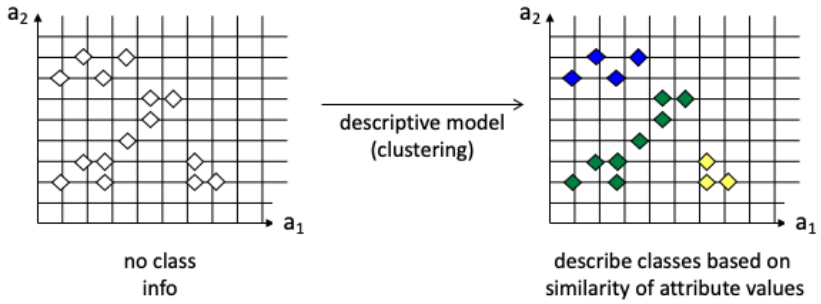
4. CLASSIFICATION

Clustering and Classification

Given a dataset of *objects* described by *attributes*, build a model that assigns objects to a *class (or label)*



no class info

describe classes based on similarity of attribute values

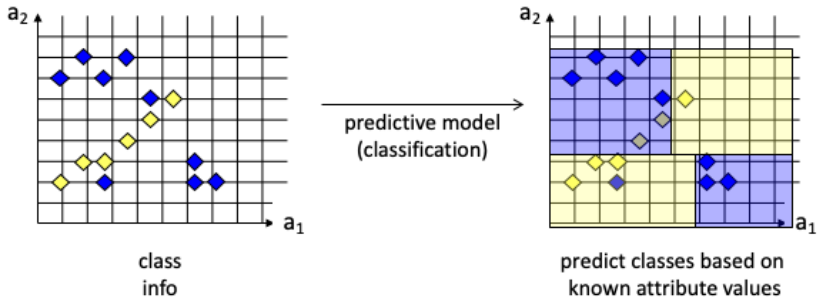
For inferring global models of data collections there exist two types of approaches: descriptive and predictive modeling. We illustrate the difference among them by an example.

We assume that a set of data items (or objects) with two attributes a1 and a2 is given. Assume the global model we are interested in is a classification (or as often said labeling) of the data items.

In descriptive modeling we just know the data items, as indicated by the points in the 2-dimensional grid. A descriptive modeling technique, such as clustering, produces classes, which are not known in advance. For doing this it relies on some criteria that specify when two data items probably belong to the same class. Such a criteria is usually based on a similarity measure.

Clustering and Classification

Given a dataset of *objects* described by *attributes*, build a model that assigns objects to a *class*



class info

predict classes based on known attribute values

A predictive modeling technique, such as classification, starts from a given classification (or labeling) of data items. Using that classification of the dataset the classification method infers conditions on the properties of the data objects, that allow to predict the membership to a specific class. For example, the prediction could be based on a partitioning of the attribute values along each dimension, as shown in the figure on the right. There, first attribute a1 is partitioned into two intervals, and for each of the intervals a different partitioning of the attribute a2 is used to determine the regions corresponding to classes. Misclassifications may occur as seen in the example.

Classification Problem

Input: set of objects with categorical/numerical attributes and one class label

Output: A model that returns the class label given the object attributes

- Model is a function represented as rules, decision trees, formulae, neural networks

Classification belongs to *supervised* ML

- Objects have class information

Classification creates a **global model**, that is used for predicting the class label of unknown data. Since the classification function is learned from existing data, this approach is also called a supervised learning approach.

Classification is clearly useful in many decision problems, where for a given data item a decision is to be made (which depends on the class to which the data item belongs). Classification is also often called predictive analytics.

Classification: Basic Approach

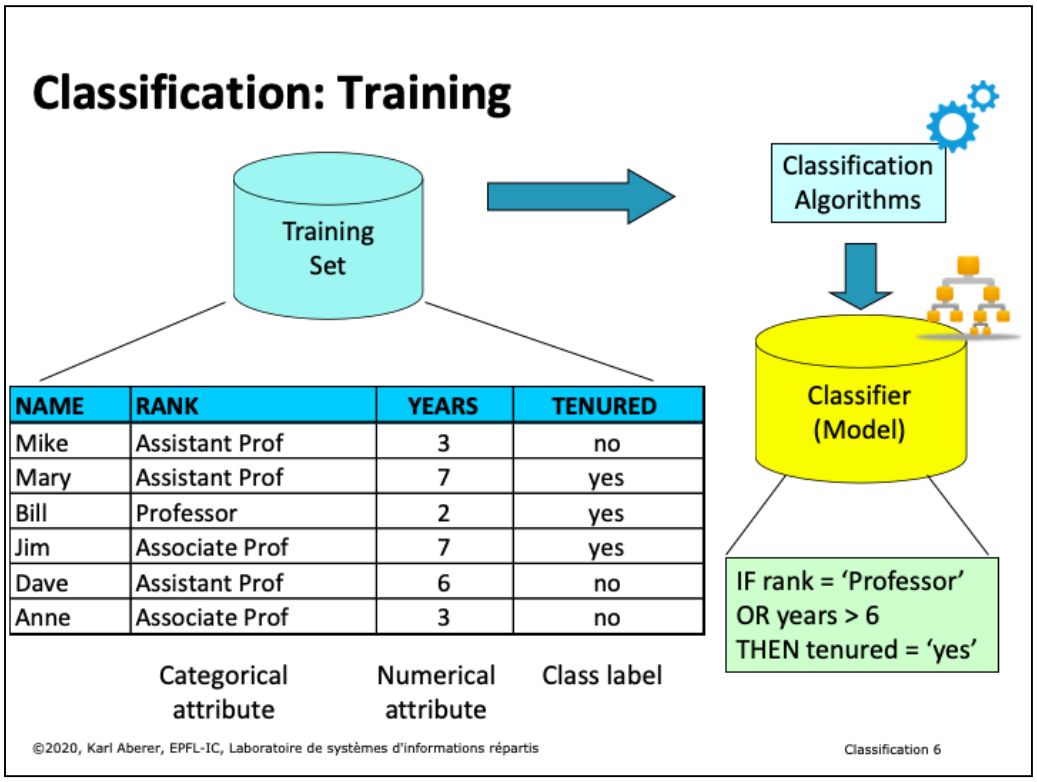
Model is learnt from a set of objects with known labels: **training set**

The quality of the model is evaluated by comparing the predicted class labels with those from a set of objects with known labels: **test set**

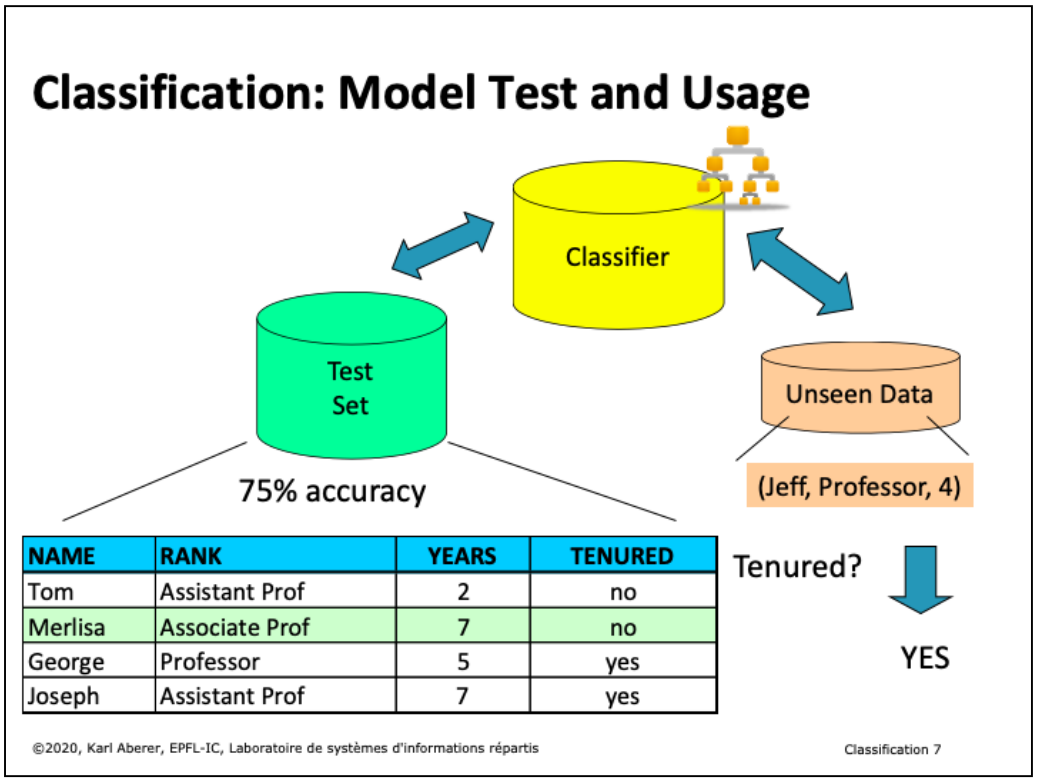
- Test set is independent of training set, otherwise over-fitting will occur

The model is applied to data with unknown labels: **prediction**

In order to test the quality of a model it is tested by using a test set. Assuming that a set of objects with known labels is available, it is split into (typically larger) set that is used for learning model, which is the training set, and a (typically smaller) set of that is used to test the quality of the model, the test set.



In classification the classes are known and given by the class label attributes. In this example, for the given data collection TENURED would be the class label attribute. The goal of classification is to determine rules on the other attributes that allow to predict the class label attribute, as the one shown right on the bottom.



In order to determine the quality of the rules derived from the training set, the test set is used. We see that the classifier that has been found is correct in 75% of the cases. If rules are of sufficient quality they are used in order to classify data that has not been seen before. Since the reliability of the rule has been evaluated as 75% by testing it against the test set and assuming that the test set is a representative sample of all data, then the accuracy of the rule applied to unseen data should be the same.

Classification: Problem Formulation

Problem

Given a database D with n data items described by d categorical/numerical attributes and one categorical attribute (class label C)

Find

A function f: $X^d \rightarrow C$

rules
decision tree
formula

Such that

classifies *accurately* the items in the *training* set

generalises well for the (unknown) items in the *test* set

We formulate the classification problem formally.

Characteristics of Classification Methods

Predictive accuracy

Speed and scalability

- Time to build the model
- Time to use the model
- In memory vs. on disk processing

Robustness

- Handling noise, outliers and missing values

Interpretability

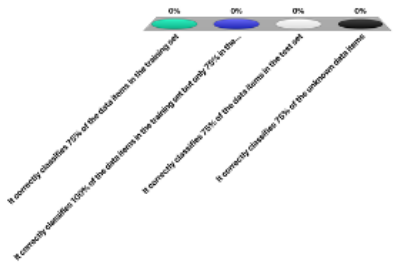
- Understanding the model and its decisions (black box) vs. white box
- Compactness of the model

As for clustering methods, we can also identify for classification methods a range of criteria to assess and compare the properties of different approaches.

Predictive accuracy is the natural main objective to optimize for a classifier. It characterizes how well the classifier performs its job. Often we encounter a trade off between predictive accuracy and the speed and scalability of the method. Methods that achieve high accuracy tend also to be more expensive. As for clustering, also for classification noise and outliers can pose additional problems affecting accuracy. Finally, a very important criterion is the interpretability of the model. In many concrete applications, it is critical that humans are able to understand based on which criteria a classifier takes a decision, e.g. for accountability. Imagine a case, where an assurance policy is refused based on the decision taken by a classifier, and the client would oppose in court. Only with human-interpretable methods the decision could be argued for. However, the most powerful classifiers today tend to produce models that are very hard to interpret for humans, as they represent very complex functions.

If a classifier has 75% accuracy, it means that ...

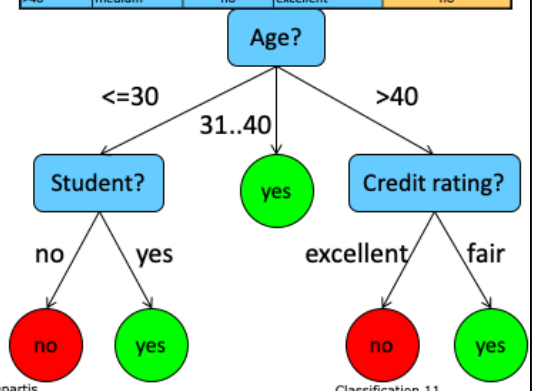
A. It correctly classifies 75% of the data items in the training set
B. It correctly classifies 100% of the data items in the training set but only 75% in the test set
C. It correctly classifies 75% of the data items in the test set
D. It correctly classifies 75% of the unknown data items



Decision Trees

- Nodes are tests on a single attribute
- Branches are attribute values
- Leaves are marked with class labels

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no



A standard type of classification function is a decision tree. In a basic decision tree, at each level one of the available attributes is used to partition the data set based on the different attribute values. At the leaf level of the decision tree, the values of the class label attribute are found. Thus, for a given data item with unknown class label attribute, by traversing the tree from the root to the leaf according to its data values, its class can be determined by choosing the class label found at the leaf level. Note that in different branches of the tree, different attributes may be used for classification.

A decision tree is constructed in a top-down manner, by recursively splitting the training set using conditions on the attributes. How these conditions are determined is one of the key questions for decision tree induction. After the decision tree construction, it may occur that at the leaf level the granularity is too fine, i.e., many leaves correspond to outliers in the data. Thus, in a second phase such leaves are identified and eliminated.

The key problem in constructing a decision tree is thus to determine the attributes that are used to partition the data set at each level of the decision tree.

Decision Tree Induction: Algorithm

Tree construction (top-down divide-and-conquer strategy)

- At the beginning, all training samples belong to the root
- Examples are partitioned recursively based on a selected "most discriminative" attribute
- Discriminative power determined based on information gain (ID3/C4.5)

Partitioning stops if

- All samples belong to the same class → assign the class label to the leaf
- There are no attributes left → majority voting to assign the class label to the leaf
- There are no samples left

The basic algorithm for decision tree induction proceeds in a greedy manner. First the set of all data objects are associated with the root. Among all attributes one is chosen to partition the set. The criterion that is applied to select the attribute is based on measuring the information gain that can be achieved, or how much uncertainty on the classification of the data objects is removed by the partitioning.

Three conditions can occur such that no further partitions can be performed:

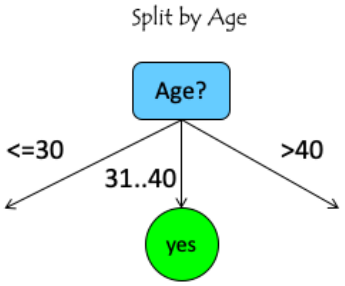
(1) all data objects are in the same class, therefore further splitting makes no sense,

(2) no attributes are left which can be used to split. Still data objects from different classes can be in the leaf, then majority voting is applied.

(3) no data objects are left.

Example: Decision Tree Induction

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

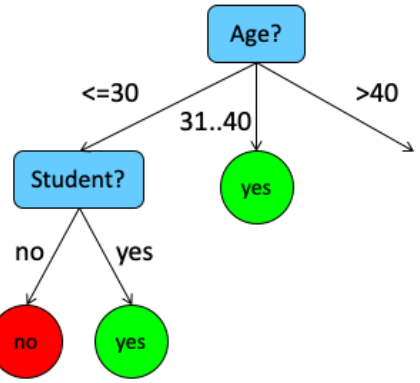


All samples belong to same class

Based on this approach for attribute selection, we can now illustrate the induction of the decision tree. In a first step, age is chosen for a split. The partition 31..40 contains after the split only instances from one class, the positive class, thus for this branch of the tree the induction terminates.

Example: Decision Tree Induction

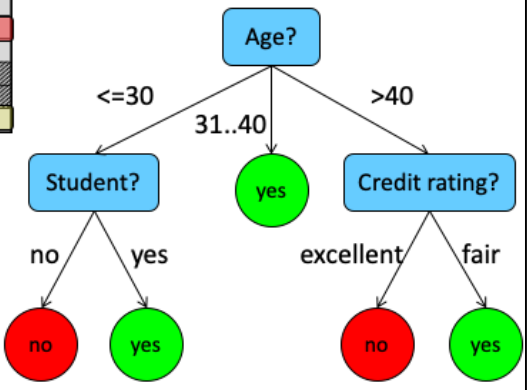
age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31..40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31..40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31..40	medium	no	excellent	yes
31..40	high	yes	fair	yes
>40	medium	no	excellent	no



For the partition <= 30 we find that the student attribute is the best to be chosen for further splitting. Further splitting makes no more sense, as the two resulting partitions, after splitting by the student attribute, are consisting of either positive or negative instances only.

Example: Decision Tree Induction

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31..40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31..40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31..40	medium	no	excellent	yes
31..40	high	yes	fair	yes
>40	medium	no	excellent	no



Similarly, for the partition >40 we find that credit rating gives the largest information gain. As before, further splitting is no more needed, as the resulting partitions contain only positive respectively negative instances.

Attribute Selection

At a given branch in the tree, the set of samples S to be classified has P positive and N negative instances

The entropy of the set S is

$$H(P, N) = -\frac{P}{P+N}\log_2\frac{P}{P+N} - \frac{N}{P+N}\log_2\frac{N}{P+N}$$

Note

- If P = 0 or N = 0 H(P, N) = 0 → no uncertainty
- If P = N H(P, N) = 1 → maximal uncertainty

Now we introduce the approach to select the split attributes during the construction of a decision tree. It is shown for the case of binary classification, and generalizes naturally to multiple class labels.

The approach is based on an information-theoretic argument. Assuming that we have a binary category, i.e., two classes **P** and **N** to which objects in S have to be assigned, we can compute the amount of information required to determine the class, by H(P, N), the standard entropy measure, where P and N denote the cardinalities of the classes **P** and **N**. Given an attribute A that can be used for partitioning the data collection, we can calculate the amount of information needed to classify the data after the split according to attribute A has been performed. This value is obtained by calculating H(P, N) for each of the partitions and weighting these values by the probability that a data item belongs to the respective partition.

Attribute Selection: Example

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$H(P,N) = H(9, 5) = 0.94$

Age	[<=30]	H(2, 3) = 0.97	Income	[high]	H(2, 2) = 1
Age	[31...40]	H(4, 0) = 0	Income	[med]	H(4, 2) = 0.92
Age	[>40]	H(3, 2) = 0.97	Income	[low]	H(3, 1) = 0.81
Student	[yes]	H(6, 1) = 0.59	Rating	[fair]	H(6, 2) = 0.81
Student	[no]	H(3, 4) = 0.98	Rating	[exc]	H(3, 3) = 1

We illustrate the attribute selection process now for our running example. Initially the data contains P = 9 positive instances and N = 5 negative instances. This results in an entropy of 0.94, i.e. 0.94 bits are required to decide the class of one instance. We compute next the entropies of all partitions that result from splitting all attributes. For example, if we split for Age, we obtain 3 partitions, each with a different distribution of positive and negative instances; and thus with different entropies.

Attribute Selection: Information Gain

Attribute A partitions S into S_1, S_2, ... S_v

Entropy of attribute A is

$$H(A) = \sum_{i=1}^{v} \frac{P_i + N_i}{P + N} H(P_i, N_i)$$

The information gain obtained by splitting S using A is

$$Gain(A) = H(P,N) - H(A)$$

Gain(Age) = 0.94 – 0.69 = 0.25 ← split on age

Gain(Income) = 0.94 – 0.91 = 0.03

Gain(Student) = 0.94 – 0.78 = 0.16

Gain(Rating) = 0.94 – 0.89 = 0.05

The information gained by a split can thus be determined as the difference of the amount of information needed for correct classification before and after the split. Thus we calculate the reduction in uncertainty that is obtained by splitting according to attribute A and select among all possible attributes the one that leads to the highest reduction. For our example we can conclude that it is best to split on attribute age.

Attribute Selection: Example

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$H(P,N) = H(9, 5) = 0.94$

$H_{Age} = p([<=30]) \cdot H(2, 3) + p([31...40]) \cdot H(4, 0) + p([>40]) \cdot H(3, 2) =$
$= 5/14 \cdot 0.97 + 4/14 \cdot 0 + 5/14 \cdot 0.97 = 0.69$

$H_{Income} = p([high]) \cdot H(2, 2) + p([med]) \cdot H(4, 2) + p([low]) \cdot H(3, 1) =$
$= 4/14 \cdot 1 + 6/14 \cdot 0.92 + 4/14 \cdot 0.81 = 0.91$

$H_{Student} = p([yes]) \cdot H(6, 1) + p([no]) \cdot H(3, 4) = 7/14 \cdot 0.59 + 7/14 \cdot 0.98 = 0.78$

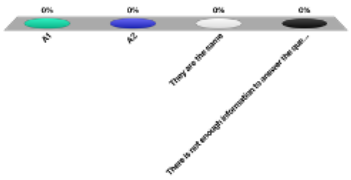
$H_{Rating} = p([fair]) \cdot H(6, 2) + p([exc]) \cdot H(3, 3) = 8/14 \cdot 0.81 + 6/14 \cdot 1 = 0.89$

Next we compute the weighted sum of all entropies of the partitions in a split. The weights correspond to the probability of an instance falling into an element of the partition. Computing this for all attributes shows, that the attribute H results in the lowest entropy, i.e., leaves the lowest remaining uncertainty about the class membership of instances after the split.

Given the distribution of positive and negative samples for attributes A_1 and A_2, which is the best attribute for splitting?

A_1	P	N
a	2	2
b	4	0
A_2	P	N
x	3	1
y	3	1

A. A1
B. A2
C. They are the same
D. There is not enough information to answer the question



Pruning

The construction phase does not filter out noise → **overfitting**

Pruning strategies

- Stop partitioning a node when large majority of samples is positive or negative, i.e., $\frac{N}{N+P}$ or $\frac{P}{N+P} > 1 - \varepsilon$
- Build the full tree, then replace nodes with leaves labelled with the majority class, if classification accuracy does not change
- Apply Minimum Description Length (MDL) principle

A common problem in classification is that a classifier may overspecialize and capture noise and outliers in the data, rather than general properties. One possibility to limit overspecialization would be to stop the partitioning of tree nodes when some specific criteria is met (e.g., number of samples assigned to the leaf node). A possible criterion is to stop partitioning when the majority of remaining samples falls into one class. However, in general it is difficult to specify a suitable criterion a priori (e.g. choosing the right value of epsilon).

Another alternative is to first build the complete classification tree, and then, in a second phase, prune subtrees that do not contribute to an efficient classification. Different approaches can be applied to that end: heuristic approaches can identify subtrees that do not contribute to the classification accuracy, and eliminate those. A more principled approach is the use of the minimum description length principle. (MDL).

Minimum Description Length Pruning

Let $M_1, M_2, ..., M_n$ be a list of candidate models (i.e., trees). The best model is the one that minimizes

$$L(M) + L(D|M)$$

where

- L(M) is the length of the description of the model in bits (#nodes, #leaves, #arcs ...)
- L(D|M) is the is the length of the description of the data when encoded with the model in bits (#misclassifications)

The MDL is based on the following consideration: if the effort in order to specify a class (the implicit description of the class extension through a decision tree) exceeds the effort to enumerate all class members (the explicit description of the class by enumerating its extension), then the subtree is over classifying and non-optimal. To measure the description cost a suitable metrics for the encoding cost, both for trees and data sets is required. For trees this can be done by suitably counting the various structural elements needed to encode the tree (#nodes, #test predicates, # arcs), whereas for explicit classification, it is sufficient to count the number of misclassifications that occur in a tree node.

Extracting Classification Rules from Trees

Represent the knowledge in the form of IF-THEN rules

- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction
- The leaf node holds the class prediction

Rules are easier for humans to understand

Example

IF *age* = "<=30" AND *student* = "*no*"	THEN *buys_computer* = "*no*"
IF *age* = "<=30" AND *student* = "*yes*"	THEN *buys_computer* = "*yes*"
IF *age* = "31…40"	THEN *buys_computer* = "*yes*"
IF *age* = ">40" AND *credit_rating* = "*excellent*"	THEN *buys_computer* = "*yes*"
IF *age* = ">40" AND *credit_rating* = "*fair*"	THEN *buys_computer* = "*no*"

A decision tree can also be seen as an implicit description of a set of classification rules. Classification rules represent the classification knowledge as IF-THEN rules and are easier to understand for human users. They can be easily extracted from the classification tree as described.

Decision Trees: Continuous Attributes

With continuous attributes we cannot have a separate branch for each value

- use **binary decision trees**

Binary decision trees

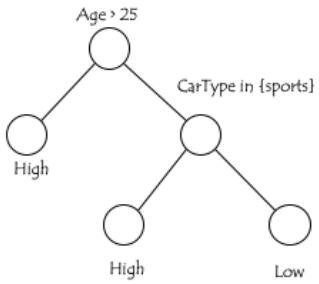
- For continuous attributes A a split is defined by val(A) < X
- For categorical attributes A a split is defined by a subset X $\subseteq$ domain(A)

With continuous attributes it does not make sense to create a separate path in the decision tree for every possible attribute value. Instead, in such a case, a binary decision tree is constructed. Binary decisions can be specified both fro continuous and categorical attributes. For continuous attributes, the binary split is performed by selecting a threshold that separates the instances in those that have a larger and a smaller value than the threshold. For categorical attributes, a subset of attribute values can be chosen that distinguishes the instances in two subsets.

Example: Binary Decision Tree

tid	Age	Car Type	Risk
0	23	family	high
1	17	sports	high
2	43	sports	high
3	68	family	low
4	32	truck	low
5	20	family	high

Training Data



Decision Tree

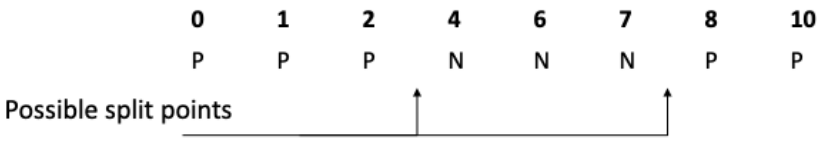
This example shows a dataset with both categorical and continuous attributes and a possible binary decision tree for such a dataset. The class label in the example is Risk.

Splitting Continuous Attributes

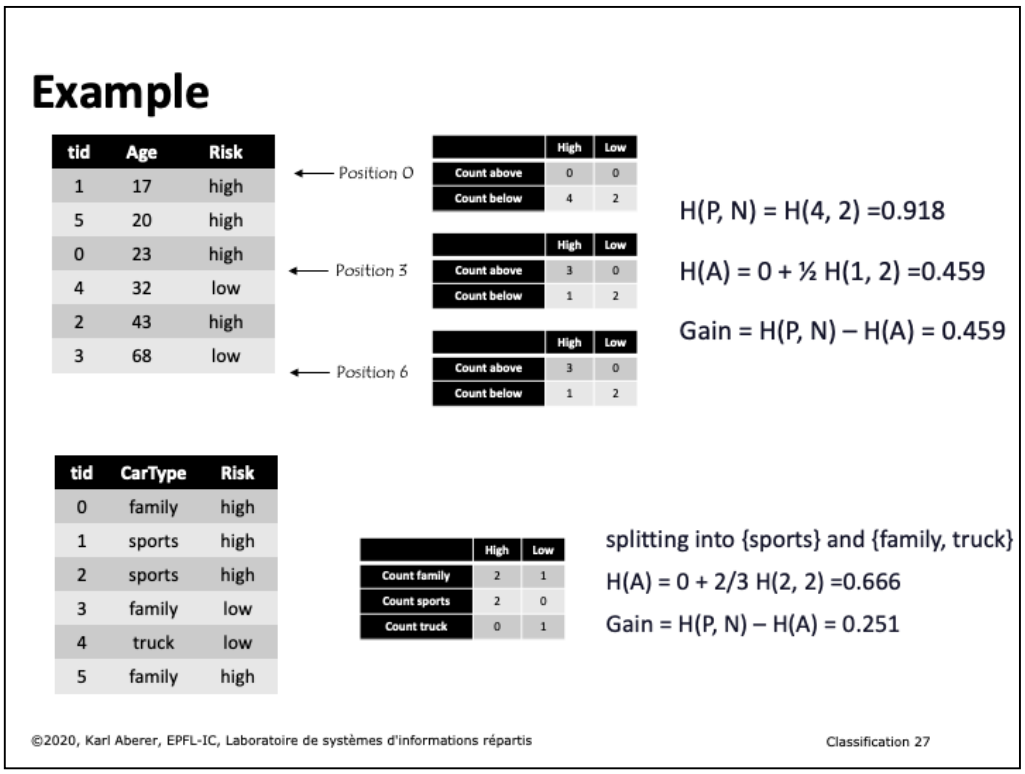
Approach

- Sort the data according to attribute value
- Determine the value of X which maximizes information gain by scanning through the data items

Only if the class label changes, a relevant decision point exists



When splitting the dataset using a continuous attribute, we need to determine which is the optimal value to split the dataset based on this attribute. To that end, first the set of attribute values is sorted. Then the class labels are traversed, and whenever it changes a possible split point is found (it can be shown that splitting where class labels do not change is provably sub-optimal). At these points the information gain needs to be computed.



This example illustrates the principle of how splitting is performed both for continuous and categorical attributes.

For reasons we will discuss later, we construct separate attribute lists for each attribute, that is used for classification. The attribute list contains the attribute for which it is constructed (i.e. Age and Car Type), the class label attribute and the transaction identifier tid. The attribute list is sorted for continuous attributes.

Now let us see of how a split point is found for the continuous attribute Age. The distribution of the class attribute for the whole data set (4 High and 2 Low) is stored in variables C_above and a pointer is positioned on top of the attribute list. Then the pointer is moved downwards. Whenever the class label changes the values C_below and C_above are updated (such that they always keep the distribution of H and L values above and below the pointer). At this step the information gain is computed, if the split were performed at that point (in the same way as done for categorical attributes before). After passing through the attribute list the optimal split value for the Age attribute is known.

For the categorical attribute we have to establish a statistics of the distribution of the classes for each of the possible attribute values and store it in a matrix. Then we check the information gain that can be obtained for each of the possible subsets of attribute values and thus determine the optimal "split" for the categorical attribute. Note that this method will be very inefficient with attributes that have large numbers of different values.

Finally, the attribute is chosen that results in the best (binary) split.

Scalability of Continuous Attribute Splits

Naive implementation

- At each step the data set is split in subsets that are associated with a tree node

Problem

- For evaluating which continuous attribute to split, data needs to be sorted according to these attributes
- Becomes dominating cost

In a naïve implementation of the splitting process, we would keep all data in a single table. This would imply that we would have for traversing the attributes in order to resort that table every time an attribute is investigated. Therefore, a more efficient approach is needed.

Scalability of Continuous Attribute Splits

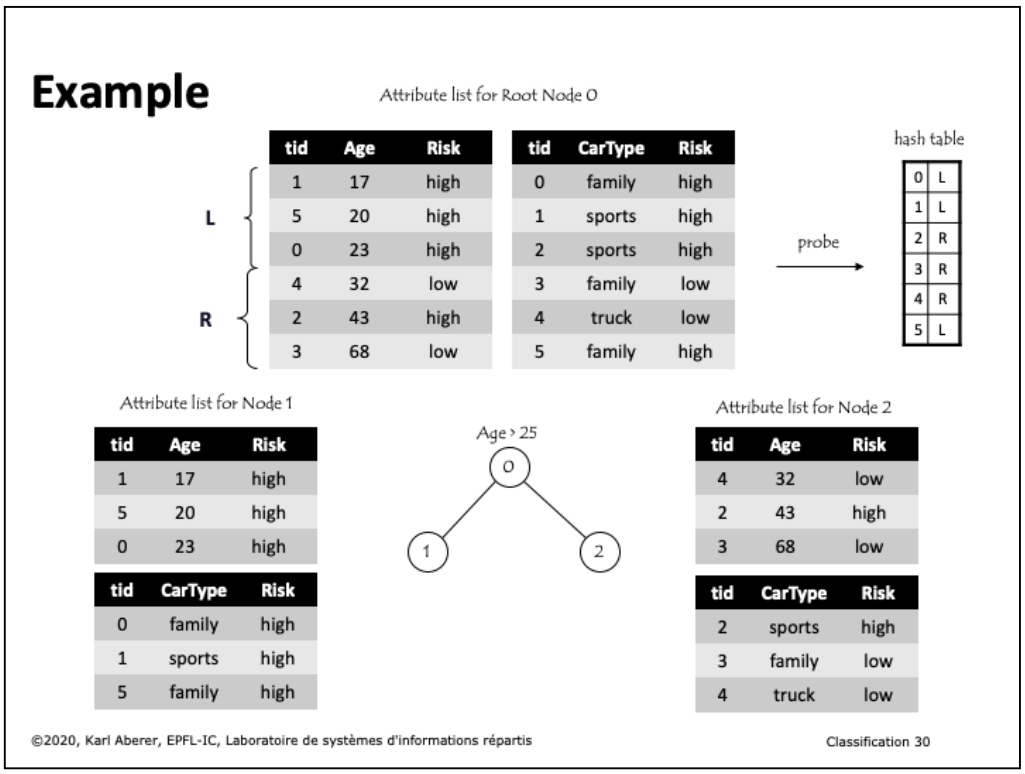
Idea: Presorting of data and maintaining order throughout tree construction

- Requires separate sorted attribute tables for each attribute

Updating attribute tables

- Attribute used for split: splitting attribute table straightforward
- Other attributes
  - Build Hash Table once associating tuple identifiers (TIDs) of data items with partitions
  - Select data from other attribute tables by scanning and probing the hash table

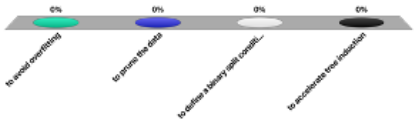
To avoid repeated resorting of data, for every attribute a separate and presorted table is kept. Once a split is chosen, we find two different situation. For the table that keeps the attribute that was used in the split, the table needs just to be partitioned into two subtables, maintaining the order. For the other attributes we have to select the subtables corresponding to the instances of the two partitions that have been formed. To that end a temporary hash table is constructed that allows to associate to each dataitem its partition. Then the attribute table is scanned and partitioned using the hashtable to decide for each entry to which partition it belongs. Note that in this approach, for continuous attributes, the resulting tables are again sorted as the order is preserved from the original table.



In this example we demonstrate of how attribute tables are split, when a decision node is introduced in the decision tree. Since the split is based on attribute Age, the table for Age can simply be split into two subtables at the threshold value. For the Car Type table we use the temporary hash table indicating partition membership to separate it into two subtables.

When splitting a continuous attribute, its values need to be sorted …

A. to avoid overfitting
B. to prune the data
C. to define a binary split condition
D. to accelerate tree induction



Characteristics of Decision Tree Induction

Strengths

- Automatic feature selection
- Minimal data preparation
- Non-linear model
- Easy to interpret and explain

Weaknesses

- Sensitive to small perturbation in the data
- Tend to overfit
- No incremental updates

We summarize here some of the major strengths and weaknesses of standard decision tree induction.

Decision trees advantages

- The information theoretic criteria used to select the most discriminative attribute is an embedded feature selection
- No data preparation is needed, such as normalisation of data
- The best aspect of using trees for analytics is that they are easy to interpret and explain, while more sophisticated ML algorithms (ANN, SVM) are seen as black-boxes that do not “explain” the classification decisions they make

Decision trees drawbacks

- They can be extremely sensitive to small perturbations in the data: a slight change can result in a drastically different tree.
- They can easily overfit. This can be compensated by validation methods and pruning, but remains a problem.
- They are not incremental. If new data is available, the existing tree cannot be incrementally modified, but the whole tree must be reconstructed from scratch

Decision Tree Induction: Properties

Model: flow-chart like tree structure

Score function: classification accuracy

Optimisation: top-down tree construction + pruning

Data Management: avoiding sorting during splits

Classification Algorithms

Decision tree induction is a (well-known) example of a classification algorithm

Alternatives

- Basic methods: Naïve Bayes, kNN, logistic regression, ..
- Ensemble methods: random forest, gradient boosting, …
- Support vector machines
- Neural networks: CNN, rNN, LSTM, …

Decision trees is one of the best known and historically first examples of a classification approach. Many other methods have been devised in studied over tie. These include basic methods (we will see some examples later), ensemble methods (discussed in the following), support vector machines (a paradigm based on splitting the space through hyper-planes), and neural networks (which are attracting recently significant attention and are nowadays among the best performing classifiers if very large training sets are available).

Ensemble Methods

Idea

- Take a collection of simple or **weak** learners
- Combine their results to make a single, **strong** learner

Types

- **Bagging:** train learners in parallel on different samples of the data, then combine outputs through voting or averaging
- **Stacking:** combine model outputs using a second-stage learner like linear regression
- **Boosting:** train learners on the filtered output of other learners

One important development in decision trees was the introduction of the idea of ensemble methods. The basic principle is simple: instead of constructing a single model, many different models are constructed independently. Even if each model is not very expressive (weak learners) their combination can be powerful (strong learner). Different ensemble methods are distinguished by the type of approach they are based on. Ensemble methods, which we will discuss in the following, learn several models in parallel, and combine then their predictions by voting or averaging. Stacking methods use more sophisticated techniques to combine model outputs, based themselves on learning methods. Finally, boosting learn models in sequence. In each step the samples of the training data are reweighted depending on whether they have been correctly classified.

Random Forests

Learn K different decision trees from independent samples of the data (bagging)

- vote between different learners, so models should not be too similar

Aggregate output: majority vote

Random forests are an ensemble method based on bagging. The principle is very simple: K different decision trees are learnt in parallel from different (independent) samples of the data, and the classification is derived from a majority vote of the predictions.

Why do Ensemble Methods Work?

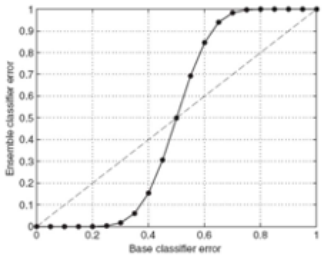
Assume there are 25 base classifiers

– Each classifier has error rate = 0.35

– Assume classifiers are independent

Probability that the ensemble classifier makes a wrong prediction

$$P(\text{wrong prediction}) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$



Tan, Steinbach, Kumar

Here we give the argument why ensemble methods work: even if the individual classifiers are not very good (e.g. make 35% errors in prediction) their aggregate will be very strong. For example, if we have 25 classifiers, the probability that a majority of them, namely at least 13, make a wrong prediction is very small, namely 6%. In general, ensemble methods work well, if the individual models are better than random guessing. The figure illustrates the relation between the classification errors of individual classifiers and the aggregate classification accuracy.

Sampling Strategies

Two sampling strategies

Sampling data

- select a subset of the data → Each tree is trained on different data

Sampling attributes

- select a subset of attributes → corresponding nodes in different trees (usually) don't use the same feature to split

For random forests the main issue is the choice of the sampling strategy, i.e., the generation of samples that are used for learning the individual models. Specifically, it consists of two different sampling strategy. The first, sampling data selects from the original dataset a sample. Thus each decision tree is trained on a different sample of data. The second, sampling attributes selects from the attributes available to take a decision a random subset. Thus even if a tree would have been constructed in the same way up to a level, the continuation might become different due to attribute sampling (e.g. the optimal attribute in one tree is not available for splitting in the other tree).

Random Forests: Algorithm

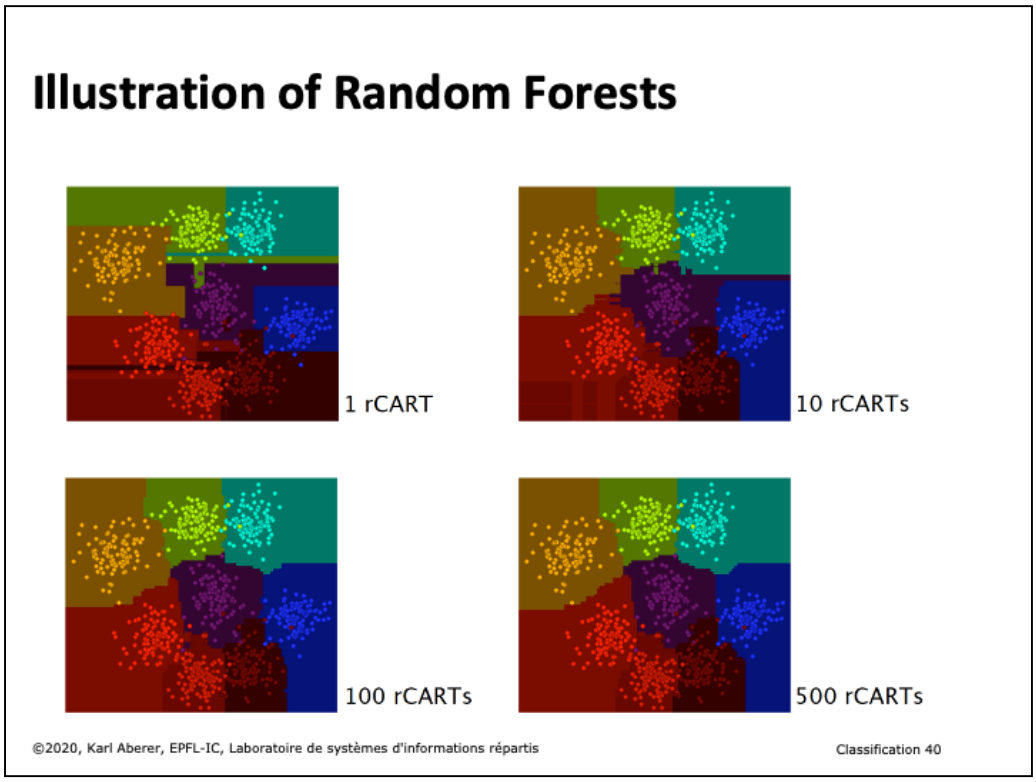
1. Draw K bootstrap **samples of size N** from original dataset, with replacement (bootstrapping)
2. While constructing the decision tree, select a random set of **m attributes** out of the p attributes available to infer split (feature bagging)

Typical parameters

- m ≈ sqrt(p), or smaller
- K ≈ 500

The random forest algorithm is based on specific choices of sampling strategies, when constructing multiple decision trees in parallel. For sampling the data, for each of the K trees to be constructed, a sample of size N (the original size of the dataset) is selected with replacement. This step is called bootstrapping. Note, since sampling is done with replacement the same object might occur repeatedly in a sample. Thus even if the sample has the same size as the original datasets, not all original data instances will occur in the sample. For selecting the attributes a proper subset of attributes is chosen to infer the next split. The number of attributes considered, is significantly smaller than the whole set, e.g. in the order of the square root of the number of all attributes.

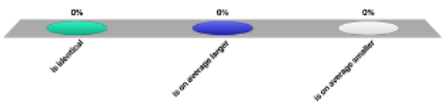
Bootstrapping has been original conceived, for training datasets that are not exceedingly large, and thus available data would have to be used very carefully. In cases were training data is available abundantly, an alternative approach, called su-bagging, can be used. There sampling is done without replacement.



Random forests allow to learn much more complex functions than basic decision trees. This fact is illustrated in this visualization. For the same training data set different numbers of decision trees are constructed (rCART is a variant of decision trees). We observe that with increasing numbers of trees the decision boundaries become increasingly more complex and smoother, and thus a better separation among different classes can be achieved.

The computational cost for constructing a RF with K as compared to constructing K decision trees on the same data

A. is identical
B. is on average larger
C. is on average smaller



Characteristics of Random Forests

Strengths

- Ensembles can model extremely complex decision boundaries without overfitting
- Probably the most popular classifier for **dense data** (≤ a few thousand features)
- Easy to implement (train a lot of trees)
- Parallelizes easily, good match for MapReduce

Random forests are a very popular method for classification due to the many advantages they offer. They are considered as the method of choice in cases where the data is dense, which means that the number of features is relatively low (in the thousands). Sparse data would be, for example, vector space representation of documents with very large vocabularies. In such cases, before applying a method such as random forests, a dimensionality reduction would have to be applied. This could be accomplished in the case of documents by creating a word embedding.

Characteristics of Random Forests

Weaknesses

- Deep Neural Networks generally do better
- Needs many passes over the data – at least the max depth of the trees
- Relatively easy to overfit – hard to balance accuracy/fit tradeoff

More recently, in cases where large training sets are available or number of features is very large, deep neural networks exhibit better performance than random forests.

References

Textbook

- Jiawei Han, Data Mining: concepts and techniques, Morgan Kaufman, 2000, ISBN 1-55860-489-8

References

- Leo Breiman (2001) “Random Forests” Machine Learning, 45, 5-32.
- Shafer, John, Rakesh Agrawal, and Manish Mehta. "SPRINT: A scalable parallel classifier for data mining." *Proc. 1996 Int. Conf. Very Large Data Bases*. 1996.