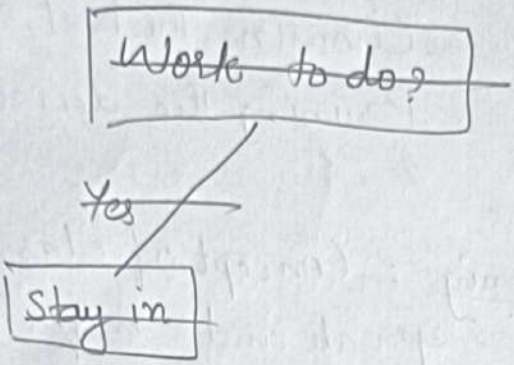
Module 3 (Data Science & ML)

Decision tree learning : Concept of decision tree,
Divide and Conquer approach,
C5.0 Decision tree algorithm,
Choosing the best split,
Pruning the decision tree.

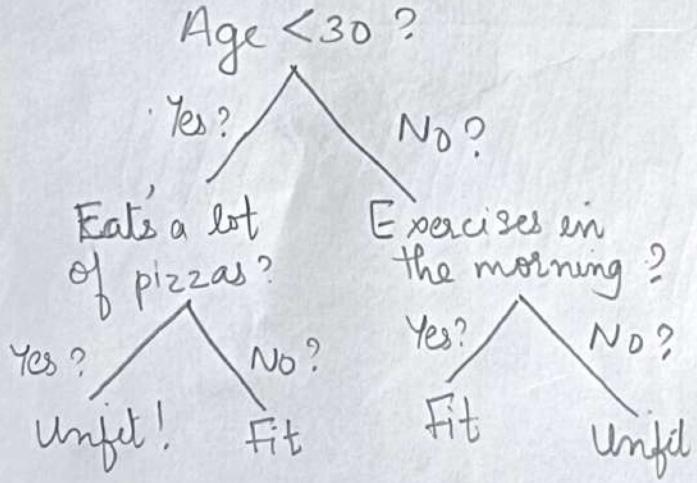
Classification rules learning : Concept of classification rules,
Separate and Conquer approach,
The IR algorithm,
Rules from decision trees.

Regression methods : Concept of regression,
Simple linear regression,
Ordinary least squares estimation,
Correlations,
Multiple linear regression.

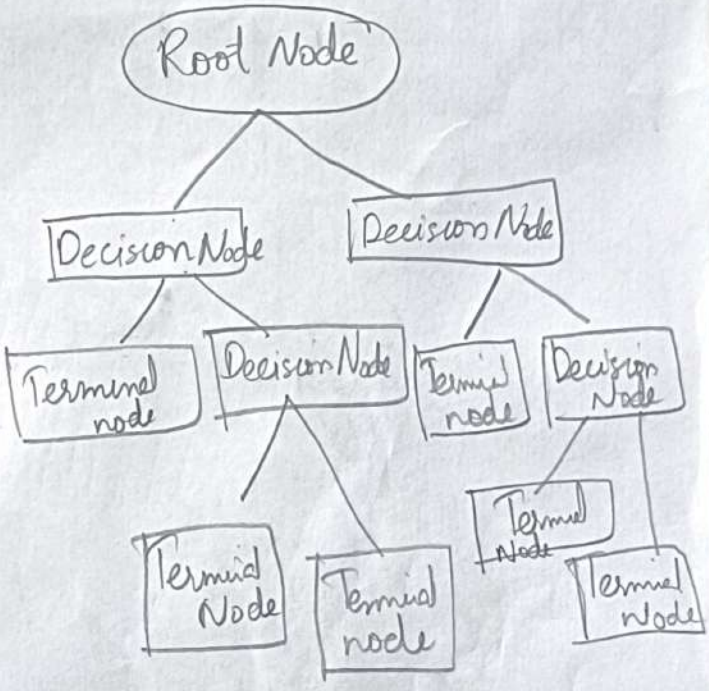
Divide and Conquer
decision trees



Is a person fit?



Sample Decision tree



KL 20 2014 Module III HDS Malu Sam

Decision Tree Learning

Divide and Conquer – Classification Using Decision Trees and Rules

decision trees (Understanding Decision tree).

- classifiers, which utilize a tree structure to model the relationships among the features and the potential outcomes
- working of a tree which predict whether a job offer should be accepted is as follows
- A job offer to be considered begins at the **root node**, where it is then passed through **decision nodes** that require choices to be made based on the attributes of the job. These choices split the data across **branches** that indicate potential outcomes of a decision, depicted here as yes or no outcomes, though in some cases there may be more than two possibilities. In the case a final decision can be made, the tree is terminated by **leaf nodes** (also known as terminal nodes) that denote the action to be taken as the result of the series of decisions. In the case of a predictive model, the leaf nodes provide the expected result given the series of events in the tree.
- A great benefit of decision tree algorithms is that the flowchart-like tree structure is not necessarily exclusively for the learner's internal use. After the model is created, many decision tree algorithms output the resulting structure in a human-readable format. This provides tremendous insight into how and why the model works or doesn't work well for a particular task.

Divide and conquer approach

Decision trees are built using a heuristic called recursive partitioning. This approach is also commonly known as divide and conquer because it splits the data into subsets, which are then split repeatedly into even smaller subsets, and so on and so forth until the process stops when the algorithm determines the data within the subsets are sufficiently homogenous, or another stopping criterion has been met.

At first, the root node represents the entire dataset, since no splitting has transpired. Next, the decision tree algorithm must choose a feature to split upon; ideally, it chooses the feature most predictive of the target class. The examples are then partitioned into groups according to the distinct values of this feature, and the first set of tree branches are formed.

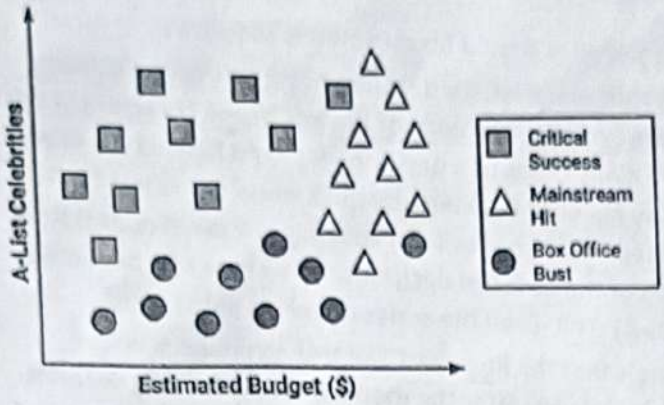
Working down each branch, the algorithm continues to divide and conquer the data, choosing the best candidate feature each time to create another decision node, until a stopping criterion is reached. Divide and conquer might stop at a node in a case that:

- All (or nearly all) of the examples at the node have the same class
- There are no remaining features to distinguish among the examples
- The tree has grown to a predefined size limit

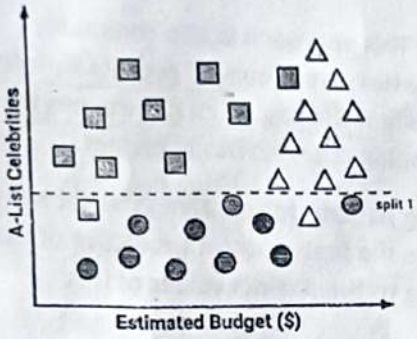
Eg: To illustrate the tree building process, let's consider a simple example. Imagine that you work for a Hollywood studio, where your role is to decide whether the studio should move forward with producing the screenplays pitched by promising new authors. After returning from a vacation, your desk is piled high with proposals. Without the time to read each proposal cover-to-cover, you decide to develop a decision tree algorithm to predict whether a potential movie would fall into one of three categories: **Critical Success, Mainstream Hit,** or **Box Office Bust.**

Job offers

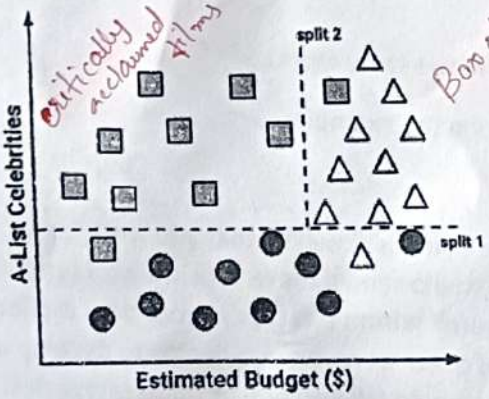
To build the decision tree, you turn to the studio archives to examine the factors leading to the success and failure of the company's 30 most recent releases. You quickly notice a relationship between the film's estimated shooting budget, the number of A-list celebrities lined up for starring roles, and the level of success. Excited about this finding, you produce a scatterplot to illustrate the pattern:



Using the divide and conquer strategy, we can build a simple decision tree from this data. First, to create the tree's root node, we split the feature indicating the number of celebrities, partitioning the movies into groups with and without a significant number of A-list stars:



Next, among the group of movies with a larger number of celebrities, we can make another split between movies with and without a high budget:



Critically acclaimed films

Box office hits with high budget & large no of celebrities

high no of celebrities low budget.

little star power but budget ranging from small to large. contains the flops

At this point, we have partitioned the data into three groups. The group at the top-left corner of the diagram is composed entirely of critically acclaimed films. This group is distinguished by a high number of celebrities and a relatively low budget. At the top-right corner, majority of movies are box office hits with high budgets and a large number of celebrities. The final group, which has little star power but budgets ranging from small to large, contains the flops.

If we wanted, we could continue to divide and conquer the data by splitting it based on the increasingly specific ranges of budget and celebrity count, until each of the currently misclassified values resides in its own tiny partition, and is correctly classified. However, it is not advisable to overfit a decision tree in this way. Though there is nothing to stop us from splitting the data indefinitely, overly specific decisions do not always generalize more broadly. We'll avoid the problem of overfitting by stopping the algorithm.

The C5.0 decision tree algorithm (developed by Computer Scientist J. Ross Quinlan)

There are numerous implementations of decision trees, but one of the most well-known implementations is the C5.0 algorithm

The C5.0 algorithm has become the industry standard to produce decision trees, because it does well for most types of problems directly out of the box.

the decision trees built by C5.0 generally perform nearly as well, but are much easier to understand and deploy

Strengths	Weaknesses
• An all-purpose classifier that does well on most problems • Highly automatic learning process, which can handle numeric or nominal features, as well as missing data • Excludes unimportant features • Can be used on both small and large datasets • Results in a model that can be interpreted without a mathematical background (for relatively small trees) • More efficient than other complex models	• Decision tree models are often biased toward splits on features having a large number of levels • It is easy to overfit or underfit the model • Can have trouble modeling some relationships due to reliance on axis-parallel splits • Small changes in the training data can result in large changes to decision logic • Large trees can be difficult to interpret and the decisions they make may seem counterintuitive

Choosing the best split

The degree to which a subset of examples contains only a single class is known as **purity**, and any subset composed of only a single class is called **pure.**

There are various measurements of purity that can be used to identify the best decision tree splitting candidate. C5.0 uses entropy, a concept borrowed from information theory that quantifies the randomness, or disorder, within a set of class values. Sets with high entropy are very diverse and provide little information about other items that may also belong in the set, as there is no apparent

commonality. The decision tree hopes to find splits that reduce entropy, ultimately increasing homogeneity within the groups.

entropy is measured in bits. If there are only two possible classes, entropy values can range from 0 to 1. For n classes, entropy ranges from 0 to log2 (n). In each case, the minimum value indicates that the sample is completely homogenous, while the maximum value indicates that the data are as diverse as possible, and no group has even a small plurality.

In the mathematical notion, entropy is specified as follows:

$$\text{Entropy}(S) = \sum_{i=1}^{c} -p_i \log_2(p_i)$$

In this formula, for a given segment of data (S), the term c refers to the number of class levels and pi refers to the proportion of values falling into class level i

To use entropy to determine the optimal feature to split upon, the algorithm calculates the change in homogeneity that would result from a split on each possible feature, which is a measure known as information gain. The information gain for a feature F is calculated as the difference between the entropy in the segment before the split (S1) and the partitions resulting from the split (S2):

$$\text{InfoGain}(F) = \text{Entropy}(S_1) - \text{Entropy}(S_2)$$

One complication is that after a split, the data is divided into more than one partition. Therefore, the function to calculate Entropy(S2) needs to consider the total entropy across all of the partitions. It does this by weighing each partition's entropy by the proportion of records falling into the partition. This can be stated in a formula as:

$$\text{Entropy}(S) = \sum_{i=1}^{n} w_i \, \text{Entropy}(P_i)$$

In simple terms, the total entropy resulting from a split is the sum of the entropy of each of the n partitions weighted by the proportion of examples falling in the partition (wi).

The higher the information gain, the better a feature is at creating homogeneous groups after a split on this feature. If the information gain is zero, there is no reduction in entropy for splitting on this feature. On the other hand, the maximum information gain is equal to the entropy prior to the split. This would imply that the entropy after the split is zero, which means that the split results in completely homogeneous groups.

Pruning the decision tree

A decision tree can continue to grow indefinitely, choosing splitting features and dividing the data into smaller and smaller partitions until each example is perfectly classified or the algorithm runs out of features to split on. However, if the tree grows overly large, many of the decisions it makes will be

overly specific and the mod
tree involves reducing its siz
One solution to th
decisi

overly specific and the model will be overfitted to the training data. The process of **pruning a decision tree** involves reducing its size such that it generalizes better to unseen data.

One solution to this problem is to stop the tree from growing once it reaches a certain number of decisions or when the decision nodes contain only a small number of examples. This is **called early stopping** or **pre-pruning** the decision tree.

An alternative, called **post-pruning**, involves growing a tree that is intentionally too large and pruning leaf nodes to reduce the size of the tree to a more appropriate level. This is often a more effective approach than pre-pruning, because it is quite difficult to determine the optimal depth of a decision tree without growing it first.

In some cases, entire branches are moved further up the tree or replaced by simpler decisions. These processes of grafting branches are known as **subtree raising** and **subtree replacement,** respectively

Understanding classification rules

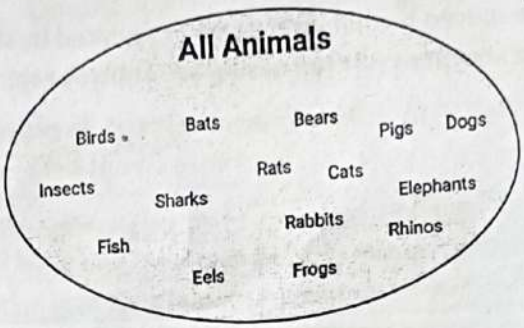
Classification rules represent knowledge in the form of logical if-else statements that assign a class to unlabeled examples. They are specified in terms of an **antecedent** and a **consequent;** these form a hypothesis stating that "if this happens, then that happens." A simple rule might state, "if the hard drive is making a clicking sound, then it is about to fail." The antecedent comprises certain combinations of feature values, while the consequent specifies the class value to assign when the rule's conditions are met.

Rule learners are often used in a manner similar to decision tree learners. Like decision trees, they can be used for applications that generate knowledge for future action

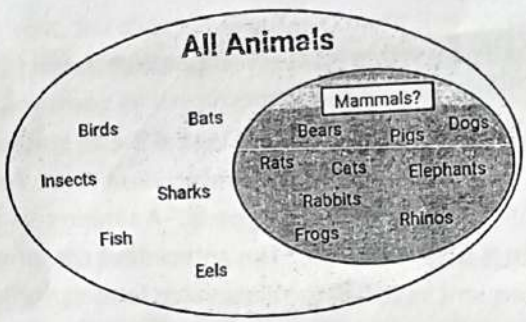
Separate and conquer

Classification rule learning algorithms utilize a heuristic known as separate and conquer. The process involves identifying a rule that covers a subset of examples in the training data, and then separating this partition from the remaining data. As the rules are added, additional subsets of the data are separated until the entire dataset has been covered and no more examples remain.

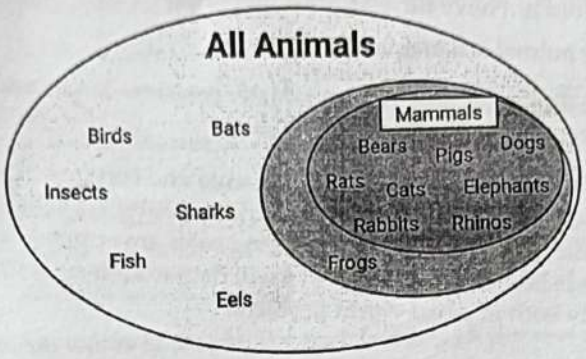
One way to imagine the rule learning process is to think about drilling down into the data by creating increasingly specific rules to identify class values. Suppose you were tasked with creating rules to identify whether or not an animal is a mammal. You could depict the set of all animals as a large space, as shown in the following diagram:



A rule learner begins by using the available features to find homogeneous groups. For example, using a feature that indicates whether the species travels via land, sea, or air, the first rule might suggest that any land-based animals are mammals:

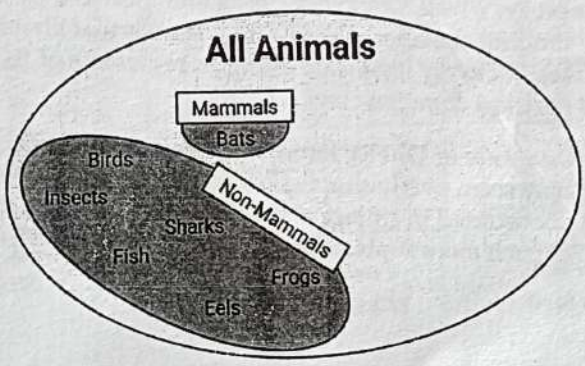


Do you notice any problems with this rule? If you're an animal lover, you might have realized that frogs are amphibians, not mammals. Therefore, our rule needs to be a bit more specific. Let's drill down further by suggesting that mammals must walk on land and have a tail:



An additional rule can be defined to separate out the bats, the only remaining mammal. Thus, this subset can be separated from the other data.

An additional rule can be defined to separate out the bats, the only remaining mammal. A potential feature distinguishing bats from the other remaining animals would be the presence of fur. Using a rule built around this feature, we have then correctly identified all the animals:



At this point, since all of the training instances have been classified, the rule learning process would stop. We learned a total of three rules:

- Animals that walk on land and have tails are mammals
- If the animal does not have fur, it is not a mammal
- Otherwise, the animal is a mammal

The previous example illustrates how rules gradually consume larger and larger segments of data to eventually classify all instances.

As the rules seem to cover portions of the data, separate and conquer algorithms are also known as **covering algorithms**, and the resulting rules are called covering rules. In the next section, we will learn how covering rules are applied in practice by examining a simple rule learning algorithm. We will then examine a more complex rule learner, and apply both to a real-world problem.

The 1R algorithm

Suppose a television game show has a wheel with ten evenly sized colored slices. Three of the segments were colored red, three were blue, and four were white. Prior to spinning the wheel, you are asked to choose one of these colors. When the wheel stops spinning, if the color shown matches your prediction, you will win a large cash prize. What color should you pick?

If you choose white, you are, of course, more likely to win the prize—this is the most common color on the wheel. Obviously, this game show is a bit ridiculous, but it demonstrates the simplest classifier, **ZeroR**, a rule learner that literally learns no rules (hence the name). For every unlabeled example, regardless of the values of its features, it predicts the most common class.

The **1R algorithm** (**One Rule** or **OneR**), improves over ZeroR by selecting a single rule. Although this may seem overly simplistic, it tends to perform better than you might expect. As demonstrated in empirical studies, the accuracy of this algorithm can approach that of much more sophisticated algorithms for many real-world tasks.

For an in-depth look at the surprising performance of 1R, see Holte RC. *Very simple classification rules perform well on most commonly used datasets.* Machine Learning. 1993; 11:63-91.

The strengths and weaknesses of the 1R algorithm are shown in the following table:

Strengths	Weaknesses
• Generates a single, easy-to-understand, human-readable rule of thumb • Often performs surprisingly well • Can serve as a benchmark for more complex algorithms	• Uses only a single feature • Probably overly simplistic

The way this algorithm works is simple. For each feature, 1R divides the data into groups based on similar values of the feature. Then, for each segment, the algorithm predicts the majority class. The error rate for the rule based on each feature is calculated and the rule with the fewest errors is chosen as the one rule.

The following tables show how this would work for the animal data we looked at earlier in this section:

Animal	Travels By	Has Fur	Mammal
Bats	Air	Yes	Yes
Bears	Land	Yes	Yes
Birds	Air	No	No
Cats	Land	Yes	Yes
Dogs	Land	Yes	Yes
Eels	Sea	No	No
Elephants	Land	No	Yes
Fish	Sea	No	No
Frogs	Land	No	No
Insects	Air	No	No
Pigs	Land	No	Yes
Rabbits	Land	Yes	Yes
Rats	Land	Yes	Yes
Rhinos	Land	No	Yes
Sharks	Sea	No	No

Full Dataset

Travels By	Predicted	Mammal	
Air	No	Yes	✖
Air	No	No	
Air	No	No	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	No	✖
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Sea	No	No	
Sea	No	No	
Sea	No	No	

Rule for "Travels By"
Error Rate = 2 / 15

Has Fur	Predicted	Mammal	
No	No	No	
No	No	No	
No	No	Yes	✖
No	No	No	
No	No	No	
No	No	No	
No	No	Yes	✖
No	No	Yes	✖
No	No	No	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	

Rule for "Has Fur"
Error Rate = 3 / 15

For the **Travels By** feature, the dataset was divided into three groups: **Air**, **Land**, and **Sea**. Animals in the **Air** and **Sea** groups were predicted to be non-mammal, while animals in the **Land** group were predicted to be mammals. This resulted in two errors: bats and frogs. The **Has Fur** feature divided animals into two groups. Those with fur were predicted to be mammals, while those without fur were not predicted to be mammals. Three errors were counted: pigs, elephants, and rhinos. As the **Travels By** feature results in fewer errors, the 1R algorithm will return the following "one rule" based on **Travels By**:

- If the animal travels by air, it is not a mammal
- If the animal travels by land, it is a mammal
- If the animal travels by sea, it is not a mammal

The algorithm stops here, having found the single most important rule.

Obviously, this rule learning algorithm may be too basic for some tasks. Would you want a medical diagnosis system to consider only a single symptom, or an automated driving system to stop or accelerate your car based on only a single factor? For these types of tasks, a more sophisticated rule learner might be useful. We'll learn about one in the following section.

The RIPPER algorithm

Early rule learning algorithms were plagued by a couple of problems. First, they were notorious for being slow, which made them ineffective for the increasing number of large datasets. Secondly, they were often prone to being inaccurate on noisy data.

A first step toward solving these problems was proposed in 1994 by Johannes Furnkranz and Gerhard Widmer. Their **Incremental Reduced Error Pruning (IREP) algorithm** uses a combination of pre-pruning and post-pruning methods that grow very complex rules and prune them before separating the instances from the full dataset. Although this strategy helped the performance of rule learners, decision trees often still performed better.

For more information on IREP, see Furnkranz J, Widmer G. *Incremental Reduced Error Pruning*. Proceedings of the 11th International Conference on Machine Learning. 1994: 70-77.

Rule learners took another step forward in 1995 when William W. Cohen introduced the **Repeated Incremental Pruning to Produce Error Reduction (RIPPER) algorithm**, which improved upon IREP to generate rules that match or exceed the performance of decision trees.

For more detail on RIPPER, see Cohen WW. *Fast effective rule induction*. Proceedings of the 12th International Conference on Machine Learning. 1995:115-123.

As outlined in the following table, the strengths and weaknesses of RIPPER are generally comparable to decision trees. The chief benefit is that they may result in a slightly more parsimonious model:

Strengths	Weaknesses
• Generates easy-to-understand, human-readable rules • Efficient on large and noisy datasets • Generally produces a simpler model than a comparable decision tree	• May result in rules that seem to defy common sense or expert knowledge • Not ideal for working with numeric data • Might not perform as well as more complex models

Having evolved from several iterations of rule learning algorithms, the RIPPER algorithm is a patchwork of efficient heuristics for rule learning. Due to its complexity, a discussion of the technical implementation details is beyond the scope of this book. However, it can be understood in general terms as a three-step process:

1. Grow
2. Prune
3. Optimize

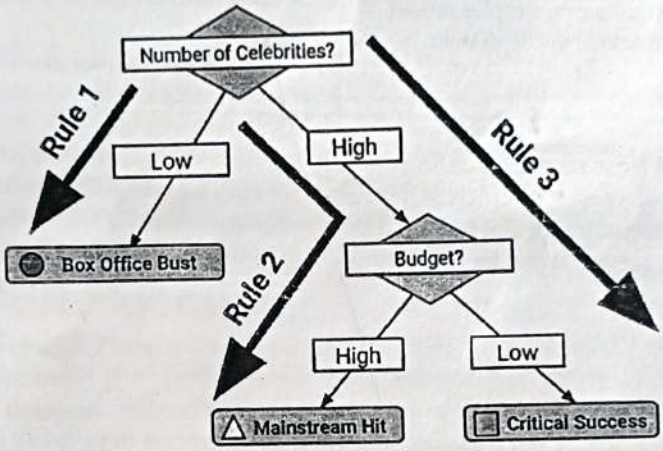
The growing phase uses the separate and conquer technique to greedily add conditions to a rule until it perfectly classifies a subset of data or runs out of attributes for splitting. Similar to decision trees, the information gain criterion is used to identify the next splitting attribute. When increasing a rule's specificity no longer reduces entropy, the rule is immediately pruned. Steps one and two are repeated until it reaches a stopping criterion, at which point the entire set of rules is optimized using a variety of heuristics.

The RIPPER algorithm can create much more complex rules than can the 1R algorithm, as in can consider more than one feature. This means that it can create rules with multiple antecedents such as "if an animal flies and has fur, then it is a mammal." This improves the algorithm's ability to model complex data, but just like decision trees, it means that the rules can quickly become more difficult to comprehend.

The evolution of classification rule learners didn't stop with RIPPER. New rule learning algorithms are being proposed rapidly. A survey of literature shows algorithms called IREP++, SLIPPER, TRIPPER, among many others.

Rules from decision trees

Classification rules can also be obtained directly from decision trees. Beginning at a leaf node and following the branches back to the root, you will have obtained a series of decisions. These can be combined into a single rule. The following figure shows how rules could be constructed from the decision tree to predict movie success:



Following the paths from the root node down to each leaf, the rules would be:

1. If the number of celebrities is low, then the movie will be a **Box Office Bust**.
2. If the number of celebrities is high and the budget is high, then the movie will be a **Mainstream Hit**.
3. If the number of celebrities is high and the budget is low, then the movie will be a **Critical Success**.

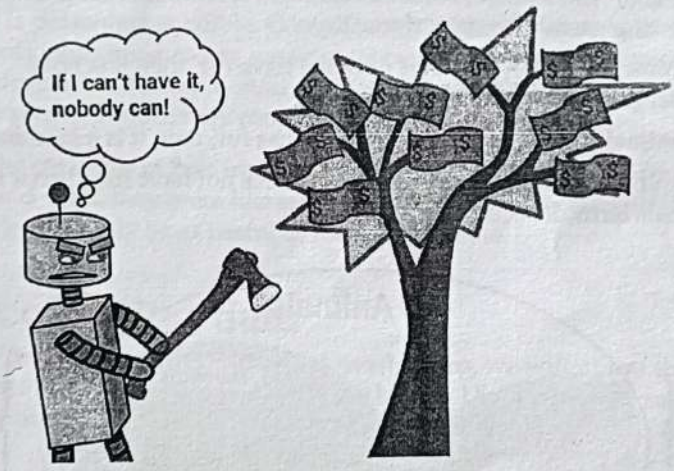
For reasons that will be made clear in the following section, the chief downside to using a decision tree to generate rules is that the resulting rules are often more complex than those learned by a rule learning algorithm. The divide and conquer strategy employed by decision trees biases the results differently than that of a rule learner. On the other hand, it is sometimes more computationally efficient to generate rules from trees.

The `C5.0()` function in the `C50` package will generate a model using classification rules if you specify `rules = TRUE` when training the model.

What makes trees and rules greedy?

Decision trees and rule learners are known as greedy learners because they use data on a first-come, first-served basis. Both the divide and conquer heuristic used by decision trees and the separate and conquer heuristic used by rule learners attempt to make partitions one at a time, finding the most homogeneous partition first, followed by the next best, and so on, until all examples have been classified.

The downside to the greedy approach is that greedy algorithms are not guaranteed to generate the optimal, most accurate, or smallest number of rules for a particular dataset. By taking the low-hanging fruit early, a greedy learner may quickly find a single rule that is accurate for one subset of data; however, in doing so, the learner may miss the opportunity to develop a more nuanced set of rules with better overall accuracy on the entire set of data. However, without using the greedy approach to rule learning, it is likely that for all but the smallest of datasets, rule learning would be computationally infeasible.



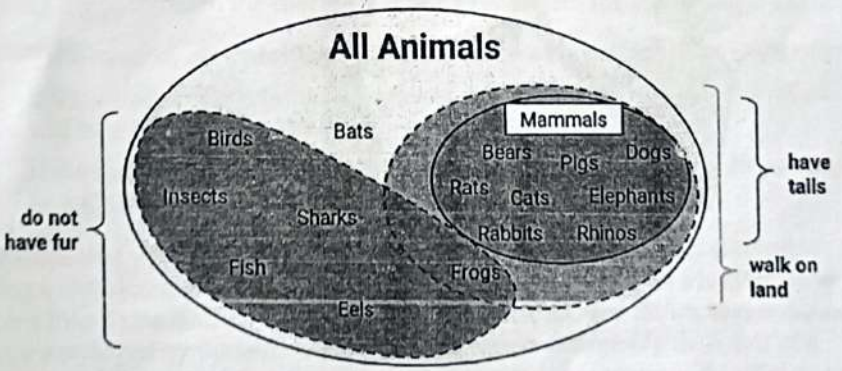
Though both trees and rules employ greedy learning heuristics, there are subtle differences in how they build rules. Perhaps the best way to distinguish them is to note that once divide and conquer splits on a feature, the partitions created by the split may not be re-conquered, only further subdivided. In this way, a tree is permanently limited by its history of past decisions. In contrast, once separate and conquer finds a rule, any examples not covered by all of the rule's conditions may be re-conquered.

To illustrate this contrast, consider the previous case in which we built a rule learner to determine whether an animal was a mammal. The rule learner identified three rules that perfectly classify the example animals:

- Animals that walk on land and have tails are mammals (bears, cats, dogs, elephants, pigs, rabbits, rats, rhinos)
- If the animal does not has fur, it is not a mammal (birds, eels, fish, frogs, insects, sharks)
- Otherwise, the animal is a mammal (bats)

In contrast, a decision tree built on the same data might have come up with four rules to achieve the same perfect classification:

- If an animal walks on land and has fur, then it is a mammal (bears, cats, dogs, elephants, pigs, rabbits, rats, rhinos)
- If an animal walks on land and does not have fur, then it is not a mammal (frogs)
- If the animal does not walk on land and has fur, then it is a mammal (bats)
- If the animal does not walk on land and does not have fur, then it is not a mammal (birds, insects, sharks, fish, eels)



The different result across these two approaches has to do with what happens to the frogs after they are separated by the "walk on land" decision. Where the rule learner allows frogs to be re-conquered by the "does not have fur" decision, the decision tree cannot modify the existing partitions, and therefore must place the frog into its own rule.

On one hand, because rule learners can reexamine cases that were considered but ultimately not covered as part of prior rules, rule learners often find a more parsimonious set of rules than those generated from decision trees. On the other hand, this reuse of data means that the computational cost of rule learners may be somewhat higher than for decision trees.

Example – identifying poisonous mushrooms with rule learners

Each year, many people fall ill and sometimes even die from ingesting poisonous wild mushrooms. Since many mushrooms are very similar to each other in appearance, occasionally even experienced mushroom gatherers are poisoned.

Unlike the identification of harmful plants such as a poison oak or poison ivy, there are no clear rules such as "leaves of three, let them be" to identify whether a wild mushroom is poisonous or edible. Complicating matters, many traditional rules, such as "poisonous mushrooms are brightly colored," provide dangerous or misleading information. If simple, clear, and consistent rules were available to identify poisonous mushrooms, they could save the lives of foragers.

Because one of the strengths of rule learning algorithms is the fact that they generate easy-to-understand rules, they seem like an appropriate fit for this classification task. However, the rules will only be as useful as they are accurate.

Step 1 – collecting data

To identify rules for distinguishing poisonous mushrooms, we will utilize the Mushroom dataset by Jeff Schlimmer of Carnegie Mellon University. The raw dataset is available freely at the UCI Machine Learning Repository (`http://archive.ics.uci.edu/ml`).

UNIT - 3

Regression Method

→ Regression is concerned with specifying the relationship between a single numeric dependent variable (the value to be predicted) and one or more numeric independent variables (the predictors).

→ The dependent variable depends upon the value of the independent variable or variables.

→ The simplest forms of regression assume that the relationship between the independent and dependent variables follows a straight line.

→ Regression analysis is commonly used for modeling complex relationships among data elements, estimating the impact of a treatment on an outcome, and extrapolating into the future.

→ Regression methods are also used for statistical hypothesis testing which determines whether a premise is likely to be true or false in light of the observed data.

Linear Regression - those that use straight lines.

* simple linear regression - when there is only a single independent variable
* multiple linear regression - In the case of two or more independent variables

Logistic Regression - used to model a binary categorical outcome

Poisson Regression - models integer count data. The method known as multinomial logistic Regression model

Simple linear regression.

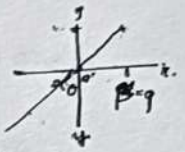
→ A simple linear regression model defines the relationship between a dependent variable and a single independent prediction variable using a line defined by an equation in the form:

$$y = \alpha + \beta x$$

(y: dependent; α: intercept; β: slop; x: independent variable.)

$y = b + wx$.

α & β → regression coefficients specifying the Y-intercept and slop of the line respectively.



Ordinary least squares estimation.

→ In order to determine the optimal estimates of α and β, an estimation method known as Ordinary Least Squares (OLS) was used.

→ In OLS regression, the slop and intercept are chosen so that they minimize the sum of the squared errors, that is the vertical distance between the predicted y value and the actual y value. These errors are known as residuals.

→ The goal of OLS regression can be expressed as the task of minimizing the following equation

$$\sum (y_i - y_1)^2 = \sum e_i^2$$

this eqn define e (the error) as the difference b/w the

actual y value and the predicted y value.

→ The error values are squared and summed across all the points in the data.

→ The soln for a depends on the value of b. It can be obtained using the formula

$$a = \bar{y} - b\bar{x}$$

→ value of b that results in the minimum squared error is

$$b = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = \frac{cov(x,y)}{var(x)}$$

$var(x) = \frac{\sum (x_i - \bar{x})^2}{n}$

$cov(x,y) = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{n}$

Correlations

→ The correlation between two variables is a number that indicates how closely their relationship follows a straight line.

→ correlation ranges between -1 & +1.

→ The extreme values indicate a perfectly linear relationship, while a correlation close to zero indicates the absence of a linear relationship

→ pearson's correlation

$$\rho_{x,y} = corr(x,y) = \frac{cov(x,y)}{\sigma_x \sigma_y}$$

$$cov(x, y) = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{n}$$

→ Measuring the correlation between two variables gives us a way to quickly guage the relationships among the independent and dependent variables.

Multiple linear regression.

→ used for most numeric prediction tasks.

Strengths

* By far the most common approach for modeling numeric data.
* can be adapted to model almost any modeling task.
* provides estimates of both the strength and size of the relationships among features and the outcome.

weaknesses

* Makes strong assumptions about the data.
* The model's form must be specified by the user in advance
* Does not handle missing data
* Only works with numeric features so categorical data requires extra processing
* Requires some knowledge of statistics to understand the mod

---

→ extension of simple linear regression.
→ The goal in both cases is similar - find values of beta coefficients that minimize the prediction error of a linear equation.
→ the key difference is that there are additional terms for additional independent variables.

→ Multiple regression equations generally follow the form of the following equation. The dependent variable y is specified as the sum of an intercept term α plus the product of the estimated β value and the x values for each of the i features.

→ An error term has been added here as a reminder that the predictions are not perfect. This represents the residual term.

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_i x_i + \varepsilon$$

$$= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_i x_i + \varepsilon$$

$$= \beta_0 x_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_i x_i + \varepsilon$$

$x_0 = 1$

Structure of regression equation is as follows