Tree induction
The task of constructing a tree from the training set is called tree induction. Most existing tree induction systems proceed in a **greedy top-down fashion**.

1 - Starting with an empty tree and the entire training set, the following algorithm is applied until no more splits are possible.

2 - If all the training examples at the current node t belong to category c, create a leaf node with the class c and halt.

3 - Otherwise, score each one of the set of possible splits S, using a goodness measure

4 - Choose the best split s* as the test at the current node, and create as many child nodes as there are distinct outcomes of s*.

Univariate and multivariate
A **univariate** decision tree is one in which the test at each internal node uses a single attribute. A **multivariate** decision tree may use as splits expressions containing multiple attributes.
A special case of multivariate trees that we are particularly interested in is **oblique** decision trees. The tests in oblique trees use linear combinations of attributes.
A univariate tree is also axis-parallel, and an oblique tree is the same as a linear tree.

Ordered vs unordered
Ordered means continuous/numerical attributes. Algorithms developed for ordered can be adapted to discrete.
Unordered means discrete. In machine learning some algorithms are developed directly for discrete.

A node in a tree with more than 2 children, considered in vector quantization literature.

Classification vs regression
a decision tree is said to perform classification if the class labels are discrete values, and regression if the class labels are continuous.

Finding splits
In case of univariate trees, finding a split amounts to finding an attribute which is the most ``useful" in discriminating the input data, and finding a decision rule using the attribute. In case of multivariate trees, finding a split can be seen as finding a ``composite" feature, a combination of (some of the) existing attributes that has good discriminatory power. In either of these cases, a basic task in tree building is to rank features (single or composite) according to their usefulness in discriminating the classes in the data.

Feature evaluation
Ben-Basset divides feature evaluation rules into three, not necessarily distinct, categories: rules derived from information theory, rules derived from distance measures and rules derived from dependence measures.

There exist several attribute selection criteria that do not clearly belong to any category in Ben-Basset's taxonomy.

Oblique splits
Most of the work on multivariate splits considered linear (oblique) trees. These are trees which have tests based on a linear combination of the attributes at some internal nodes. The problem of finding an optimal linear split (optimal with respect to any of the feature evaluation measures in Section 2.3.1) is more difficult that that of finding the optimal univariate split. In fact, finding optimal linear splits is known to be intractable for some feature evaluation rules (see Section 2.6.1 for pointers), so heuristic methods are required for finding good, albeit suboptimal, linear splits. Methods used in the literature for finding good linear tests include linear discriminant analysis, hill climbing search, linear programming, perceptron training and others.

Size
Obtaining the ``right'' sized trees may be important for several reasons, which depend on the size of the classification problem . For moderate sized problems, the critical issues are generalization accuracy, honest error rate estimation and gaining insight into the predictive and generalization structure of the data. For very large tree classifiers, the critical issue is optimizing structural properties (height, balance etc.)
Related to overfitting and generalization

Pruning
is the method most widely used for obtaining right sized trees
build the complete tree (a tree in which splitting no leaf node further will improve the accuracy on the training data) and then remove subtrees that are not contributing significantly towards generalization accuracy. It is argued that this method is better than stop-splitting rules, because it can compensate, to some extent, for the suboptimality of greedy tree induction.
For instance, if there is very good node T1 a few levels below a not-so-good node T2, a stop-splitting rule will stop tree growth at T2, whereas pruning may give a high rating for, and retain, the whole subtree at T1.
May require a pruning set, separate from training set.

Sample size vs. dimensionality

How sample size should vary according to dimensionality and *vice versa*.
Intuitively, an imbalance between the number of samples and the number of features (i.e., too many samples with too few attributes, or too few samples with too many attributes) can make induction more difficult.

- For a finite sized data with little or no *a priori* information, the ratio of the sample size to dimensionality must be as large as possible to suppress optimistically biased evaluations of the performance of the classifier.
- For a given sample size used in training a classifier, there exists an optimum feature size and quantization complexity (the latter refers to the number of ranges a dimension is split into). This result is true for both two-class problems and multi-class problems.
- The ratio of the sample size to dimensionality should vary inversely proportional to the amount of available knowledge about the class conditional densities.

In tasks where more features than the ``optimal'' are available, decision tree quality is known to be affected by the redundant and irrelevant attributes. To avoid this problem, either a feature subset selection method or a method to form a small set of composite features can be used as a preprocessing step to tree induction.
On the other hand, if the training sample has too many objects, a subsample selection method can be employed to filter out the unnecessary observations.
Techniques to search for multivariate splits can be seen as ways for constructing composite features.

Incorporating costs
In most real-world domains, attributes can have costs of measurement, and objects can have misclassification costs. If the measurement (misclassification) costs are not identical between different attributes (classes), decision tree algorithms need to be designed explicitly to prefer cheaper trees.

Missing attribute values
In real world data sets, it is often the case that some attribute values are missing from the data. Several researchers have addressed the problem of dealing with missing attribute values in the training as well as testing sets

Improving on greedy

Most tree induction systems use a greedy approach --- trees are induced top-down, a node at a time. Several authors pointed out the inadequacy of greedy induction for difficult concepts. The problem of inducing globally optimal decision trees has been addressed time and again.
Tree construction with partial or exhaustive Lookahead, not very efficient
Tree construction with two stages: 1st: Greedy construction, 2nd: Refinement.

Soft splits and fuzziness
Two common criticisms of decision trees are the following: (1) As decisions in the lower levels of a tree are based on increasingly smaller fragments of the data, some of them may not have much probabilistic significance (data fragmentation). (2) As several leaf nodes can represent the same class, unnecessarily large trees may result, especially when the number of classes is large (high class overlap). It has been shown that the use of fuzzy reasoning can help reduce both the above problems.
Several researchers have considered using *soft* splits of data for decision trees. A hard split divides the data into mutually exclusive partitions. A soft split, on the other hand, assigns a probability that each point belongs to a partition, thus allowing points to belong to multiple partitions.

Multiple trees
A known peril of decision tree construction is its variance, especially when the samples are small and the features are many. Variance can be caused by random choice of training and pruning samples, by many equally good attributes only one of which can be chosen at a node, due to cross validation or because of other reasons. A few authors suggested using a collection of decision trees, instead of just one, to reduce the variance in classification performance.
Classification results of the trees have been combined using either simplistic voting methods

Incremental tree induction
Most tree induction algorithms use batch training --- the entire tree needs to be recomputed to accommodate a new training example. A crucial property of neural network training methods is that they are incremental --- network weights can be continually adjusted to accommodate training

examples. Incremental induction of decision trees is considered by several authors. Friedman's binary tree induction method could use ``adaptive" features for some splits. An adaptive split depends on the training subsample it is splitting. (An overly simple example of an adaptive split is a test on the mean value of a feature.)

Tree quality measure

The fact that several trees can correctly represent the same data raises the question of how to decide that one tree is better than another. Several measures have been suggested to quantify tree quality. Moret [254] summarizes work on measures such as tree size, expected testing cost and worst-case testing cost. He shows that these three measures are pairwise incompatible, which implies that an algorithm minimizing one measure is guaranteed *not* to minimize the others, for some tree. Fayyad and Irani [94] argue that, by concentrating on optimizing one measure, number of leaf nodes, one can achieve performance improvement along other measures.
Generalization accuracy is a popular measure for quantifying the goodness learning systems. The accuracy of the tree is computed using a testing set that is independent of the training set or using estimation techniques like cross-validation or bootstrap, and more accurate trees are preferred to the less accurate ones.

Top down, Bottom up, mixed
Most existing tree induction systems proceed in a greedy top-down fashion [343,29,292]. Bottom up induction of trees is considered in [204]. Bottom up tree induction is also common [282] in problems such as building identification keys and optimal test sequences. A hybrid approach to tree construction, that combined top-down and bottom-up induction can be found in [179].

NP-Complete
Hyafil and Rivest [155] proved that the problem of building optimal decision trees from decision tables, optimal in the sense of minimizing the expected number of tests required to classify an unknown sample is NP-Complete. even the problem of identifying the root node in an optimal strategy is NP-hard. The problem of constructing the smallest decision tree which best distinguishes characteristics of multiple distinct groups is shown to be NP-complete in [353].
Most of the above results consider only univariate decision tree construction. Intuitively, linear or multivariate tree construction should be more difficult than univariate tree construction, as there is a much larger space of splits to be searched. As the problem of finding a single linear split is NP-hard, it is no surprise that the problem of building the optimal linear decision trees is NP-hard

top-down tree induction (using mutual information) is necessarily suboptimal in terms of average tree depth

Assumptions and biases
Assumption: Multi-stage classifiers may be more accurate than single stage classifiers. Analysis: However, the data fragmentation caused by multi-stage hierarchical classifiers may compensate for the gain in accuracy.

Bias: *Smaller consistent decision trees have higher generalization accuracy than larger consistent trees* (Occam's Razor). Analysis: Murphy and Pazzani [262] empirically investigated the truth of this bias. Their experiments indicate that this conjecture seems to be true. However, their experiments indicate that the smallest decision trees typically have lesser generalization accuracy than trees that are slightly larger

Oblique trees

Many variants of decision tree (DT) algorithms have concentrated on decision trees in which each node checks the value of a single attribute. This class of decision trees may be called *axis-parallel*, because the tests at each node are equivalent to axis-parallel hyperplanes in the attribute space.

Obliques are Trees that test a linear combination of the attributes at each internal node. (Not necessarily all the attributes)
Because these tests are equivalent to hyperplanes at an oblique orientation to the axes, we call this class of decision trees *oblique* decision trees. (Trees of this form have also been called ``linear'' (Section ) and ``multivariate''. We prefer the term ``oblique'' to aid geometric intuition and because ``multivariate'' includes non-linear combinations of the variables, i.e., curved surfaces.) It is clear that these are simply a more general form of axis-parallel trees

It should be intuitively clear that when the underlying concept is defined by a polygonal space partitioning, it is preferable to use oblique decision trees for classification. For example, there exist many domains in which one or two oblique hyperplanes will be the best model to use for classification. In such domains, axis-parallel methods will have to approximate the correct model with a staircase-like structure, while an oblique tree-building method could capture it with a tree that was both smaller and more accurate.

The ability to generate oblique trees often produces very small trees compared to axis-parallel methods. When the underlying problem requires an oblique split, oblique trees are also more accurate than axis-parallel trees. Allowing a tree-building system to use both oblique and axis-parallel splits broadens the range of domains for which the system should be useful.

In Axis parallel, at each node it is actually possible to enumerate all possible splittings and choose the best one.
In oblique, there are too many possible splittings, therefore it is an NP hard problem.

The problem of inducing the smallest axis-parallel decision tree is known to be NP-hard
Note that one can generate the smallest axis-parallel tree that is consistent with the training set in polynomial time *if* the number of attributes is a constant. But when the tree uses oblique splits, it is not clear, even for a fixed number of attributes, how to generate an optimal (e.g., smallest) decision tree in polynomial time. Goodrich [35] showed that the problem of inducing the smallest oblique decision tree is NP-hard even in three dimensions. This suggests that the complexity of constructing good oblique trees is greater than that for axis-parallel trees.