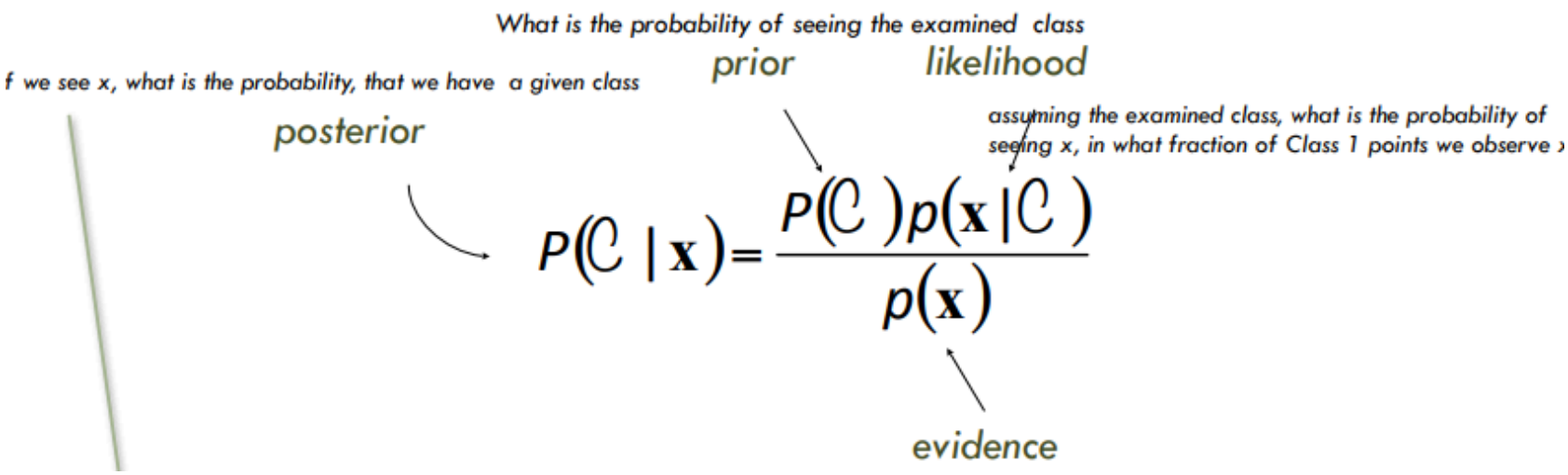
Chapter 1:
Bayesian Decision Theory

Context: Bayesian Decision Theory is commonly used in classification tasks within supervised learning to predict the class of an input based on observed data, rule generation, and decision-making.
Algorithm: Naive Bayes
Goal: Choose the class with the highest posterior probability, minimizing the expected loss.

Bayes Rule



Evidence (Marginal Probability): Without respect to any class, how likely is it to see x?

Decision Making

Classification type	Rule
Binary	Choose class 1 $if\ P(C = 1\|x) > P(C = 0\|x)$.

Multiclass	Choose the class i with the highest posterior probability.

Losses and Risks

Definition: A loss function quantifies the cost associated with making incorrect decisions.
Purpose: Helps in making decisions that minimize the expected cost or risk.

$$R(\alpha_i|x) = \sum_{k=1}^{K} \lambda_{ik} P(C_k|x)$$

Reject Option

Definition: Introduces the option to reject making a decision when the risk of misclassification is too high.
Mechanism:
If the maximum posterior probability is below a certain threshold, reject making a classification.
Helps in avoiding high-cost errors when the classifier is uncertain.

Utility Theory:

An alternative to the loss function focuses on maximizing the expected utility and benefits rather than minimizing costs.
Utility Function: Represents the benefit of making a certain decision under specific circumstances.

Discriminant Functions

A **discriminant function** is a function that provides a score for each class, given an input data point. The scores are used to make classification decisions.

For a given data point x and classes C_1, C_2,..., C_k the discriminant function for class C_i is denoted as $g_i(x)$. The data point x is assigned to the class with the highest discriminant function value:

$$\text{Assign } x \text{ to class } C_i \; if \; g_i(x) > g_j(x) \; for \; all \; j \neq i$$

Relationship to Bayesian Decision Theory

In Bayesian Decision Theory, the discriminant function is typically derived from the posterior probabilities $P(C_i|x)$ of the classes given the data point x. The goal is to maximize the posterior probability.

For classification, the denominator $P(x)$ is the same for all classes and can be ignored. The discriminant function is then based on the numerator which is the product of the likelihood and the prior:

$g_i(x) = P(x|C_i)P(C_i)$

To simplify computations, especially when dealing with continuous distributions, the logarithm of the discriminant function is often used. The logarithmic discriminant function is:

$g_i(x) = logP(x|C_i) + logP(C_i)$

	Descritpiton	Discriminant Function	Decision Rule
Dichotomizer (k=2)	Binary classifier for two classes	$g(x) = g_1(x) - g_2(x)$ $g(x) = log\frac{P(C_1\|x)}{P(C_2\|x)}$	Assign x to $C_1 \; if \; g(x) > 0 \; , \; C_2 \; otherwise$
Polychotomizer (k>2)	Multi-class classifier for more than two classes	$g_i(x) = logP(x\|C_i) + logP(C_i)$	Assign x to class $C_i \; if \; g_i(x) > g_j(x) \; for \; all \; j \neq i$

Chapter 2:
Parametric Models

Parametric Models

Parametric models are statistical models that summarize data with a fixed number of parameters.
They assume a specific form for the underlying function or distribution (linear, polynomial, normal distribution …)

	Algorithm	Description
Regression	Linear Regression	Assumes a linear relationship between input variables and the output.
	Polynomial Regression	
Classification	Logisitc Regression	Assumes a logistic relationship for binary classification.
	Linear Discriminant Analysis LDA	Assumes that different classes generate data based on different Gaussian distributions with the same covariance matrix.
	Quadratic Discriminant QDA	Similar to LDA but assumes each class has its own covariance matrix.
	Naive Bayes	Assumes independence between the features
	Mixtures of Gaussians	Assumes the data is generated from a mixture of Gaussian distributions.

- **Hidden Markov Models**
  Assumes the data is generated from a of Gaussian distributions.

Non-parametric models

Non-parametric models are statistical models that do not assume a specific form for the function or distribution generating the data. Instead, they allow the model complexity to grow with the amount of data.

	Algorithm	Description
Regression	k-Nearest Neighbors (k-NN) Regression	Assumes a linear relationship between input variables and the output.
	Kernel Regression	
	Decision Trees	
	Random Forests	
Classification	k-Nearest Neighbors (k-NN) Classification	Assumes a logistic relationship for binary classification.
	Kernel Density Estimation (KDE)	Assumes that different classes generate data based on different Gaussian distributions with the same covariance matrix.
	Decision Trees	Similar to LDA but assumes each class has its own covariance matrix.
	Random Forests	Assumes independence between the features
	Support Vector Machines (SVM) with Non-Linear Kernels	

Parametric Vs. Non-Parametric models

Parametric	**Non-parametric**
Strong assumptions about the data's distribution or function	Few or no assumptions about the data's distribution
Fixed number of parameters	No fixed number of parameters
Requires less data to estimate parameters accurately	Requires more data for accurate modeling
Generally computationally efficient	Can be computationally intensive
Less adaptable Less Flexible	Adaptable Highly flexible

Parameters Vs. Hyperparameters

Whether a model has a fixed or variable number of parameters determines whether it may be referred to as “parametric” or “nonparametric”.

	Parameters	**Hyperparameters**
Definition	Internal values learned from training data	External configurations set before training
Optimization	Adjusted during the training process using optimization algorithms (E.g. gradient descent)	Set before training, tuned through hyperparamter tuning methods (E.g.)
Influence	Directly affect the model’s predictions	Control the training process and model structure.

Examples	- Weights in neural networks - Coefficients in linear or logistic regression - Support vectors in SVM	- Learning rate in neural netwroks - C regularization hyperparameter in SVM - K in k-nearest neighbors

Parameter estimation involves using statistical techniques to estimate the values of parameters in a model. Various methods are used to estimate these parameters depending on the type of model and data.

Parameter Estimation techniques

Maximum Likelihood Estimation

Maximum likelihood estimation is a method that determines values for the parameters θ of a model by maximizing the likelihood function.

Likelihood function $L(\theta)$ represents the probability of the observed data given the parameters θ

$X = \{x_1, x_2, ..., x_n\}$

$$L(\theta) = p(X|\theta) = \prod_{i=1}^{n} p(x_i|\theta)$$

Log likelihood: To simplify calculations, we take a logarithm to change the product to summation. This does not change the optimal estimator since the logarithm is a monotonic function.

$$LL(X|\theta) = \sum_{t} log\, p(x_t|\theta)$$

Parametric Density Estimation techniques

	Assumption	Parameters	Estimation	Density Fucntion

Gaussian(Normal) Distribution	The data follows a Gaussian (normal) distribution.	- Mean μ - Standrad deviation σ	Parameters are estimated using MLE. - $\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$ - $\widehat{\sigma^2} = \frac{1}{N}\sum_{i=1}^{N} (x - \hat{\mu})^2$	$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$
Multinomial Distribution	Suitable for categorical data where features represent counts or frequencies.	- Probabilities of each category (more than 2 states)	Parameters are estimated using MLE. - $\hat{p}_i = \frac{1}{N}\sum_t x_i^t$	$P(P\|x_1, x_2,..., x_n) = \prod_i p_i^{x_i}$
Bernoulli Distribution	Models binary outcomes (success/failure)	- Probabilities of 2 states (success or failure)	Parameters are estimated using MLE. - $\hat{p} = \frac{1}{N}\sum_t x^t$	$P(x) = p^x(1 - p)^{1-x}$

Chapter 3: Probabilistic Models

Graphical Models

Probabilistic models take into consideration the uncertainty inherent in real-world data. These models make predictions based on probability distributions, rather than absolute values, allowing for a more nuanced and accurate understanding of complex systems. One common approach is **Bayesian inference**, where prior knowledge is combined with observed data to

make predictions. Another approach is **maximum likelihood estimation**, which seeks to find the model that best fits observational data.

A graphical model, probabilistic graphical model (PGM), or structured probabilistic model is a **probabilistic model** for which a graph expresses the **conditional dependence** structure between random variables. They are commonly used in probability theory, statistics—particularly **Bayesian statistics**—and machine learning.

Nodes: Hypotheses
Labels:
- On nodes: Probability $P(X)$
- On Edges: Conditional probability $P(X|Y)$

Arcs/Edges: Represent direct influences between hypotheses
Directed Acyclic Graph (DAG): The structure of graphical models.

Conditional Independence

Independence:

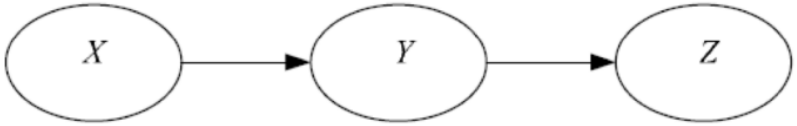
X and Y are independent if P(X, Y) = P(X).P(Y)
P(X, Y) : Joint probability

Conditional Independence:

Two variables are conditionally independent given a third if knowing the third variable makes the two independent.
X and Y are conditionally independent given Z if:
P(X,Y|Z) = P(X|Z).P(Y|Z)

Canonical cases:

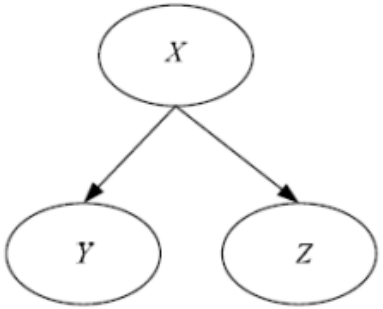
HEAD-TO-TAIL



- X and Z are conditionally independent given Y
- If Y is unknown, it block the path between X and Z

$P(X,Y,Z)=P(X)P(Y|X)P(Z|Y)$

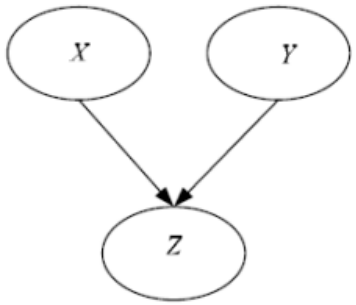
TAIL-TO-TAIL



- Having knowledge about Y can propagate to X then to Z.
- Y and Z become independent known X.
- If X is unknown, it blocks the path between Y and Z.

$P(X,Y,Z) = P(X).P(Y|X).P(Z|X)$

HEAD-TO-HEAD

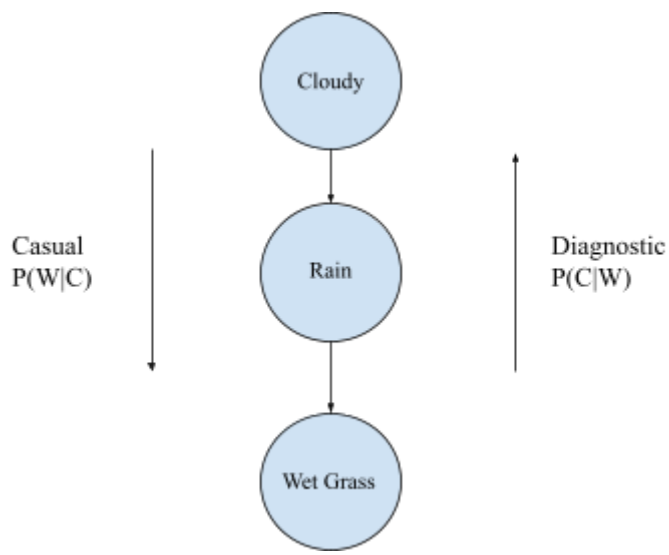


- Knowledge about x can be propagated to Z then to Y.
- X and Y are dependent through Z when Z is observed
- If Z is unknown, it blocks the path between X and Y

P(X,Y,Z) = P(X).P(Y).P(Z|X,Y)

Casual vs. Diagnostic Inference

Type	Definition	Formula
Causal	Propagate the info forward through the network. From cause to effect	$P(W\|C) = P(W\|R).P(R\|C) + P(W\|\overline{R}).P(\overline{R}\|C)$
Diagnostic	Propagate the evidence back through the network using the Bayes Rule. From effect to cause	$P(C\|W) = \frac{P(W\|R)\times P(R)}{P(W)}$



d-Separation:

d-Separation is a concept in graphical models used to determine whether two sets of nodes are conditionally independent given a third set of nodes. It is a key tool for understanding the structure and dependencies within a Bayesian network.
Two nodes A and B are d-separated by a set of nodes C if every path between A and B is blocked by C.

Junction Trees

Junction Trees are a data structure used in graphical models to perform efficient inference. They provide a way to convert a complex network into a simpler, tree-like structure that allows for efficient computation of marginal and conditional probabilities.

Key Concepts

1. **Clique**: A subset of nodes in a graph where every pair of nodes is connected by an edge.
2. **Separator**: A set of nodes that separates two cliques in a junction tree.
3. **Tree Structure**: Junction trees are acyclic and form a tree structure where each node is a clique of the original graph.

Applications of Junction Trees

- **Inference in Bayesian Networks**: Efficiently compute marginal and conditional probabilities.
- **Probabilistic Reasoning**: Simplify complex probabilistic models into manageable structures.
- **Hidden Markov Models (HMMs)**: Used for exact inference in models with hidden variables.

Chapter 4: Probabilistic Models

Hidden Markov Models

A Hidden Markov Model (HMM) is a **statistical model** that represents systems with **underlying hidden states** using observable sequences. It is often used in situations where the underlying system or process that generates the observations is unknown or hidden, hence it has the name "Hidden Markov Model."
An HMM consists of two types of variables: hidden states and observations.

- **The hidden states** are the underlying variables that generate the observed data, but they are not directly observable.
- **The observations** are the variables that are measured and observed.

The Hidden Markov Model (HMM) is the relationship between the hidden states and the observations using two sets of probabilities:

- **The transition probabilities** describe the probability of transitioning from one hidden state to another.
- **The emission probabilities** describe the probability of observing an output given a hidden state.

Markov Property:

- The transition probabilities depend only on the current state, not on the sequence of events that preceded it.

Key Components of HMM:

Component	**Description**
States	Not directly observable (hidden) $S = \{S_1, S_2, S_3..., S_N\}$

Observations	Observations are visible and are linked to the hidden states. Each state produces an observation according to a probability distribution. $O = \{O_1, O_2, O_3..., O_M\}$
Transisition probabilities Matrix (N by N)	Probability of transitioning from one state to another. $A = \left[a_{ij}\right]$
Emission probabilities Matrix (N by M)	Probability of observing a symbol given a state. $B = \left[b_j(m)\right]$
Initial probabilities (N by 1)	Initial probability distribution vector over states. $\Pi = \left[\pi_i\right]$

Hidden Markov Model Algorithm

1. Define the state space S of all possible hidden states and observation space O of all possible observations.
2. Define the initial state distribution Π
   This is the probability distribution over the initial state.
3. Define the state transition probabilities Matrix A
4. Define the observation likelihoods B
5. Train the model
   The parameters of the state transition probabilities and the observation likelihoods are estimated using the following algorithms by iteratively updating the model params until convergence and maximizing the likelihood of the observed data.:
   - The Baum-Welch algorithm
   - The Forward-backward algorithm

6. Decode the most likely sequence of hidden states given the model and an observation sequence.
   Given the observed data, we use the Viterbi algorithm to compute the most likely sequence hidden states.
7. Evaluate the model
   The performance of the HMM can be evaluated using various metrics, such as accuracy, precision, recall or F1-score.

Fundamental Problems of HMM:

1. Evaluation Problem

How likely is it that a given sequence of observations will happen?

Objective: Compute the probability of an observation sequence given the model parameters.

Forward Algorithm:

1. **Initialization**
   Start with the initial state probabilities and the probability of the first observation given each state.
2. **Recursion**
   Iteratively compute the probabilities for the sequence up to the next time step for each state, using the probabilities from the **previous** time step.
3. **Termination**
   Sum the probabilities of being in any state at the final time step, which gives the total probability of the observation sequence.

Backward Algorithm

1. **Initialization**
   Set the probability of ending in each state to 1.

2. **Recursion**
   Iteratively compute the probabilities for the sequence from the next time step to the end for each state, using the probabilities from the following time step.
3. **Termination**
   Combine the results with the forward algorithm to compute the total probability of the observation sequence.

2. Decoding Problem

What is the most probable sequence of hidden states given an observation sequence?

Objective: Identify the sequence of states that most likely generated the observed sequence.

Viterbi Algorithm

1. **Initialization**
   Start with the initial state probabilities and the probability of the first observation given each state.
2. **Recursion**
   Iteratively compute the highest probability path to each state at the next time step, using the highest probability paths from the previous time step.
3. **Termination**
   Identify the highest probability path ending in any state at the final time step
4. **Path Backtracking**
   Trace back through the computed paths to determine the most likely sequence of states

3. Learning Problem

How can we learn the HMM parameters (transition probabilities, emission probabilities, initial state probabilities) from observed data?

Objective: Estimate the model parameters that maximize the likelihood of the observed data.

Baum-Welch Algorithm

1. **Expectation Step**
   - Compute the forward probabilities (probability of being in each state up to each time step)
   - Compute the backward probabilities(probability of being in each state from each time step to the end)
   - Use these probabilities to estimate the expected number of transitions and emissions for each state.
2. **Maximization Step**
   - Update the transition probabilities based on the expected number of transitions between states
   - Update the emission probabilities based on the expected number of emissions from each state
3. **Iteration**

   Repeat steps 1 and 2 until the parameters converge to stable values.

Chapter 5:
Lazy Learning

Lazy learning is a type of machine learning that doesn't process training data until it needs to make a prediction. Instead of building models during training, lazy learning algorithms wait until they encounter a new query. This method stores and compares training examples when making predictions. It's also called **instance-based** or **memory-based** learning.

Non – parametric density estimation methods:

1. Histogram Estimator

A histogram estimator divides the range of the data into bins and counts the number of data points in each bin. The density is estimated as the count in each bin divided by the total number of data points and the bin width.

How?

1. **Bin Selection**: Divide the data range into equal-width bins.
2. **Counting**: Count the number of data points in each bin.
3. **Density Calculation**: Estimate the density in each bin as: $\hat{f}(x) = \frac{Number\ of\ points\ in\ bin}{N\ (Total\ number\ of\ points) \times h\ (Bin\ width)}$

2. Naive Estimator

A naive density estimator uses a fixed-width window (or interval) centered around the point of interest to count the number of data points within that window.

How?

1. **Bandwidth h selection**: determines the size of the window
   - Large h will produce a smoother estimate.
   - Small h will produce a more detailed but potentially noisy estimate.
2. **Counting:** Count the number of data points within the window
3. **Density Calculation:** Estimate the density at each point as: $\hat{f}(x) = \frac{\left[number\ of\ x_i\ in\ (x-h, x+h)\right]}{2Nh}$

3. Kernel Density Estimator

Kernel estimator is used to smoothen the probability distribution function (pdf) and cumulative distribution function (CDF) graphics by averaging the contributions of each data point using a kernel function.
How?

1. **Kernel Function:** Choose a kernel function K(x)
2. **Bandwidth Selection:** Choose a bandwidth h that controls the smoothness.
3. **Density Calculation:** Estimate the density at each point x as: $\hat{f}(x) = \frac{1}{Nh}\sum_{i=1}^{N} K(\frac{x-x_i}{h})$

4. KNN Density Estimator

Unlike the previous methods of fixing the bin width h, in this estimation we fix the nearest neighbors k.

The k-NN density estimator estimates the density at a point x by considering the distance to the kth nearest neighbor. The volume around x containing its k nearest neighbors is used to compute the density.
How?

1. **K selection:** Choose the number of neighbors k.
2. **Distance Calculation:** For a given point x, find the distance d_k to its kth nearest neighbor.
3. **Density Calculation:** Estimate the density at x as: $\hat{f}(x) = \frac{k}{2Nd_k(x)}$

Prototype-based models

Prototype methods are machine learning methods that use data prototypes. A data prototype is a data value that reflects other values in its class,e.g., the centroid in a K-means clustering problem.

1. K-means Clustering
2. Learning Vector Quantization (Optimization)

A supervised learning algorithm used for classification tasks. It combines concepts from neural networks and vector quantization. LVQ is particularly useful for applications where interpretability and simplicity are important, such as in pattern recognition and signal processing.

Parameters:

The number of prototype vectors and the learning rate α need to be chosen carefully, which may require cross validation.

Algorithm:

1. **Initialization**:
   Initialize a set of prototype vectors to represent the different classes in the feature space.
2. **Training**:
   For each training sample x with label y:
   - Find the closest prototype vector w to the sample x
   - Adjust the prototype vector w based on wether it is correctly or incorrectly classifies the sample:
     If the prototype's class w_{class} matches the label y: $w \leftarrow w + \alpha(x - w)$
     If the prototype's class w_{class} does not match the label y: $w \leftarrow w - \alpha(x - w)$
     Here α is the learning rate, which typically decreases over time.
3. **Iteration**:
   Repeat the training step for a predefined number of iterations or until convergence (when changes to the prototype vectos are minimal)

3. Gaussian Mixtures

A type of probabilistic model that assumes data is generated from a mixture of several Gaussian distributions with unknown parameters. Primarily used for clustering but can also be used for Density Estimation.

Algorithm

Expectation-Maximization (EM) algorithm is commonly used to estimate the parameters of the Gaussian distributions (means, covariances, and mixture weights).

1. **Intialization**
   Initialize the parameters of the Gaussian distributions (means, covariances, and mixture weights)
2. **Exepectation step (E-step)**
   Compute the probability that each data point belongs to each cluster
3. **Maximization step (M-step)**
   Update the parameters of the Gaussian distributions to maximize the likelihood of the data given the current probabilities.
4. **Iteration**
   Repeat the E-step and M-step until convergence

Hashing

Hashing is a technique used to map data of arbitrary size to fixed-size values (hash codes) using a hash function.

Local sensitivity hashing

Locality-Sensitive Hashing (LSH) is a special type of hashing designed to maximize the probability that similar items are mapped to the same hash bucket. It is used to approximate nearest neighbor searches in high-dimensional spaces.

Key concepts

	Description
Similarity Measure	LSH is based on a similarity measure, such as cosine similarity, Euclidean distance, or Jaccard similarity.
Hash Functions	LSH uses a family of hash functions where similar items have a higher probability of colliding (i.e., being assigned to the same bucket) than dissimilar items.
Buckets	Each hash function divides the data space into regions (buckets). Multiple hash functions are used to ensure similar items end up in the same or neighboring buckets.
Parameters	k: Number of hash functions used to form a hash key (dimension) L: Number of hash tables used to ensure robustness (buckets)

Use cases:

- **Nearest Neighbor Search:**
  Nearest Neighbor search is a technique that can be used to implement the k-Nearest Neighbors (k-NN) algorithm in a more efficient manner. By leveraging hashing, similar items tend to receive the same hash value, thereby streamlining the process of finding the nearest neighbors.
- **Clustering:**
  Clustering involves grouping similar items together. In the context of hashing, items that fall into the same hash bucket are considered similar and can be treated as belonging to the same cluster.
- **Dimensionality Reduction**
  Dimensionality reduction involves reducing the number of random variables under consideration. The hash value assigned to an item can serve as a lower-dimensional representation of that item, preserving essential similarities while reducing complexity.

Algorithm

Nearest Neighbors Algorithms

K-NN

The effect of K:

- Small K: More detailed and localized estimate
- Large K: A smoother estimate that is less affected by local fluctuations

1-NN

Condensed Nearest Neighbor (CNN)

Condensed Nearest Neighbor is a technique designed to reduce the size of the training dataset k-NN classification while maintaining the classification accuracy. It aims to find a subset of the original training data that can represent the entire dataset.

Algorithm:

1. **Initialization:** Start with an empty set *S* and add one randomly chosen instance fron the training set.
2. **Iteration:**
    - For each instance x in the training set, check if it is correctly classified by the current subset S.
    - If x is misclassified, add x to S
3. **Repeat** until no more instances are misclassified

Adaptive Nearest Neighbor (ANN)

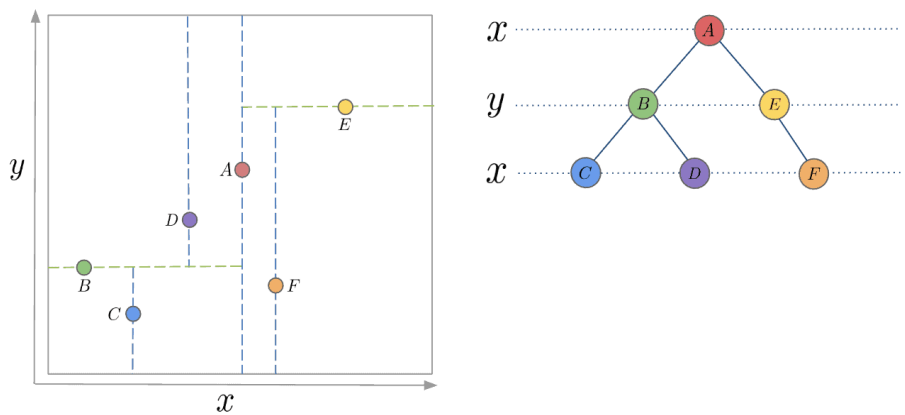
Adaptive Nearest Neighbor methods adjust the number of neighbors k or the distance metric based on the local density and structure of the data. This adaptability allows the algorithm to perform better in heterogeneous or complex data spaces.

Voronoi Diagrams

A **Voronoi diagram** is a fundamental geometric data structure that partitions a space into regions based on the distance to a specific set of points. Each region, called a **Voronoi cell**, contains all the points closer to one particular point than to any other point. In the context of nearest neighbor search, Voronoi diagrams provide a way to visualize and understand the structure of the space regarding proximity to a set of points.

K-d Trees

A K-D Tree is a binary tree in which each node represents a k-dimensional point. Every non-leaf node in the tree acts as a hyperplane, dividing the space into two partitions. This hyperplane is perpendicular to the chosen axis, which is associated with one of the K dimensions.



Chapter 6:
Decision Trees

Decision Trees (DTs) are a **non-parametric supervised learning** method used for **classification** and **regression**. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Basic Decision Tree Terminologies

Parent and Child Node: A node that gets divided into sub-nodes is known as Parent Node, and these sub-nodes are known as Child Nodes.
Root Node: The topmost node of a decision tree. It does not have any parent node. It represents the entire population or sample.
Leaf / Terminal Nodes: Nodes of the tree that do not have any child node.

- Classification: class labels or proportions
- Regression: numeric, average, local fit.

Internal decision nodes

Univariate: uses a single attribute

- Numeric: Binary split (x>w)
- Discrete: n-way split for n possible values

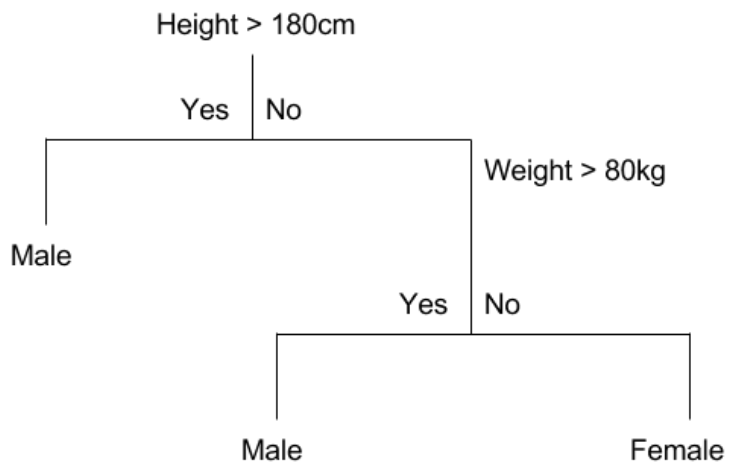
Multivariate: uses all attributes

Decision Tree Algorithms

1. From the root down - greedy approach
2. Established basic algorithms including ID3 and C4.5

CART (Classification and Regression Trees)

The representation for the CART model is a binary tree.
Each **root node** represents a single input variable (x) and a split point on that variable (assuming the variable is numeric).
The **leaf nodes** of the tree contain an output variable (y) which is used to make a prediction.
E.g: Given a dataset with two inputs (x) of height in centimeters and weight in kilograms the output of sex as male or female, below is a crude example of a binary decision tree



Creating a CART model involves selecting input variables and split points on those variables until a suitable tree is constructed. For that, we use a **greedy algorithm** to minimize a cost function.

Tree Induction:

The process of building the decision tree is called **Tree Induction**

1. Greedy Splitting:

   Creating a binary decision tree is a process of dividing up the input space.
   A greedy approach is used to divide the space where all the values are lined up and different split points are tried and tested using a cost function.
   Select the Best Split:

Split Method	Task	Definition
Gini Impurity/ Gini Index	Classification	Gini is the probability of correctly labeling a randomly chosen element if it is randomly labeled according to the distribution of labels in the node. The lower the Gini Impurity, the higher the homogeneity of the node. The Gini Impurity of a pure node is zero. $Gini = \sum_{i=1}^{n} p_i^2$ Gini Impurity = 1- Gini **Gini Impurity is preferred to Information Gain because it does not contain logarithms which are computationally intensive.**
Information gain/ Entropy	Classification	Entropy is used for calculating the purity of a node. The lower the value of entropy, the higher the purity of the node. The entropy of a homogeneous/pure node is zero. Since we subtract entropy from 1, the Information Gain is higher for the purer nodes with a maximum value of 1. $Entropy = -\sum_{i=1}^{n} p_i log_2(p_i)$ $Information\ Gain = 1 - Entropy$
Variance Reduction	Regression	Variance is used for calculating the homogeneity of a node. If a node is entirely

		homogeneous, then the variance is zero. $Variance = \frac{\sum(X-\mu)^2}{N}$

2. Recursive Partitioning
    a. Start with the entire dataset.
    b. Select the best feature and threshold to split the data based on the chosen metric.
    c. Divide the dataset into subsets (left and right child nodes).
    d. Repeat the process recursively for each subset until a **stopping criterion** is met (e.g., maximum depth, minimum samples per leaf, no further improvement).

Tree Pruning

We use tree pruning to avoid overfitting by simplifying the tree.

Method		Advantage
Pre-Prunning (Early Stopping)	Stop splitting when a criterion is met (e.g., maximum depth, minimum number of samples required to split).	Faster
Post-Pruning	Growing the whole tree then prune subtrees that overfit using the separate validation pruning set.	More accurate

Rule Induction

Rule induction is the process of extracting useful IF-THEN rules from a dataset. These rules are often derived from decision trees or other models and can be used for classification or decision-making purposes.

Why use decision rules?

- More compact
- More understandable

How to learn Decision rules?

- Convert trees to rules
- Use specific Rule learning methods like **Sequential Covering Algorithms** (Generate rules one at a time until all positive examples are covered.) E.g: Ripper, IREP

Algorithm	Description	Steps
IREP	IREP is an algorithm designed to create rules for classification by incrementally growing and pruning them. The main goal of IREP is to generate rules that generalize well on unseen data by reducing overfitting through pruning.	1. **Initialization** Stat with an empty set of rules Split the training data into a growing set and a pruning set 2. **Rule Generation (Growing Phase)** Select a Class Grow rule by adding conditions to maximize accuracy the the growing set. 3. **Rule Pruning (Pruning Phase)** Simplify the rule by removing conditions to improve its accuracy on the pruning set using Description Length. 4. **Add rule to Rule Set** Add pruned rule to rule set, remove covered examples from training data 5. **Repeat** Repeat until no more examples can be covered or predefined size is reached 6. **Termination** Stop when no further rules can be generated or a specific number if iterations is reached.
RIPPER	RIPPER is an extension and improvement of IREP. It includes additional steps and modifications to improve the accuracy and efficiency of the rule induction process.	Same as IREP algorithm bus introduces an **Optimization Phase** after Rule Pruning that is post pruning to further refine the rule set

	RIPPER handles both binary and multi-class classification problems effectively.	

Role of Description Length

Definition:

Description length is a measure of the complexity of the rule and the number of misclassifications it produces.

Formula:

$DL(rule) = Length\ of\ Rule + Error\ of\ Rule$

Where:

$Length\ of\ Rule$: The number of conditions in the rule.

$Error\ of\ Rule$: The number of misclassified examples in the pruning set.

Usage:

During the pruning phase, the algorithm evaluates the description length of the rule after removing each condition.

- The condition is retained if its removal increases the description length because a long DL indicates that the rule has become less effective or more complex.
- The condition is removed if it decreases the description length because a short DL indicates that the rule has become simpler or more accurate.

Tree induction = Breadth First

Rule induction = Depth First (one rule at a time)

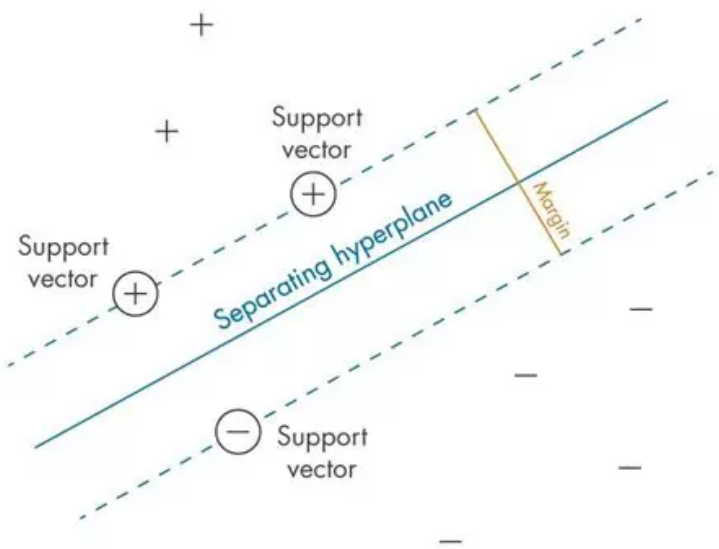
Chapter 7:
Kernel Machines

Kernel machines are models that use the Kernel function - that **maps data from one space to another space.**
Kernel Methods involve using linear classifiers to solve nonlinear problems by finding the hyperplane that separates the data points of different classes.

Advantages:

- Instances don't need to be represented as vectors.
- Effective in high dimensional cases
- Memory efficient as it uses a a subset of training points in the decision function (support vectors)

Basic Kernel Machines Terminologies:



Term	Definition
Hyperplane	A decision boundary that separets data points of different classes in a feature space
Support Vectors	Support vectors are the closest data points to the hyperplace which makes a critical role in deciciding the hyperplane and margin.
Margin	The distance between the support vectors and the hyperplane. The main objective is to **maximize** the margin because the wider the margin the better the classification performance.
Kernel Trick	A method used to enable kernel methods to classify non-linear data using a linear classifier by applying a kernel function. It maps the input data into a higher-dimensional space where a hyperplance is used to divide the classes. This mapping is computationally efficient because it avoids the direct calculation of the coordinates in this higher space.
Kernel Function	A mathematical function used to measure how **similar** two data points are in the input space by mapping the original input data into a high-dimensional feature space making it easier to find a hyperplace that separates the classes. **Linear Kernel:** No transformation is actually performed, the original feature space is used as it is. $K(w,x) = w^T x + b$ **Polynomial kernel:** Maps the original feature space to a higher-dimensional polynomial space. $K(w,x) = (\gamma w^T x + b)^N$ **Radial Basis Function (RBF)/ Gaussian Kernel:** Maps the original feature space to an infinite-dimensional space. $K(w,x) = exp\left(-\gamma\left\|x_i - x_j\right\|^2\right)$
Hard Margin	The maximum-margin hyperplane that separates the data points without any misclassifications
Soft Margin	Used when the data in not perfectly seperable or contains outliers. Introduces slack variables to allow for some misclassifications. The objective is to find a balance between maximizing the margin and minimizing classification errors.
C	Regularization Parameter that controls the trade-off between maximizing the margin and minimizing classification errors **Effect:** - **Large C value:** Imposes a higher penalty for misclassifications, resulting in a smaller margin and fewer

	missclassifications - **Small C value:** Allows a larger margin with more potential misclassifications.
Hinge Loss	a loss function used to penalizes misclassifications and correctly classified predictions that are too close to the decision boundary therefore it ensures a larger margin for robust classification.
Dual Problem	An alternative formulation of the SVM optimization problem that focuses on finding

Kernel Machines Comparison:

	Training Data	Objective	Common algorithms	Application
One-Class	Primarily one class (normal data)	An unsupervised algorithm that learns a decision function for novelty detection: classifying new data as similar or different to the training set.	One-Class SVM, isolation Forest	Fault Detection
Binary-Class	Two Distinct Classes	Classify instances into one of two classes	SVM, Logistic Regression, Decision trees	Email spam detection
Multi-Class	Multiple distinct classes	Classify instances into one of multiple classes	One-vs-All, One-vs-One	Digit recognition

Kernel Algorithms:

1. SVM
2. Kernel PCA
3. Kernel LDA

Multiple Kernel Learning

Multiple Kernel Learning (MKL) is a machine learning approach that uses a predefined set of kernels and learns the optimal combination of these kernels, either linear or non-linear, as part of the algorithm.
Advantages:

- Optimal kernel and parameter selection.
- Combining data from different sources that may require different kernels.

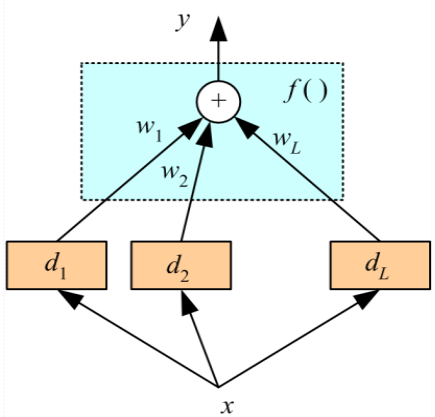
Types of kernel combinations

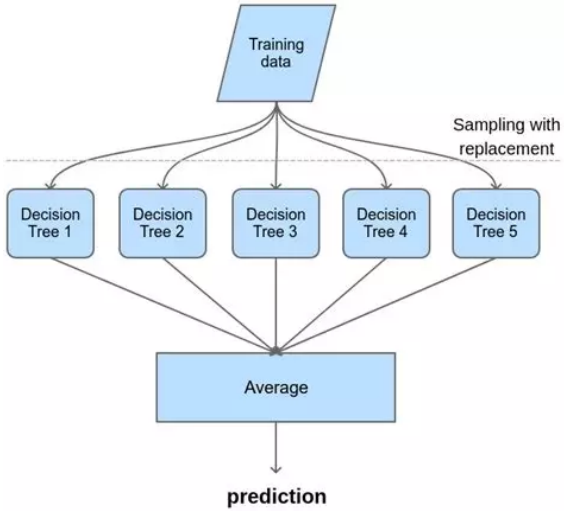
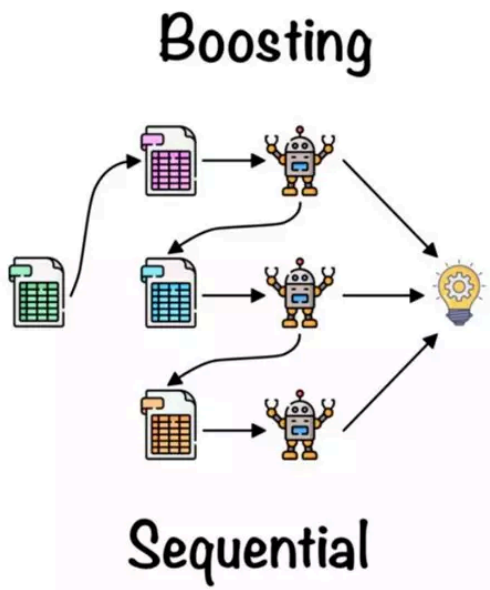
	Description	Pros	Cons
Fixed Kernel Combination	Preselects a set of kernels representing different similarity measures and combines them using predefined fixed weights.	- Easy to implement.	- May not adapt well to specific data characterisitcs.
Adaptive Kernel Combination	Learns the optimal weights for the kernels directly during training.	- Adapts to the data, potentially improving performance.	- Requires additional computation to learn the weights.
Adaptive Kernel Combination	Partitions the feature space into regions and assign different kernels to each region capturing local variations.	- Localized adaptation.	- More complex as it requires determining which kernel is relevant in which region.

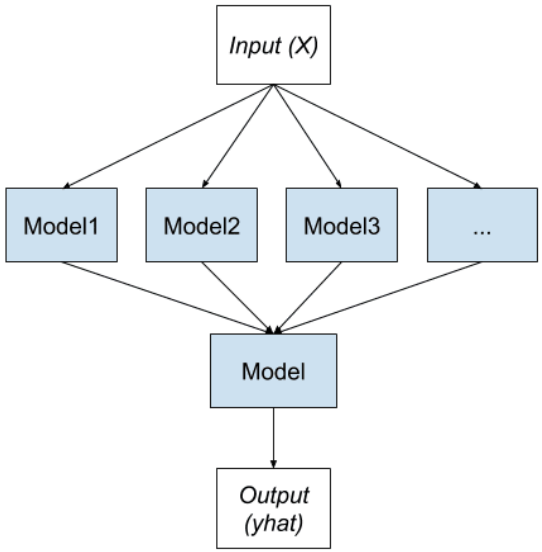
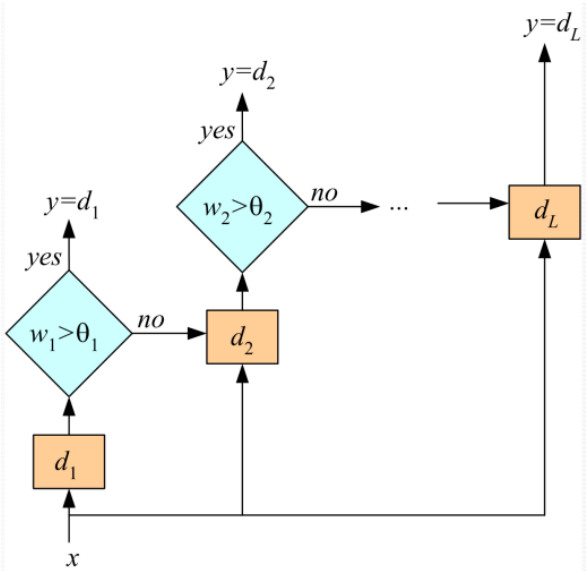
Chapter 8:
Combining Multiple Learners

Ensemble Methods are echnique that combines several base models in order to produce one optimal predictive model.
To ensure a **bias-variance tradeoff**

Ensemble Methods

	Descritption	
Voting	Voting is an ensemble method that combines the predictions of multiple models by taking a vote on the most popular class label or averaging the predictions for regression tasks. **Types:** • **Hard Voting:** Each model votes for a class, and the class with the most votes is chosen. • **Soft Voting:** Each model provides a probability distribution over classes, and the probabilities are averaged to make the final prediction.	

Bagging (Bootstrap Aggregating)	Bagging is an ensemble technique that trains multiple models in **parallel** on different subsets of the training data, created by random sampling **with replacement.** The final prediction is the average of individual predictions (for regression) or the majority vote (for classification). An algorithm is called **unstable** if small changes to the training set cause large changes in the learned classifier. If the learning algorithm is unstable, then bagging almost always improves performance because it reduces the variance of the model hence prevent overfitting.	
Boosting	Boosting is an iterative ensemble technique that **sequentially** trains models, each trying to correct the errors of its predecessor. It focuses on training the next model to emphasize the mistakes made by the previous models. **Types**: • **AdaBoost**: Assigns weights to each instance, increasing weights for misclassified instances and decreasing weights for correctly classified ones. • **Gradient Boosting**: Sequentially fits new models to the residual errors of the combined ensemble model.	

Stacking	Stacking is an ensemble method that combines multiple models using a meta-model (also called a combiner or a second-level model) to learn how to best combine the base models' predictions.	
Cascading	Cascading is an ensemble method where models are arranged in a **sequence**, and the output of one model is used as the input for the next model in the sequence.	

Mixture of Experts	Mixture of experts is an ensemble learning technique that involves decomposing predictive modeling tasks into sub-tasks training an expert model on each. Then the models are combined using gating that learns to assign weights to the predictions of each model.