Machine Learning

Gabriel Rovesti

Contents

1 Introduction to Machine Learning

1.1 Definition and Basic Concepts

Machine Learning (ML) is the field of study that gives computers the ability to learn without being explicitly programmed (Arthur Samuel, 1959). Tom Mitchell defined machine learning as: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

The fundamental assumption in machine learning is that there exists a (stochastic) process that explains the observed data. While we may not know the details, the aim of ML is to build a good approximation of this process.

1.2 Types of Reasoning

1.2.1 Deductive Reasoning

Deductive reasoning (Aristotle 384 BC – 322 AD) follows the pattern:

- RULE(C→R): All men are mortal
- CASE(C_1): Socrates is a man
- RESULT(R_1): Socrates is mortal

Deductive reasoning is the foundation of mathematical proofs and theorems. Starting from true assumptions, new theorems are inferred by means of logical consequences.

1.2.2 Inductive Reasoning

Inductive reasoning (F. Bacon, philosopher 1561–1626) follows the pattern:

- CASE(C_1): Socrates is a man
- RESULT(R_1): Socrates is mortal
- RULE(C→R): All men are mortal

In inductive reasoning, the premises provide evidence supporting the conclusion but do not guarantee its truth. This is the type of reasoning used in machine learning.

1.2.3 Abductive Reasoning

Abductive reasoning (C.S. Peirce, 1839–1914) follows the pattern:

- RULE(C→R): All men are mortal
- RESULT(R_1): Socrates is mortal
- CASE(C_1): Socrates is a man

In abductive reasoning, we assume an implication to be valid in reverse, "moving laterally" rather than generalizing.

1.3 Machine Learning Approaches

1.3.1 When to Use Machine Learning

Machine learning is particularly useful when:

- Exact problem characterization is impossible
- Input/output contains noise and uncertainty
- Solution formulation has high complexity
- Solution is inefficient
- There is a lack of compiled knowledge about the problem

1.3.2 Importance of Learning

Learning is important when a system needs to:

- Adapt to its operating environment
- Improve performance on a specific task
- Discover regularities and new information from empirical data

1.4 Applications of Machine Learning

Machine learning has numerous applications including:

- Face recognition
- Named entity recognition
- Document classification
- Game playing and opponent profiling
- Bioinformatics
- Recommender systems
- Speech recognition
- Handwritten recognition
- Social network analysis

Notable achievements include:

- Deep Blue defeating world chess champion Garry Kasparov in 1997
- AlphaGo defeating champion Lee Sedol at Go in 2016
- Generative adversarial networks creating realistic images
- Large language models like ChatGPT generating human-like text

2 Mathematical Foundations

2.1 Probability Theory

2.1.1 Basic Definitions

A random experiment is one whose outcome is not predictable with certainty in advance. Probability can be interpreted either as a frequency or as a degree of belief.

2.1.2 Axioms of Probability

- $0 \leq P(E) \leq 1$. If E_1 cannot possibly occur, then $P(E_1) = 0$. If E_2 is certain to occur, $P(E_2) = 1$.
- If S is the sample space containing all possible outcomes, $P(S) = 1$.
- If E_i, $i = 1, \ldots, n$ are mutually exclusive, we have $P(\cup_{i=1}^{n} E_i) = \sum_{i=1}^{n} P(E_i)$.
- In particular, $P(E^c) = 1 - P(E)$ holds if E^c denotes the complement of E.
- If the intersection of E and F is not empty, we have: $P(E \cup F) = P(E) + P(F) - P(E \cap F)$.

2.1.3 Conditional Probability

$P(E|F)$ (or posterior probability of E given F) is the probability of the occurrence of event E given that F occurred:

$$P(E|F) = \frac{P(E \cap F)}{P(F)} \tag{1}$$

Since $P(F)P(E|F) = P(E \cap F) = P(E)P(F|E)$, we obtain the Bayes formula:

$$P(F|E) = \frac{P(E|F)P(F)}{P(E)} \tag{2}$$

Two events E, F are said to be independent when $P(E|F) = P(E)$. When events are independent, $P(E \cap F) = P(E)P(F)$.

2.1.4 Mean and Variance

The mean (expected value) of a random variable X, $E[X]$, is the average value of X in a large number of experiments:

$$E[X] = \sum_i x_i P(x_i) \tag{3}$$

Properties:

- $E[aX + b] = aE[X] + b$
- $E[X + Y] = E[X] + E[Y]$
- $E[g(X)] = \sum_i g(x_i)P(x_i)$
- $E[X^n] = \sum_i x_i^n P(x_i)$ (nth moment)

The variance measures how much X varies around the expected value:

$$Var(X) = E[(X - \mu)^2] = E[X^2] - \mu^2 \tag{4}$$

The standard deviation is defined as $\sigma(X) = \sqrt{Var(X)}$.

2.1.5 Common Distributions

Discrete distributions:

- Bernoulli - Output 1 (success), 0 (failure) with $P(X = 1) = p$ and $P(X = 0) = 1 - p$. $E[X] = p$, $Var(X) = p(1 - p)$.
- Binomial - Number of successes in N independent Bernoulli trials: $P(X = i) = \binom{N}{i} p^i (1 - p)^{N-i}$, $i = 0, \ldots N$. $E[X] = Np$, $Var(X) = Np(1 - p)$.

Continuous distributions:

- Uniform in interval $[a, b]$: $p(x) = \frac{1}{b-a}$ if $a \leq x \leq b$ and $p(x) = 0$ otherwise. $E[X] = \frac{a+b}{2}$, $Var(X) = \frac{(b-a)^2}{12}$.
- Normal (Gaussian) with mean μ and variance σ^2, $N(\mu, \sigma^2)$: $p(x) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(x-\mu)^2}{2\sigma^2})$.

2.2 Linear Algebra

2.2.1 Vectors

An n-dimensional vector $x \in \mathbb{R}^n$ is a collection of n scalar values arranged in a column:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \tag{5}$$

Vector addition is performed element-wise, and scalar multiplication is distributed across all elements.

2.2.2 Dot Product

Given two n-dimensional vectors x, z, their dot product is a scalar:

$$x \cdot z = \sum_i x_i z_i \tag{6}$$

The square of the length of a vector x is:

$$|x|^2 = \sum_i x_i^2 \tag{7}$$

The dot product has a natural geometric interpretation:

$$x \cdot z = |x||z| \cos(\theta) \tag{8}$$

where θ is the angle between the vectors.

2.2.3 Binary Vectors and Sets

For binary vectors $x \in \{0, 1\}^n$, a vector can be interpreted as a set by considering a universe of n elements, and x represents the set containing elements corresponding to the ones in the vector.

The projection of a vector x along the direction of another vector z is:

$$x_z = \frac{(x \cdot z)}{|z|} \frac{z}{|z|} = \frac{x \cdot z}{z \cdot z} z = \alpha z \tag{9}$$

2.2.4 Matrices

A matrix $A \in \mathbb{R}^{m \times n}$ is a collection of scalar values arranged in a rectangle of m rows and n columns:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \tag{10}$$

Matrix addition is element-wise: $[A + B]_{ij} = [A]_{ij} + [B]_{ij} = a_{ij} + b_{ij}$.

Matrix multiplication for $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$ is defined as:

$$[AB]_{ij} = \sum_{q=1}^{k} [A]_{iq}[B]_{qj} = \sum_{q=1}^{k} a_{iq} b_{qj} \tag{11}$$

2.2.5 Transpose

The transpose $A^\top \in \mathbb{R}^{m \times n}$ of a matrix $A \in \mathbb{R}^{n \times m}$ is defined by:

$$[A^\top]_{ij} = A_{ji} \tag{12}$$

Properties:

- $(A^\top)^\top = A$
- $(AB)^\top = B^\top A^\top$

A matrix A is symmetric if $A = A^\top$.

2.2.6 Inverse

The identity matrix $I \in \mathbb{R}^{n \times n}$ has ones on the diagonal and zeros elsewhere.

The inverse matrix of a square matrix $A \in \mathbb{R}^{n \times n}$ is a matrix A^{-1} such that $AA^{-1} = I = A^{-1}A$.

An inverse exists only if the determinant of A is not zero (i.e., A has full rank).

For inverse matrices, $(AB)^{-1} = B^{-1}A^{-1}$.

For rectangular matrices, if $AA^\top$ is invertible, then the matrix $A^\dagger = A^\top(AA^\top)^{-1}$ (called the pseudo-inverse) satisfies $AA^\dagger = I$.

2.2.7 Solving Linear Systems

The linear system $Ax = b$ has solution $x = A^{-1}b$ when A is invertible.

2.2.8 Positive Definite Matrices

A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite if $x^\top Ax \geq 0$ for any vector $x \in \mathbb{R}^n$ (eigenvalues ≥ 0).

A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive definite if $x^\top Ax > 0$ for any vector $x \in \mathbb{R}^n$ (eigenvalues > 0).

Positive definite matrices are always invertible.

2.3 Optimization

2.3.1 Optimization Problems

Optimization problems involve maximizing or minimizing a real function $f : \mathbb{R}^n \to \mathbb{R}$ with respect to some variables, potentially subject to constraints.

They can be divided into:

- Discrete Optimization: variables have discrete values
- Continuous Optimization: variables have continuous values

A general form is $f(x) = g(x) + \beta h(x)$, where the parameter β controls the influence of $h(x)$ versus $g(x)$.

2.3.2 Linear and Quadratic Optimization

Linear Optimization Problem:

$$\min_x f(x) = a \cdot x \tag{13}$$

$$\text{subject to: } Ax \leq b \tag{14}$$

$$Bx = d \tag{15}$$

Quadratic Optimization Problem:

$$\min_x f(x) = x^\top Ax + bx \tag{16}$$

$$\text{subject to: } Bx \leq c \tag{17}$$

If matrix A is positive definite, the function $f(x)$ is convex and the optimization problem has a unique minimum.

2.3.3 Optimization Techniques

Basic Gradient Descent Gradient descent is an iterative optimization algorithm for finding a local minimum of a differentiable function. The basic steps are:

1. Start at a point x_0
2. Compute the gradient $\nabla f(x_k)$
3. Take a step in the negative gradient direction: $x_{k+1} = x_k - \eta \nabla f(x_k)$
4. Repeat until convergence

Where $\eta > 0$ is the learning rate that determines the step size.

Stochastic Gradient Descent (SGD) Instead of computing the gradient on the entire dataset, SGD computes the gradient on a single randomly selected example:

$$w_{t+1} = w_t - \eta \nabla L_i(w_t) \tag{18}$$

Mini-batch SGD A compromise between batch and stochastic approaches, using a small random subset of examples:

$$w_{t+1} = w_t - \eta \frac{1}{|B|} \sum_{i \in B} \nabla L_i(w_t) \tag{19}$$

where B is a mini-batch of examples.

SGD with Momentum Adds a fraction of the previous update to the current one:

$$v_{t+1} = \gamma v_t + \eta \nabla L(w_t) \tag{20}$$

$$w_{t+1} = w_t - v_{t+1} \tag{21}$$

where $\gamma \in [0, 1]$ is the momentum parameter. This helps accelerate convergence and reduces oscillation.

Adam Optimizer Adaptive Moment Estimation combines momentum with adaptive learning rates for each parameter:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla L(w_t) \tag{22}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla L(w_t))^2 \tag{23}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{24}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{25}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{26}$$

where β_1 and β_2 are decay rates for moment estimates, and ϵ is a small constant for numerical stability.

Learning Rate Scheduling Techniques to adjust the learning rate during training:

- Step decay: Reduce learning rate by a factor after a fixed number of epochs
- Exponential decay: $\eta_t = \eta_0 \cdot e^{-kt}$
- $1/t$ decay: $\eta_t = \frac{\eta_0}{1+kt}$
- Cosine annealing: $\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\frac{t\pi}{T}))$

Early Stopping Stop training when performance on a validation set stops improving, to prevent overfitting. The best model found during training is retained.

3 Learning Theory

3.1 PAC Learning

3.1.1 The Bin Experiment Analogy

Consider a bin with red and green marbles where:

- $P(\text{red}) = \pi$
- $P(\text{green}) = 1 - \pi$

If we pick N marbles (our sample) from the bin and find that a fraction σ of them are red, what does σ tell us about π?

In a large sample, the value σ is likely close to π (within ϵ). Formally, by Hoeffding's Inequality:

$$P(|\sigma - \pi| > \epsilon) \leq 2e^{-2\epsilon^2 N} \tag{27}$$

This means that $\sigma = \pi$ is Probably Approximately Correct (PAC).

3.1.2 Connection to Learning

In supervised learning:

- The bin represents our input space X
- Given a hypothesis h, green marbles correspond to instances where the hypothesis is correct: $h(x) = f(x)$
- Red marbles correspond to instances where the hypothesis is wrong: $h(x) \neq f(x)$

For a specific hypothesis h, the in-sample error $E_i(h)$ (equivalent to σ) approximates the out-of-sample error $E_o(h)$ (equivalent to π):

$$P(|E_i(h) - E_o(h)| > \epsilon) \leq 2e^{-2\epsilon^2 N} \tag{28}$$

3.1.3 Multiple Hypotheses

When choosing from multiple hypotheses, the problem becomes more complex. The unluckiest learner would always choose a hypothesis where the "bad event" occurs.

Using the Union Bound:

$$P[|E_i(g) - E_o(g)| > \epsilon] \leq P[|E_i(h_1) - E_o(h_1)| > \epsilon \text{ or } \ldots \text{ or } |E_i(h_M) - E_o(h_M)| > \epsilon] \tag{29}$$

$$\leq \sum_{m=1}^{M} P[|E_i(h_m) - E_o(h_m)| > \epsilon] \leq 2Me^{-2\epsilon^2 N} \tag{30}$$

3.2 VC Dimension

3.2.1 Growth Function

The number of hypotheses M can be very large or even infinite. We can replace M with $m_H(N)$, the growth function, defined as:

$$m_H(N) = \max_{x_1,\ldots,x_N} |H(x_1, \ldots, x_N)| \leq 2^N \tag{31}$$

where $|H(x_1, \ldots, x_N)|$ is the number of dichotomies possible on $x_1, \ldots, x_N$ using hypotheses from H.

3.2.2 Shattering

A set $S \subset X$ is shattered by hypothesis space H if for every possible binary labeling of instances in S, there exists a hypothesis $h \in H$ that perfectly classifies them.

3.2.3 VC-Dimension Definition

The VC-dimension of a hypothesis space H defined over an instance space X is the size of the largest finite subset of X shattered by H:

$$VC(H) = \max_{S \subseteq X} |S| : S \text{ is shattered by } H \tag{32}$$

3.2.4 Example: VC-Dimension of Hyperplanes

For hyperplanes in $\mathbb{R}^2$:

$$H_1 = \{f_{(\vec{w},b)}(\vec{y}) | f_{(\vec{w},b)}(\vec{y}) = \text{sign}(\vec{w} \cdot \vec{y} + b), \vec{w} \in \mathbb{R}^2, b \in \mathbb{R}\} \tag{33}$$

The VC-dimension is 3, as we can shatter 3 points but not 4 points.

More generally, for hyperplanes in $\mathbb{R}^n$, $VC(H) = n + 1$.

3.3 Generalization and Structural Risk Minimization

3.3.1 VC Bound

For a binary classification learning problem with training set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, hypothesis space H, and learning algorithm L that returns the hypothesis $g = h_\theta^*$ minimizing the empirical error on S, an upper bound of the ideal error is:

$$\text{error}(g) \leq \text{error}_S(g) + F\left(\frac{VC(H)}{n}, \frac{1}{\delta}\right) \tag{34}$$

where δ is the confidence parameter and F is a function of the VC-dimension and sample size.

3.3.2 Analysis of the Bound

The bound has two terms:

- $A = \text{error}_S(g)$: Depends on the hypothesis returned by the learning algorithm
- $B = F(VC(H)/n, 1/\delta)$: The VC-confidence, which depends on the VC-dimension, training size, and confidence parameter

As the VC-dimension grows, the empirical risk (A) decreases, but the VC confidence (B) increases.

3.3.3 Structural Risk Minimization

Structural Risk Minimization (SRM) aims to minimize the right-hand side of the confidence bound, achieving a tradeoff between empirical error and complexity.

Consider a sequence of hypothesis spaces H_i such that:

- $H_1 \subseteq H_2 \subseteq \cdots \subseteq H_n$
- $VC(H_1) \leq \cdots \leq VC(H_n)$

SRM selects the hypothesis with the smallest bound on the true risk.

4 Supervised Learning

4.1 Fundamentals of Supervised Learning

4.1.1 Formalization and Terminology

In supervised learning:

- Input: $x \in X$ (e.g., email representation)
- Output: $y \in Y$
  - Binary classification: $Y \equiv \{-1, +1\}$
  - Multi-class classification: $Y \equiv \{1, \ldots, m\}$
  - Regression: $Y \equiv \mathbb{R}$
- Oracle/Nature: Either a deterministic target function $f : X \to Y$ or probability distributions $P(x), P(y|x)$
- Data: Independent and identically distributed (i.i.d.) data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$
- Hypothesis space: A predefined set of hypotheses/functions $H \equiv \{h|h : X \to Y\}$

4.1.2 Learning Process

- A training set of pairs (x_i, y_i) is available, generated according to unknown probability function $P(x, y) = P(x)P(y|x)$
- A 'plausible' hypothesis $g : X \to Y$ is selected from the hypothesis space H using training data
- The error on training data is called empirical error/risk
- The expected error on new data is called ideal error/risk

4.1.3 The Inductive Bias

For learning to be feasible, assumptions must be made about the complexity of the target function and the hypothesis space. The inductive bias consists of:

- The hypothesis space: how the space H is defined
- The learning algorithm: how the space H is explored

Example: K-Nearest Neighbor (KNN) has the inductive bias that nearby instances are expected to have the same label.

4.1.4 Polynomial Regression Example

Given training data $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, $x \in \mathbb{R}$, $y \in \mathbb{R}$, we want to find a polynomial of degree p:

$$h_w(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_p x^p \tag{35}$$

The empirical error is:

$$\text{error}_S(h_w) = \frac{1}{n} \sum_{i=1}^{n} (h_w(x_i) - y_i)^2 \tag{36}$$

Given a fixed p, we can find the weights w using ridge regression:

$$w = (X^\top X + \alpha I)^{-1} X^\top y \tag{37}$$

where α is a regularization parameter controlling the tradeoff between fitting the data and keeping the weights small.

4.2 Decision Trees

4.2.1 Introduction to Decision Trees

Decision trees (DTs) learn discrete decision functions/rules represented by a tree. They are particularly useful for:

- Instances represented as attribute-value pairs
- Fixed sets of attributes and values
- Few possible values for attributes
- Discrete values for attributes (with extensions to real-valued)
- Discrete output values
- Target concepts describable by disjunctions of boolean functions
- Data with noise and missing values

DTs are easily convertible to if-then rules, making them highly interpretable, which is valuable for applications in medicine, biology, and finance.

4.2.2 Structure of Decision Trees

In a decision tree:

- Inner nodes specify tests on attributes
- Branches correspond to possible attribute values
- Leaf nodes assign classifications

To classify an instance:

1. Start at the root
2. Select the attribute at the current node
3. Follow the branch corresponding to the attribute's value
4. If a leaf is reached, return its label; otherwise, repeat from step 2

4.2.3 Decision Trees and Boolean Functions

A decision tree can be represented as a Boolean function:

- Each path from root to leaf represents a conjunction of constraints
- Different paths to the same classification represent disjunctions
- Together, they form Disjunctive Normal Form (DNF) formulas

4.2.4 ID3 Algorithm

The ID3 algorithm builds decision trees through a top-down, greedy search:

Algorithm 1 ID3 Algorithm

ID3S: Sample, A: Attributes Create a root node T **if** examples in S are all of the same class c **then**
return leaf node T with label c
end if
if A is empty **then**
return leaf node T with the majority class in S
end if
Select $a \in A$, the optimal attribute in A
Partition S according to values of a: $S_{a=v_1}, \ldots, S_{a=v_m}$
return tree T with sub-trees from recursively calling ID3($S_{a=v_j}$,$A - a$)

4.2.5 Attribute Selection in ID3

ID3 uses entropy to measure the "impurity" of a sample S:

$$E(S) = -\sum_{c=1}^{m} p_c \log(p_c), \text{ where } p_c = \frac{|S_c|}{|S|} \tag{38}$$

For binary classification:

$$E(S) = -p_- \log(p_-) - p_+ \log(p_+) \tag{39}$$

The optimal attribute maximizes the Information Gain:

$$G(S, a) = E(S) - \sum_{v \in V(a)} \frac{|S_{a=v}|}{|S|} E(S_{a=v}) \tag{40}$$

Alternative criteria include:

- Gini Index: $I_G = 1 - \sum_{c=1}^{m} p_c^2$
- Misclassification: $I_E = 1 - \max_c(p_c)$

4.2.6 Decision Trees for Real-Valued Attributes

For continuous attributes, a new Boolean pseudo-attribute is created:

$$A_c = \begin{cases} \text{true} & \text{if } A < c \\ \text{false} & \text{otherwise} \end{cases} \tag{41}$$

The optimal threshold c is the one that maximizes information gain, typically at a boundary between classes.

4.2.7 Handling Missing Values

When attribute values are missing, possible approaches include:

- Using the most frequent value for the attribute
- Using the most frequent value considering only examples with the same class
- Creating fractional examples with weights based on the probability distribution of the attribute values

4.2.8 Pruning Techniques

To address overfitting, various pruning methods can be applied:

- Reduced Error Pruning: Replace subtrees with leaves when this improves performance on a validation set
- Rule-Post Pruning: Convert the tree to rules, prune the rules independently, and order them by performance

4.3 Neural Networks

4.3.1 Background and Motivation

Artificial Neural Networks (ANNs) are inspired by the human brain but focus on understanding the general computational principles rather than precisely replicating biological processes.

NNs are particularly useful when:

- Input data has high dimensionality with discrete and/or real values
- The form of the target function is unknown
- Long learning times are acceptable but quick evaluation is needed
- Interpretability is not a requirement

4.3.2 Single Neuron - Perceptron

A perceptron can be described as:

$$h(x) = \text{sign}(w \cdot x + b) = \text{sign}(\sum_{i=0}^{n} w_i x_i) \tag{42}$$

where $w_0 = b$ and $x_0 = 1$.

Algorithm 2 Perceptron Learning Algorithm

Input: Training set $S = \{(x, t)\}$, $x \in \mathbb{R}^{n+1}$, $t \in \{-1, +1\}$, $\eta > 0$ (learning rate)
Initialize weights w randomly
repeat
  Select a training example (x, t)
  if $o = \text{sign}(w \cdot x) \neq t$ **then**
    $w \leftarrow w + \eta(t - o)x$
  end if
until all instances are correctly classified

4.3.3 Perceptron Learning Algorithm

If the training set is linearly separable, the Perceptron algorithm converges in a finite number of steps.

4.3.4 Delta Rule

The Delta rule uses gradient descent to find the weights that minimize the mean squared error:

$$E[w] = \frac{1}{2N} \sum_{(x^{(s)}, t^{(s)}) \in S} (t^{(s)} - o(x^{(s)}))^2 \tag{43}$$

For a linear activation function $o(x) = w \cdot x$:

$$\Delta w_i = \eta(t - o)x_i \tag{44}$$

For a sigmoid activation function $o(x) = \sigma(w \cdot x)$ where $\sigma(y) = \frac{1}{1+e^{-y}}$:

$$\Delta w_i = \eta(t - o)\sigma(y)(1 - \sigma(y))x_i \tag{45}$$

4.3.5 Multilayer Neural Networks

Multilayer neural networks consist of:

- Input units representing input variables
- Hidden units encoding correlations among inputs
- Output units representing output variables
- Connections with adaptable weights

Activation Functions Common activation functions include:

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$, range (0, 1)
- Hyperbolic tangent: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, range (-1, 1)
- ReLU (Rectified Linear Unit): $\text{ReLU}(x) = \max(0, x)$
- Leaky ReLU: $\text{LeakyReLU}(x) = \max(\alpha x, x)$, typically $\alpha = 0.01$
- ELU (Exponential Linear Unit): $\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$
- Softmax (for output layer in classification): $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$

Loss Functions Common loss functions include:

- Mean Squared Error (regression): MSE $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Cross-Entropy (binary classification): CE $= -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$
- Categorical Cross-Entropy (multi-class): CCE $= -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(\hat{y}_{ij})$

Regularization in Neural Networks Techniques to prevent overfitting:

- Weight decay: L1 or L2 regularization on weights
- Dropout: Randomly set a fraction of inputs to zero during training

$$y = f(Wx) \Rightarrow y = f(W(r * x)) \tag{46}$$

  where r is a vector of Bernoulli random variables with probability p

- Batch Normalization: Normalize activations within a mini-batch

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{47}$$

$$y_i = \gamma \hat{x}_i + \beta \tag{48}$$

  where μ_B and σ_B are the mean and variance of the mini-batch, and γ and β are learnable parameters

4.3.6 Backpropagation Algorithm

The backpropagation algorithm trains multilayer networks by minimizing the mean squared error:

$$E(w) = \frac{1}{2cN}\sum_{s=1}^{N}\sum_{k=1}^{c}(t_k^{(s)} - z_k^{(s)})^2 \tag{49}$$

For output units, the error signal is:

$$\delta_k = z_k(1 - z_k)(t_k - z_k) \tag{50}$$

For hidden units, the error signal is:

$$\delta_j = y_j(1 - y_j)\sum_{k=1}^{c} w_{kj}\delta_k \tag{51}$$

Weight updates are:

$$\Delta w_{kj} = \eta \delta_k y_j \tag{52}$$

$$\Delta w_{ji} = \eta \delta_j x_i \tag{53}$$

4.3.7 Practical Considerations

- Batch vs. Stochastic Gradient Descent: Stochastic updates weights after each example; batch updates after all examples
- Local Minima: Adding momentum or using stochastic training can help escape local minima
- Overfitting: Can be addressed by monitoring validation error or using regularization
- Hidden Layer Representation: Hidden units learn an effective encoding of the input that simplifies classification

4.4 Linear and Generalized Linear Models

4.4.1 Linear Models Overview

Linear models have the form:

$$f_{w,b}(x) = \sum_{i=1}^{m} w_i x_i + b = w \cdot x + b \tag{54}$$

For classification, we return the sign: $h(x) = \text{sign}(f_w(x))$
For regression, we return the function value: $h(x) = f_w(x)$

4.4.2 Logistic Regression

Logistic regression is a generalized linear model for binary classification that outputs a probability instead of a direct class prediction:

$$P(y = 1|x) = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} \tag{55}$$

where σ is the sigmoid function.
For multi-class classification, this extends to the softmax function:

$$P(y = j|x) = \frac{e^{w_j \cdot x + b_j}}{\sum_{k=1}^{K} e^{w_k \cdot x + b_k}} \tag{56}$$

The loss function used for logistic regression is the cross-entropy loss:

$$L(w) = -\sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{57}$$

For multi-class classification with softmax:

$$L(w) = -\sum_{i=1}^{n} \sum_{j=1}^{K} y_{ij} \log(\hat{y}_{ij}) \tag{58}$$

where y_{ij} is 1 if the i-th example belongs to class j, and 0 otherwise.

4.4.3 Regularization Techniques

To prevent overfitting, regularization terms are added to the loss function:
L2 Regularization (Ridge):

$$L(w) = \text{Loss}(w) + \lambda \sum_{j=1}^{m} w_j^2 = \text{Loss}(w) + \lambda ||w||_2^2 \tag{59}$$

L1 Regularization (Lasso):

$$L(w) = \text{Loss}(w) + \lambda \sum_{j=1}^{m} |w_j| = \text{Loss}(w) + \lambda ||w||_1 \tag{60}$$

Elastic Net combines both L1 and L2:

$$L(w) = \text{Loss}(w) + \lambda_1 ||w||_1 + \lambda_2 ||w||_2^2 \tag{61}$$

L1 regularization tends to produce sparse solutions (feature selection), while L2 regularization shrinks all weights but rarely to exactly zero.

4.4.4 Margin of a Hyperplane

For a hyperplane $w \cdot x + b = 0$, the distance of a point x from the hyperplane is:

$$r = \frac{g(x)}{||w||} = \frac{w \cdot x + b}{||w||} \tag{62}$$

When properly scaled so that $g(x) = 1$ for the closest positive example and $g(x) = -1$ for the closest negative example, the margin is:

$$\rho = 2r = \frac{2}{||w||} \tag{63}$$

4.4.5 Optimal Hyperplane - SVM

The Support Vector Machine (SVM) finds the hyperplane with the maximum margin, which can be formulated as a constrained quadratic optimization problem:

$$\min_{w,b} \frac{1}{2}||w||^2 \tag{64}$$

$$\text{subject to: } \forall i \in \{1, \ldots, n\} : y_i(w \cdot x_i + b) \geq 1 \tag{65}$$

This problem has a dual formulation:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) \tag{66}$$

$$\text{subject to: } \forall i \in \{1, \ldots, n\} : \alpha_i \geq 0 \text{ and } \sum_{i=1}^{n} y_i \alpha_i = 0 \tag{67}$$

The solution is:

$$w = \sum_{i=1}^{n} y_i \alpha_i x_i \tag{68}$$

$$b = y_k - w \cdot x_k \text{ for any } x_k \text{ such that } \alpha_k > 0 \tag{69}$$

4.4.6 Non-Separable Case - Soft Margin SVM

For non-linearly separable data, slack variables $\xi_i \geq 0$ are introduced:

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} \xi_i \tag{70}$$

$$\text{subject to: } \forall i \in \{1, \ldots, n\} : y_i(w \cdot x_i + b) \geq 1 - \xi_i \tag{71}$$

The parameter $C > 0$ controls the tradeoff between margin maximization and classification errors.

4.4.7 Kernel Methods

Kernel methods map the input into a higher-dimensional feature space where linear separation is possible:

$$\phi : X \to \mathcal{F} \tag{72}$$

Rather than computing $\phi(x)$ explicitly, we use the kernel trick with a kernel function $K(x, z) = \phi(x) \cdot \phi(z)$.

Common kernel functions:

- Linear kernel: $K(x, z) = x \cdot z$
- Polynomial kernel: $K(x, z) = (x \cdot z + c)^s$, $c > 0$
- Radial basis function (RBF) kernel: $K(x, z) = \exp(-\gamma||x - z||^2)$, $\gamma > 0$

4.4.8 SVM for Regression

SVM can be adapted for regression using the ϵ-insensitive loss function:

$$L_\epsilon(y, f(x)) = \begin{cases} 0 & \text{if } |y - f(x)| \leq \epsilon \\ |y - f(x)| - \epsilon & \text{otherwise} \end{cases} \tag{73}$$

4.5 Preprocessing and Feature Selection

4.5.1 Types of Data and Encoding

- Categorical variables:
  - Nominal (no order): e.g., country, brand, color
  - Ordinal (ordered but distances not preserved): e.g., military ranks
- Quantitative variables:
  - Interval (enumerable): e.g., ratings from 0 to 10
  - Ratio (real): e.g., weight, height

4.5.2 Encoding Categorical Variables

One-hot encoding represents categorical variables as binary vectors, with one component for each possible value:

- For k possible values, create a k-dimensional binary vector
- Set the component corresponding to the value to 1, all others to 0

4.5.3 Preprocessing Continuous Variables

Common transformations for continuous variables:

- Centering: $c(x_{ij}) = x_{ij} - \hat{x}_j$
- Standardization: $s(x_{ij}) = \frac{c(x_{ij})}{\sigma_j}$
- Scaling to a range: $h(x_{ij}) = \frac{x_{ij} - x_{\min,j}}{x_{\max,j} - x_{\min,j}}$
- Normalization: $g(x) = \frac{x}{||x||}$

4.5.4 Feature Selection and Extraction

Feature selection reduces dimensionality by removing irrelevant or redundant features, while feature extraction creates new features by combining original ones.

Feature selection methods:

- Filter methods: Use scoring functions like Mutual Information or Chi-squared test
- Wrapper methods: Evaluate the predictor on feature subsets
- Embedded methods: Selection occurs during model creation (e.g., regularization)

Feature extraction methods:

- Principal Component Analysis (PCA): Creates linearly uncorrelated features
- Neural networks: Hidden layers can extract useful features

4.6 Model Selection and Evaluation

4.6.1 Underfitting and Overfitting

- Underfitting: Model is too simple to capture the underlying pattern (high bias)
- Overfitting: Model fits the training data too closely, capturing noise (high variance)

4.6.2 Bias-Variance Decomposition

For a target function $y = f(x) + \epsilon$ and an approximation $\hat{f}(x; D)$:

$$E[(y - \hat{f}(x; D))^2] = \text{Bias}[\hat{f}(x; D)]^2 + \text{Var}[\hat{f}(x; D)] + \sigma^2 \tag{74}$$

where:

$$\text{Bias}[\hat{f}(x; D)] = E_D[\hat{f}(x; D)] - f(x) \tag{75}$$

$$\text{Var}[\hat{f}(x; D)] = E[(\hat{f}(x; D) - E[\hat{f}(x; D)])^2] \tag{76}$$

4.6.3 Basic Model Selection Techniques

- Hold-out method: Reserve a subset of training data for validation
- K-fold Cross-Validation: Partition data into k sets, train on $k-1$ sets and validate on the remaining set, rotating k times
- Leave-One-Out Cross-Validation: Special case with k equal to the dataset size

4.6.4 Advanced Model Selection

Nested Cross-Validation When both hyperparameter tuning and model evaluation are needed:

1. Outer loop: Split data into training and test folds
2. Inner loop: For each training fold, perform k-fold CV to select hyperparameters
3. Evaluate final model on the outer test fold

This prevents information leakage from test to training.

Stratified Cross-Validation Ensures that each fold maintains the same class distribution as the full dataset, critical for imbalanced datasets.

Time Series Cross-Validation For time series data, maintains temporal ordering:

- Expanding window: Training set grows as time progresses
- Sliding window: Fixed-size training window moves forward in time

4.6.5 Hyperparameter Tuning

Grid Search Exhaustively search through a manually specified subset of hyperparameter space:

- Define grid of hyperparameter values
- Train model for each combination of hyperparameters
- Select combination with best validation performance

Computational complexity grows exponentially with the number of hyperparameters.

Random Search Sample hyperparameter values randomly from specified distributions:

- Define probability distribution for each hyperparameter
- Randomly sample from these distributions
- More efficient than grid search when some hyperparameters have minimal impact

Bayesian Optimization Sequential model-based optimization approach:

1. Build a probabilistic model (often Gaussian Process) of the objective function
2. Use an acquisition function to determine which hyperparameters to evaluate next
3. Update the probabilistic model with new results
4. Repeat until convergence or budget exhaustion

More efficient than random or grid search, especially for expensive evaluations.

4.6.6 Handling Class Imbalance

Techniques to address imbalanced datasets:

Resampling Techniques

- Under-sampling: Reduce majority class samples
- Over-sampling: Increase minority class samples
- SMOTE (Synthetic Minority Over-sampling Technique): Generate synthetic minority class samples

Cost-sensitive Learning Assign different misclassification costs to different classes:

- Weighted loss functions
- Class-weight adjustments in algorithms

Ensemble Methods for Imbalanced Data

- SMOTE + Boosting
- Balanced Random Forest
- Easy Ensemble: Train on balanced bootstrap samples from majority class with all minority samples

4.6.7 Evaluation Metrics for Classification

For binary classification:

- Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision: $\frac{TP}{TP+FP}$
- Recall: $\frac{TP}{TP+FN}$
- F-measure: $\frac{(1+\beta^2)\pi\rho}{\beta^2\pi+\rho}$, where β controls the precision-recall tradeoff

4.6.8 Multiclass Classification

Approaches to multiclass classification:

- One-vs-Rest: Train one classifier per class, against all other classes
- One-vs-One: Train a classifier for each pair of classes

Evaluation metrics for multiclass:

- Confusion Matrix: Shows predicted vs. actual classes
- Precision and Recall per class
- Micro-averaging: Calculate from grand totals
- Macro-averaging: Average metrics across classes

4.7 Bayesian Learning

4.7.1 Bayes Theorem and Hypotheses

Bayes theorem relates the posterior probability of a hypothesis given the data to the likelihood and prior:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} = \frac{P(D|h)P(h)}{\sum_{h'} P(D|h')P(h')} \tag{77}$$

where:

- $P(h)$: Prior probability of hypothesis h
- $P(D)$: Prior probability of training data
- $P(h|D)$: Posterior probability of h given D
- $P(D|h)$: Likelihood - probability of D given h

4.7.2 Maximum a Posteriori (MAP) and Maximum Likelihood (ML)

The maximum a posteriori hypothesis:

$$h_{MAP} = \arg\max_{h \in H} P(h|D) = \arg\max_{h \in H} P(D|h)P(h) \tag{78}$$

With uniform priors, this simplifies to the maximum likelihood hypothesis:

$$h_{ML} = \arg\max_{h \in H} P(D|h) \tag{79}$$

4.7.3 Bayesian Justification for Common Error Functions

For a regression problem with Gaussian noise, maximizing the likelihood is equivalent to minimizing the squared error:

$$h_{ML} = \arg\min_{h \in H} \sum_{i=1}^{m} (d_i - h(x_i))^2 \tag{80}$$

For binary classification with a probabilistic function, maximizing the likelihood is equivalent to minimizing the cross-entropy:

$$h_{ML} = \arg\max_{h \in H} \sum_{i=1}^{m} d_i \ln(h(x_i)) + (1 - d_i) \ln(1 - h(x_i)) \tag{81}$$

4.7.4 Bayes Optimal Classification

The optimal Bayesian prediction averages over all hypotheses weighted by their posterior probabilities:

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) \tag{82}$$

The Bayes optimal classification maximizes this probability:

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) \tag{83}$$

4.7.5 Naive Bayes Classifier

The Naive Bayes classifier applies Bayes' theorem with the "naive" assumption that features are conditionally independent given the class:

$$P(a_1, a_2, \ldots, a_n|v_j) = \prod_i P(a_i|v_j) \tag{84}$$

Classification is performed by:

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i|v_j) \tag{85}$$

4.7.6 Generative Models

Gaussian Mixture Models (GMM) A GMM models the probability density of the data as a weighted sum of Gaussian distributions:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{86}$$

where π_k are the mixture weights, μ_k are the means, and Σ_k are the covariance matrices of the K Gaussian components.

Expectation Maximization (EM) Algorithm The EM algorithm is used to estimate the parameters of latent variable models, such as GMM:

1. Initialize parameters $\theta = \{\pi_k, \mu_k, \Sigma_k\}$
2. **E-step**: Compute the responsibilities (posterior probabilities)

$$\gamma_{ik} = P(z_i = k|x_i, \theta) = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} \tag{87}$$

3. **M-step**: Update parameters to maximize the expected log-likelihood

$$N_k = \sum_{i=1}^{N} \gamma_{ik} \tag{88}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{89}$$

$$\mu_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} x_i \tag{90}$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} (x_i - \mu_k^{new})(x_i - \mu_k^{new})^T \tag{91}$$

4. Iterate until convergence

Hidden Markov Models (HMM) HMMs model sequential data with hidden states:

- States $S = \{s_1, \ldots, s_N\}$
- Observations $O = \{o_1, \ldots, o_T\}$
- Initial state probabilities $\pi = \{\pi_i\}$
- State transition probabilities $A = \{a_{ij}\}$
- Emission probabilities $B = \{b_i(o_t)\}$

Key algorithms:

- Forward-Backward algorithm: Computes the probability of an observation sequence
- Viterbi algorithm: Finds the most likely state sequence
- Baum-Welch algorithm: EM algorithm for HMMs to estimate parameters

Variational Autoencoders (VAE) VAEs are generative models that learn a probabilistic mapping between data and a latent space:

- Encoder: Maps input x to parameters of a distribution $q_\phi(z|x)$ in latent space
- Decoder: Maps samples z from the latent space to reconstructed data $p_\theta(x|z)$
- Training objective: Evidence Lower Bound (ELBO)

$$\mathcal{L}(\theta, \phi; x) = E_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z)) \tag{92}$$

  where the first term is the reconstruction term and the second is a regularization term.

Generative Adversarial Networks (GAN) GANs learn to generate data through a two-player game:

- Generator G tries to produce data indistinguishable from real data
- Discriminator D tries to distinguish between real and generated data
- Training objective: Min-max game

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{93}$$

Key variants:

- DCGAN: Uses convolutional architectures
- Conditional GAN: Generates data conditioned on class labels
- Wasserstein GAN: Uses Wasserstein distance instead of JS divergence
- StyleGAN: Separates high-level attributes and stochastic variations

Differences between VAE and GAN

- VAEs are trained by maximizing a lower bound on data likelihood
- GANs are trained through adversarial training without explicit density estimation
- VAEs tend to produce blurrier samples but offer a more stable training process
- GANs can produce sharper, more realistic samples but suffer from mode collapse and training instability

4.8 Ensemble Methods

4.8.1 Motivation for Ensemble Methods

Ensemble methods combine multiple models to improve performance. Dietterich's "three reasons" for their effectiveness:

- Statistical: Reduces the risk of choosing a wrong classifier
- Computational: Helps escape local optima
- Representational: Expands the space of representable functions

4.8.2 Types of Ensembles

- Parallel ensembles: Base learners are independent
  - Voting: Select the prediction with the most votes
  - Bagging: Build models from different samples of the training set
- Sequential ensembles: Base learners depend on previous ones
  - Boosting: Focus on examples that previous models misclassified

4.8.3 Bagging (Bootstrap Aggregating)

Bagging creates diversity through bootstrapping:

1. Create k bootstrap samples (sampling with replacement)
2. Train a distinct classifier on each sample
3. Classify by majority vote (classification) or average (regression)

Bagging works best with unstable base learners and reduces variance at the expense of slightly increased bias.

4.8.4 Random Forests

Random forests extend bagging for decision trees:

1. Use k bootstrap replicates to train k different decision trees
2. At each node, pick a random subset of features
3. Aggregate predictions across all trees

4.8.5 Boosting and AdaBoost

AdaBoost (Adaptive Boosting) sequentially builds classifiers with increased focus on difficult examples:

1. Initialize a uniform distribution over training examples
2. For $t = 1$ to T:
   - Train a weak classifier h_t on the weighted training set
   - Calculate the weighted error ϵ_t
   - Set the classifier weight $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
   - Update example weights, increasing weights for misclassified examples
3. Final classifier: $H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x))$

AdaBoost minimizes an exponential loss function: $E = \sum_{i=1}^{n} e^{-y_i H(x_i)}$.

4.8.6 Stacking

Stacking combines an arbitrary set of base models using a meta-model:

1. Train multiple base models on the training set
2. Generate predictions from each base model
3. Train a meta-model on these predictions

Common meta-models include averaging, majority voting, linear regression, and logistic regression.

5 Unsupervised Learning

5.1 Clustering

5.1.1 The Clustering Problem

Clustering groups similar objects together without class labels. Formally, given:

- A set of instances $X = \{x_1, \ldots, x_n\}$
- A measure of similarity
- A desired number of clusters K

The goal is to find an assignment function $\gamma : X \to \{1, \ldots, K\}$ that optimizes some criterion.

5.1.2 Evaluation of Clustering

Clustering quality can be assessed through:

- Internal criteria: Based on the chosen similarity measure
  - Example: $V(X, \gamma) = \sum_{k=1}^{K} \sum_{i:\gamma(x_i)=k} ||x_i - c_k||^2$
- External criteria: Comparison to a ground truth
  - Purity: Ratio of dominant class elements to cluster size
  - Rand Index: Proportion of correctly grouped pairs
  - Adjusted Rand Index: Corrects for chance agreement

5.1.3 K-Means Clustering

K-Means optimizes the squared distance between examples and cluster centroids:

Algorithm 3 K-Means Algorithm

```
Generate K initial centroids (e.g., random examples)
repeat
    Assign each example to the closest centroid
    Recalculate centroids as means of assigned examples
until centroids stabilize
```

5.1.4 Hierarchical Clustering

Hierarchical clustering builds a tree-based taxonomy (dendrogram):

- Agglomerative (bottom-up): Start with singleton clusters and merge the closest pairs
- Divisive (top-down): Start with one cluster and recursively split

Inter-cluster distance measures:

- Single-link: Distance between closest points
- Complete-link: Distance between furthest points
- Centroid: Distance between centroids
- Average-link: Average distance between all pairs

5.1.5 Advanced Clustering Methods

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) A density-based clustering algorithm that groups points that are closely packed together:

- Core points: Points with at least minPts neighbors within distance ϵ
- Directly density-reachable: A point q is directly reachable from p if p is a core point and q is within distance ϵ from p
- Density-reachable: A point q is reachable from p if there is a path of directly reachable points from p to q
- Density-connected: Points p and q are connected if there exists a point o such that both p and q are density-reachable from o
- Clusters: Maximal sets of density-connected points
- Noise: Points not belonging to any cluster

Algorithm:

1. For each unvisited point p:
    - Mark p as visited
    - Find all points density-reachable from p
    - If p is a core point, form a new cluster
    - If p is not a core point, mark as noise

Advantages:

- No need to specify number of clusters
- Can find arbitrarily shaped clusters
- Can identify noise points

Spectral Clustering Leverages the spectrum (eigenvalues) of the similarity matrix of the data:

1. Construct a similarity graph G from the data (e.g., k-nearest neighbors or ϵ-neighborhood)
2. Compute the Laplacian matrix $L = D - W$ where W is the adjacency matrix and D is the degree matrix
3. Compute the first k eigenvectors of L (or normalized variants of L)
4. Form a matrix $U \in \mathbb{R}^{n \times k}$ containing these eigenvectors as columns
5. Apply k-means or another clustering algorithm on the rows of U

Advantages:

- Can detect complex cluster shapes
- Often outperforms traditional clustering when clusters have complicated structures
- Mathematically well-founded

Mean Shift An iterative, non-parametric clustering technique that seeks modes in a density function:

1. For each point x_i:
   - Compute the mean shift vector $m(x_i) = \frac{\sum_{x_j} K(x_j - x_i) x_j}{\sum_{x_j} K(x_j - x_i)} - x_i$
   - Update $x_i \leftarrow x_i + m(x_i)$
   - Repeat until convergence
2. Group points that converge to the same mode

Where K is a kernel function (e.g., Gaussian kernel).

5.2 Representation Learning

5.2.1 Introduction to Representation Learning

Representation learning aims to transform data into representations that make subsequent tasks easier. Good representations have properties like smoothness, multiple explanatory factors, hierarchical organization, and sparsity.

5.2.2 Principal Component Analysis (PCA)

PCA finds a linear transformation that projects data onto orthogonal axes of maximum variance:

1. Center the data by subtracting the mean
2. Compute the covariance matrix $S_X = \frac{1}{n} X X^\top$
3. Find the eigenvectors and eigenvalues of S_X
4. Project data onto the eigenvectors with largest eigenvalues

PCA minimizes the squared reconstruction error while maximizing variance along the principal components.

5.2.3 Autoencoders

Autoencoders are neural networks trained to reconstruct their input:

- Encoder: Maps input to a code (typically smaller dimension)
- Decoder: Reconstructs input from the code
- Bottleneck layer: Forces the network to learn a compact representation

The loss function is typically the reconstruction error:

$$L(x, \tilde{x}) = ||x - \tilde{x}||_2^2 \tag{94}$$

Variants include:

- Sparse autoencoders: Add regularization to encourage sparse representations
- Denoising autoencoders: Train to reconstruct clean inputs from corrupted ones
- Variational autoencoders: Learn a probabilistic latent space

5.2.4 Deep Learning Architectures

Convolutional Neural Networks (CNN) in Detail CNNs are specialized neural networks designed for grid-like data (e.g., images) that use convolution operations instead of general matrix multiplication. The key components are:

- **Convolution layers:** Apply learnable filters to input data
  - Input: $I \in \mathbb{R}^{W \times H \times C}$ (Width, Height, Channels)
  - Filters: $F \in \mathbb{R}^{K \times K \times C \times N}$ ($K \times K$ kernel size, C input channels, N filters)
  - Output: $O \in \mathbb{R}^{W' \times H' \times N}$ where $W' = \frac{W-K+2P}{S} + 1$ and $H' = \frac{H-K+2P}{S} + 1$
  - P is padding, S is stride
- **Padding:** Adding zeros around the input
  - Valid padding (no padding): Output size smaller than input
  - Same padding: Output size equals input size (typically $P = \frac{K-1}{2}$ for odd kernel sizes)
- **Stride:** Step size when applying the filter
  - Stride = 1: Check every position
  - Stride > 1: Skip positions, reducing spatial dimensions
- **Pooling layers:** Reduce spatial dimensions and create invariance
  - Max pooling: Take maximum value in each window
  - Average pooling: Take average value in each window
  - Global pooling: Reduce entire feature map to a single value
- **Fully connected layers:** Combine features for classification

CNNs progressively learn more abstract features:

- Early layers: Edges and simple patterns
- Middle layers: Textures and parts
- Deep layers: Objects and concepts

Recurrent Neural Networks (RNN) RNNs are designed to work with sequential data by maintaining a hidden state:

$$h_t = f(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \tag{95}$$

$$y_t = g(W_{yh}h_t + b_y) \tag{96}$$

where h_t is the hidden state at time t, x_t is the input, y_t is the output, and W and b are weights and biases.

Standard RNNs suffer from vanishing and exploding gradient problems during training.

Long Short-Term Memory (LSTM) LSTM networks address the vanishing gradient problem through gating mechanisms:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad \text{(forget gate)} \tag{97}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad \text{(input gate)} \tag{98}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad \text{(candidate cell state)} \tag{99}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad \text{(cell state update)} \tag{100}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad \text{(output gate)} \tag{101}$$

$$h_t = o_t * \tanh(C_t) \quad \text{(hidden state)} \tag{102}$$

Gated Recurrent Unit (GRU) GRU is a simplified version of LSTM with fewer parameters:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \quad \text{(update gate)} \tag{103}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \quad \text{(reset gate)} \tag{104}$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t * h_{t-1}, x_t] + b_h) \quad \text{(candidate hidden state)} \tag{105}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \quad \text{(hidden state update)} \tag{106}$$

Transformers Transformers rely entirely on attention mechanisms instead of recurrence:

- **Self-Attention:** Allow each position to attend to all positions in the sequence

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{107}$$

  where Q (queries), K (keys), and V (values) are projections of the input.

- **Multi-Head Attention:** Apply attention multiple times in parallel
- **Position Encoding:** Add positional information since there's no inherent order
- **Feed-Forward Networks:** Apply point-wise fully connected layers
- **Layer Normalization and Residual Connections:** Stabilize training

Transformer architecture consists of an encoder and a decoder stack, each with multiple identical layers.

5.2.5 Word Embeddings - Word2Vec

Word2Vec learns vector representations of words by predicting:

- CBOW (Continuous Bag of Words): Context words predict target word
- Skip-gram: Target word predicts context words

Word embeddings capture semantic and syntactic relationships, allowing operations like "King - Man + Woman ≈ Queen".

5.2.6 Knowledge Graph Embeddings

Knowledge graph embeddings represent entities and relations as vectors:

- TransE: Represents a relation as a translation in the embedding space
- The goal is that $h + r \approx t$ for valid triples (h, r, t)

These embeddings enable link prediction and knowledge base completion.

6 Recommender Systems

6.1 Introduction to Recommender Systems

6.1.1 Definition and Purpose

Recommender Systems (RSs) are software tools that provide suggestions for items that are likely of interest to users. They help address the "paradox of choice" where too many options can overwhelm users.

6.1.2 Taxonomy of Recommender Systems

- Non-personalized: Same recommendations for all users
  - Most popular items
  - Highest rated items
- Personalized
  - Content-based: Recommend similar items to those the user liked
  - Collaborative filtering: Use patterns in user-item interactions
  - Context-aware: Consider contextual information (time, location, etc.)
  - Hybrid: Combine multiple approaches

6.1.3 Types of Feedback

- Explicit feedback: Direct ratings or reviews (reliable but hard to collect)
- Implicit feedback: Behavioral signals like clicks or purchases (easier to collect but noisy)

6.1.4 The Rating Matrix

The core of many recommender systems is the rating matrix:

- Users are arranged in rows, items in columns
- Entry r_{ui} is the rating of user u for item i
- Typically very sparse ($< 0.01\%$ density)
- User activity and item popularity follow long-tail distributions

6.1.5 Recommendation Tasks

- Rating prediction: Estimate missing values in the rating matrix
- Top-N recommendation: Suggest N items the user will like most

6.1.6 Evaluation of Recommender Systems

Evaluation methods:

- Offline: Use historical data split into training and test sets
  - Rating prediction: MAE, RMSE
  - Top-N: Precision, Recall, AUC, DCG, MRR
  - Beyond accuracy: Diversity, Novelty
- Online: Involve actual users
  - Direct feedback through questionnaires
  - A/B testing comparing systems

6.2 Collaborative Filtering

6.2.1 Approaches to Collaborative Filtering

Collaborative Filtering can be categorized as:

- Similarity-wise
  - Item-based: Recommendations based on item similarity
  - User-based: Recommendations based on user similarity
- Algorithm-wise
  - Memory-based: Use rating data directly to compute similarities
  - Model-based: Develop models to predict ratings

6.2.2 Notation

- U is the set of users, with $|U| = n$
- I is the set of items, with $|I| = m$
- R is the set of ratings/interactions
- $R \in \mathbb{R}^{n \times m}$ is the rating matrix
- I_u is the set of items rated by user u
- U_i is the set of users who rated item i
- I_{uv} is the set of items rated by both users u and v
- U_{ij} is the set of users who rated both items i and j

6.2.3 Computing Similarity

For implicit feedback, cosine similarity is commonly used:

- User-based: $s_{uv} = \cos(r_u, r_v) = \frac{r_u r_v^\top}{||r_u|| \cdot ||r_v||} = \frac{|I_u \cap I_v|}{\sqrt{|I_u| \cdot |I_v|}}$
- Item-based: $s_{ij} = \cos(r_i, r_j) = \frac{r_i^\top r_j}{||r_i|| \cdot ||r_j||} = \frac{|U_i \cap U_j|}{\sqrt{|U_i| \cdot |U_j|}}$

For explicit feedback, Pearson correlation is often used:

- User-based: $s_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)(r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2 \cdot \sum_{i \in I_{uv}} (r_{vi} - \mu_v)^2}}$
- Item-based: $s_{ij} = \frac{\sum_{u \in U_{ij}} (r_{ui} - \mu_i)(r_{uj} - \mu_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \mu_i)^2 \cdot \sum_{u \in U_{ij}} (r_{uj} - \mu_j)^2}}$

Adjusted cosine similarity centers ratings on their user mean rather than item mean.

6.2.4 K-Nearest Neighbors

K-NN in collaborative filtering:

- User-based: $N_i^k(u) \equiv \{v \in U_i \mid |\{v' \in U_i \mid s_{uv'} > s_{uv}\}| < k\}$
- Item-based: $N_u^k(i) \equiv \{j \in I_u \mid |\{j' \in I_u \mid s_{ij'} > s_{ij}\}| < k\}$

Recommendations are computed as weighted averages:

- Explicit feedback:
  - User-based: $\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} s_{uv} \cdot r_{vi}}{\sum_{v \in N_i^k(u)} s_{uv}}$
  - Item-based: $\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} s_{ij} \cdot r_{uj}}{\sum_{j \in N_u^k(i)} s_{ij}}$
- Implicit feedback:
  - User-based: $\hat{r}_{ui} = \frac{1}{k} \sum_{v \in N_i^k(u)} s_{uv}$
  - Item-based: $\hat{r}_{ui} = \frac{1}{k} \sum_{j \in N_u^k(i)} s_{ij}$

6.2.5 Matrix Factorization

Matrix factorization decomposes the rating matrix into lower-dimensional factors:

- Each item i is associated with a vector $q_i \in \mathbb{R}^k$
- Each user u is associated with a vector $p_u \in \mathbb{R}^k$
- The dot product $p_u q_i^\top$ captures their interaction

The objective function is:

$$L(P, Q) = \min_{P,Q} \sum_{(u,i) \in Tr} \left(r_{ui} - p_u q_i^\top\right)^2 + \lambda(||p_u||^2 + ||q_i||^2) \tag{108}$$

This can be optimized using:

- Stochastic Gradient Descent (SGD)

$$p_u \leftarrow p_u + 2\eta(\epsilon_{ui} q_i - \lambda p_u) \tag{109}$$
$$q_i \leftarrow q_i + 2\eta(\epsilon_{ui} p_u - \lambda q_i) \tag{110}$$

- Alternating Least Squares (ALS)

$$p_u = r_u Q(Q^\top Q + \lambda I)^{-1} \tag{111}$$

$$q_i = r_i^\top P(P^\top P + \lambda I)^{-1} \tag{112}$$

7 Reinforcement Learning

7.1 Introduction to Reinforcement Learning

Reinforcement Learning (RL) is a paradigm where an agent learns to make decisions by taking actions in an environment to maximize cumulative reward. Unlike supervised learning, there are no labeled training examples; instead, the agent learns from feedback received through interaction with the environment.

7.1.1 Components of RL

- Agent: The decision-maker or learner
- Environment: Everything the agent interacts with
- State (s): Complete description of the environment
- Action (a): Choices available to the agent
- Reward (r): Scalar feedback signal indicating how well the agent is doing
- Policy (π): Strategy the agent follows to determine actions
- Value function (V or Q): Prediction of future rewards
- Model: Agent's representation of the environment

7.1.2 Markov Decision Processes (MDPs)

MDPs provide a formal framework for RL problems with the Markov property (future states depend only on the current state and action, not on past history):

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Transition probability function $P(s'|s,a)$
- Reward function $R(s,a,s')$
- Discount factor $\gamma \in [0,1]$

The goal is to find a policy π that maximizes the expected cumulative discounted reward:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{113}$$

7.2 Value Functions and Bellman Equations

7.2.1 Value Functions

- State-value function: Expected return starting from state s and following policy π

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] \tag{114}$$

- Action-value function: Expected return starting from state s, taking action a, and then following policy π

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \tag{115}$$

7.2.2 Bellman Equations

Recursive relationships for value functions:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')] \tag{116}$$

$$Q^\pi(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s',a')] \tag{117}$$

Bellman optimality equations (for the optimal policy π^*):

$$V^*(s) = \max_a \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')] \tag{118}$$

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')] \tag{119}$$

7.3 Dynamic Programming Methods

7.3.1 Policy Evaluation

Compute V^π for a given policy π:

1. Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$
2. Repeat until convergence:

$$V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V(s')] \tag{120}$$

7.3.2 Policy Improvement

Improve the policy using the value function:

$$\pi'(s) = \arg\max_a \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')] \tag{121}$$

7.3.3 Policy Iteration

Alternating policy evaluation and policy improvement:

1. Initialize π arbitrarily
2. Repeat until convergence:
   - Policy Evaluation: Compute V^π
   - Policy Improvement: Compute π' and update $\pi \leftarrow \pi'$

7.3.4 Value Iteration

Combine policy evaluation and improvement into a single step:

1. Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$
2. Repeat until convergence:

$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V(s')] \tag{122}$$

7.4 Model-Free Methods

7.4.1 Monte Carlo Methods

Learn directly from episodes of experience without knowing the MDP:

- First-visit MC: Average returns from first visit to each state in episodes
- Every-visit MC: Average returns from all visits to each state in episodes

MC Control:

1. Initialize $Q(s, a)$ arbitrarily
2. Repeat for each episode:
   - Generate an episode following π
   - For each state-action pair (s, a) in the episode:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[G_t - Q(s, a)] \tag{123}$$

   - Improve policy: $\pi(s) \leftarrow \arg\max_a Q(s, a)$

7.4.2 Temporal Difference Learning

Learn from incomplete episodes by bootstrapping:

- TD(0) for prediction:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{124}$$

- SARSA (on-policy TD control):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{125}$$

- Q-learning (off-policy TD control):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{126}$$

7.5 Function Approximation in RL

7.5.1 Linear Function Approximation

Represent value functions as linear combinations of features:

$$\hat{V}(s, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s) \tag{127}$$

$$\hat{Q}(s, a, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s, a) \tag{128}$$

where $\mathbf{x}(s)$ or $\mathbf{x}(s, a)$ are feature vectors and $\mathbf{w}$ are weights.

7.5.2 Deep Reinforcement Learning

Use neural networks to approximate value functions or policies:

- Deep Q-Networks (DQN):
  - Use neural network to approximate $Q(s, a)$
  - Experience replay to break correlations in the training data
  - Target network to stabilize learning

- Policy Gradient Methods:
  - Directly optimize the policy without value functions
  - REINFORCE algorithm updates policy parameters θ to maximize expected return:
$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) G_t] \tag{129}$$
- Actor-Critic Methods:
  - Combine policy gradient (actor) with value function approximation (critic)
  - Actor updates policy parameters θ based on the advantage function:
$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) A(s,a)] \tag{130}$$
  - Critic updates value function parameters w to minimize TD error

8 Ethics and Interpretability in Machine Learning

8.1 Model Interpretability

8.1.1 Importance of Interpretability

Interpretable ML is critical for:

- Trust: Users need to understand why models make certain decisions
- Debugging: Identifying and fixing errors in models
- Regulatory compliance: Many domains require explainable decisions
- Scientific discovery: Extracting insights from complex relationships

8.1.2 Interpretable Models

Some models are inherently more interpretable:

- Linear/logistic regression: Coefficients indicate feature importance
- Decision trees: Decision paths represent logical rules
- Rule-based systems: Explicit if-then rules
- Generalized additive models (GAMs): Show contribution of each feature independently

8.1.3 Post-hoc Explanation Methods

Techniques to explain black-box models after training:

- Feature importance:
  - Permutation importance: Randomly shuffle features and measure impact on performance
  - Partial dependence plots: Show relationship between features and predictions
- Local explanations:
  - LIME (Local Interpretable Model-agnostic Explanations): Approximates complex models locally with simpler models

  - SHAP (SHapley Additive exPlanations): Uses game theory to assign importance values to features

- Global explanations:
  - Global surrogate models: Train interpretable models to mimic black-box models
  - Rule extraction: Derive rules from model behavior

8.2 Fairness and Bias

8.2.1 Sources of Bias

Biases can enter ML systems through various channels:

- Data collection bias: Unrepresentative sampling
- Prejudice bias: Historical discrimination reflected in data
- Measurement bias: Different accuracy across groups
- Algorithmic bias: Model design choices amplify bias

8.2.2 Fairness Metrics

Quantitative measures of fairness:

- Group fairness:
  - Demographic parity: $P(\hat{Y} = 1|A = a) = P(\hat{Y} = 1|A = b)$
  - Equalized odds: $P(\hat{Y} = 1|Y = y, A = a) = P(\hat{Y} = 1|Y = y, A = b)$
  - Equal opportunity: $P(\hat{Y} = 1|Y = 1, A = a) = P(\hat{Y} = 1|Y = 1, A = b)$
- Individual fairness: Similar individuals should receive similar predictions

8.2.3 Bias Mitigation Strategies

Approaches to reduce bias in ML systems:

- Pre-processing: Modify training data to remove bias
- In-processing: Incorporate fairness constraints during model training
- Post-processing: Adjust model outputs to ensure fairness

8.3 Privacy and Security

8.3.1 Privacy Concerns

ML systems can compromise privacy through:

- Membership inference attacks: Determining if data was used in training
- Model inversion attacks: Reconstructing training data
- Attribute inference: Inferring sensitive attributes

8.3.2 Privacy-Preserving ML

Techniques to protect privacy:

- Differential privacy: Add noise to ensure individual data cannot be identified
- Federated learning: Train across devices without sharing raw data
- Secure multi-party computation: Compute on encrypted data
- Homomorphic encryption: Perform computations on encrypted data

9 Conclusion

Machine Learning has evolved into a rich field with diverse algorithms and approaches. From supervised learning methods like decision trees, neural networks, and SVMs to unsupervised techniques like clustering and representation learning, the field offers powerful tools for pattern recognition, prediction, and recommendation.

The theoretical foundations of learning, including PAC learning and VC theory, provide a rigorous understanding of generalization and model complexity. Practical issues like preprocessing, feature selection, and model evaluation are crucial for successful application.

Advanced topics such as deep learning architectures, reinforcement learning, and generative models have expanded the capabilities of machine learning systems. Meanwhile, concerns about interpretability, fairness, and privacy have led to new research directions aimed at making machine learning more trustworthy and responsible.

As machine learning continues to advance, the integration of these diverse approaches and considerations is pushing the boundaries of what's possible, enabling systems that can learn from data in increasingly sophisticated and responsible ways.