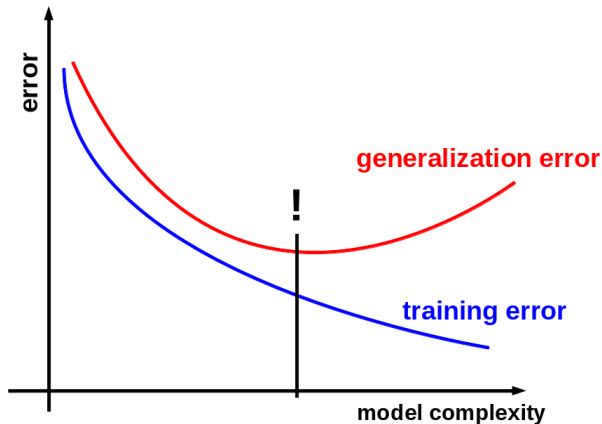
- for **regression**: **mean squared error** $\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$
- for **classification**: **classification error** $= \frac{1}{n} \sum_{i=1}^n \mathbb{I}(\hat{y}_i \neq y_i)$

Generalization error (aka “true error” or “expected error”) measures how well a hypothesis h generalizes beyond the used training data set, to unseen data with distribution \mathcal{D} . Usually it is defined as follows

- for **regression**: $\text{error}_{\mathcal{D}}(h) = \mathbb{E} (\hat{y}_i - y_i)^2$
- for **classification**: $\text{error}_{\mathcal{D}}(h) = \Pr(\hat{y}_i \neq y_i)$

Minimizing generalization error vs. overfitting

Finding a model that minimizes generalization error
... is one of central goals of the machine learning process



Learning a decision tree – key problems

Each node of a decision tree is associated with a subset of training data

Building a decision tree means to make a hierarchical sequence of splits. Each practical algorithm must be able to efficiently decide the following key questions:

- (1) How to choose a suitable splitting condition?**
- (2) When to stop the splitting process?**

A practical answer to problem (1) is to employ entropy or another similar measure. Each node is defined by an associated subset of examples with a specific distribution of target values. After a split, the entropy in child nodes should decrease in comparison with entropy in the parent node.

The splitting process should be duly stopped just to not produce model that overfits the training data. To avoid overfitting, practical implementations usually use pruning after building a relatively deep tree.

Historical excursion

Decision trees concept
(Hunt, 1962)



ID3 (Quinlan, 1979)



C4.5 (Quinlan, 1993)

AID (Morgan, 1964)



CART (Breiman, 1984)

- ID3 ~ Iterative Dichotomiser
- AID ~ Automatic Interaction Detection
- CART ~ Classification and Regression Trees

Probably most well-known is the “C5.0” algorithm (Quinlan), which has become the industry standard.

Packages in R: `rpart`

Building a classification tree from training data

We work with decisions on the value of only a single feature

- For each categorical feature A_j having values $Values(A_j) = \{b_1, b_2, \dots, b_L\}$

is $x_j = b_i?$ as $i = 1, \dots, L$

- For each categorical feature A_j

is $x_j \in$ a subset $\in 2^{Values(A_j)}$?

- For each numerical feature A_j

is $x_j \leq k?$, $k \in (-\infty, +\infty)$

Which decision is the best?

- Focus on the distribution of target class values in the associated subset of training examples.
- Then select the decision that splits training data into subsets as pure as possible.

Building a classification tree from training data

Which decision is the best?

We say a data set is **pure** (or **homogenous**) if it contains only a single class. If a data set contains several classes, then the data set is **impure** (or **heterogenous**).

Example:

| | | |
|---|--|---|
| $\oplus: 5, \ominus: 5$ | | $\oplus: 9, \ominus: 1$ |
| heterogenous high degree of impurity | | almost homogenous low degree of impurity |

Which decision is the best?

1. **Define** a candidate set S of splits at each node using possible decisions. $s \in S$ splits t into two subsets t_1 and t_2 .
2. **Define** the node proportions $p(y_j|t), j = 1, \dots, k$, to be the proportion of instances $\langle \mathbf{x}, y_j \rangle$ in t .
3. **Define** an **impurity measure** $i(t)$, i.e. **splitting criterion**, as a non-negative function Φ of the $p(y_1|t), p(y_2|t), \dots, p(y_k|t)$,

$$i(t) = \Phi(p(y_1|t), p(y_2|t), \dots, p(y_k|t)), \quad (1)$$

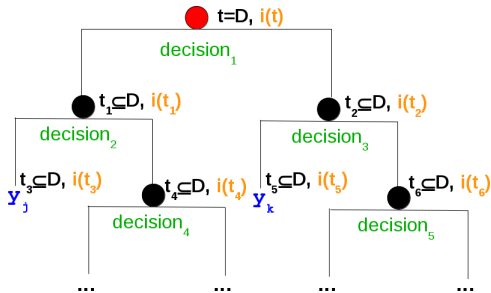
such that

- $\Phi(\frac{1}{k}, \frac{1}{k}, \dots, \frac{1}{k}) = \max$, i.e. the node impurity is largest when all examples are equally mixed together in it.
- $\Phi(1, 0, \dots, 0) = 0, \Phi(0, 1, \dots, 0) = 0, \dots, \Phi(0, 0, \dots, 1) = 0$, i.e. the node impurity is smallest when the node contains instances of only one class

Building a classification tree from training data

Which decision is the best?

4. Define the **goodness of split s** to be the decrease in impurity
$$\Delta i(s, t) = i(t) - (p_1 * i(t_1) + p_2 * i(t_2)),$$
where p_i is the proportion of instances in t that go to t_i .
5. Find split s^* with the largest decrease in impurity:
$$\Delta i(s^*, t) = \max_{s \in S} \Delta i(s, t).$$
6. Use splitting criterion $i(t)$ to compute $\Delta i(s, t)$ and get s^* .



Which decision is the best?

Splitting criteria – examples that are really used

- Misclassification Error – $i(t)_{ME}$
- Information Gain – $i(t)_{IG}$
- Gini Index – $i(t)_{GI}$

Building a classification tree from training data

Which decision is the best?

Splitting criteria — Misclassification Error $i(t)_{ME}$

$$i(t)_{ME} = 1 - \max_{j=1, \dots, k} p(y_j | t) \quad (2)$$

Example:

| | | | | |
|-------------|-------------------------|--------------------------|--------------------------|-------------------------|
| | $\oplus: 0, \ominus: 6$ | $\oplus: 1, \ominus: 5$ | $\oplus: 2, \ominus: 4$ | $\oplus: 3, \ominus: 3$ |
| $i(t)_{ME}$ | $1 - \frac{6}{6} = 0$ | $1 - \frac{5}{6} = 0.17$ | $1 - \frac{4}{6} = 0.33$ | $1 - \frac{3}{6} = 0.5$ |

Building a classification tree from training data

Which decision is the best?

Splitting criteria — Information Gain $i(t)_{IG}$

$$i(t)_{IG} = - \sum_{j=1}^k p(y_j|t) * \log p(y_j|t). \quad (3)$$

Recall the notion of entropy $H(t)$, $i(t)_{IG} = H(t)$.

$$Gain(s, t) = \Delta i(s, t)_{IG} \quad (4)$$

Building a classification tree from training data

Which decision is the best?

Splitting criteria — Gini Index $i(t)_{GI}$

$$i(t)_{GI} = 1 - \sum_{j=1}^k p^2(y_j|t) = \sum_{j=1}^k p(y_j|t)(1 - p(y_j|t)). \quad (5)$$

Interpretation

Use the rule that assigns an instance selected at random from the node to class i with probability $p(i|t)$. The estimated probability that the item is actually in class j is $p(j|t)$. The estimated probability of misclassification is the Gini index. In other words, Gini can be interpreted as expected error rate.

Building a classification tree from training data

Which decision is the best?

Splitting criteria – a comparison example

| | $\oplus: 0$ $\ominus: 6$ | $\oplus: 1$ $\ominus: 5$ | $\oplus: 2$ $\oplus: 4$ | $\oplus: 3$ $\oplus: 3$ |
|---------|-----------------------------|-----------------------------|----------------------------|----------------------------|
| Gini | 0 | 0.278 | 0.444 | 0.5 |
| Entropy | 0 | 0.65 | 0.92 | 1.0 |
| ME | 0 | 0.17 | 0.333 | 0.5 |

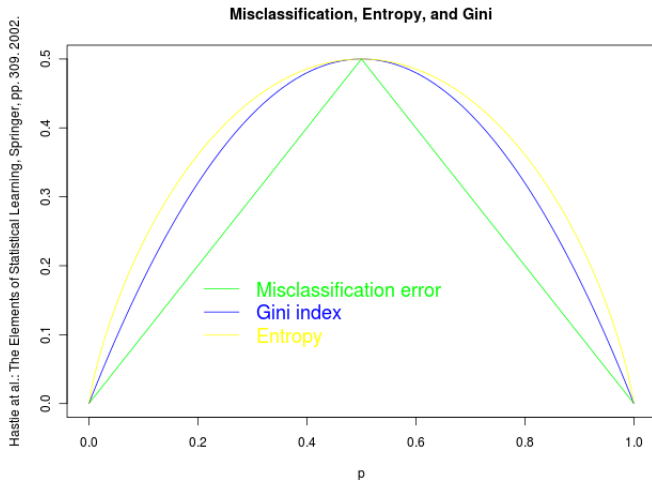
For two classes ($k = 2$), if p is the proportion of the class "1", the measures are:

- Misclassification error: $1 - \max(p, 1 - p)$
- Entropy: $-p * \log p - (1 - p) * \log(1 - p)$
- Gini: $2p * (1 - p)$

Building a classification tree from training data

Which decision is the best?

Splitting criteria



Building a *regression* tree from training data

Again, we work with decisions on the value of only a single feature

Which decision is the best?

Splitting criterion – usually used

- Squared Error – $i(t)_{SE}$

$$i(t)_{SE} = \frac{1}{|t|} \sum_{\mathbf{x}_i \in t} (y_i - y^t)^2,$$

where $y^t = \frac{1}{|t|} \sum_{\mathbf{x}_i \in t} y_i$.

Building decision tree from training data

When to stop the splitting process?

The recursive binary splitting is stopped when a stopping criterion is fulfilled. Then a leaf node is created with an output value.

Stopping criteria, e.g.

- the leaf node is associated with less than five training instances
- the maximum tree depth has been reached
- the best splitting criteria is not greater than a certain threshold

Building a decision tree from training data

How to avoid overfitting?

Overfitting can be avoided by

- applying a stopping criterion that prevents some sets of training instances from being subdivided,
- removing some of the structure of the decision tree after it has been produced.

Preferred strategy

Grow a large tree T_0 , stop the splitting process when only some minimum node size (say 5) is reached. Then prune T_0 using some pruning criteria.

Decision trees learning parameters

2 phases of decision tree learning:

- growing
- pruning

Learning parameters are used to control these two phases:

- when to stop growing
- how much to prune the tree

... to avoid overfitting and improve performance

Decision trees — implementation in R

There are two widely used packages in R

- `rpart`
- `tree`

The algorithms used are very similar.

References

- An Introduction to Recursive Partitioning Using the RPART Routines by Terry M. Therneau, Elizabeth J. Atkinson, and Mayo Foundation (available online)
- *An Introduction to Statistical Learning with Application in R* Chapters 8.1, 8.3.1, and 8.3.2 by Gareth James, Daniela Witten, Trevor Hastie and Rob Tibshirani (available online)
- R packages documentation — `rpart`, `tree` (available online)

Example heuristic — implementation in R

Learning parameters in `rpart()`

`rpart.control`

minsplit

- the minimum number of observations that must exist in a node in order for a split to be attempted

cp

- complexity parameter, influences the depth of the tree

... and others, see `?rpart.control`

T: try to set different `cp` and `minsplit` values in the M1 model learning and observe the resulting tree

Meaning of the cp parameter

- Any split that does not decrease the **relative training error** by a factor of cp is not attempted

⇒ That means, the learning algorithm measures for each split how it improves the tree relative error and if the improvement is too small, the split will not be performed.

Relative error is the error relative to the misclassification error (without any splitting relative error is 100%)

How to choose the optimal cp value?

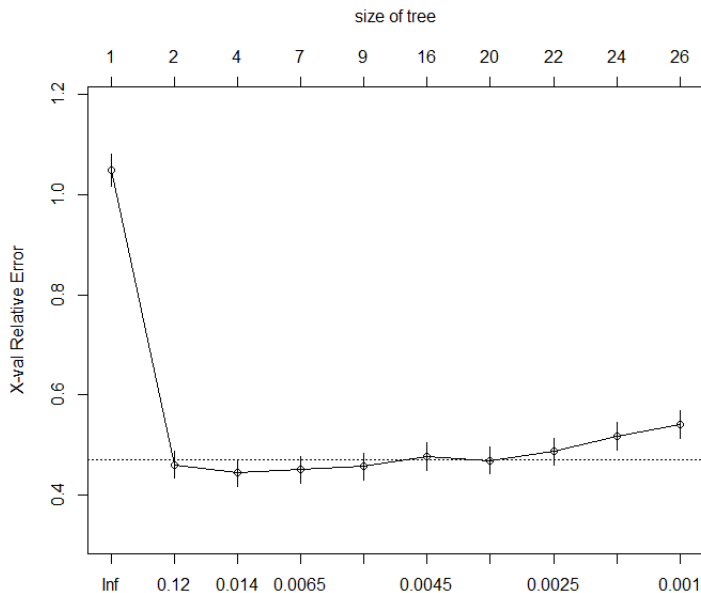
```
> m = rpart(profits ~ category + sales + assets + marketvalue,
            data=F[data.train, 1:8], cp=0.001)
> m$cptable
      CP nsplit rel error   xerror   xstd
1  0.543259557     0 1.0000000 1.0482897 0.03178559
2  0.027162978     1 0.4567404 0.4607646 0.02673551
3  0.007042254     3 0.4024145 0.4446680 0.02640028
4  0.006036217     6 0.3762575 0.4507042 0.02652763
5  0.005030181     8 0.3641851 0.4567404 0.02665301
6  0.004024145    15 0.3279678 0.4768612 0.02705703
7  0.003018109    19 0.3118712 0.4688129 0.02689795
8  0.002012072    21 0.3058350 0.4869215 0.02725122
9  0.001006036    23 0.3018109 0.5171026 0.02780383
10 0.001000000    25 0.2997988 0.5412475 0.02821490
```

rel error relative error on training data

xerror relative error in x-fold **cross-validation**

xstd standard deviation of xerror on x validation folds

How to choose the optimal cp value?



Decision Trees – weak spots

- **data splitting**
 - deeper nodes can learn only from small data portions
- **sensitivity to training data set (unstable algorithm)**
 - learning algorithm is called unstable if small changes in the training set cause large differences in generated models

Random Forests — an extension of Decision Trees

Resampling approach

Resampling can be used as a way to produce diversity among base learners

- Distribute the training data into K portions
- Run the learning process to get K different models
- Collect the output of the K models use a combining function to get a final output value

Bootstrapping principle

- New data sets $Data_1, \dots, Data_K$ are drawn from $Data$ with replacement, each of the same size as the original $Data$, i.e. n .
- In the i -th step of the iteration, $Data_i$ is used as a training set, while the examples $\{\mathbf{x} \mid \mathbf{x} \in Data \wedge \mathbf{x} \notin Data_i\}$ form the test set.
- The probability that we pick an instance is $1/n$, and the probability that we do not pick an instance is $1 - 1/n$. The probability that we do not pick it after n draws is $(1 - 1/n)^n \approx e^{-1} \doteq 0.368$.
- It means that for training the system will not use 36.8% of the data, and the error estimate will be pessimistic. So the solution is to repeat the process many times.

Random Forests

- an ensemble method based on decision trees and bagging
- builds a number of random decision trees and then uses voting
- introduced by L. Breiman (2001), then developed by L. Breiman and A. Cutler
- very good (state-of-the-art) prediction performance
- a nice page with description
`www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm`
- important: Random Forests helps to
 - avoid overfitting (by random sampling the training data set)
 - select important/useful features (by random sampling the feature set)

Building Random Forests

The algorithm for building a tree in the ensemble

- ① Having a training set of the size n , sample n cases at random – with replacement, and use the sample to build a decision tree.
- ② If there are M input features, choose a less number $m \ll M$. When building the tree, at each node a random sample of m features is selected as split candidates from the full set of M available features. Then the best split on these m features is used to split the node. A fresh sample of m features is taken at each split.
 - m is fixed for the whole procedure
- ③ Each tree is grown to the largest extent possible. There is no pruning.

**The more trees in the ensemble, the better.
There is no risk of overfitting!**

R packages for Random Forests

- **randomForest**: Breiman and Cutler's random forests for classification and regression
 - Classification and regression based on a forest of trees using random inputs.
- **RRF**: Regularized Random Forest
 - Feature Selection with Regularized Random Forest. This package is based on the 'randomForest' package by Andy Liaw. The key difference is the RRF function that builds a regularized random forest.
 - <http://cran.r-project.org/web/packages/RRF/index.html>
- **party**: A Laboratory for Recursive Partytioning
 - a computational toolbox for recursive partitioning
 - `cforest()` provides an implementation of Breiman's random forests
 - extensible functionality for visualizing tree-structured regression models is available

Examination requirements

- You should understand the basic ideas of building and using Decision Trees for both classification and regression task.
 - Decision Trees – splitting criteria: typical heuristics
 - Decision Trees – pruning and overfitting: the complexity parameter
 - Decision Trees – practical use of the `rpart()` package
- You should understand Random Forests, which is an important and effective extension of simple Decision Trees.
- You should be able to practically use `rpart()` and `randomForest()` packages in R.
- Also, later we will discuss Random Forests again, in connection with more general ensemble methods.

References – more details on Decision Trees

- Breiman Leo, Friedman Jerome H., Olshen Richard A., Stone Charles J. *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.
- Hunt, E. B. *Concept Learning: An Information Processing Problem*, Wiley. 1962.
- Morgan, J. N., Sonquist, J. A. Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association* 58, pp. 415–434. 1963.
- Quinlan, J. R. Discovering rules from large collections of examples: A case study, in D. Michie, ed., *Expert Systems in the Micro Electronic Age*. Edinburgh University Press. 1979.
- Quinlan, J. R. *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, California. 1993.