
Classification



Content

- Classification: Basic Concepts
 - Decision Trees
 - Perceptrons
 - Neural Network
 - Rule-based Classifier,
 - Nearest-Neighbor Classifier
 - Bayesian Classifier,
 - Principles of Modelling:
 - Train
 - Test
 - Cross-Validation.
-



(a) A spiral galaxy.



(b) An elliptical galaxy.


Figure 3.1. Classification of galaxies from telescope images taken from the NASA website.

Classification task

The data for a classification task consists of a collection of instances (records).



Each such instance is characterized by the tuple (x, y) , where x is the set of attribute values that describe the instance and y is the class label of the instance.



The attribute set x can contain attributes of any type, while the class label y must be categorical.

Classification task



Figure 3.2. A schematic illustration of a classification task.

Classification model

An abstract representation of the relationship between the attribute set and the class label.

The model can be represented in many ways, e.g., as a tree, a probability table, or simply, a vector of real-valued parameters.

Mathematically as a target function f that takes as input the attribute set x and produces an output corresponding to the predicted class label.

The model is said to classify an instance (x, y) correctly if $f(x) = y$.

Examples of classification task

Task	Attribute set	Class label
Spam filtering	Features extracted from email message header and content	spam or non-spam
Tumor identification	Features extracted from magnetic resonance imaging (MRI) scans	malignant or benign
Galaxy classification	Features extracted from telescope images	elliptical, spiral, or irregular-shaped

Example: Vertebrate Classification

Classifying vertebrates into mammals, reptiles, birds, fishes, and amphibians.

The attribute set includes characteristics of the vertebrate such as its body temperature, skin cover, and ability to fly.

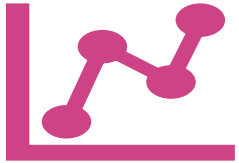
The data set can also be used for a binary classification task

- Mammal classification, by grouping the reptiles, birds, fishes, and amphibians into a single category called nonmammals.

Table 3.2. A sample data for the vertebrate classification problem.

Vertebrate Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibia
komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard	cold-blooded	scales	yes	yes	no	no	no	fish
shark								
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibia

Types of Classification



Predictive model

used as a predictive model to classify previously unlabelled instances.

A good classification model must provide accurate predictions with a fast response time.



Descriptive model

to identify the characteristics that distinguish instances from different classes.

This is particularly useful for critical applications, such as medical diagnosis

General Framework for Classification

Classification is the task of assigning labels or categories to unlabelled data instances.

Classifiers are used to perform classification tasks and are typically built using a learning algorithm.

Learning Algorithm and Induction:

- Learning a classification model involves using a learning algorithm to build a model from a labelled training dataset.
- The systematic approach for building this model from the training set is known as induction or learning a model.
- The trained model captures patterns and relationships between attributes and class labels.

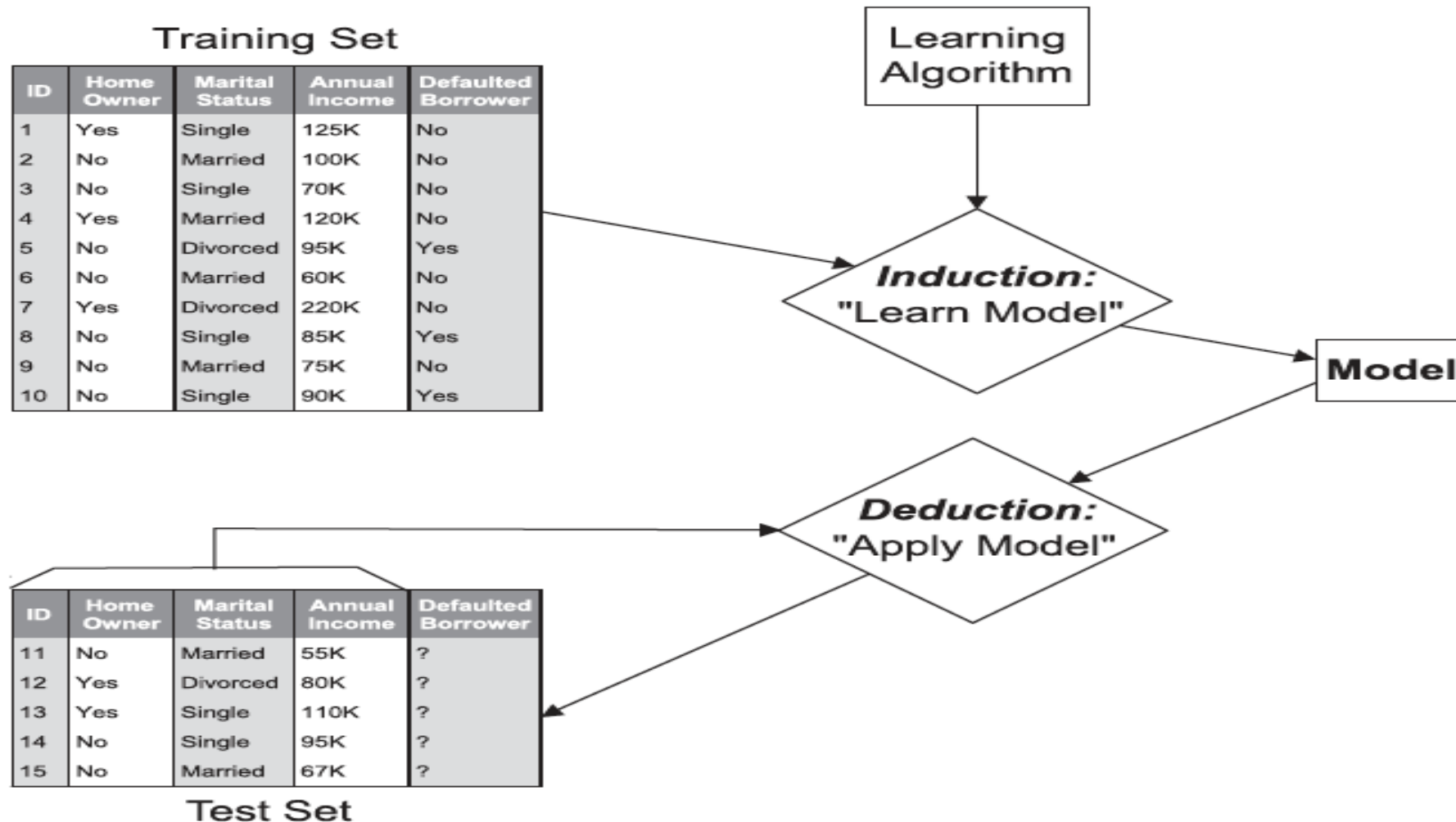
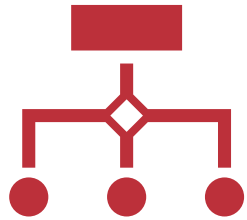


Figure 3.3. General framework for building a classification model.

Deduction and Predictions

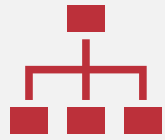


Once the model is trained, it can be applied to assign labels to new, unlabelled instances.
This process is called deduction or prediction.



Classification involves two main steps:
Learning a model
Applying the model to predict class labels.

Classification Techniques:



Classification techniques refer to general approaches for solving classification problems.



Each technique consists of a family of related models and algorithms for learning these models.



Examples of classification techniques

decision trees
neural networks
support vector machines.

Classifier vs. Model Terminology:



The terms "classifier" and "model" are often used interchangeably.



While every model defines a classifier, not every classifier is defined by a single model.



Some classifiers do not explicitly build a model, while others, like ensemble classifiers, combine multiple models.

Independence of Training and Test Sets:



The training and test datasets should be independent of each other to ensure that the induced model can generalize well to unseen instances.



Good generalization performance means that the model can accurately predict the class labels of new instances.

Performance Evaluation and Confusion Matrix

Model performance is evaluated by comparing predicted labels against true labels.

A confusion matrix summarizes the model's performance in a binary classification problem.

The matrix contains entries representing the number of instances from one class predicted as another class.

Confusion Matrix

- The confusion matrix for a binary classification problem.
- Each entry f_{ij} denotes the number of instances from class i predicted to be of class j .
- For example, f_{01} is the number of instances from class 0 incorrectly predicted as class 1.
- The number of correct predictions made by the model is $(f_{11} + f_{00})$ and the number of incorrect predictions is $(f_{10} + f_{01})$.

Table 3.4. Confusion matrix for a binary classification problem.

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

Evaluation metric

summarizing this information into a single number makes it more convenient to compare the relative performance of different models

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}. \quad (3.1)$$

For binary classification problems, the accuracy of a model is given by

$$\text{Accuracy} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}. \quad (3.2)$$

Error rate

another related metric, which is defined as follows for binary classification problems:

$$\text{Error rate} = \frac{\text{Number of wrong predictions}}{\text{Total number of predictions}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}. \quad (3.3)$$

Decision tree

a simple classification technique the decision tree classifier.

1.Classification Problem:

1. Consider the example of distinguishing mammals from non-mammals using a dataset of vertebrate species.

2.Asking Questions: Decision trees work by posing a series of questions about the attributes of a data point (in this case, a species). Each question helps narrow down the possibilities until a definitive class label can be assigned.

3.Hierarchy of Questions: These questions and their possible answers are organized into a hierarchical structure called a decision tree.

Decision tree



Types of Nodes in a Decision Tree:

Root Node: The starting point of the tree with no incoming links and multiple outgoing links.

Internal Nodes: These nodes have one incoming link and two or more outgoing links. They represent attribute test conditions.

Leaf Nodes: Terminal nodes with one incoming link and no outgoing links. They are associated with a class label.

Attribute Test Conditions:

Non-terminal nodes (root and internal nodes) contain attribute test conditions, typically defined using a single attribute. Each outcome of the test condition leads to one of the child nodes.

Classifying with a Decision Tree:

To classify a new data point (e.g., a new species), start at the root node, apply the attribute test condition, and follow the appropriate branch based on the outcome of the test. Continue this process until a leaf node is reached, and assign the class label associated with that leaf node to the data point.

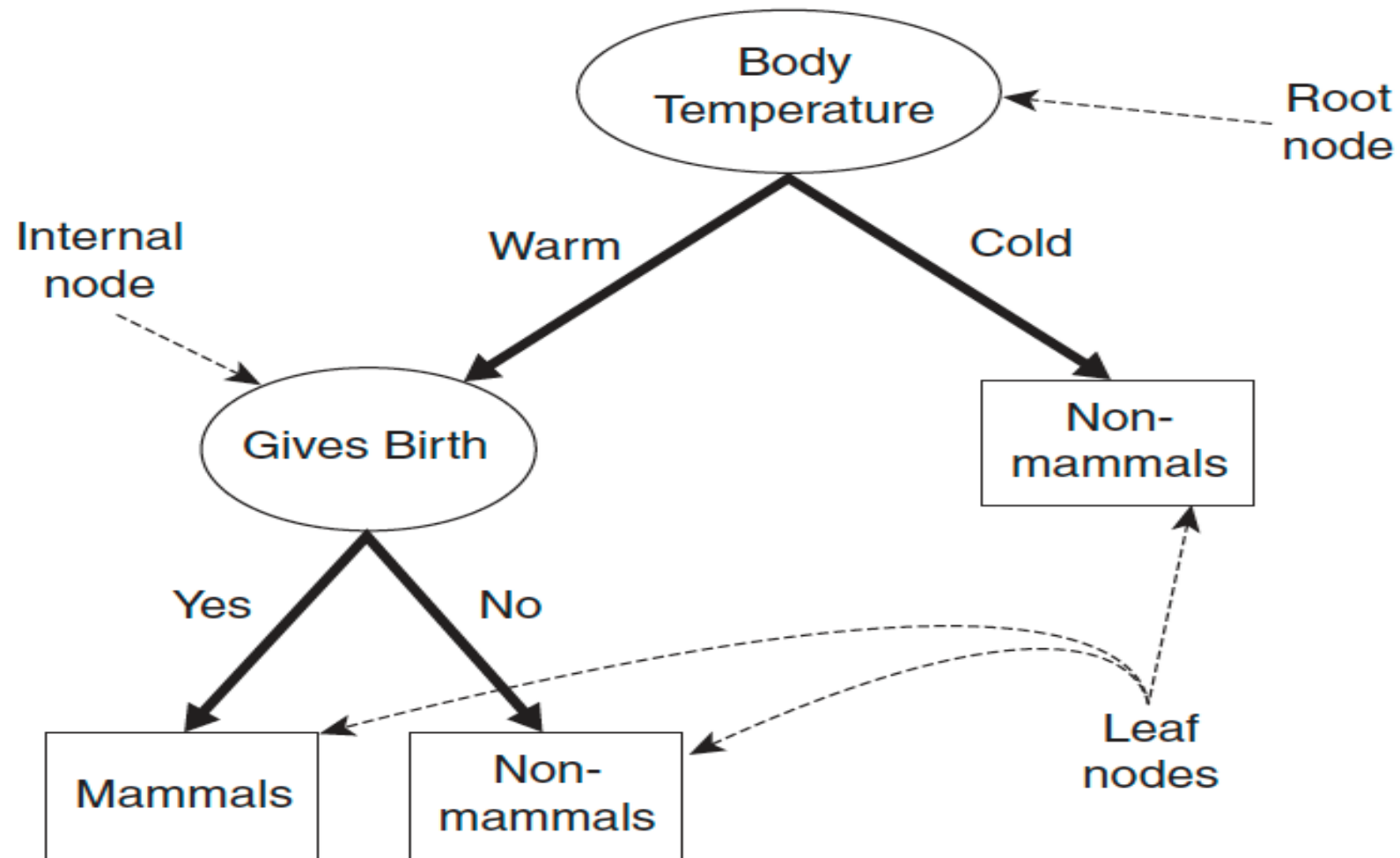


Figure 3.4. A decision tree for the mammal classification problem.

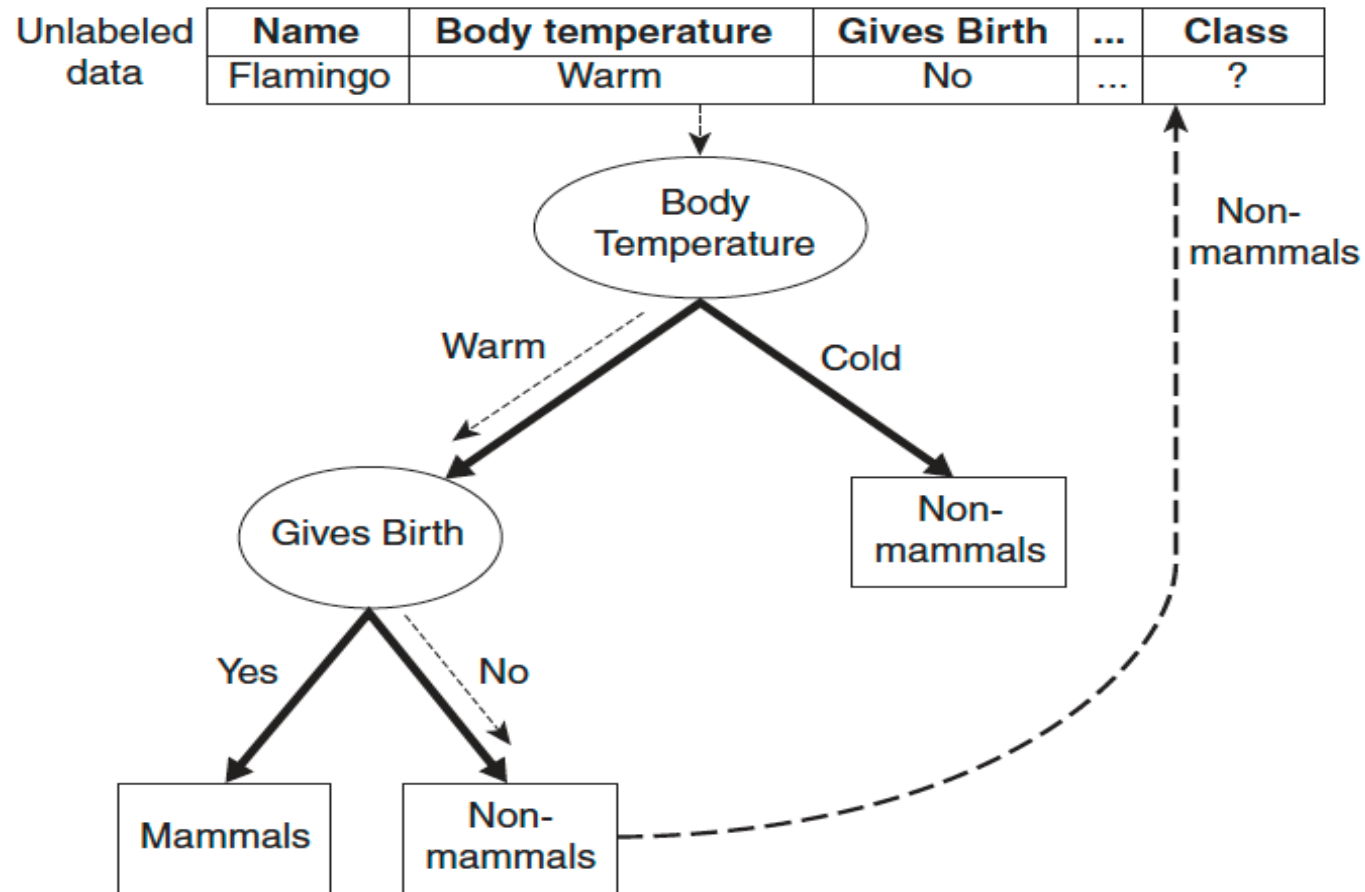


Figure 3.5. Classifying an unlabeled vertebrate. The dashed lines represent the outcomes of applying various attribute test conditions on the unlabeled vertebrate. The vertebrate is eventually assigned to the `Non-mammals` class.

A	B	C	D	Class
T	T	T	T	T
T	T	T	F	F
T	T	F	T	F
T	T	F	F	T
T	F	T	T	F
T	F	T	F	T
T	F	F	T	T
T	F	F	F	F
F	T	T	T	F
F	T	T	F	T
F	T	F	T	T
F	T	F	F	F
F	F	T	T	T
F	F	T	F	F
F	F	F	T	F
F	F	F	F	T

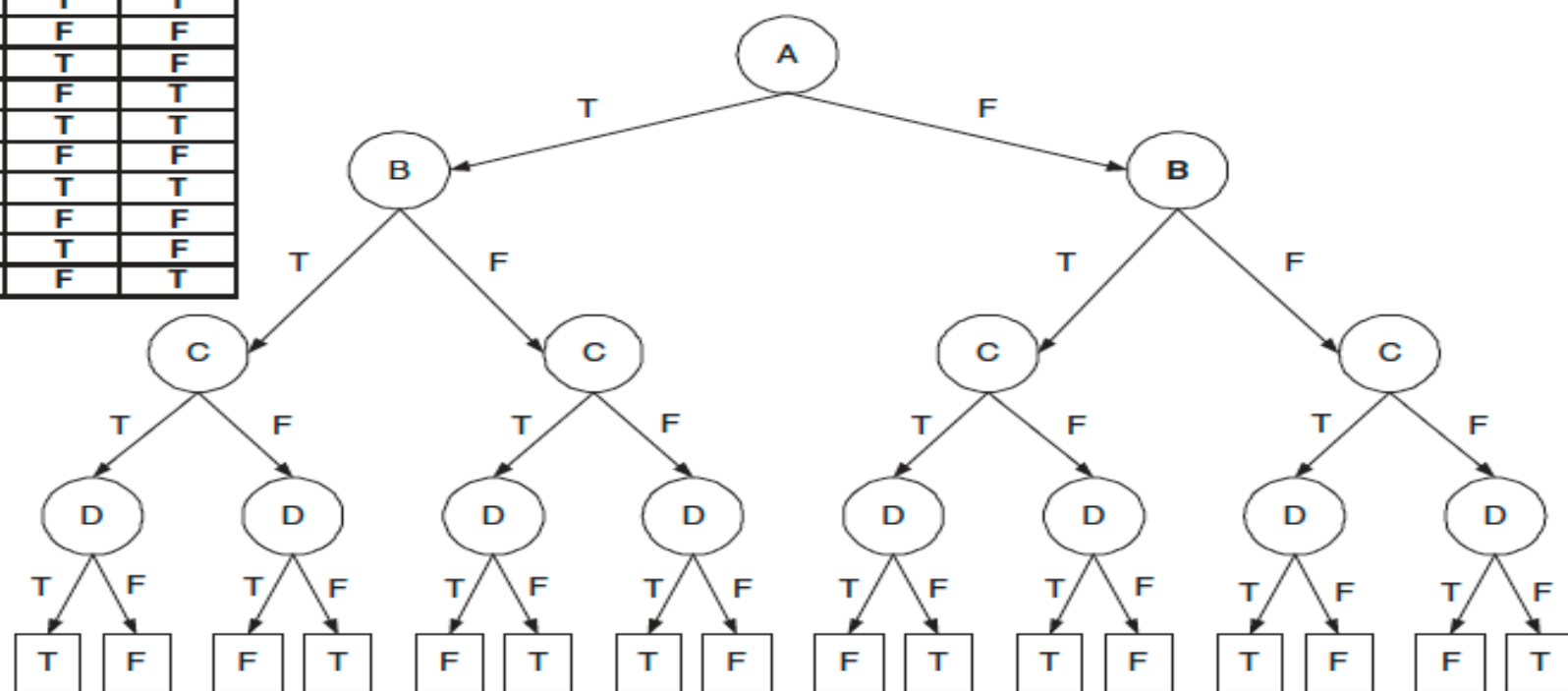


Figure 4.1. Decision tree for parity function of four Boolean attributes.

DECISION TREE INDUCTION ALGORITHM



Hunt's Algorithm:

it serves as a basis for some of the more complex algorithms.



CART (Classification and regression trees)

a non-parametric technique, uses the Gini index to select attribute for 1st split



ID3, C4.5:

Picks the attribute with the highest reduction in entropy to determine which attribute should the data be split with first



SLIQ, SPRINT

Scalable algorithms that have been proposed to deal with the issues the greedy algorithms above present

Hunt's Algorithm

a decision tree algorithm that builds a decision tree by partitioning a training dataset into increasingly pure subsets.

In a top-down, recursive manner

It follows a greedy approach to partition the training data based on attribute tests.

Steps in hunt's algo

Initialization

- Start with a single root node that represents all the training instances.

Check for Homogeneity:

- If all the training instances at a node belong to the same class
- create a leaf node for that class and stop expanding this branch

Attribute Selection:

- If the training instances at a node belong to multiple classes, select an attribute to split the data.
- This attribute selection is often based on a splitting criterion like information gain (used in ID3), gain ratio (used in C4.5), or Gini impurity (used in CART).
- The attribute that best separates the instances into different classes is chosen for splitting.

Steps in hunt's algo

Splitting:

- Create child nodes for each possible outcome of the selected attribute.
- Instances are distributed to these child nodes based on their attribute values.
- Continue the recursive expansion of the tree for each child node, considering only the instances associated with that node.

Stopping Criteria:

- The recursion stops when one of the following conditions is met
- All instances at a node belong to the same class, and a pure leaf node is created.
- No further attributes are available for splitting (e.g., all attributes have been used, or the depth limit of the tree is reached).
- A predefined stopping criterion is met (e.g., a minimum number of instances at a node or a minimum information gain threshold).

Assigning Class Labels:

- When the recursion stops for a particular branch, a class label is assigned to each leaf node.
- The label is determined based on the majority class of the training instances associated with that node.

Hunt's algo

X_t : the set of training records for node t

$y = \{y_1, \dots, y_c\}$: class labels

- Step 1: If all records in X_t belong to the same class y_t , then t is a leaf node labeled as y_t
 - Step 2: If X_t contains records that belong to more than one class,
 - select attribute test condition to partition the records into smaller subsets
 - Create a child node for each outcome of test condition
 - Apply algorithm recursively for each child
-

Example- Hunt's Algo

- **Initial Leaf Node (Figure 3.6(a)):**
 - The tree starts with a single leaf node, and it is labeled as "Defaulted = No" because it represents the majority class in the training data. The training error of this initial tree is 30% because three out of ten training instances have the class label "Defaulted = Yes."
 - Since the leaf node contains instances from more than one class, it is not pure and can be further expanded.
 - **Attribute Selection (Choosing "Home Owner"):**
 - The algorithm selects the "Home Owner" attribute as the attribute to split the training instances. The specific criteria for choosing this attribute are not provided in the description but might be based on information gain, gain ratio, or another criterion.
 - **Binary Split on "Home Owner" (Figure 3.6(b)):**
 - The "Home Owner" attribute is used to split the training data into two subsets. Instances with "Home Owner = Yes" go to the left child of the root node, while the rest go to the right child.
 - **Recursive Expansion:**
 - For the left child, all instances have "Home Owner = Yes," and since this subset is pure (all instances have the same class label, "Defaulted = No"), a leaf node labeled "Defaulted = No" is created, and no further splitting is required.
 - For the right child, it contains instances from both class labels, so further splitting is needed. The specific attribute and criteria for the next split are not provided in the description, but the algorithm continues to recursively expand this branch to create additional nodes and leaf nodes until certain stopping criteria are met.
-

	binary	categorical	continuous	class
Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Figure 4.6. Training set for predicting borrowers who will default on loan payments.

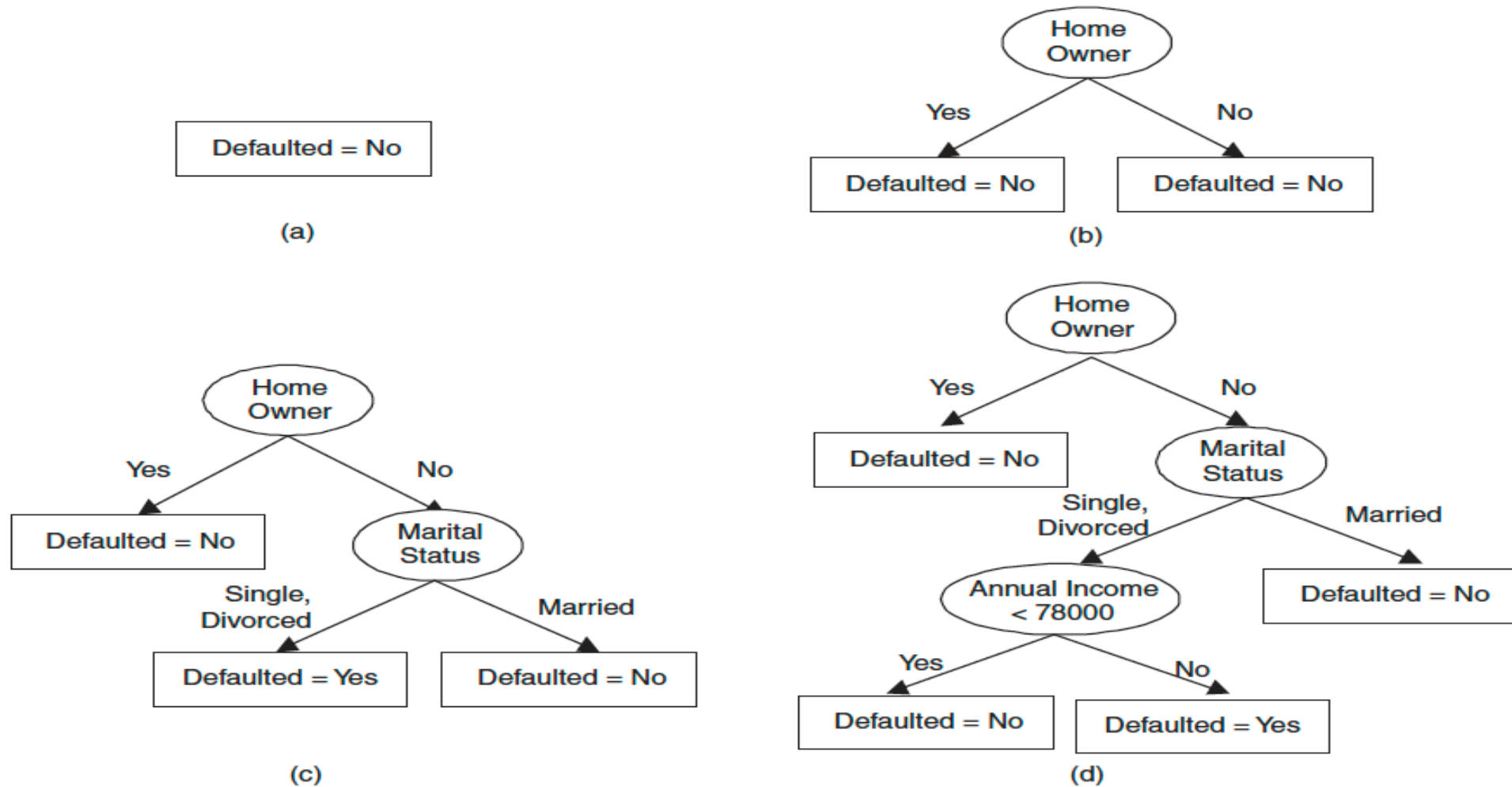


Figure 3.6. Hunt's algorithm for building decision trees.

DATA SET

Table 4.2. Data set for Exercise 3.

Instance	a_1	a_2	a_3	Target Class
1	T	T	1.0	+
2	T	T	6.0	+
3	T	F	5.0	—
4	F	F	4.0	+
5	F	T	7.0	—
6	F	T	3.0	—
7	F	F	8.0	—
8	T	F	7.0	+
9	F	T	5.0	—

QUESTION

1. 1. Consider the training examples shown in Table 4.2 for a binary classification problem.
 1. What is the entropy of this collection of training examples with respect to the positive class?
 2. What are the information gains of a_1 and a_2 relative to these training examples?
 3. For a_3 , which is a continuous attribute, compute the information gain for every possible split.
 4. What is the best split (among a_1 , a_2 , and a_3) according to the information gain?
What is the best split (between a_1 and a_2) according to the classification error rate?
-

ANSWER 1

There are four positive examples and five negative examples. Thus, $P(+) = 4/9$ and $P(-) = 5/9$. The entropy of the training examples is $-4/9 \log_2(4/9) - 5/9 \log_2(5/9) = 0.9911$.

ANSWERS 2

For attribute a_1 , the corresponding counts and probabilities are:

a_1	+	-
T	3	1
F	1	4

The entropy for a_1 is

$$\begin{aligned} & \frac{4}{9} \left[- (3/4) \log_2(3/4) - (1/4) \log_2(1/4) \right] \\ + & \frac{5}{9} \left[- (1/5) \log_2(1/5) - (4/5) \log_2(4/5) \right] = 0.7616. \end{aligned}$$

Therefore, the information gain for a_1 is $0.9911 - 0.7616 = 0.2294$.

For attribute a_2 , the corresponding counts and probabilities are:

a_2	+	-
T	2	3
F	2	2

The entropy for a_2 is

$$\begin{aligned} & \frac{5}{9} \left[- (2/5) \log_2(2/5) - (3/5) \log_2(3/5) \right] \\ + & \frac{4}{9} \left[- (2/4) \log_2(2/4) - (2/4) \log_2(2/4) \right] = 0.9839. \end{aligned}$$

Therefore, the information gain for a_2 is $0.9911 - 0.9839 = 0.0072$.

ANSWERS 3

a_3	Class label	Split point	Entropy	Info Gain
1.0	+	2.0	0.8484	0.1427
3.0	-	3.5	0.9885	0.0026
4.0	+	4.5	0.9183	0.0728
5.0	-			
5.0	-	5.5	0.9839	0.0072
6.0	+	6.5	0.9728	0.0183
7.0	+			
7.0	-	7.5	0.8889	0.1022

The best split for a_3 occurs at split point equals to 2.

ANSWERS 4

Answer:

For attribute a_1 : error rate = $2/9$.

For attribute a_2 : error rate = $4/9$.

Therefore, according to error rate, a_1 produces the best split.

Assumptions made in Hunt's algorithm and how to handle them

Empty Child Nodes:

If a child node has no training instances due to a particular attribute value not being present in the dataset, you can treat it as a leaf node.

Assign the class label that is most frequent among the instances of the parent node.

This way, you maintain the structure of the tree and ensure that the decision tree covers all possible attribute values.

Identical Attribute Values, Different Class Labels:

When all training instances at a node have identical attribute values but different class labels, you can handle this by creating a leaf node for that subset.

Assign the class label that is most frequent among the instances at that node.

This ensures that the tree provides a prediction based on the most common class within that subset.

Characteristics of Decision Tree Classifiers

Applicability:

- Decision trees are a nonparametric approach, versatile for building classification models.
- They don't require prior assumptions about the data's probability distribution
- Handle both categorical and continuous data without the need for extensive data pre-processing.
- They can also handle multiclass problems without decomposition and are relatively easy to interpret.

Expressiveness:

- Decision trees can represent discrete-valued functions universally.
- They can encode any function of discrete-valued attributes and provide compact representations when multiple attribute combinations lead to the same outcome.
- However, not all decision trees can be simplified for certain complex functions.

Computational Efficiency:

- Decision tree algorithms often use heuristic-based approaches to efficiently search for the best tree structure.
- They can quickly construct reasonably good trees, even with large training sets.
- Once constructed, classifying a test record is fast, with a time complexity of $O(w)$, where w is the maximum depth of the tree.

Characteristics of Decision Tree Classifiers

Handling Missing Values:

- Probabilistic splits (e.g., C4.5), surrogate splits (e.g., CART), and the separate class method (e.g., CHAID) for test instances.
- Strategies during training include excluding instances with missing values for attribute selection and propagating instances with missing values to child nodes.

Handling Interactions among Attributes:

- Decision trees might overlook attributes that are individually weak but powerful when used together.
- Greedy nature of attribute selection can result in overly complex trees.

Handling Irrelevant Attributes:

- Irrelevant attributes are those that do not contribute to the classification task.
- Decision trees can handle these attributes, but in complex classification problems with many irrelevant attributes, they may be selected by chance.
- Feature selection techniques can help address this issue.

Characteristics of Decision Tree Classifiers

Handling Redundant Attributes:

- Redundant attributes, which are strongly correlated with others.
- Only one of the correlated attributes will be chosen for splitting.

Using Rectilinear Splits:

- Use single attributes for test conditions, resulting in rectilinear (axis-aligned) decision boundaries.
- While this limits expressiveness in representing complex decision boundaries, oblique decision trees, which use multiple attributes for test conditions.

Choice of Impurity Measure:

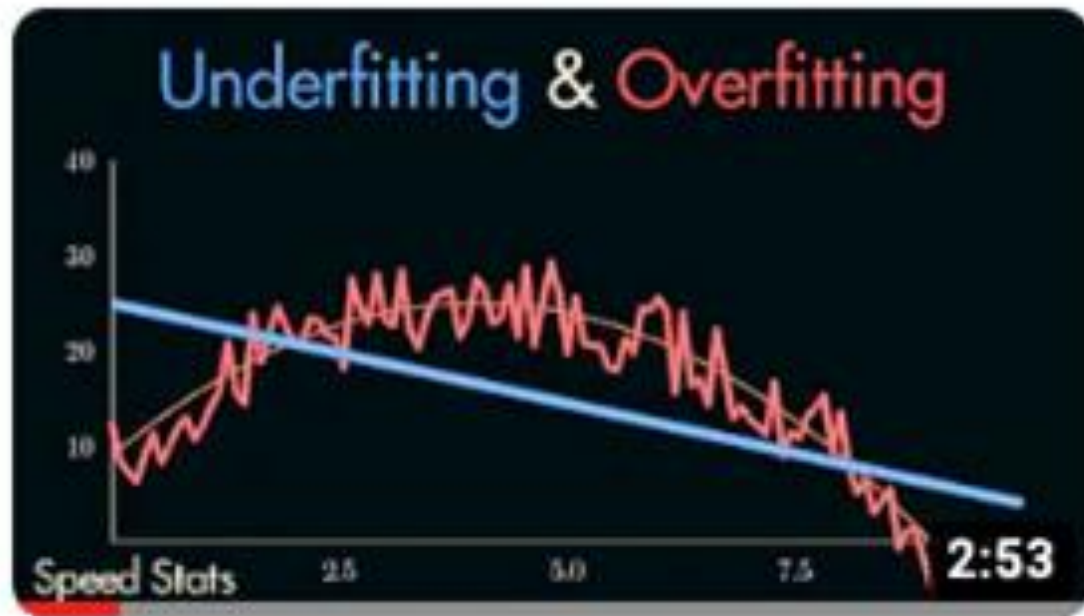
- The choice of impurity measure, such as Gini impurity or entropy, may not significantly impact the performance of decision trees.
- The pruning strategy used to optimize the tree has a more substantial effect on the final model.

MODEL OVER FITTING

A 3D rendering of a red puzzle piece standing out from a field of white puzzle pieces. The red piece is in the center, slightly raised, and has a glossy finish. The white pieces are arranged in a grid-like pattern around it, with some pieces missing, creating a sense of incompleteness. The lighting is soft, casting gentle shadows on the surface.

Video link for model overfitting

https://youtu.be/o3DztvnfAJg?si=hpObjxOO7f_DFItv



Underfitting & Overfitting - Exp

NStatum • 5.6K views

Underfitting and overfitting are some of the statistical/machine learning model. It is ther

Model under fitting



Data Description:

The dataset contains two classes, denoted as '+' and 'o.'

The 'o' class instances are generated from a uniform distribution, while the '+' class instances are a mix of data generated from a Gaussian distribution centered at (10, 10) with unit variance and data from the same uniform distribution as the 'o' class.



Training and Testing:

The data is split into a 10% training set and a 90% testing set.



Decision Trees:

Decision trees are constructed using the Gini index as the impurity measure.

The size of the trees, in terms of the number of leaf nodes, is varied from 1 to 150.



Model Underfitting:

When the tree has only one or two leaf nodes, both training and test error rates are high.

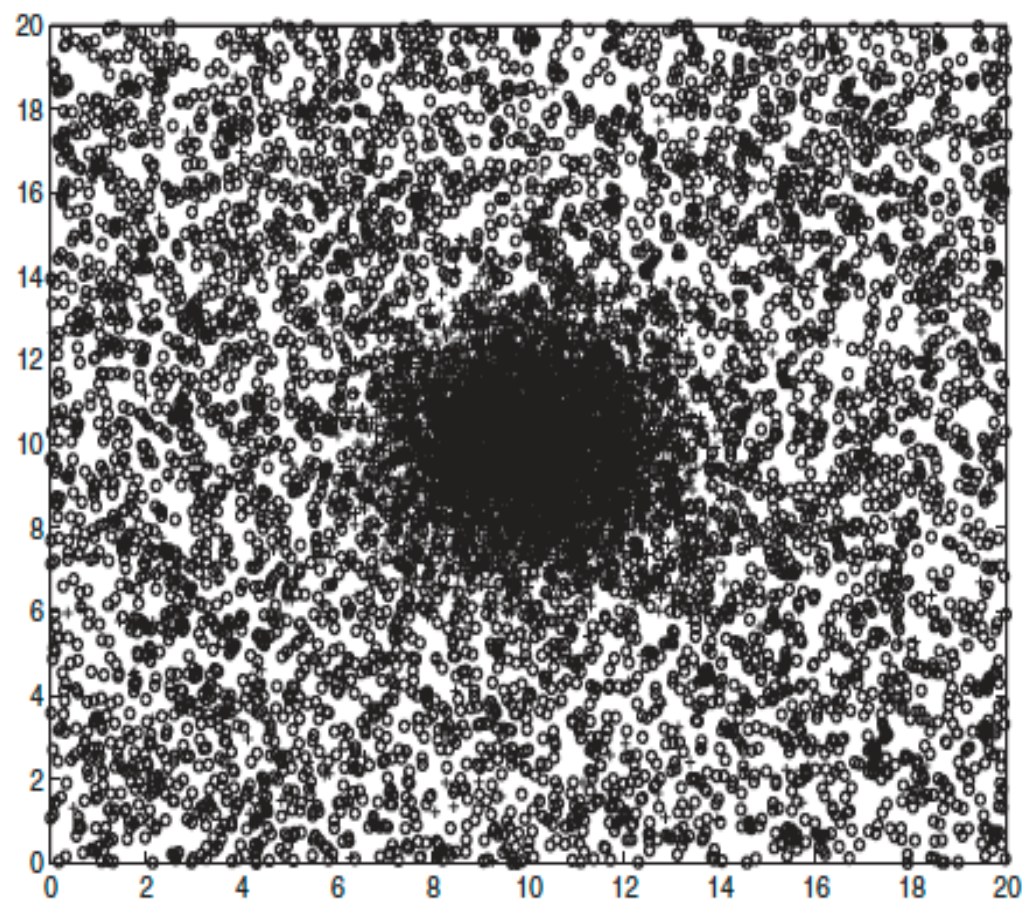
This is referred to as model underfitting. Underfitting occurs when the model is too simplistic to capture the underlying patterns in the data.



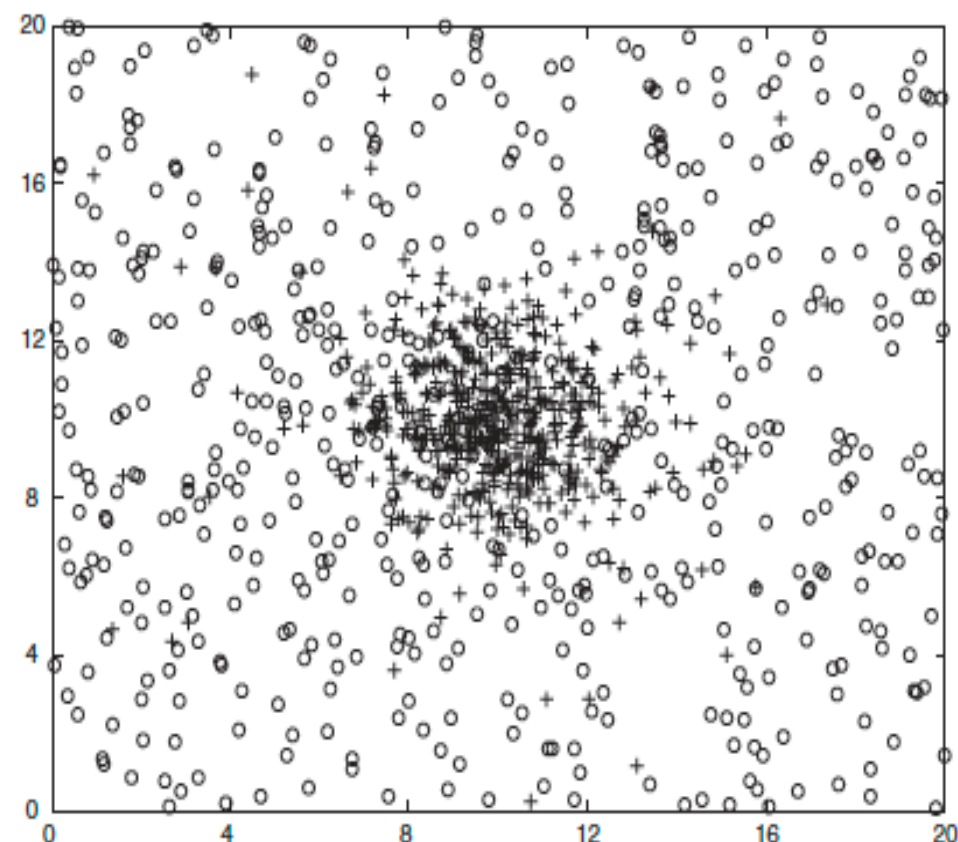
Model Improvement:

As the tree size increases from 1 to 8, both the training and test error rates decrease.

This means that larger trees can capture more complex decision boundaries, and the performance on the training set becomes representative of the generalization performance.



(a) Example of a 2-D data.



(b) Training set using 10% data.

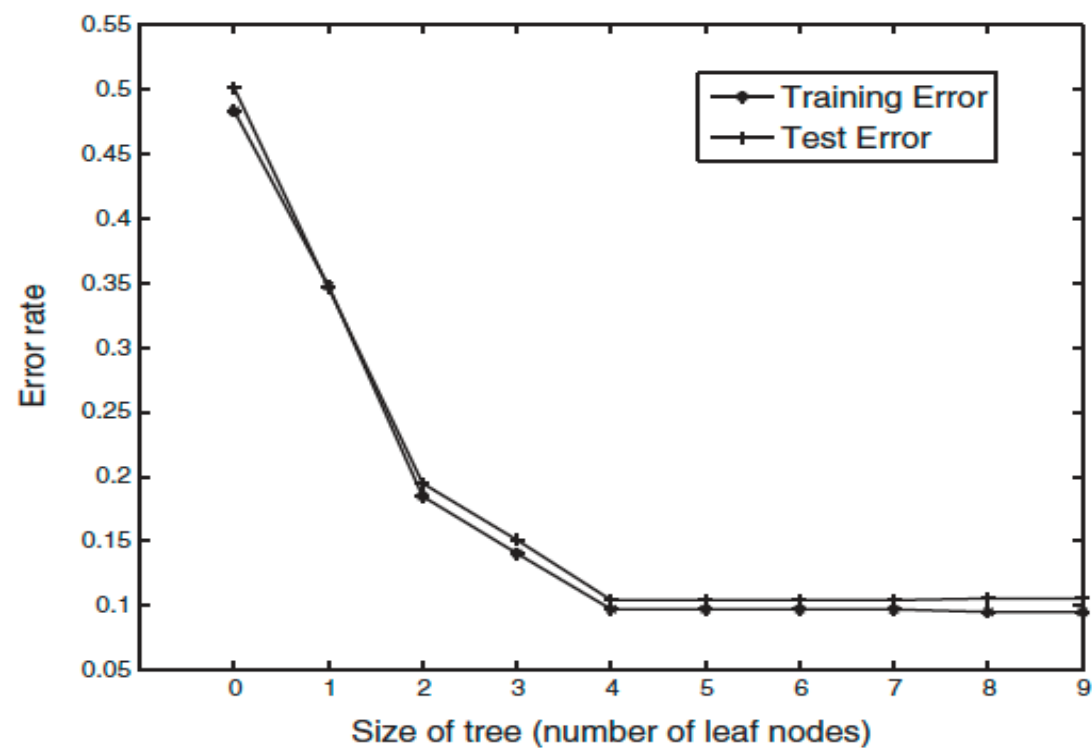
Model overfitting

Beyond a certain tree size, in this case, beyond 8, the training error continues to decrease and eventually reaches zero, while the test error rate starts to increase.

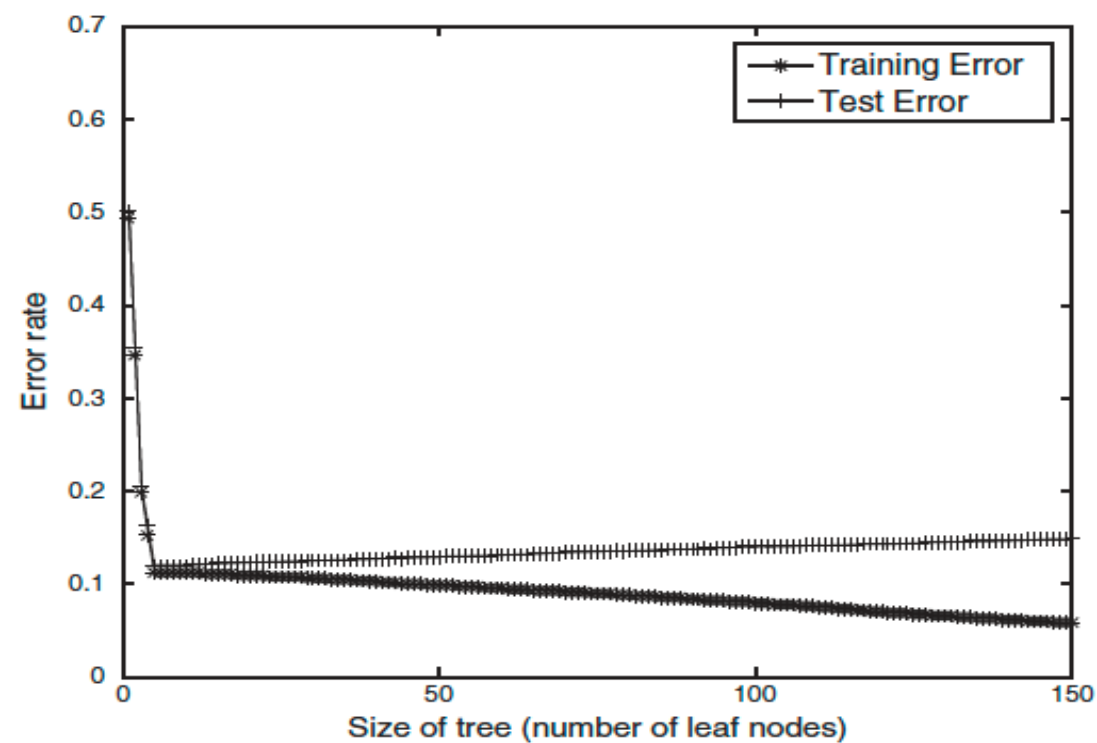
This widening gap between training and test error rates is indicative of model overfitting.

Model overfitting occurs when the model becomes too complex and starts fitting the training data noise rather than the actual underlying patterns.

As a result, it performs poorly on unseen data.



(a) Varying tree size from 1 to 8.



(b) Varying tree size from 1 to 150.

Figure 3.23. Effect of varying tree size (number of leaf nodes) on training and test errors.

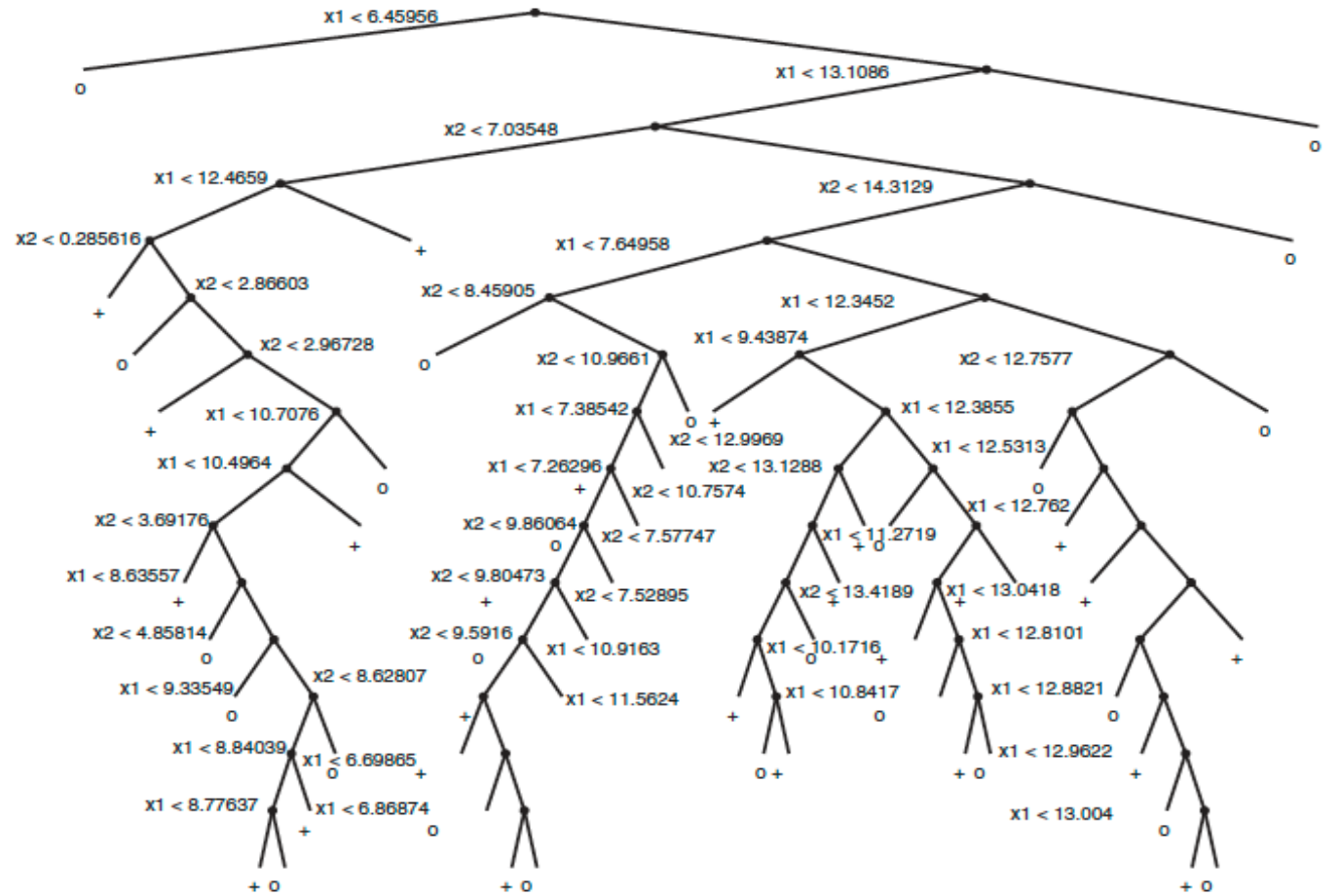
Reason

Illustration with Decision Trees:

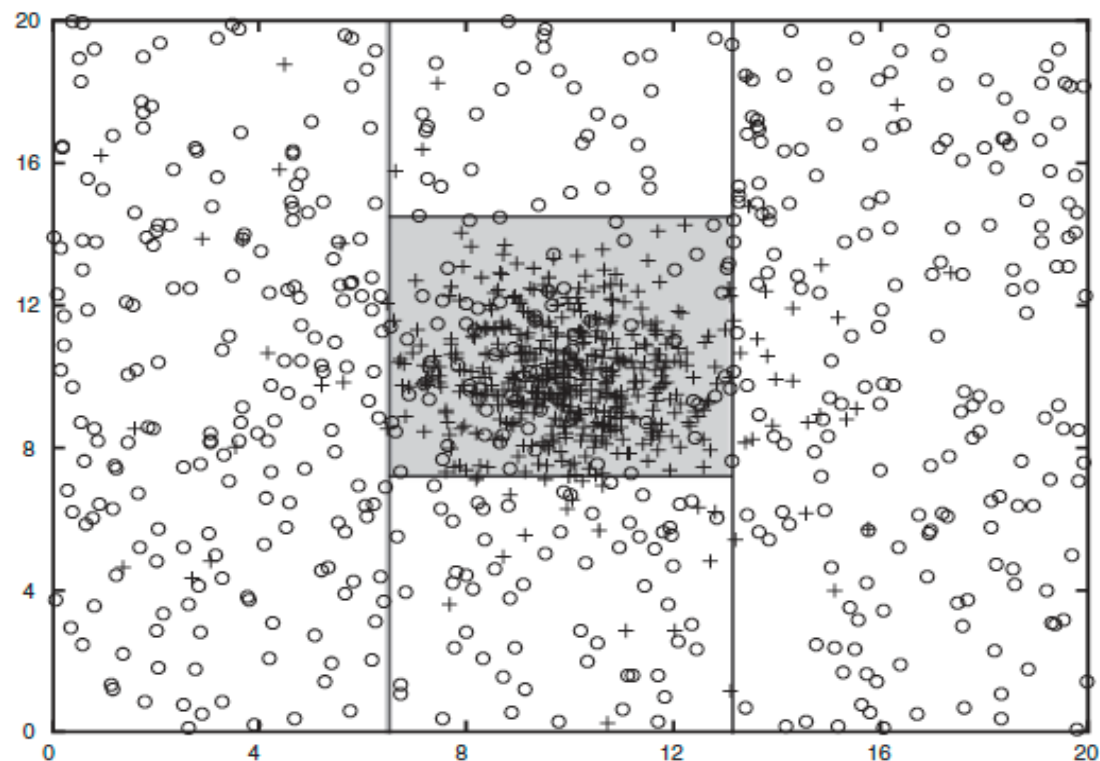
- Two trees of sizes 5 and 50 are compared.
- The tree of size 5 appears simpler, with decision boundaries that provide a reasonable approximation of the ideal decision boundary (in this case, a circle centred around a Gaussian distribution).
- Both its training and test error rates are low and close to each other, indicating good generalization.

The Complex Tree:

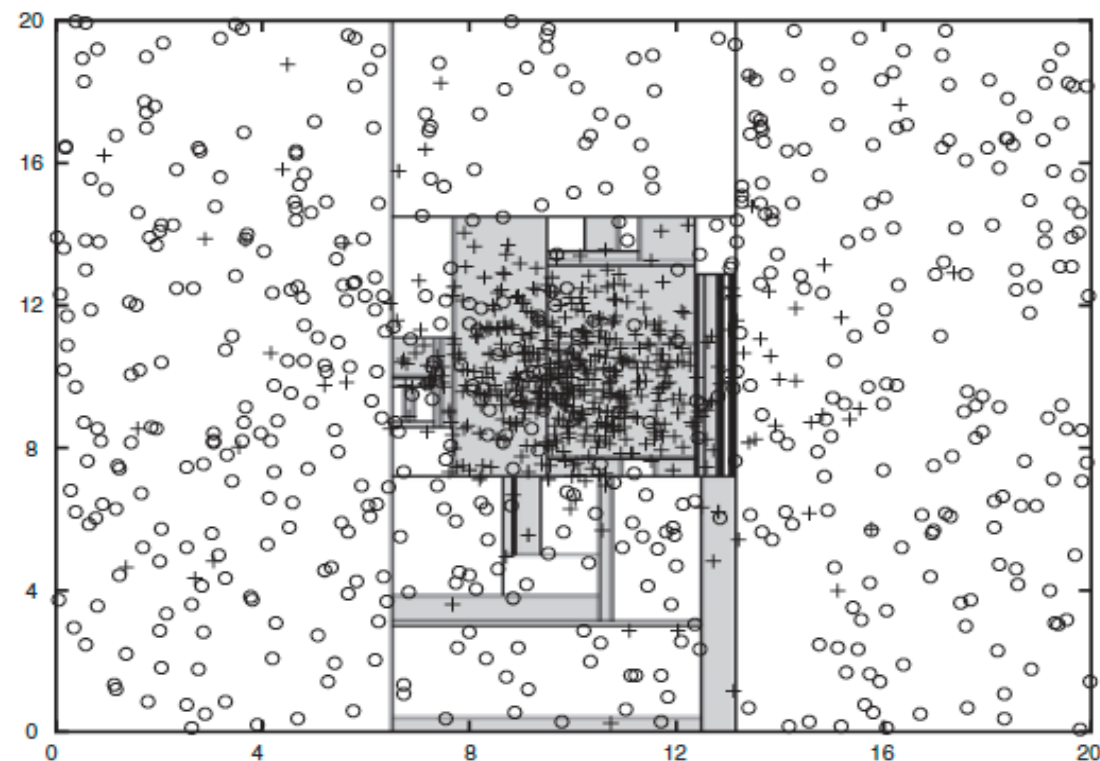
- In contrast, the tree of size 50 is much more complex, with intricate decision boundaries.
 - Some of its decision boundaries attempt to cover narrow regions in the input space that contain only a few training instances of the '+' class.
 - These specific patterns are highly specific to the training set and may not represent the true data distribution.
-



(b) Decision tree with 50 leaf nodes.



(c) Decision boundary for tree with 5 leaf nodes.



(d) Decision boundary for tree with 50 leaf nodes.

Figure 3.24. Decision trees with different model complexities.

Factors Influencing Overfitting

Limited Training Size:

- A training set with a finite number of instances can only provide a limited representation of the overall data.
- Therefore, patterns learned from a small training set may not fully represent the true patterns in the overall data, leading to overfitting.
- Increasing the training size by using more instances can reduce the effect of overfitting.

High Model Complexity:

- A model that is too complex, such as the tree of size 50, may try to fine-tune itself to specific training patterns, leading to poor generalization.
 - This can be mitigated by using simpler models or by having enough training data to support the complexity.
-

Example: Effect of Training Size

Effect of Training Size:

- in the example is provided to demonstrate the effect of training size.
- When the training size is increased to 20% of the data (compared to 10% in a previous experiment), the training and test error rates show differences.
- The training error rate decreases more slowly with increasing tree size, and the gap between the training and test error rates is smaller.
- This suggests that using more training data makes the patterns learned by the model more generalizable.

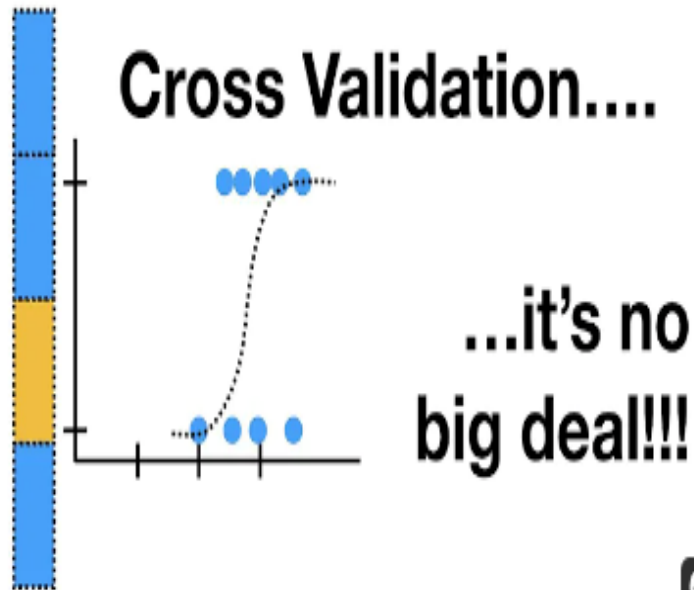
Decision Boundaries:

- It also highlights that the tree of size 50, trained with 20% of the data, is more effective in capturing the general boundaries of the '+' instances without being overly specific to the training set's noise.

MODEL EVALUATION

Video for Model Evaluation

<https://youtu.be/fSytzGwwBVw?si=U9hglavuHyJLrT7T>



Machine Learning Fundamentals: Cross Validation

919K views • 5 years ago



StatQuest with Josh Starmer ✓

One of the fundamental concepts in machine learning is Cross Validation. It's how we decide which machine learning method ...

Subtitles

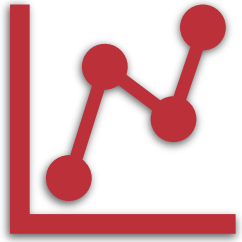


Motivation for using Cross Validation

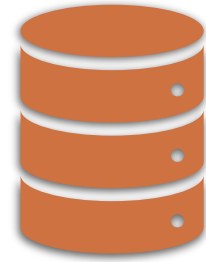
6 chapters ▾

6:05

MODEL EVALUATION



Model evaluation is essential to assess the performance of a classification model on unseen data, separate from the data used for model training.



Two primary approaches for partitioning the data into training and test sets and computing the test error rate are described:

1. *Holdout method*
2. *k-fold cross-validation*

Holdout Method



The labelled dataset is randomly split into two disjoint subsets:

The training set (D_{train})

The test set (D_{test}).



A classification model is trained using D_{train} , and its error rate on D_{test} (err_{test}) is used as an estimate of the model's generalization error.



The proportion of data reserved for training and testing is typically at the discretion of the analysts (e.g., two-thirds for training and one-third for testing).

k-Fold Cross-Validation



k-fold cross-validation aims to make efficient use of all labelled instances in the dataset.



The labelled data (D) is divided into k equal-sized partitions (or folds).

During each run, one partition is used as the test set, while the remaining partitions are used for training.



The procedure is repeated k times, with each partition serving as the test set once.



The total test error rate (err_{test}) is computed by aggregating the errors from each run and dividing by the total number of instances.



The choice of k depends on the specific problem, with smaller k resulting in a larger estimate of the generalization error and larger k reducing bias in the estimate.



Common values for k are between 5 and 10.



The *leave-one-out approach* ($k = N$) uses each data instance for testing once and the rest for training.

Types of K-fold cross validation

Stratified Cross-Validation:

- In k-fold cross-validation, it is important to ensure that the fraction of positive and negative instances in every partition is similar to the overall dataset.
- Stratified cross-validation achieves this by stratified sampling of positive and negative instances into k partitions.

Complete Cross-Validation:

- This method involves running k-fold cross-validation for every possible partitioning of the data into k partitions.
- It provides a more robust estimate of generalization error and its variance but is computationally expensive, especially for large datasets.

Error Estimate in k-fold cross- validation

Variance of Generalization Error Estimate:

- Running k-fold cross-validation multiple times with different random partitioning can be used to estimate the generalization error rate and its variance.

Interpretation of `errtest` in k-Fold Cross-Validation:

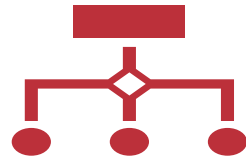
- In k-fold cross-validation, `errtest` does not reflect the generalization error rate of any specific model but represents the expected generalization error of the model selection approach when applied to a training set of the same size as one of the training folds.
- In practice, `errtest` is often used as an estimate of the generalization error of a model built on the entire dataset, particularly when k is large.



Types of Classifier

Different Classification Techniques

Binary vs. Multiclass Classifiers



Binary Classifiers:

These classifiers assign each data instance to one of two possible labels, typically denoted as +1 and -1.

They are designed for problems where there are only two classes (e.g., spam vs. non-spam).



Multiclass Classifiers:

Multiclass classifiers handle problems with more than two possible labels.

They can be adapted from binary classifiers to work with multiple classes.

Deterministic vs. Probabilistic Classifiers:

Deterministic Classifiers:

- These classifiers provide a discrete label for each data instance they classify.

Probabilistic Classifiers:

- Probabilistic classifiers assign a continuous score between 0 and 1 to indicate the likelihood of an instance belonging to a particular class. These scores sum up to 1.
 - They are used when you need information about the confidence in class assignments.
-

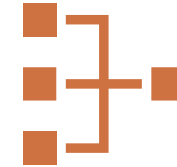
Linear vs. Nonlinear Classifiers:



Linear Classifiers:

Linear classifiers use a linear separating hyperplane to distinguish instances from different classes.

They are less flexible but less prone to overfitting.



Nonlinear Classifiers:

Nonlinear classifiers allow the construction of more complex, nonlinear decision surfaces.

They are more flexible but can be susceptible to overfitting.

Feature transformation can make linear classifiers more adaptable to nonlinear data.

Global vs. Local Classifiers



Global Classifiers:

Global classifiers fit a single model to the entire dataset.

They may not be effective when the relationship between attributes and class labels varies across the input space.



Local Classifiers:

Local classifiers divide the input space into smaller regions and fit distinct models to each region.

They are more flexible but can overfit, especially when regions have few training examples.

An example is the k-nearest neighbour classifier

Generative vs. Discriminative Classifiers



Generative Classifiers:

Generative classifiers not only predict class labels but also seek to describe the underlying mechanism that generates instances for each class.

They provide insights into class-specific data characteristics.

Examples include Naïve Bayes and Bayesian networks.



Discriminative Classifiers:

Discriminative classifiers directly predict class labels without explicitly modelling the distribution of each class.

They focus on class separation and are simpler.

Examples include decision trees, rule-based classifiers, nearest neighbour classifiers, artificial neural networks, and support vector machines.

Rule based Classifier

Discriminative Classifiers

Rule-Based Classifier

Classify records by using a collection of “if...then...” rules

Rule: $(Condition) \rightarrow y$

- where
 - *Condition* is a conjunctions of attributes
 - y is the class label
- *LHS*: rule *antecedent or condition*
- *RHS*: rule *consequent*
- Examples of classification rules:
 - $(\text{Blood Type}=\text{Warm}) \wedge (\text{Lay Eggs}=\text{Yes}) \rightarrow \text{Birds}$
 - $(\text{Taxable Income} < 50\text{K}) \wedge (\text{Refund}=\text{Yes}) \rightarrow \text{Evade}=\text{No}$

Rule-based Classifier (Example)

- R1: (Give Birth = no) \wedge (Can Fly = yes) \rightarrow Birds**
R2: (Give Birth = no) \wedge (Live in Water = yes) \rightarrow Fishes
R3: (Give Birth = yes) \wedge (Blood Type = warm) \rightarrow Mammals
R4: (Give Birth = no) \wedge (Can Fly = no) \rightarrow Reptiles
R5: (Live in Water = sometimes) \rightarrow Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
human	warm	yes	no	no	mammals
python	cold	no	no	no	reptiles
salmon	cold	no	no	yes	fishes
whale	warm	yes	no	yes	mammals
frog	cold	no	no	sometimes	amphibians
komodo	cold	no	no	no	reptiles
bat	warm	yes	yes	no	mammals
pigeon	warm	no	yes	no	birds
cat	warm	yes	no	no	mammals
leopard shark	cold	yes	no	yes	fishes
turtle	cold	no	no	sometimes	reptiles
penguin	warm	no	no	sometimes	birds
porcupine	warm	yes	no	no	mammals
eel	cold	no	no	yes	fishes
salamander	cold	no	no	sometimes	amphibians
gila monster	cold	no	no	no	reptiles
platypus	warm	no	no	no	mammals
owl	warm	no	yes	no	birds
dolphin	warm	yes	no	yes	mammals
eagle	warm	no	yes	no	birds

Application of Rule-Based Classifier

- A rule r **covers** an instance x if the attributes of the instance **satisfy the condition** of the rule

R1: (Give Birth = no) \wedge (Can Fly = yes) \rightarrow Birds

R2: (Give Birth = no) \wedge (Live in Water = yes) \rightarrow Fishes

R3: (Give Birth = yes) \wedge (Blood Type = warm) \rightarrow Mammals

R4: (Give Birth = no) \wedge (Can Fly = no) \rightarrow Reptiles

R5: (Live in Water = sometimes) \rightarrow Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
hawk	warm	no	yes	no	?
grizzly bear	warm	yes	no	no	?

The rule R1 covers a hawk \Rightarrow Bird

The rule R3 covers the grizzly bear \Rightarrow Mammal

Rule Coverage and Accuracy

- **Coverage** of a rule:
 - Fraction of records that satisfy the antecedent of a rule
- **Accuracy** of a rule:
 - Fraction of records that satisfy both the antecedent and consequent of a rule (over those that satisfy the antecedent)

the rule

"Gives Birth = yes) \wedge (Body Temperature = warm-blooded) \rightarrow Mammals"

- **Coverage** of 33% because it matches five out of fifteen instances.
- The **accuracy** of this rule is 100% because all five instances covered by the rule are indeed mammals.

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
human	warm	yes	no	no	mammals
python	cold	no	no	no	reptiles
salmon	cold	no	no	yes	fishes
whale	warm	yes	no	yes	mammals
frog	cold	no	no	sometimes	amphibians
komodo	cold	no	no	no	reptiles
bat	warm	yes	yes	no	mammals
pigeon	warm	no	yes	no	birds
cat	warm	yes	no	no	mammals
leopard shark	cold	yes	no	yes	fishes
turtle	cold	no	no	sometimes	reptiles
penguin	warm	no	no	sometimes	birds
porcupine	warm	yes	no	no	mammals
eel	cold	no	no	yes	fishes
salamander	cold	no	no	sometimes	amphibians
gila monster	cold	no	no	no	reptiles
platypus	warm	no	no	no	mammals
owl	warm	no	yes	no	birds
dolphin	warm	yes	no	yes	mammals
eagle	warm	no	yes	no	birds

How does Rule-based Classifier Work?

R1: (Give Birth = no) \wedge (Can Fly = yes) \rightarrow Birds

R2: (Give Birth = no) \wedge (Live in Water = yes) \rightarrow Fishes

R3: (Give Birth = yes) \wedge (Blood Type = warm) \rightarrow Mammals

R4: (Give Birth = no) \wedge (Can Fly = no) \rightarrow Reptiles

R5: (Live in Water = sometimes) \rightarrow Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
lemur	warm	yes	no	no	?
turtle	cold	no	no	sometimes	?
dogfish shark	cold	yes	no	yes	?

- A lemur triggers rule R3, so it is classified as a mammal
 - A turtle triggers both R4 and R5
 - A dogfish shark triggers none of the rules
-

Decider Data for Rule-Based Classifier



Mutually exclusive rules

No two rules are triggered by the same record.

This ensures that every record is covered by **at most** one rule.



Exhaustive rules

There exists a rule for each combination of attribute values.

This ensures that every record is covered by **at least** one rule.



Together these properties ensure that every record is covered by exactly one rule.

Rules

Non mutually exclusive rules

- A record may trigger more than one rule
- Solution?
 - Ordered rule set

Non exhaustive rules

- A record may not trigger any rules
- Solution?
 - Use a default class

Ordered Rule Set

Rules are ranked ordered according to their priority (e.g. based on their quality)

- An ordered rule set is known as a **decision list**

When a test record is presented to the classifier

- It is assigned to the class label of the highest ranked rule it has triggered
- If none of the rules fired, it is assigned to the default class

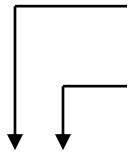
R1: (Give Birth = no) \wedge (Can Fly = yes) \rightarrow Birds

R2: (Give Birth = no) \wedge (Live in Water = yes) \rightarrow Fishes

R3: (Give Birth = yes) \wedge (Blood Type = warm) \rightarrow Mammals

R4: (Give Birth = no) \wedge (Can Fly = no) \rightarrow Reptiles

R5: (Live in Water = sometimes) \rightarrow Amphibians



Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
turtle	cold	no	no	sometimes	?

Direct Methods for Rule Extraction

Sequential Covering Algorithm:

- The algorithm begins with an empty decision list (R) and extracts rules for each class based on class prevalence.
- Iteratively, rules are generated for each class using the Learn-One-Rule function.
- Once a rule for a class is found, the training instances covered by the rule are removed, and the rule is added to the decision list (R).
- This process continues until a stopping criterion is met, and the algorithm proceeds to generate rules for the next class.

Learn-One-Rule Function:

- Finding an optimal rule is computationally expensive due to the exponential search space.
- The Learn-One-Rule function starts with an initial rule, which has an empty antecedent and a positive class in the consequent.
- The rule is iteratively refined until a stopping criterion is met.
- Foil's information gain is used to choose the best conjunct to add to the rule's antecedent based on both accuracy and support.

Direct Methods for Rule Extraction

Rule Pruning:

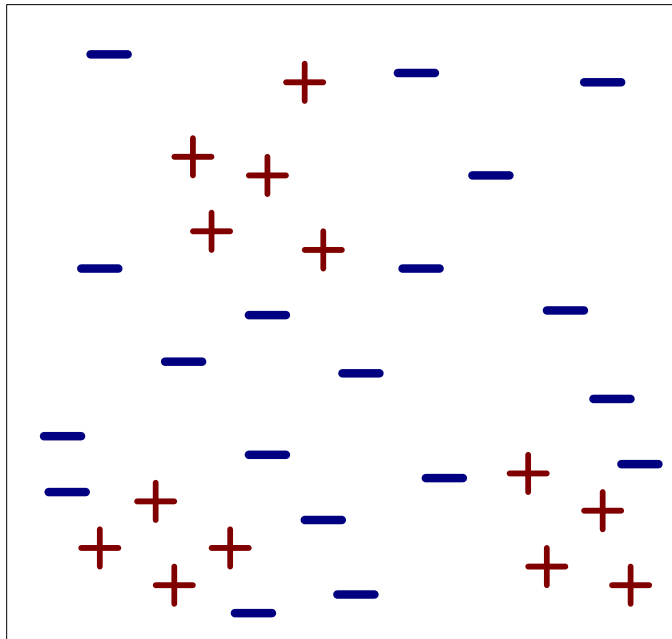
- Rules generated by the Learn-One-Rule function can be pruned based on their performance on a validation set.
- Pruning decisions are based on the $(p - n) / (p + n)$ metric, which is related to rule accuracy on the validation set.
- Pruning starts from the last conjunct added to the rule.
- Pruning aims to improve the generalization of rules by removing fewer effective conjuncts.

Building the Rule Set:

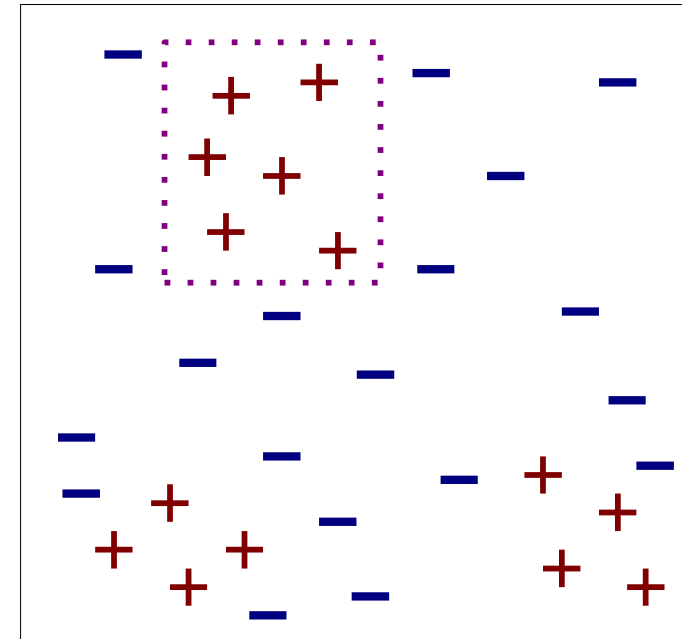
- After generating a rule, positive and negative examples covered by the rule are eliminated.
 - The rule is added to the rule set, provided it doesn't violate a stopping condition, often based on the minimum description length principle.
 - Another stopping condition is that the error rate of the rule on the validation set must not exceed 50%.
-

Building Classification Rules: Sequential Covering

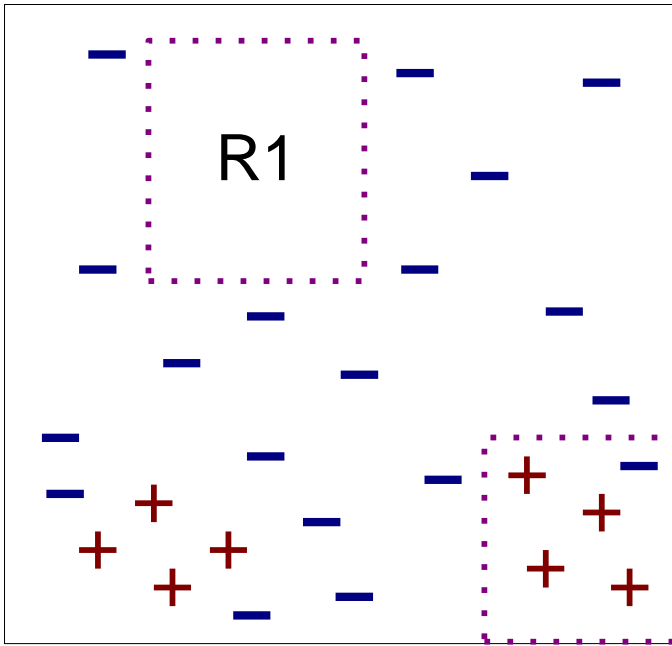
1. Start from an empty rule
2. Grow a rule using some **Learn-One-Rule** function
3. Remove training records **covered** by the rule
4. Repeat Step (2) and (3) until stopping criterion is met



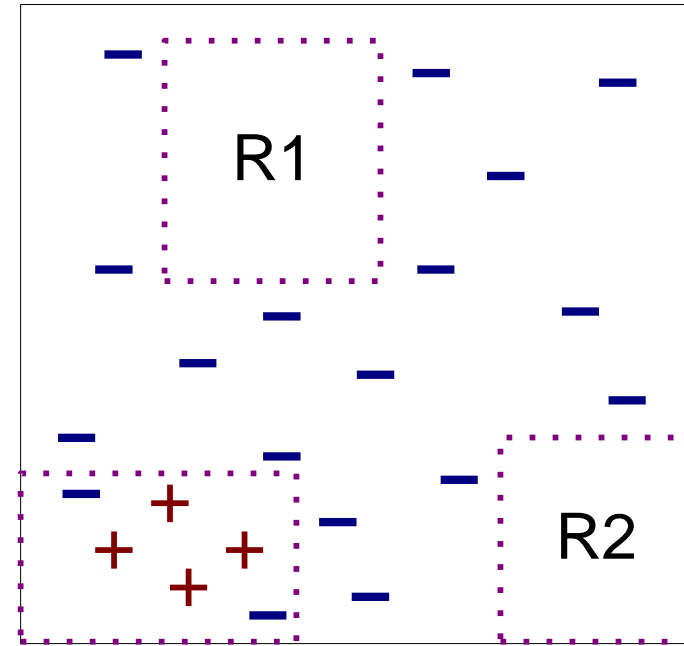
(i) Original Data



(ii) Step 1



(iii) Step 2



(iv) Step 3

- This approach is called a **covering** approach because at each stage a rule is identified that covers some of the instances

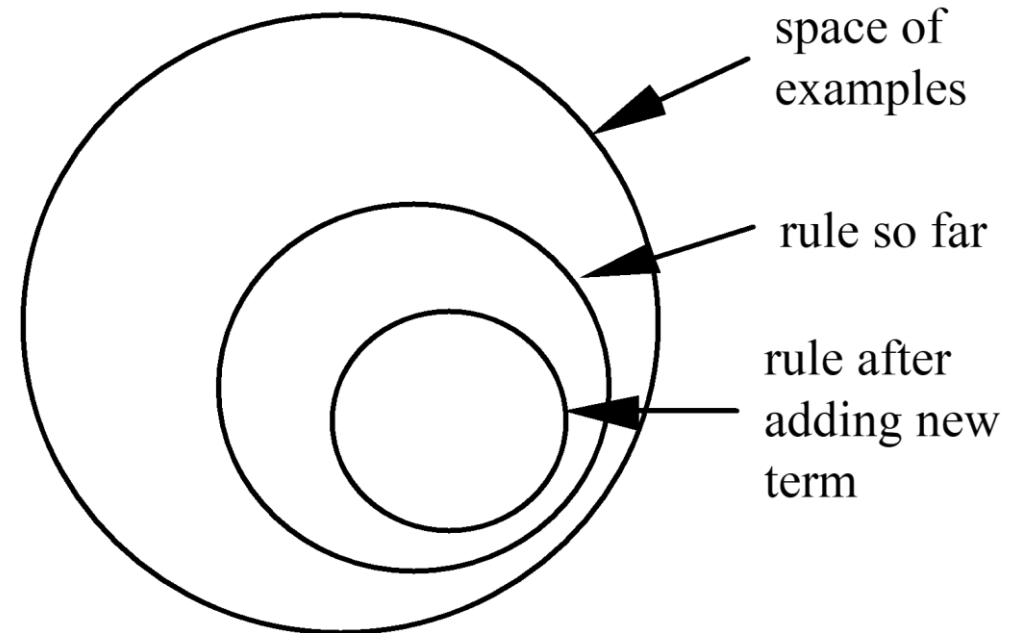
A simple covering algorithm

Generates a rule by adding tests that maximize rule's accuracy

Similar to situation in decision trees:
problem of selecting an attribute to split on.

- But: decision tree inducer maximizes overall purity

Here, each new test (growing the rule) reduces rule's coverage.



Sequential covering algorithm (RIPPER'S Algorithm)

- 1: Let E be the training instances and A be the set of attribute-value pairs, $\{(A_j, v_j)\}$.
 - 2: Let Y_0 be an ordered set of classes $\{y_1, y_2, \dots, y_k\}$.
 - 3: Let $R = \{ \}$ be the initial rule list.
 - 4: for each class $y \in Y_0 - \{y_k\}$ do
 - 5: while stopping condition is not met do
 - 6: $r \leftarrow \text{Learn-One-Rule}(E, A, y)$.
 - 7: Remove training instances from E that are covered by r .
 - 8: Add r to the bottom of the rule list: $R \leftarrow R \vee r$.
 - 9: end while
 - 10: end for
 - 11: Insert the default rule, $\{ \} \rightarrow y_k$, to the bottom of the rule list R .
-

Indirect Methods for Rule Extraction

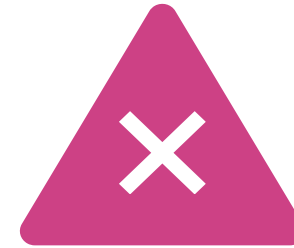


Principle of Rule Generation from Decision Trees:

Each path from the root node to a leaf node in a decision tree can be represented as a classification rule.

The conditions encountered along the path are used as conjuncts in the rule antecedent.

The class label assigned to the leaf node becomes the rule's consequent.

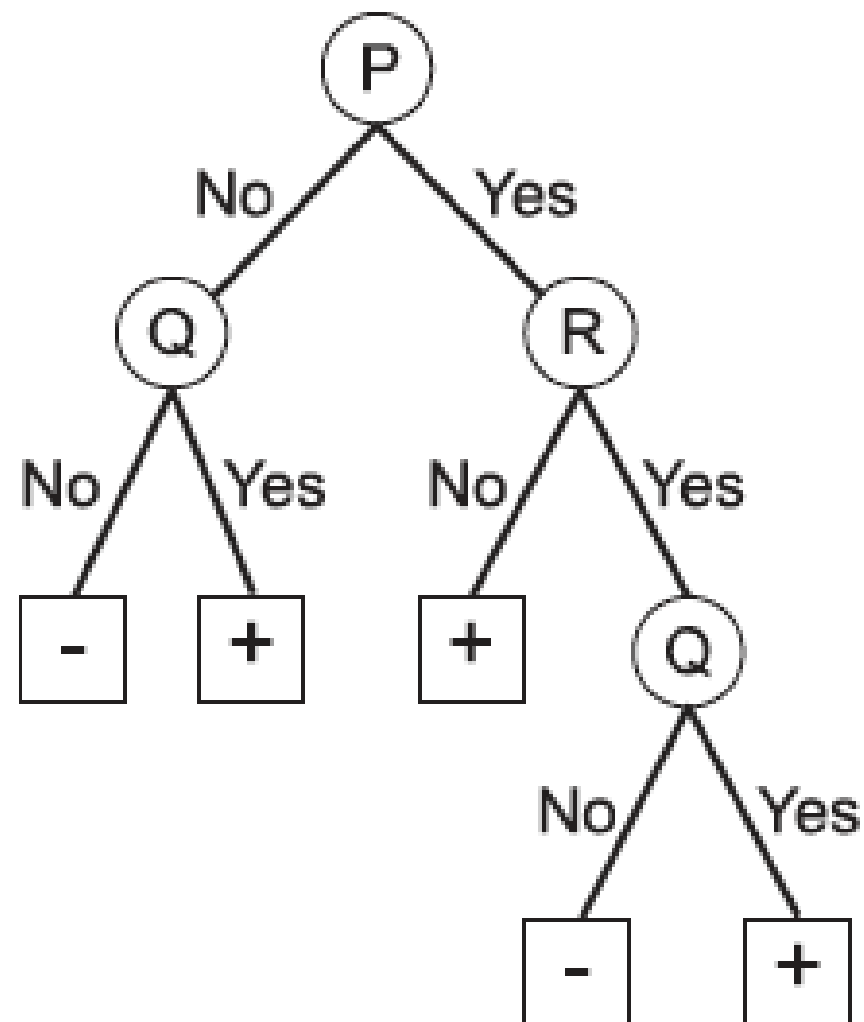


Simplification of Rules:

While the rule set generated from a decision tree can be exhaustive and mutually exclusive, some rules can be simplified.

Simplification involves reducing complexity and making rules easier to interpret.

For example, if it's observed that a rule set always predicts a positive class when a certain condition is met (e.g., $Q = \text{Yes}$), the rules can be simplified by removing unnecessary conjuncts.



Rule Set

r1: (P=No,Q=No) ==> -

r2: (P=No,Q=Yes) ==> +

r3: (P=Yes,Q=No) ==> +

r4: (P=Yes,R=Yes,Q=No) ==> -

r5: (P=Yes,R=Yes,Q=Yes) ==> +

Figure 6.4. Converting a decision tree into classification rules.

Example of Rule Simplification:



The example provided includes three rules with conditions related to attributes P, Q, and R.



It's observed that the rules always predict a positive class when Q = Yes.



As a result, the rules can be simplified to retain only the necessary conjuncts: $r2 : (Q = \text{Yes}) \rightarrow +$ and $r3: (P = \text{Yes}) \wedge (R = \text{No}) \rightarrow +$.

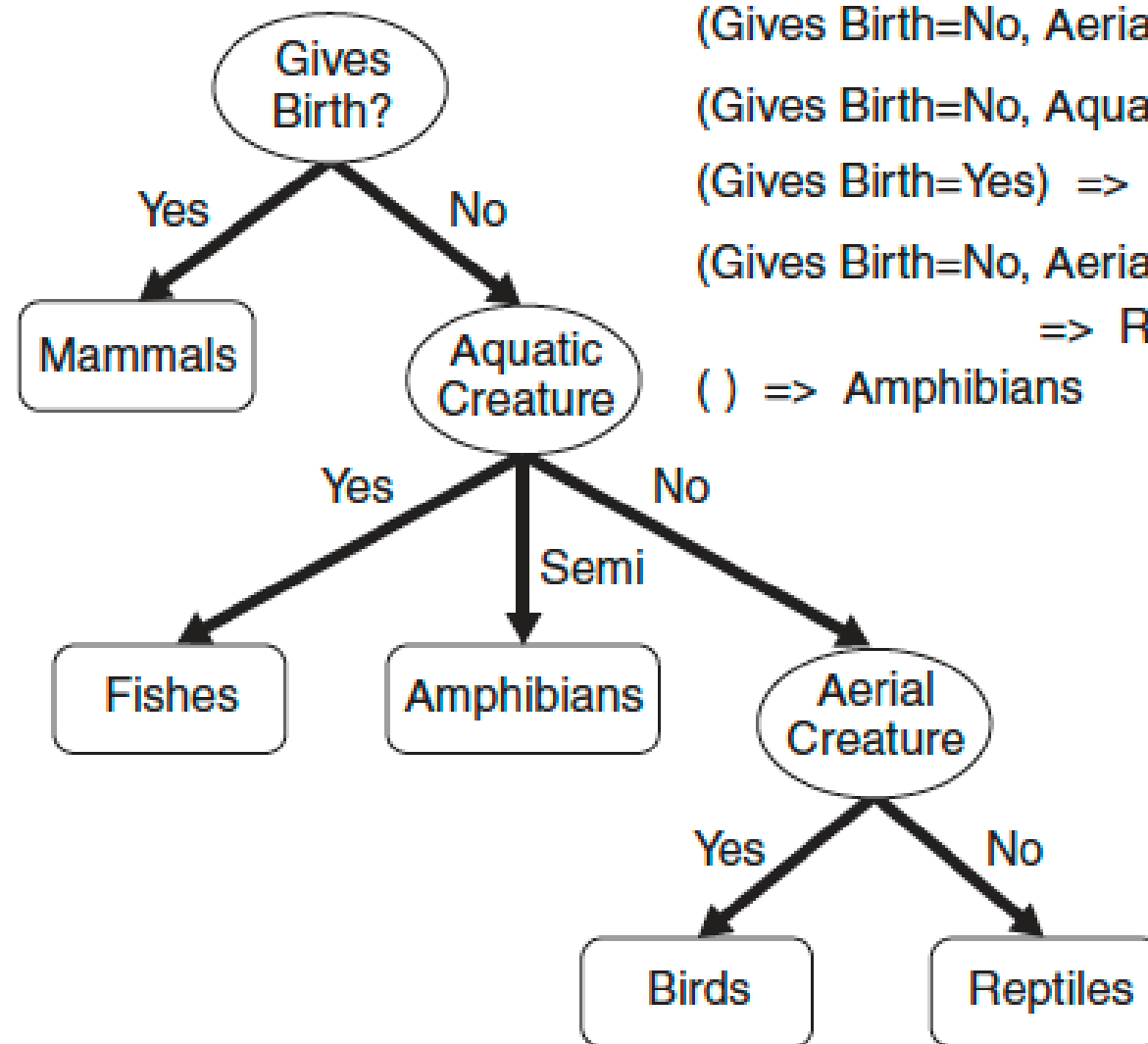


This simplification makes the rules less complex and more interpretable.



C4.5rules Algorithm:

The C4.5rules algorithm is introduced as an approach to generate rule sets from decision trees.



(Gives Birth=No, Aerial Creature=Yes) => Birds
(Gives Birth=No, Aquatic Creature=Yes) => Fishes
(Gives Birth=Yes) => Mammals
(Gives Birth=No, Aerial Creature=No, Aquatic Creature=No)
=> Reptiles
() => Amphibians

Figure 6.5. Classification rules extracted from a decision tree for the vertebrate classification problem.

Rule Generation & Ordering

Rule Generation:

- Rules are extracted for every path from the root node to a leaf node in the decision tree.
- Simplified rules (with one conjunct removed) are considered, and the rule with the lowest pessimistic error rate is retained if its error rate is lower than the original rule.
- The rule-pruning step is repeated until further improvement in the pessimistic error is not possible.

Rule Ordering:

- After generating the rule set, C4.5rules orders the rules using a class-based ordering scheme.
- Rules predicting the same class are grouped together into the same subset.
- The total description length for each subset is computed.
- Classes are arranged in increasing order of their total description length.
- The class with the smallest description length is given the highest priority, as it is expected to contain the best set of rules.
- The total description length includes encoding misclassified examples ($L_{exception}$) and the model (L_{model}), along with a tuning parameter (g) that depends on the presence of redundant attributes in the model.
- The tuning parameter value is smaller when there are more redundant attributes.

Characteristics of Rule-Based Classifiers

Similarity to Decision Trees:

- Rule-based classifiers share similarities with decision trees.
- A rule set can be as expressive as a decision tree because a decision tree can be represented by a set of mutually exclusive and exhaustive rules. Both methods create partitions in the attribute space and assign classes to these partitions.
- However, rule-based classifiers can trigger multiple rules for a given instance, enabling them to learn more complex models than decision trees.

Handling Attributes:

- Rule-based classifiers, like decision trees, can handle both categorical and continuous attributes.
- They are also suitable for multiclass classification scenarios.
- Rule-based models are often preferred for their interpretability, while still providing competitive performance compared to decision tree classifiers.

Handling Redundant Attributes:

- Rule-based classifiers can effectively handle redundant attributes that are highly correlated.
- When an attribute is used as a conjunct in a rule antecedent, remaining redundant attributes may show little to no information gain and can be ignored.

Characteristics of Rule-Based Classifiers

Avoiding Irrelevant Attributes:

- Rule-based classifiers are adept at avoiding irrelevant attributes, as they typically show poor information gain.
- However, in complex scenarios with interacting attributes that collectively distinguish between classes, there is a chance that an irrelevant attribute may be favored over a relevant one by random chance.
- This can lead to poorer performance when interacting attributes are present, especially if there are many irrelevant attributes.

Handling Imbalanced Datasets:

- Some rule-based classifiers, like RIPPER, use a class-based ordering strategy that prioritizes rare classes.
- This approach is effective in handling training datasets with imbalanced class distributions.

Handling Missing Values:

- Rule-based classifiers may struggle with missing values in the test set.
 - The position of rules in a rule set follows a specific ordering strategy.
 - If a test instance is covered by multiple rules, they may assign different class labels based on their positions in the rule set.
 - Consequently, dealing with missing attribute values in a test instance can be challenging, as ignoring a rule due to a missing attribute could lead to incorrect class assignments.
-

Naive Bayes Classification Algorithm

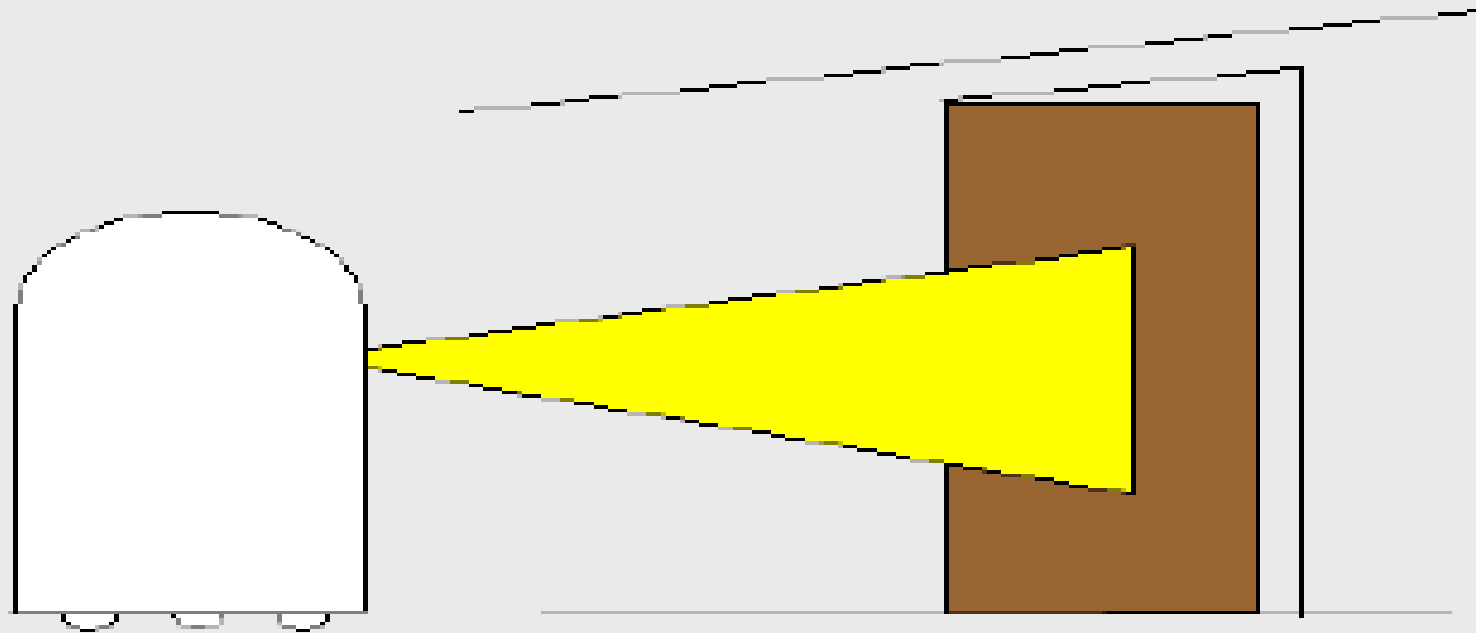
An abstract geometric composition featuring a red sphere with a gradient, resting on a dark purple rectangular prism. Behind the prism is a larger, lighter purple hexagonal prism. The background is a solid dark purple. Three horizontal white lines are present: one at the top, one below the text, and one at the bottom.

Probability Basics

- Prior, conditional and joint probability
 - Prior probability: $P(X)$
 - Conditional probability: $P(X_1 | X_2), P(X_2 | X_1)$
 - Joint probability: $\mathbf{X} = (X_1, X_2), P(\mathbf{X}) = P(X_1, X_2)$
 - Relationship: $P(X_1, X_2) = P(X_2 | X_1)P(X_1) = P(X_1 | X_2)P(X_2)$
 - Independence: $P(X_2 | X_1) = P(X_2), P(X_1 | X_2) = P(X_1), P(X_1, X_2) = P(X_1)P(X_2)$
-

Simple Example of State Estimation

- Suppose a robot obtains measurement z
- What is $P(\text{doorOpen} | z)$?



Causal vs. Diagnostic Reasoning

- $P(open|z)$ is diagnostic.
 - $P(z|open)$ is causal.
 - Often causal knowledge is easier to obtain.
 - Bayes rule allows us to use causal
- knowledge:

$$P(open | z) = \frac{P(z | open)P(open)}{P(z)}$$

count frequencies!

Example

- $P(z|open) = 0.6$ $P(z|\neg open) = 0.3$
- $P(open) = P(\neg open) = 0.5$

$$P(open | z) = \frac{P(z | open)P(open)}{P(z | open)p(open) + P(z | \neg open)p(\neg open)}$$

$$P(open | z) = \frac{0.6 \cdot 0.5}{0.6 \cdot 0.5 + 0.3 \cdot 0.5} = \frac{2}{3} = 0.67$$

- z raises the probability that the door is open.

Naive Bayes Classification Algorithm



Naïve Bayes is a data-classification algorithm that is based on probabilistic analysis.



The term probability is often associated with the term event.

“The probability of Event X is so and so.”



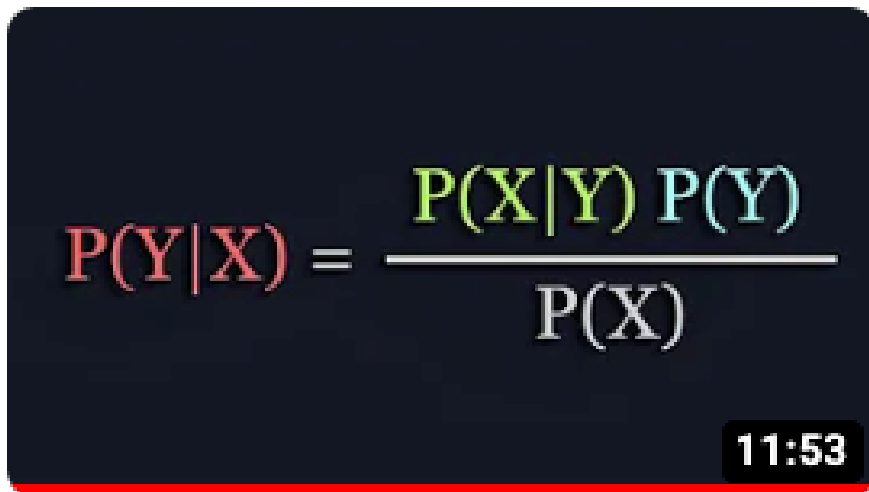
The probability of Event X is a numerical value.



You can calculate this numerical value by dividing the number of times Event X can occur by the number of events that are possible in the same circumstances.

Video explaining Bayes theorem and Naïve Bayesian classifier

<https://youtu.be/IFJbZ6LVxN8?si=za-ZWX0io3YHQhdi>



A video thumbnail with a dark blue background. It displays the Bayes theorem formula: $P(Y|X) = \frac{P(X|Y) P(Y)}{P(X)}$. The terms are color-coded: $P(Y|X)$ is red, $P(X|Y)$ is green, $P(Y)$ is teal, and $P(X)$ is white. In the bottom right corner, there is a black box with the white text "11:53".

The Math Behind Bayesian Classifiers Clearly Explained!

Normalized Nerd • 60K views

In this video, I've explained the math behind Bayes classifiers with an example. I've also covered the Naive Bayes model. #machinelearni...

Defining a Classification problem

Suppose we have feature X :

$$X = (x_1, x_2, \dots, x_n)$$

and we need correct label that is represented by Y

where X & Y random variables.

Then,

$$P(Y=y \mid X = x_1, x_2, \dots, x_n),$$

A conditional probability which finds the probability of Y is equal to y , given that $X = x_1, x_2, \dots, x_n$

So, we need to find the value of y for which this conditional probability becomes maximum.

Bayes theorem

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

$$Posterior = \frac{Likelihood * Prior}{Evidence}$$

$P(Y)$ → *Prior* : probability before having any evidences.

$P(X|Y)$. → *Likelihood* : Probability of X happening for Y.

$P(X)$ → *Evidence* : Probability event happened

$P(Y|X)$ → *Posterior*: probability after having some evidences

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

$P(\text{Home Owner}=\text{Yes}|\text{No}) = 3/7$
 $P(\text{Home Owner}=\text{No}|\text{No}) = 4/7$
 $P(\text{Home Owner}=\text{Yes}|\text{Yes}) = 0$
 $P(\text{Home Owner}=\text{No}|\text{Yes}) = 1$
 $P(\text{Marital Status}=\text{Single}|\text{No}) = 2/7$
 $P(\text{Marital Status}=\text{Divorced}|\text{No}) = 1/7$
 $P(\text{Marital Status}=\text{Married}|\text{No}) = 4/7$
 $P(\text{Marital Status}=\text{Single}|\text{Yes}) = 2/3$
 $P(\text{Marital Status}=\text{Divorced}|\text{Yes}) = 1/3$
 $P(\text{Marital Status}=\text{Married}|\text{Yes}) = 0$

For Annual Income:

If class=No: sample mean=110
sample variance=2975

If class=Yes: sample mean=90
sample variance=25

EXAMPLE Naive Bayes Classifier

- Consider the data set shown in Figure 6.9(a), where the target class is Defaulted Borrower, which can take two values Yes and No.
 - We can compute the class-conditional probability for each categorical attribute and the sample mean and variance for the continuous attribute, as summarized in Figure 6.9(b).
 - We are interested in predicting the class label of a test instance $x = (\text{Home Owner}=\text{No}, \text{Marital Status} = \text{Married}, \text{Annual Income} = \$120\text{K})$.
 - To do this, we first compute the prior probabilities by counting the number of training instances belonging to every class.
 - We thus obtain $P(\text{Yes}) = 0.3$ and $P(\text{No}) = 0.7$
-

EXAMPLE Naive Bayes Classifier

Next, we can compute the class-conditional probability as follows:

$$P(x|\text{No}) =$$

$$\begin{aligned} & P(\text{Home Owner} = \text{No}|\text{No}) \times P(\text{Status} = \text{Married}|\text{No}) \times P(\text{Annual Income} = \$120\text{K}|\text{No}) \\ &= 4/7 \times 4/7 \times 0.0072 = 0.0024. \end{aligned}$$

$$P(x|\text{Yes}) =$$

$$\begin{aligned} & P(\text{Home Owner} = \text{No}|\text{Yes}) \times P(\text{Status} = \text{Married}|\text{Yes}) \times P(\text{Annual Income} = \$120\text{K}|\text{Yes}) \\ &= 1 \times 0 \times 1.2 \times 10^{-9} = 0. \end{aligned}$$

EXAMPLE Naive Bayes Classifier

- Notice that the class-conditional probability for class Yes has become 0 because there are no instances belonging to class Yes with Status = Married in the training set.
- Using these class-conditional probabilities, we can estimate the posterior probabilities as

$$P(\text{No}|\mathbf{x}) = 0.7 \times 0.0024$$

$$P(\mathbf{x}) = 0.0016\alpha$$

$$P(\text{Yes}|\mathbf{x}) = 0.3 \times 0$$

$$P(\mathbf{x}) = 0.$$

where $\alpha = 1/P(X)$ is a normalizing constant. Since $P(\text{No}|\mathbf{x}) > P(\text{Yes}|\mathbf{x})$, the instance is classified as No.

Characteristics of Naïve Bayes Classifiers

Probabilistic classification models:

- Able to quantify the uncertainty in predictions by providing posterior probability estimates.
- They are also generative classification models as they treat the target class as the causative factor for generating the data instances.
- Hence, apart from computing posterior probabilities, also attempt to capture the underlying mechanism behind the generation of data instances belonging to every class.

Naive Bayes assumption:

- They can easily compute class conditional probabilities even in high-dimensional settings, provided that the attributes are conditionally independent of each other given the class labels.
- This property makes Naïve Bayes classifier a simple and effective classification technique that is commonly used in diverse application problems, such as text classification.

Robust to isolated noise points

- because such points are not able to significantly impact the conditional probability estimates, as they are often averaged out during training.

Characteristics of Naïve Bayes Classifiers

Handling missing values:

- in the training set by ignoring the missing values of every attribute while computing its conditional probability estimates.
- By using only, the non-missing attribute values while computing posterior probabilities.

Robust to irrelevant attributes

- If X_i is an irrelevant attribute, then $P(X_i|Y)$ becomes almost uniformly distributed for every class y .
- The class-conditional probabilities for every class thus receive similar contributions of $P(X_i|Y)$, resulting in negligible impact on the posterior probability estimates.

Underperformed for Correlated attributes

- Correlated attributes can degrade the performance of Naive Bayes classifiers because the Naïve Bayes assumption of conditional independence no longer holds for such attributes.



Classification

Neural networks

Neural networks



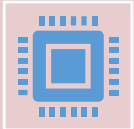
A more complex algorithm, biologically inspired by the structure of the human brain.



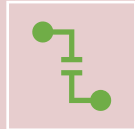
A neural network provides a very simple model in comparison to the human brain, but it works well enough for our purposes.



Widely used for data classification, neural networks process past and current data to estimate future values — discovering any complex correlations hidden in the data — in a way analogous to that employed by the human brain.



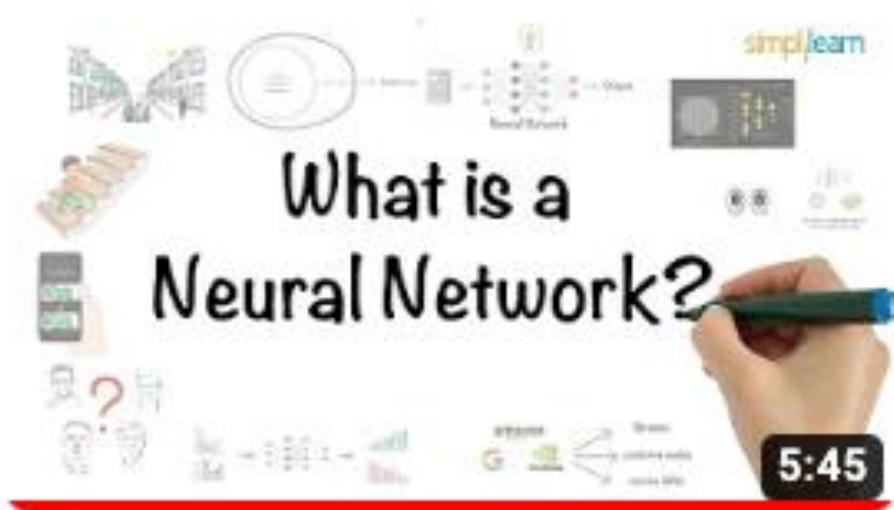
Neural networks can be used to make predictions on time series data such as weather data.



A neural network can be designed to detect pattern in input data and produce an output free of noise.

Video for brief explanation of Neural Network

<https://youtu.be/bfmFfD2RIcg?si=IbGjoKEChMpLeaEU>



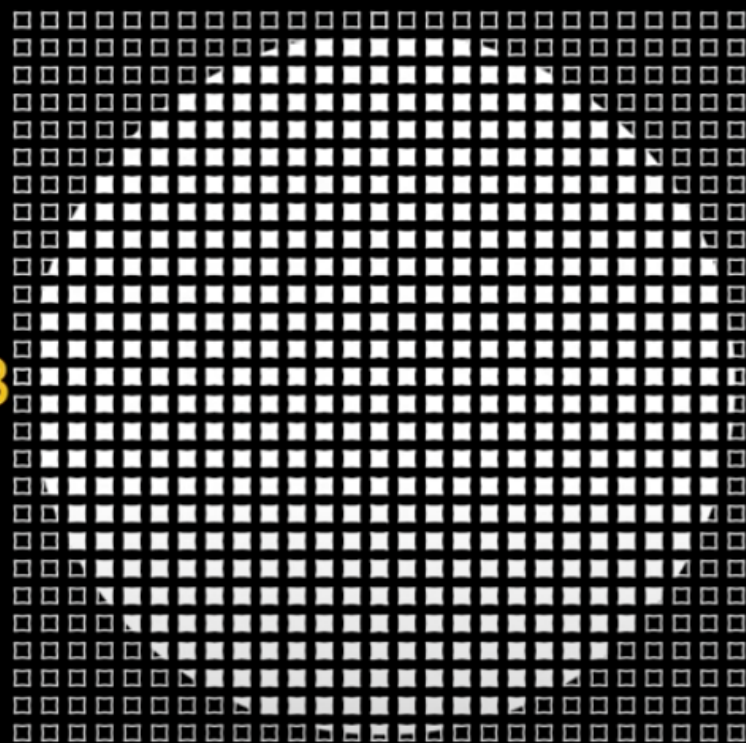
Neural Network In 5 Minutes | What Is A Neural Network? | Ho...

Simplilearn ✓ 1.1M views

🔥 Become An AI & ML Expert Today:

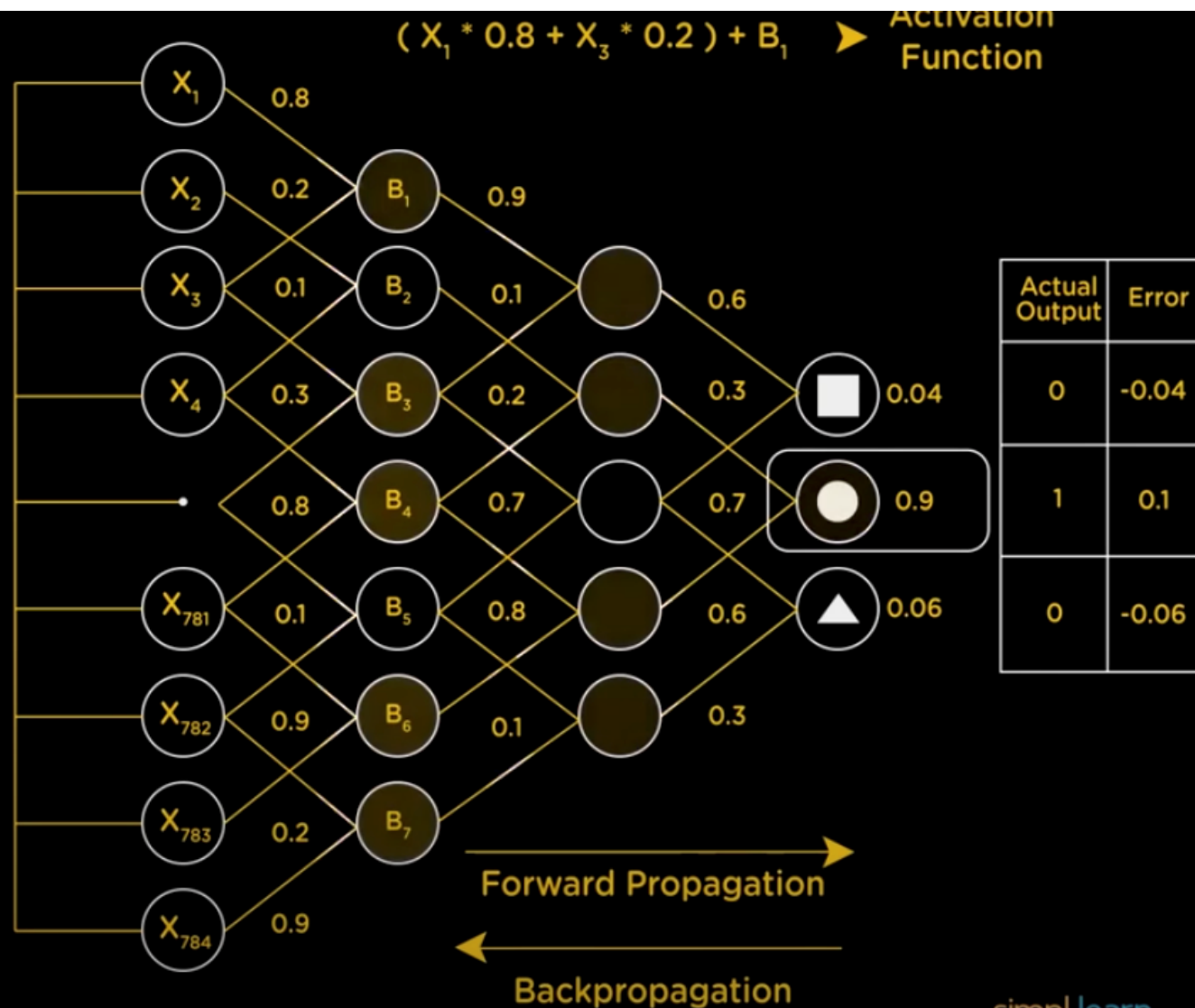
https://taplink.cc/simplilearn_ai_ml This video on "What is a Neural..."

28

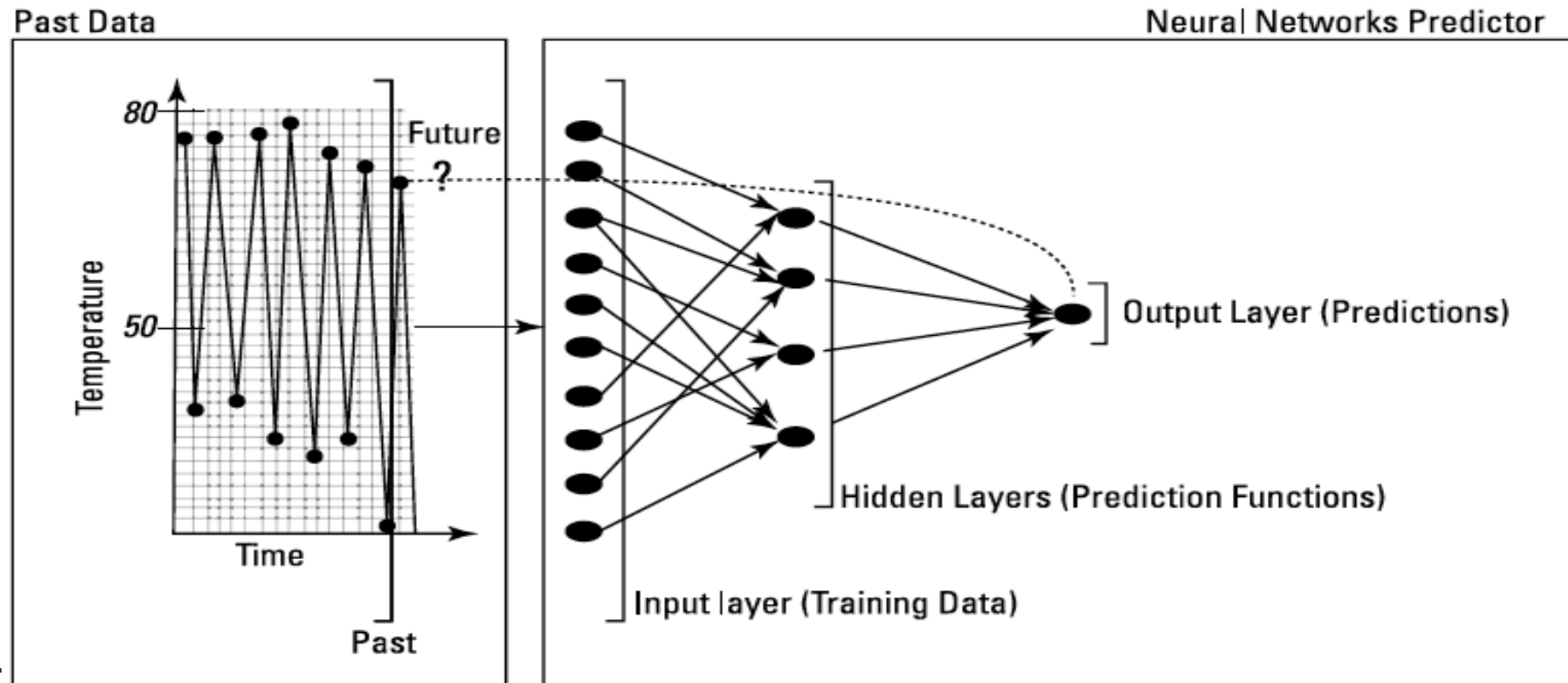


28

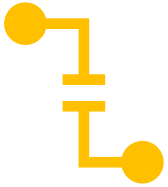
28 x 28 = 784 Pixels



Structure of a neural-network algorithm



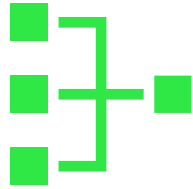
Structure of a neural-network algorithm



Input layer:

feeds past data values into the next (hidden) layer.

The black circles represent nodes of the neural network.

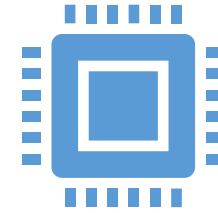


Hidden layer:

Encapsulates several complex functions that create predictors; functions are hidden from the user.

A set of nodes (black circles) at the hidden layer represents mathematical functions that modify the input data

these functions are called *Neurons*.



Output layer:

collects the predictions made in the hidden layer and produces the result:

- the model's prediction.

The Training process

Hidden Layer:

- The hidden layer in a neural network is crucial because it contains neurons that perform calculations to produce the output.
- Neurons take input values, each associated with *a weight and a bias*.
- Their output is a function of the *weighted sum of inputs plus the bias*.

Activation Functions:

- Neural networks typically use mathematical functions to activate neurons. It uses the *sigmoid functions*, which are often referred to as *logistic functions*.
- The sigmoid function is used to map inputs to outputs and is represented by the formula:

$$f(\text{input}) = \frac{1}{1 + e^{\text{output}}}$$

- Sigmoid functions are preferred because of their characteristics like being continuous, smooth, and bounded, and having easily calculable derivatives, which are vital for training.

Weights and Bias:

- Each neuron has associated weights and biases.
 - These values are adjusted during the training process.
 - Weights determine the importance of different inputs, and biases introduce a level of flexibility to the neuron's operation.
-

The Training process

Neural networks are trained using either supervised or unsupervised methods

Supervised Training:

- Weights are derived by providing the network with sample inputs and their corresponding outputs until the weights are tuned to achieve a close match between inputs and outputs.
- This process is used for tasks like classification and regression.

Unsupervised Training:

- The network is only given inputs, and the algorithm generates corresponding outputs.
 - When the algorithm can produce new outputs for new, similar inputs that are in line with previous outputs, it indicates that the weights have been tuned.
 - Unsupervised training is often used for tasks like clustering or feature learning.
-

The Training process

Advantages

- Neural networks have the advantage of being able to work well even when the data contains a significant amount of noise.
- This means they can discover relationships in the data despite some inaccuracies or variability.
- This ability to generalize is a key strength of neural networks.

Disadvantage:

- The accuracy of neural network predictions may be valid only within the time period during which the training data was gathered.
 - This is known as the issue of data stationarity.
 - Neural networks might not perform as well when applied to data outside the time they were trained on because the underlying patterns in the data may change.
-
