Beyond Decision Trees - Bayesian, Nearest Neighbor, and Linear Classifiers

Lecture Title

Duration:
1 Hour

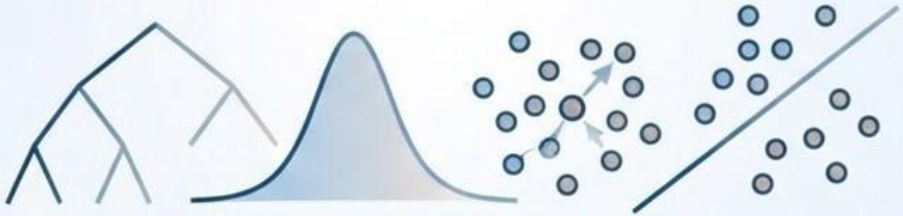
Target Audience:
Data Science/Computer Science students

Prerequisites:

- Basic probability
- Understanding of classification concepts
- Linear algebra fundamentals

Beyond the Decision Tree

Probabilistic, Lazy, and Linear Models



Slide 1: Introduction to Advanced Classification

Part 1: Introduction & Learning Objectives

The Opening Hook

When Rules Are Not Enough

The Limitation

Decision trees give us highly interpretable rules, but they struggle with nuance.

The Questions

- What if we want to explicitly incorporate prior knowledge into our model?
- What if we need to quantify our uncertainty with exact probabilities?
- What if we want to classify a new case simply based on its similarity to known examples?

The Solution

Today we move beyond trees to explore three fundamentally different machine learning approaches:

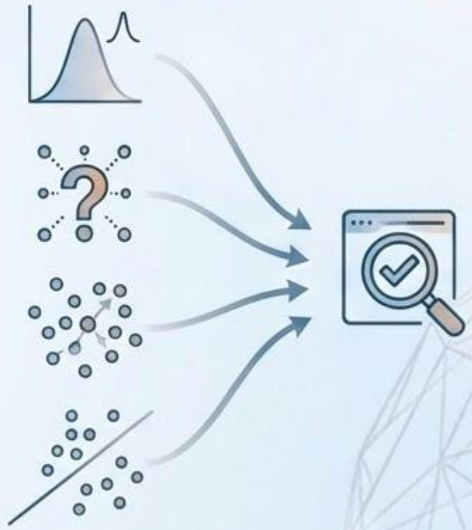
- **Probabilistic Methods** that quantify uncertainty.
- **Lazy Learners** that defer computation until the last moment.
- **Linear Models** that draw optimal boundaries through space.

Learning Objectives

What We Will Cover Today

By the end of this lecture, you will be able to:

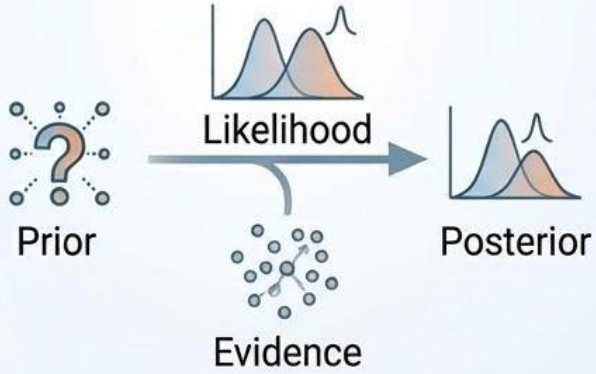
- **Apply Bayes' Theorem**: Calculate conditional probabilities for probabilistic classification.
- **Implement Naïve Bayes**: Understand the mechanics and assumptions of Naïve Bayesian classifiers.
- **Master k-Nearest Neighbor (k-NN):** Execute similarity-based classification and understand its key variants.
- **Understand Linear Classifiers:** Grasp the math and function of models like the Perceptron and Logistic Regression.
- **Evaluate Paradigms**: Compare the distinct strengths, weaknesses, and ideal use cases of these different classification families.



Part 2: Bayes Classification Methods

Quantifying Uncertainty

Learning from Evidence Using Bayes' Theorem



The Fundamental Concept

Updating Beliefs with Evidence

The Concept

Bayes' theorem describes the probability of an event based on prior knowledge of conditions that might be related to the event.

Intuitive Understanding

Think of it as a mathematical learning process:

Start with an initial belief (Prior).

Observe new evidence (Data).

Update your belief (Posterior).

The Mathematical Formulation

The Core Equation

$$P(H|X) = \frac{P(X|H) \times P(H)}{P(X)}$$

The Terms

$\mathcal{P}(H|X)$: The Posterior Probability of hypothesis H given evidence X.

$\mathcal{P}(X|H)$: The Likelihood of evidence X given that hypothesis H is true.

$\mathcal{P}(H)$: The Prior Probability of hypothesis H (before seeing any evidence).

$\mathcal{P}(X)$: The Marginal Probability of evidence X.

Simple Example: Medical Testing

The Base Rate Fallacy

The Setup

- **1%** (Prior = 0.01)
- **95%** Accurate (Sensitivity = 0.95)
- **5%** False Positive Rate (Specificity = 0.95)

The Question

If a person tests positive, what is the probability they actually have the disease?

The Math

$$P(\text{disease}|\text{positive}) = \frac{0.95 \times 0.01}{(0.95 \times 0.01) + (0.05 \times 0.99)}$$

$$= \frac{0.0095}{0.059} = \mathbf{0.161}$$

The Result

Surprisingly, only **16.1%**! Assuming the person is definitely sick ignores the low prior probability (1%), a cognitive trap known as the "Base Rate Fallacy."

Why This Matters for Classification

The MAP Decision Rule

The Classification Goal

We want to find the most probable class C_i given a specific data instance $\boldsymbol{X}$.

The Optimization

Since the denominator $P(\boldsymbol{X})$ is constant for all classes being compared, we can safely ignore it. We only need to maximize the numerator: $P(X|C_i) \times P(C_i)$.

Maximum A Posteriori (MAP) Decision Rule

$$C_{\text{MAP}} = \underset{C_i}{\operatorname{argmax}}\, P(C_i|X)$$
$$= \underset{C_i}{\operatorname{argmax}}\, P(X|C_i) \times P(C_i)$$

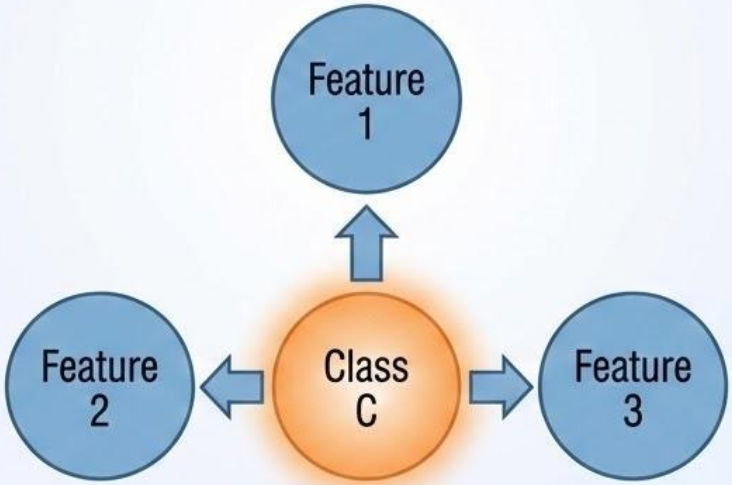
The Translation

"Classify this new data point into whichever category has the highest combined likelihood and prior probability."

Part 2: Naïve Bayesian Classification

The Power of the Independence Assumption

Fast, Scalable, and Probabilistic Predictions



The “Naïve” Assumption & Algorithm

Simplifying Complex Probabilities

The Naïve Assumption

The algorithm assumes that all predictor attributes are conditionally independent given the class label.

Why is it “Naïve”?

In real life, attributes are almost always correlated (e.g., “Owning a House” and “Credit Score” are linked). Despite ignoring this reality, the classifier works surprisingly well!

The Mathematics

Because of this assumption, we can simply multiply the probabilities together:

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

The Algorithm

Training Phase: Estimate the prior probabilities $P(C_i)$ and the conditional probabilities $P(x_k|C_i)$ from the historical dataset.

Prediction Phase: For a new instance X, compute the probability for each class and assign the highest one:

$$P(C_i|X) \propto P(C_i) \times \prod_{k=1}^{n} P(x_k|C_i)$$

Worked Example: Loan Application (Setup)

Predicting a New Applicant

Historical Training Data:

ID	Has Job	Owns House	Credit Score	Loan Status
1	Yes	Yes	Good	Approved
2	Yes	No	Good	Approved
3	No	Yes	Fair	Approved
4	Yes	No	Poor	Rejected
5	No	Yes	Poor	Rejected
6	No	No	Fair	Rejected
7	Yes	No	Fair	Approved
8	No	No	Good	Rejected

New Applicant (X):

{Has Job = Yes, Owns House = No, Credit Score = Fair}

Step 1: Calculate Prior Probabilities

$P(\text{Approved}) = 4/8 = 0.5$

$P(\text{Rejected}) = 4/8 = 0.5$

Worked Example (The Calculation)

Computing the Posteriors

Step 2: Calculate Conditional Probabilities

For Approved:
P(Has Job = Yes | Approved) = 3/4 = 0.75
P(Owns House = No | Approved) = 2/4 = 0.5
P(Credit Score = Fair | Approved) = 1/4 = 0.25

For Rejected:
P(Has Job = Yes | Rejected) = 1/4 = 0.25
P(Owns House = No | Rejected) = 2/4 = 0.5
P(Credit Score = Fair | Rejected) = 1/4 = 0.25

Step 3 & 4: Calculate Posteriors and Normalize

Approved:
$P(\text{Approved}|X) \propto 0.5 \times 0.75 \times 0.5 \times 0.25 = 0.046875$

Rejected:
$P(\text{Rejected}|X) \propto 0.5 \times 0.25 \times 0.5 \times 0.25 = 0.015625$

Normalize:
0.046875 / (0.046875 + 0.015625) = 0.75

Prediction: Loan APPROVED with 75% confidence.

Handling Zero Probabilities

The Laplace Smoothing Solution

The Problem:

- In Naïve Bayes, we multiply probabilities. If a particular attribute value never appears in the training data for a specific class, its probability becomes zero ($P(x_k|C_i) = 0$).

Multiplying by zero wipes out all other evidence, making the entire posterior probability zero!

The Solution:

Add small pseudocounts to the data, a technique known as **Laplace Smoothing**.

$$P(x_k|C_i) = \frac{\text{count}(x_k, C_i) + \alpha}{\text{count}(C_i) + \alpha \times v}$$

(Where v is the number of possible values for the attribute, and α is the smoothing parameter, typically set to 1).

Advantages & Disadvantages

Evaluating Naïve Bayes

Advantages

- **Simple & Fast:** Linear time complexity makes it highly scalable.
- **Handles Missing Values:** Can simply ignore missing attributes during calculation.
- **Incremental Learning:** Can easily update probabilities as new data arrives without retraining from scratch.
- **Small Data Champion:** Less prone to overfitting on small datasets.

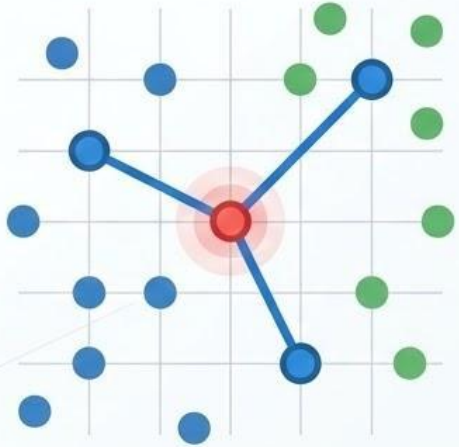
Disadvantages

- **Naïve Assumption:** The independence assumption is frequently violated in real-world data.
- **Zero Frequency Problem:** Requires smoothing to prevent complete failure on unseen categorical values.
- **Double-Counting:** Highly sensitive to correlated features (e.g., it will "double-count" the evidence if two features mean the exact same thing).

Part 3: Lazy Learners

Learning by Proximity

The k-Nearest-Neighbor (k-NN) Algorithm



Context: Part 3

The Philosophy & The Algorithm

"Tell me who your neighbors are, and I'll tell you who you are."

The Key Characteristic

k-NN is a "lazy learner." It does not build a generalized mathematical model during training. It simply stores the training data and defers all actual computation until it is asked to make a prediction.

The k-NN Algorithm

Training Phase: Store all historical/training instances in memory (zero actual "learning" happens here).

Prediction Phase: When given a new, unknown instance (x_q):

1. Compute the distance between the new instance and all stored instances.
2. Select the *k* closest instances (the "nearest neighbors").
3. **Vote:** Assign the most common class among the neighbors (for classification) or calculate the average (for regression).

Distance Measures

How Do We Mathematically Define 'Near'?

To find the closest neighbors, the algorithm relies on specific distance formulas to measure the space between data points:

Euclidean Distance

The standard "straight-line" distance between two points.

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (x_{ir} - x_{jr})^2}$$

Manhattan Distance

The "city block" distance, calculated along axes at right angles.

$$d(x_i, x_j) = \sum_{r=1}^{n} |x_{ir} - x_{jr}|$$

Minkowski Distance

$\sqrt[p]{p^p}$ A generalized mathematical formula containing a parameter p (where $p = 1$ is Manhattan and $p = 2$ is Euclidean).

Hamming Distance

0100100
0101100
0100110
0101010

Used specifically to calculate the distance between categorical (non-numeric) attributes.

Worked Example: *k*-NN Setup (*k*=3)

Calculating the Distances

Let's classify a new point, P (4, 4), based on 5 historical data points using Euclidean distance.

Training Data:

- A (2, 3) → Class Red
- B (3, 4) → Class Red
- C (5, 6) → Class Blue
- D (6, 5) → Class Blue
- E (4, 5) → Class Blue

Step 1: Calculate Distances from P (4, 4) to all points

$$d(P, A) = \sqrt{(4-2)^2 + (4-3)^2} = \sqrt{4+1} = \sqrt{5} \approx 2.24$$

$$d(P, B) = \sqrt{(4-3)^2 + (4-4)^2} = \sqrt{1+0} = 1$$

$$d(P, C) = \sqrt{(4-5)^2 + (4-6)^2} = \sqrt{1+4} = \sqrt{5} \approx 2.24$$

$$d(P, D) = \sqrt{(4-6)^2 + (4-5)^2} = \sqrt{4+1} = \sqrt{5} \approx 2.24$$

$$d(P, E) = \sqrt{(4-4)^2 + (4-5)^2} = \sqrt{0+1} = 1$$

Worked Example: Voting

Selecting the Neighbors and Making the Call

Step 2: Find the 3 Nearest Neighbors

- **Point B**: Distance = 1 (Class Red)
- **Point E**: Distance = 1 (Class Blue)
- **Tie**: Points A, C, and D are all tied at a distance of 2.24.

Step 3: The Vote

To get our 3rd neighbor, we break the tie (e.g., picking the first one evaluated, Point A).

Our 3 neighbors are **B** (Red), **E** (Blue), and **A** (Red).

Red Votes: 2 **Blue** Votes: 1

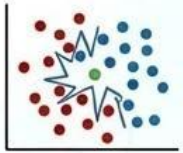
Final Prediction: Point P is classified as **RED**.

Choosing 'k' and Distance Weighting

Fine-Tuning the Algorithm

The Impact of 'k':

- **Small k** (e.g., k=1): Highly sensitive to noise and outliers; creates jagged, complex decision boundaries.



- **Large k**: Creates a smoother boundary, but risks losing nuanced, local patterns.

Rule of Thumb:

A common starting point is setting $k \approx \sqrt{n}$ (where n is the total number of training instances).

Best practice is to use Cross-Validation to test multiple values and pick the best performer.

Distance Weighting:

Instead of a simple democratic vote, we can give closer neighbors more voting power by weighting them by their inverse distance:

$$w_i = \frac{1}{d(x_q, x_i)^2}$$

Advantages & Disadvantages

Evaluating k-NN

Advantages

- **Simple & Intuitive**: Incredibly easy to understand, explain, and implement from scratch.
- **Instant 'Training'**: Zero wait time to build the model; you just dump the data into memory.
- **Highly Adaptable**: New data can be added on the fly without needing to retrain a model.
- **Non-Linear**: Easily models highly complex, irregular decision boundaries.

Disadvantages

- **Slow Prediction**: Unlike trees or Bayes, it must compute complex distances to every single historical instance at prediction time.
- **Memory Intensive**: Requires storing the entire historical dataset in RAM.
- **Curse of Dimensionality**: Distance calculations are easily skewed by irrelevant attributes.
- **Requires Scaling**: Features with large ranges (e.g., Income) will completely dominate smaller features (e.g., Age) unless mathematically normalized first.

Part 3: Case-Based Reasoning

Learning from Experience

Solving New Problems with Past Solutions

Section 6.4.2: Extending the philosophy of Lazy Learners into complex, real-world problem-solving.

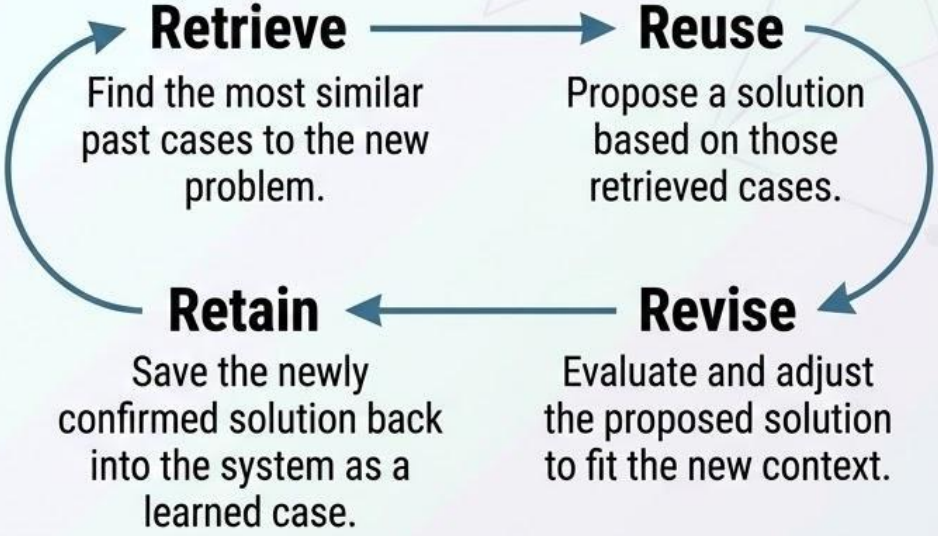
What is Case-Based Reasoning?

The ‘4R’ Cycle

The Definition

Case-based reasoning (CBR) is a problem-solving paradigm that uses past experiences (cases) to solve new problems. Think of it as an advanced extension of k-NN that uses richer representations and actually adapts its answers.

The CBR Cycle



The Components of CBR

Under the Hood of the System

Case Representation

Unlike simple rows of numbers, each CBR case contains rich structure:

- **Problem description** (the features/symptoms).
- **Solution** (the class or action taken).
- **Outcome** (feedback on whether the solution actually worked).

Retrieval & Reuse

Uses similarity metrics (like k-NN) combined with domain-specific knowledge to find matches.

It then **Copies**, **Adapts**, or **Composes** (combines) solutions to fit the current need.

Revise & Retain

The system tests the adapted solution (via simulation or expert manual correction).

If **successful**, the system **Retains** it, organically growing its knowledge base.

Worked Example: Tech Support

CBR in Action at the Help Desk

The Historical Case Base

Case 1:
Problem: Printer won't print
Context: Windows 10
Solution: Restart spooler

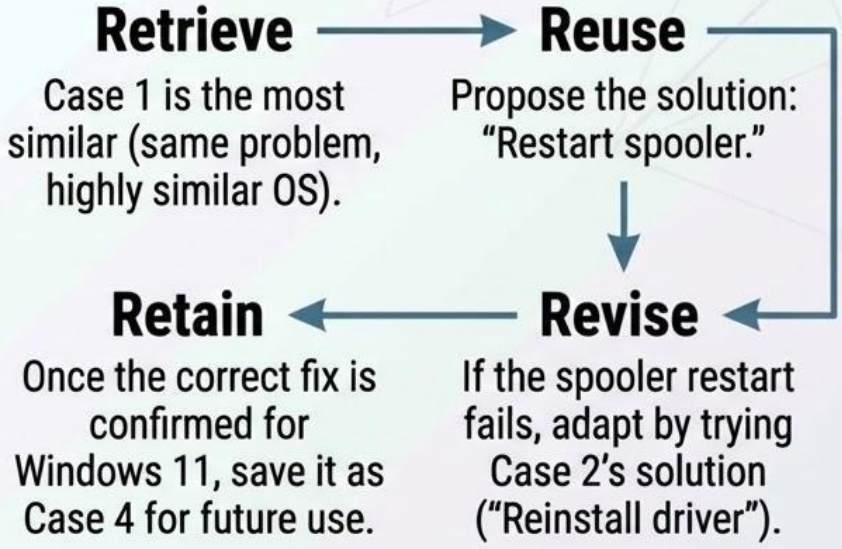
Case 2:
Problem: Printer won't print
Context: Mac OS
Solution: Reinstall driver

Case 3:
Problem: Slow printing
Context: Windows 10
Solution: Clear queue

The New Problem

Problem: Printer won't print
Context: Windows 11

CBR Cycle Application



CBR vs. k-NN

Evolving Beyond Simple Proximity

Aspect	k-Nearest-Neighbor (k-NN)	Case-Based Reasoning (CBR)
Case Representation	Simple feature vectors (numbers).	Rich structure, context, and outcomes.
Similarity	Generic mathematical distance metrics.	Domain-specific, contextual measures.
Adaptation	None (just relies on a simple vote).	Sophisticated modification and adaptation.
Learning	Just stores new instances as data points.	Actively retains successful adaptations.
Knowledge	Purely data-driven.	Can easily incorporate human/domain knowledge.

Applications of CBR

Where is this Paradigm Used?

Because CBR mimics human learning, it excels in complex, nuanced fields:

Medical Diagnosis

Matching a new patient's complex array of symptoms to similar historical patient cases.

Legal Reasoning

Finding historical legal precedents to build arguments for new, similar court cases.

Customer Support

Using past successful help-desk tickets to instantly solve new user issues.

Design & Architecture

Adapting successful historical blueprints to meet new spatial requirements.

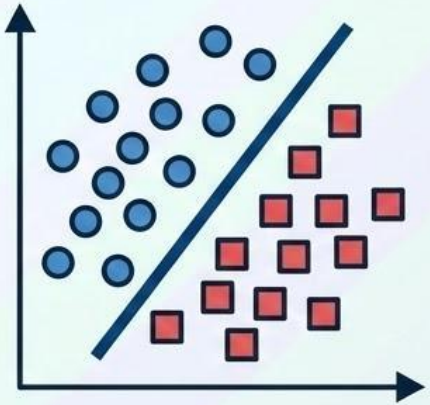
Logistics & Planning

Reusing successful supply-chain plans plans with minor modifications for new routes.

Part 4: Linear Classifiers

Drawing the Line

The Mathematical Foundation of Classification Boundaries



The Foundation - Linear Regression

Predicting Continuous Outcomes

Before we can classify data with linear models, we must understand the foundation: Linear Regression.

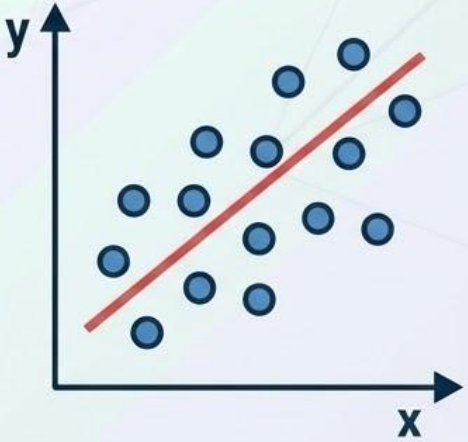
The Concept: Linear regression predicts a continuous output by calculating a linear combination of the input features.

Mathematical Form:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Vector Form:

$$y = \mathbf{w}^T \mathbf{x} + w_0$$



(Where **w** represents the learned weights, **x** represents the input features, and $\mathbf{w}_0$ is the bias or intercept).

The Learning Objective -

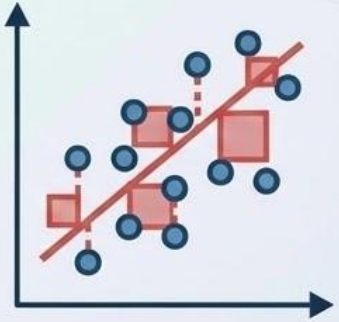
Finding the Line of Best Fit

The Goal: How does the model actually "learn"?

It must find the specific set of weights (w) that absolutely minimizes minimizes the prediction error across all historical data points.

The Least Squares Solution:
The standard mathematical approach is to minimize the sum of the squared differences between the actual known values (y_i) and the model's predicted values ($w^T \mathbf{x}_i$).

$$\min_{\mathbf{w}} \sum_{i=1}^{m} (y_i - w^T \mathbf{x}_i)^2$$



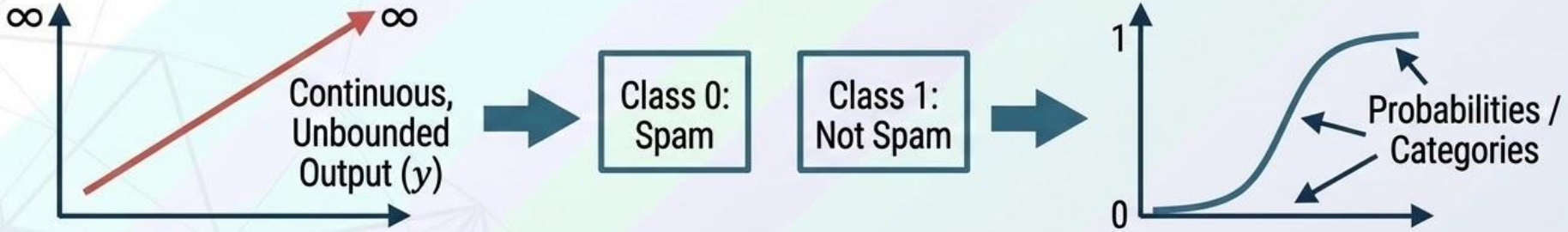
The Limitation for Classification -

Why Standard Regression Isn't Enough

The Problem: The output of linear regression (y) is a continuous, unbounded number (e.g., predicting a house price of $350,000 or a temperature of 72.5 degrees).

The Disconnect: Classification requires predicting discrete, distinct class labels (e.g., "Spam" or "Not Spam", "Approved" or "Rejected").

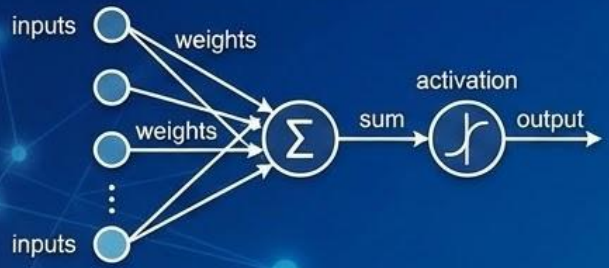
The Bridge to Classification: To use linear models for classification tasks, we must introduce mathematical functions that squash these continuous outputs into strict categories or probabilities.



Part 4: The Perceptron

THE FIRST NEURAL NETWORK

Turning Linear Regression into Classification



Section 6.5.2: Introduced by Frank Rosenblatt in 1958, this is the simplest neural network model and the foundational stepping stone for modern deep learning.

Architecture & Mathematical Form

How the Model is Built

The Architecture:

- **Inputs** (x): The features of your data.
- **Weights** (w): The learned importance of each feature.
- **Summation** (Σ): The linear combination of inputs and weights.
- **Activation Function** (f): The mechanism that turns the continuous sum into a discrete classification.

The Mathematical Form:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

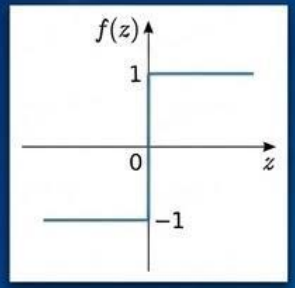
Activation & The Decision Boundary

Drawing the Line in the Sand

The Step Function:

For standard classification, the activation function f acts as a strict step function, transforming the sum into a hard label (e.g., $+1$ or -1).

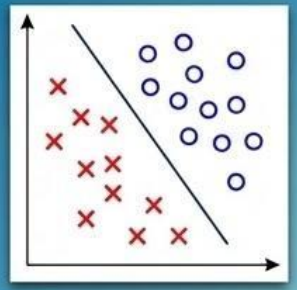
$$f(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{otherwise} \end{cases}$$



The Decision Boundary:

This mathematical combination creates a strict linear hyperplane that divides the data space into two distinct classes. The boundary itself exists where the equation equals zero:

$$w^T x + w_0 = 0$$



The Learning Algorithm

How the Perceptron Corrects its Mistakes

The perceptron learns by actively predicting and adjusting its weights when it gets the answer wrong.

1. **Initialize**: Start with random, very small weight values.
2. **Predict**: For each training example (x, y), compute the model's prediction: $\hat{y} = f(w^T x)$
3. **Update**: If the prediction is incorrect, update the weights using the following rule:

$$w_{new} = w_{old} + \eta \times (y - \hat{y}) \times x$$

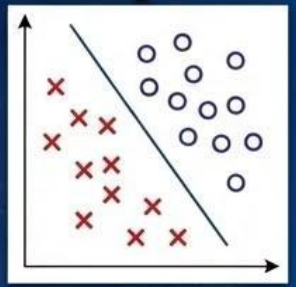
(Where η represents the "learning rate"—a small positive number that controls how drastically the weights change after an error).

Convergence & Limitations

The Fatal Flaw of Early AI

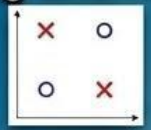
The Perceptron Convergence Theorem:

A mathematical guarantee stating that if a dataset is linearly separable (meaning a perfectly straight line can divide the classes), the perceptron will successfully converge to a solution in a finite number of steps.



The Limitations:

- **Strictly Linear:** It only works on perfectly linearly separable data.
- **The XOR Problem:** It cannot learn the famous XOR (Exclusive OR) function, a limitation that effectively killed neural network funding and research in the 1970s.



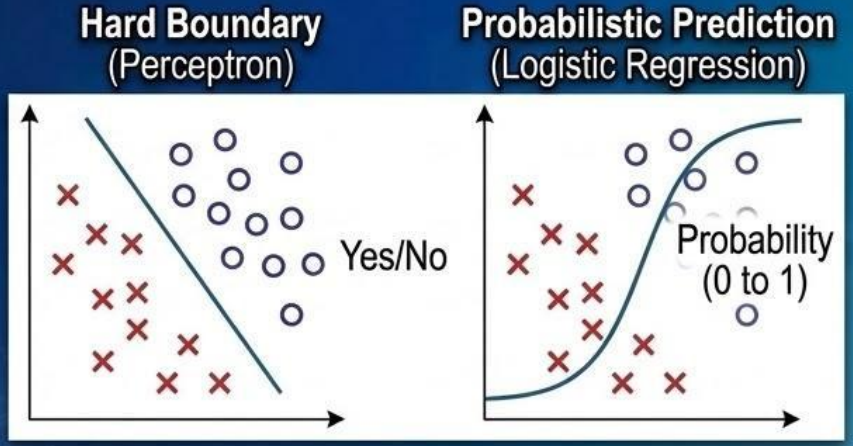
- **Hard Output:** It provides a strict, hard decision (e.g., "Yes" or "No") with absolutely no probability or confidence score attached.

Part 4: Logistic Regression

The Curve of Confidence

From Hard Boundaries to Probabilistic Predictions

Section 6.5.3: Solving the Perceptron's biggest limitation by predicting the probability of a class rather than just a hard "Yes" or "No".

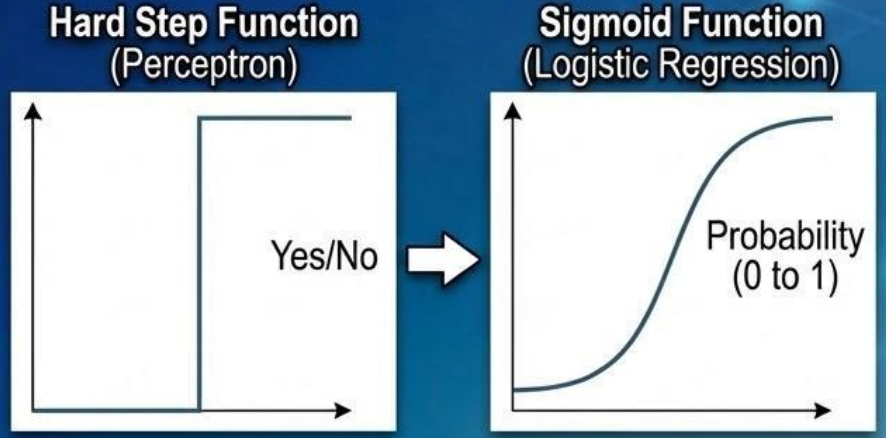


The Problem & The Sigmoid Solution

Softening the Step Function

The Problem with the Perceptron:

Hard decisions are brittle. A model that only outputs a strict 1 or 0 gives us no indication of how confident it is in that prediction. We want probabilities!



The Solution:

Take the exact same linear output used in the Perceptron, but pass it through a Sigmoid (Logistic) Function instead of a hard step function.

The Mathematical Form:

$$p(y=1|\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+w_0)}} = \sigma(\mathbf{w}^T\mathbf{x} + w_0)$$

Sigmoid Properties & The Decision Rule

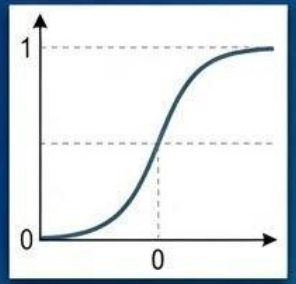
Translating Math into Probability

Properties of the Sigmoid Function:

Bounded: The output is strictly squeezed between 0 and 1, making it perfectly interpretable as a probability percentage.

Smooth & Differentiable: Unlike a jagged step function, the smooth curve allows for complex calculus-based optimization.

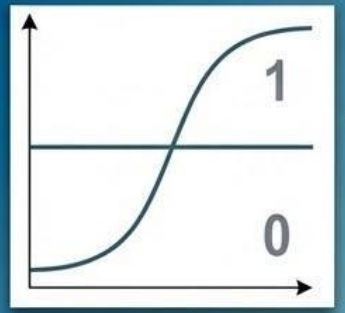
Monotonic: A larger input number will always reliably result in a larger probability.



The Decision Rule:

Once we have our probability, we set a threshold (usually 50%) to make our final classification.

$$\hat{y} = \begin{cases} 1 & \text{if } p(y = 1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$



Threshold (0.5)

Learning Objective & Optimization

Maximizing the Likelihood

The Goal:

Instead of just minimizing errors, Logistic Regression seeks to maximize the likelihood that the model accurately predicts the training data.

The Log-Likelihood: $\prod_{i=1}^{m} \rightarrow \sum \log$

Because multiplying tiny probabilities together causes computers to round down to zero, we take the natural logarithm. This turns the multiplication into addition, making it mathematically stable and much easier to optimize:

Optimization:

The model uses algorithms like Gradient Ascent (or Gradient Descent on the negative log-likelihood) to find the perfect weights.

$$L(w) = \prod_{i=1}^{m} p(y_i|x_i, w) \qquad \ell(w) = \sum_{i=1}^{m} \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right]$$

Worked Example: Credit Scoring

Putting It Into Practice

The Historical Data:

Applicant	Income ($K)	Debt ($K)	Defaulted?
1	50	10	No (0)
2	30	25	Yes (1)
3	80	5	No (0)
4	40	30	Yes (1)
5	60	15	No (0)

The Trained Model Formula:

$$p(\text{default} = 1) = \sigma(-0.05 \times \text{Income} + 0.1 \times \text{Debt} + 1.5)$$

Worked Example: Scoring a New Applicant

Calculating the Risk

New Applicant Data:

Income = \$45K
Debt = \$20K

Step 1: Compute the linear combination (z):

$$z = -0.05(45) + 0.1(20) + 1.5$$
$$= -2.25 + 2.0 + 1.5$$
$$= 1.25$$

Step 2: Apply the Sigmoid function to z:

$$p = \frac{1}{1 + e^{-1.25}}$$
$$= \frac{1}{1 + 0.2865} \approx 0.777$$

The Interpretation:

The model calculates a 77.7% probability that this applicant will default. Because 0.777 > 0.5, we classify this applicant as High Risk (1).

Key Insights & Takeaways

The Core Lessons of Machine Learning

- **The "No Free Lunch" Theorem:** There is no single 'perfect' algorithm. Different data structures and business problems require entirely different mathematical approaches.
- **The Accuracy vs. Interpretability Trade-Off:** The simpler a model is to explain to a stakeholder, the less complex data it can handle (and vice versa).
- **Lazy vs. Eager Learners:** You must choose whether you want to pay the computational cost upfront during training (Eager) or later during prediction (Lazy).
- **The Value of Probability:** Hard decisions are brittle. Algorithms that provide probabilistic outputs (like Logistic Regression and Naïve Bayes) offer crucial confidence scores for decision-makers.
- **Feature Scaling is Critical:** For distance-based algorithms (k-NN) and linear models, failing to normalize your features will mathematically destroy your model's accuracy.