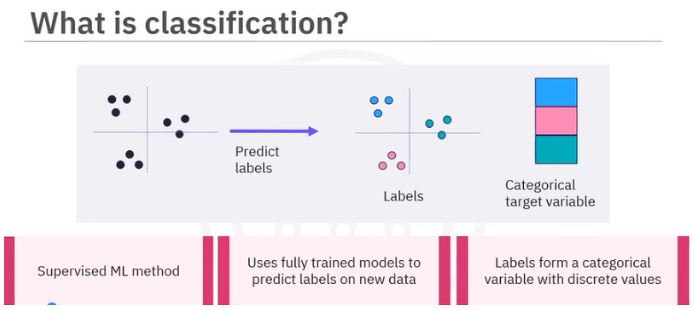
Machine Learning with Python(M1)

Created	@December 3, 2025 8:14 PM
Module Code	IBMM01: Introduction to Machine Learning

Module 3: Building Supervised Learning Model

Classification

Machine learning (ML) is a subset of AI that uses algorithms to learn from data. It typically requires practitioners to manually engineer features. Deep learning, by contrast, uses multi-layered neural networks that automatically extract features from complex, unstructured data.


What is classification?
Predict labels
Labels
Categorical target variable
Supervised ML method
Uses fully trained models to predict labels on new data
Labels form a categorical variable with discrete values

1. What is Classification?

Classification is a **supervised machine learning method** where the model is trained on labeled data to predict **categorical outputs** (discrete labels).

Examples of labels: *spam/not spam*, *loan default/no default*, *drug A/B/C*.

The model learns patterns from historical data and uses them to classify new, unseen data.

---

2. Applications of Classification

Classification is widely used across industries. Examples:

- **Email filtering:** spam vs. not spam
- **Speech-to-text**, **handwriting recognition**
- **Biometric ID**, **document classification**
- **Churn prediction:** Will a customer leave?
- **Customer segmentation:** Which group does the customer belong to?
- **Marketing:** Will a customer respond to a campaign?

More detailed examples:

- **Loan default prediction:** Predict whether a customer will repay a loan → **binary classification** (two classes)
- **Drug recommendation:** Predict which drug suits a patient → **multi-class classification** (more than two classes)

---

3. Common Classification Algorithms

Algorithms used to build classification models include:

- **Logistic Regression**
- **Naive Bayes**
- **Decision Trees**
- **K-Nearest Neighbors (KNN)**
- **Support Vector Machines (SVM)**
- **Neural Networks**

Some of these directly support multi-class (e.g., Decision Trees, KNN), while others need special strategies.

---

4. Multi-Class Prediction Strategies

Some algorithms naturally support multi-class classification, but others don't. For binary-only algorithms, we extend them using:

A. One-vs-All (OvA)

- Create **k binary classifiers** for **k classes**.
- Each classifier answers:

  “Is this point in class i or not?”
- The class whose classifier is most confident is chosen.

Pros: Simple, works well

Cons: Some points may not belong to any class → useful for detecting outliers

---

B. One-vs-One (OvO)

- Build a classifier for **every pair of classes**.
- Each classifier answers:

  “Is it class A or class B?”
- Final label decided by **majority vote**.
- If there’s a tie → use classifier confidence scores.

Pros: Very accurate

Cons: For many classes, number of models becomes large

---

5. Key Takeaways

- **Classification** predicts **categorical labels** using supervised learning.
- Widely used in **finance, healthcare, marketing, and security**.
- Algorithms include Logistic Regression, KNN, Decision Trees, SVM, etc.
- Multi-class classification can be solved using **One-vs-All** or **One-vs-One** strategies.
- Voting or confidence-based methods choose final class in OvO.

Decision Trees

Decision Tree

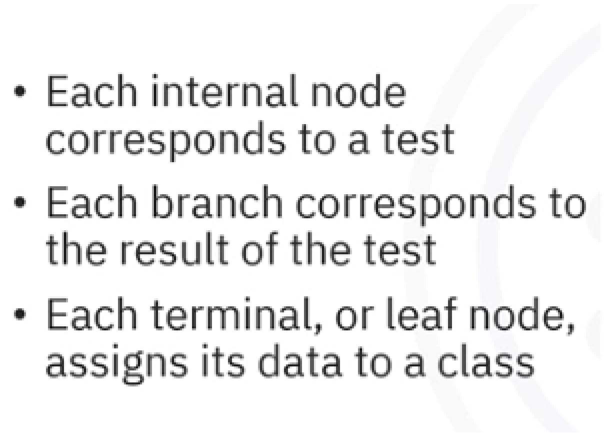
is a machine learning model that works like a

flowchart

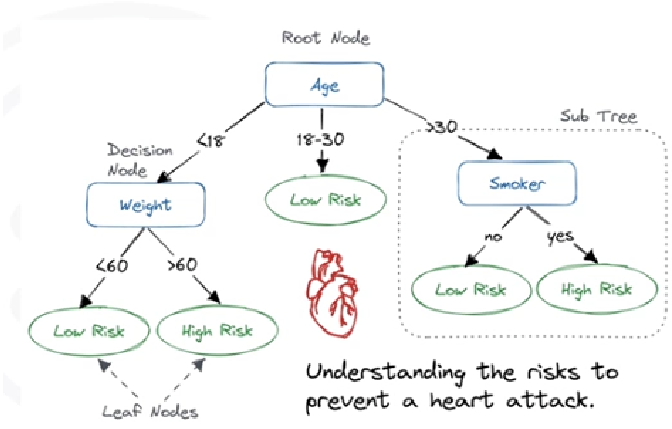
At each step, it asks a **question about a feature**

(like age, gender, cholesterol).

Depending on the answer, you follow a branch until you reach a **leaf node**, which gives the final **prediction/class**



- Each internal node corresponds to a test
- Each branch corresponds to the result of the test
- Each terminal, or leaf node, assigns its data to a class



🟦 How a Decision Tree Works (Simple Terms)

1. Start with full training data

You begin with all the labeled samples (patients + the drug they responded to).

2. Pick the best feature to split

The tree chooses a feature that best **separates the classes** (drug A or B).

How does it choose?

Using:

- **Entropy** (measures impurity/disorder)
- **Information Gain** (how much entropy is reduced)
- or **Gini impurity**

The best feature = the one that reduces impurity the most.

3. Split the dataset

Example:

Split by **Gender → Male / Female**

Each branch gets its part of the data.

4. Repeat recursively

For each branch:

- Pick the best next feature that separates the remaining data.
- Keep splitting until:
  - all samples in a node belong to one class, or
  - no features left, or
  - a stopping rule is reached.

This is called **recursive partitioning**.

5. Stop the tree (pruning)

We stop or cut branches to avoid:

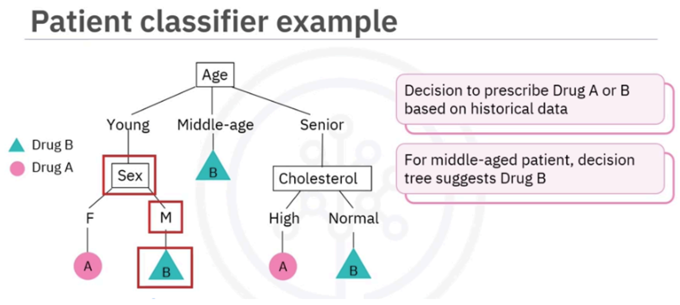
- **Overfitting**
- **Capturing noise**
- **Making the tree too complex**

Stopping rules include:

- Max depth reached
- Minimum samples per node
- Minimum samples per leaf
- Max leaf nodes reached
- No significant improvement

Pruning makes the tree:

- simpler
- more generalizable
- easier to read
- more accurate on new data
-



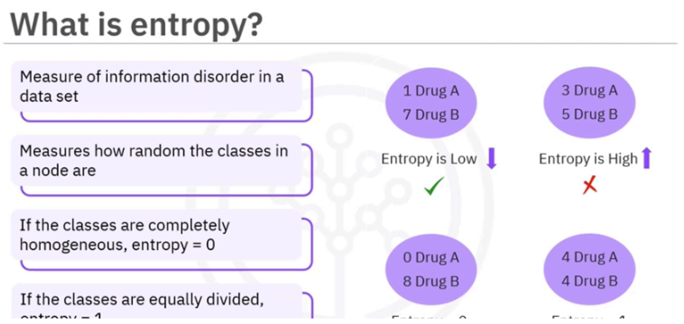
Patient classifier example
Age
Young
Middle-age
Senior
Drug B
Drug A
Sex
F
M
A
B
B
Cholesterol
High
Normal
A
B
Decision to prescribe Drug A or B based on historical data
For middle-aged patient, decision tree suggests Drug B

🟩 Entropy & Information Gain (Very Simple)

- **Entropy** = how mixed the classes in a node are
  - pure node (all same class) → entropy = 0
  - 50-50 classes → entropy = 1 (maximum disorder)
- **Information Gain** = how much entropy decreases after a split

  The higher the gain → the better the feature.

Decision Trees pick features that give **highest information gain**.



What is entropy?

Measure of information disorder in a data set

Measures how random the classes in a node are

If the classes are completely homogeneous, entropy = 0

If the classes are equally divided,

1 Drug A
7 Drug B

Entropy is Low ✓

3 Drug A
5 Drug B

Entropy is High ✗

0 Drug A
8 Drug B

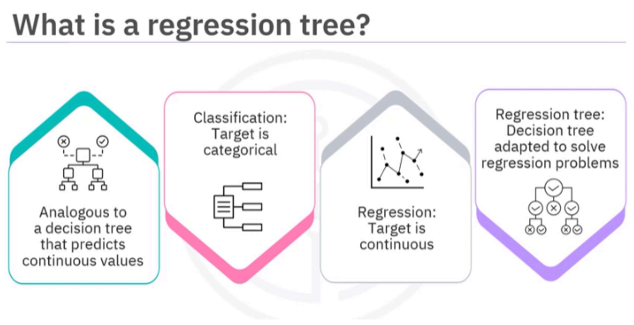
4 Drug A
4 Drug B

Why Decision Trees are Useful

- Very **interpretable** → you can see how decisions are made
- No need to scale features
- Works with numerical + categorical data
- Shows feature importance
- Easy to visualize

Regression Trees

Regression trees are created by recursively splitting the dataset into subsets to minimize the variance or Mean Squared Error (MSE) of the target values. This process generates a tree-like structure and reduces the spread of the predicted values within each node



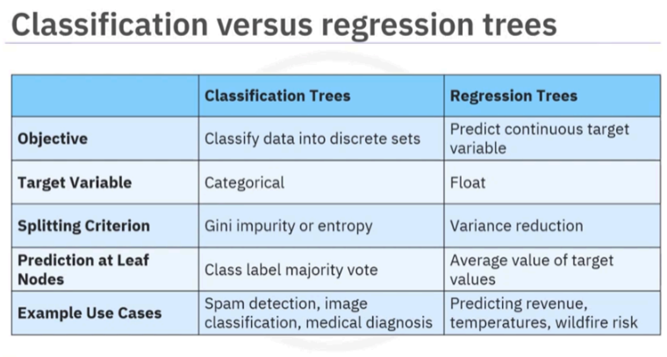
What is a regression tree?

Analogous to a decision tree that predicts continuous values

Classification: Target is categorical

Regression: Target is continuous

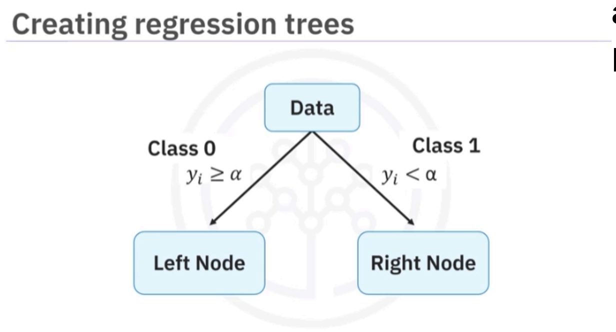
Regression tree: Decision tree adapted to solve regression problems



Classification versus regression trees

	Classification Trees	Regression Trees
Objective	Classify data into discrete sets	Predict continuous target variable
Target Variable	Categorical	Float
Splitting Criterion	Gini impurity or entropy	Variance reduction
Prediction at Leaf Nodes	Class label majority vote	Average value of target values
Example Use Cases	Spam detection, image classification, medical diagnosis	Predicting revenue, temperatures, wildfire risk

How a Regression Tree Makes Predictions?



At each leaf node → prediction = **average of all target (y) values** in that leaf.

Example: If leaf has targets: 10, 12, 15

Prediction = (10 + 12 + 15)/3 = **12.33**

If data is skewed → can use **median** instead of mean.

The tree wants each group to have **low variation**. If values in a node are like: 30, 31, 29, 30 → very similar → good group

If they are: 30, 100, 10, 60 → totally different → bad group

(You can't predict well using their average).

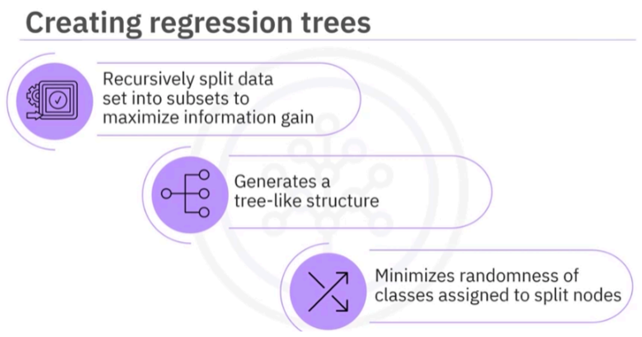
To measure this variation → we use **MSE (Mean Squared Error)**

but think of it simply as:

👉 A number that tells: "How scattered are the values?"

Low MSE = values are close

High MSE = values are far apart

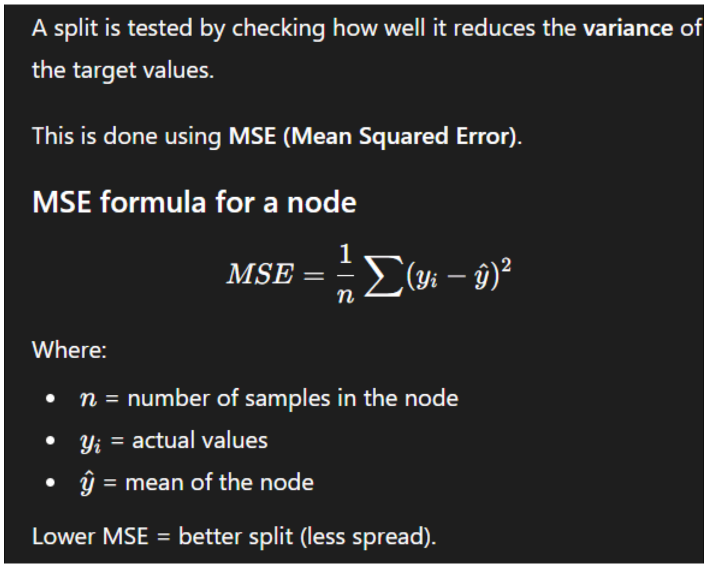


Creating regression trees

Recursively split data set into subsets to maximize information gain

Generates a tree-like structure

Minimizes randomness of classes assigned to split nodes



A split is tested by checking how well it reduces the **variance** of the target values.

This is done using **MSE (Mean Squared Error)**.

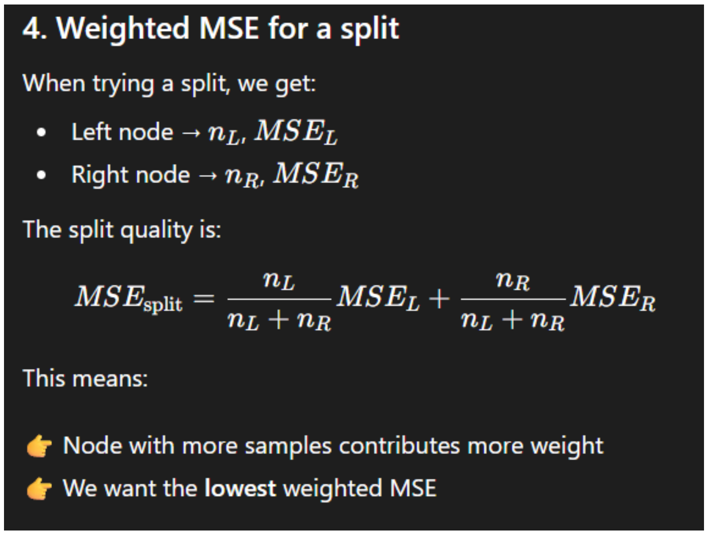
MSE formula for a node

$$MSE = \frac{1}{n} \sum (y_i - \hat{y})^2$$

Where:

- n = number of samples in the node
- y_i = actual values
- $\hat{y}$ = mean of the node

Lower MSE = better split (less spread).



4. Weighted MSE for a split

When trying a split, we get:

- Left node → n_L, MSE_L
- Right node → n_R, MSE_R

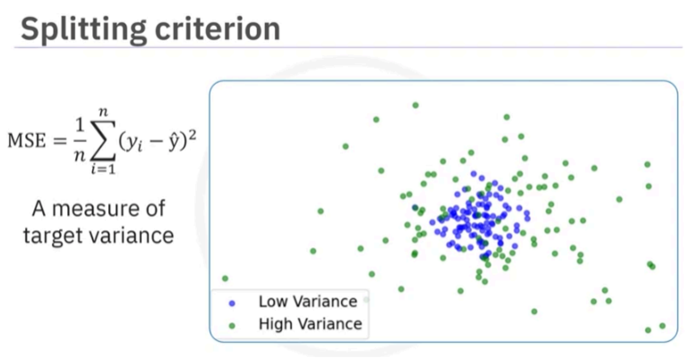
The split quality is:

$$MSE_{\text{split}} = \frac{n_L}{n_L + n_R} MSE_L + \frac{n_R}{n_L + n_R} MSE_R$$

This means:

👉 Node with more samples contributes more weight

👉 We want the **lowest** weighted MSE


Splitting criterion

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2$$

A measure of target variance

How does the tree find the best threshold?

For continuous feature X:

Steps

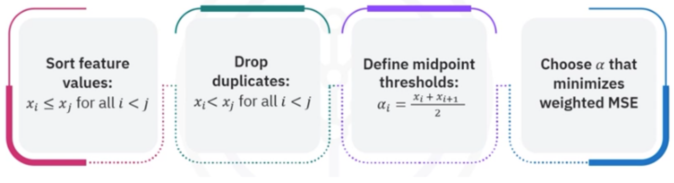
1. Sort the unique values of that feature

   X1<X2<X3<...X_1 < X_2 < X_3 < ...X1<X2<X3<...

2. Compute midpoints between each pair :
3. Try splitting at each α
4. For each α → calculate weighted MSE
5. Choose the α with the **lowest** weighted MSE

This is **exhaustive search** → accurate but slow for big data.

For large data → sample some thresholds instead of using all.


Sort feature values: $x_i \le x_j$ for all $i < j$
Drop duplicates: $x_i < x_j$ for all $i < j$
Define midpoint thresholds: $\alpha_i = \frac{x_i + x_{i+1}}{2}$
Choose α that minimizes weighted MSE

✔ 6. Binary and Multiclass Features

Binary feature → only 2 splits possible

Simple: values go to left or right. Only 1 possible split.

Multiclass feature →

For categorical features with multiple levels, evaluate different binary partitions of the categories and select the split that minimizes the weighted MSE

⭐ Final Summary

- Regression tree predicts numbers (continuous output).
- At each leaf → prediction = average y.
- Splits chosen by minimizing weighted MSE (variance).
- Thresholds for continuous features are midpoints between sorted values.
- Find the threshold giving the lowest weighted MSE → best split.

Support Vector Machines

1. What SVM Does

SVM is a supervised machine learning method used mainly for classification.

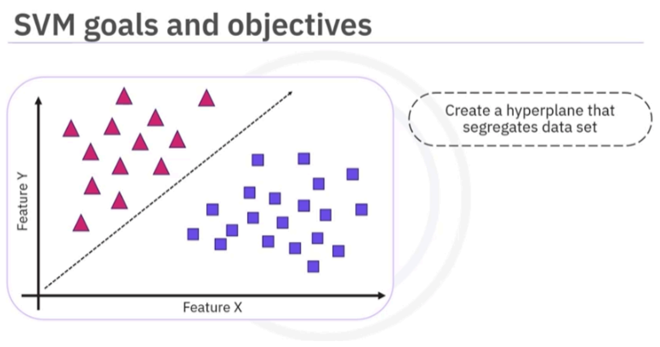
The goal of SVM:

1. Find a boundary (line in 2D, plane in higher dimensions) that separates two classes.
2. Ensure this boundary is as far away as possible from the nearest points of each class(margin).
3. These nearest points are called **support vectors**.
4. The distance between support vectors and the boundary is the **margin**.

SVM chooses the boundary that **maximizes the margin**.

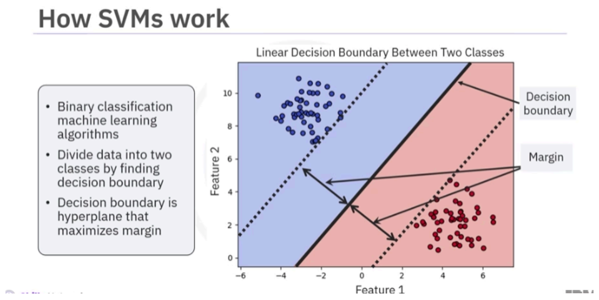
The hyperplane is a linear line that segregates and classifies the dataset.

A larger margin usually means better generalization on new data.



Hard Margin vs Soft Margin

1. **Hard Margin**
    - No misclassification allowed.
    - Works only if data is perfectly separable.
2. **Soft Margin**
    - Allows some misclassification using slack variables.
    - Controlled by parameter **C**.
        - Large C → strict boundary.
        - Small C → more relaxed boundary.



3. When Data Is Not Linearly Separable

Some datasets cannot be separated by a straight line.

SVM solves this using **kernel functions**:

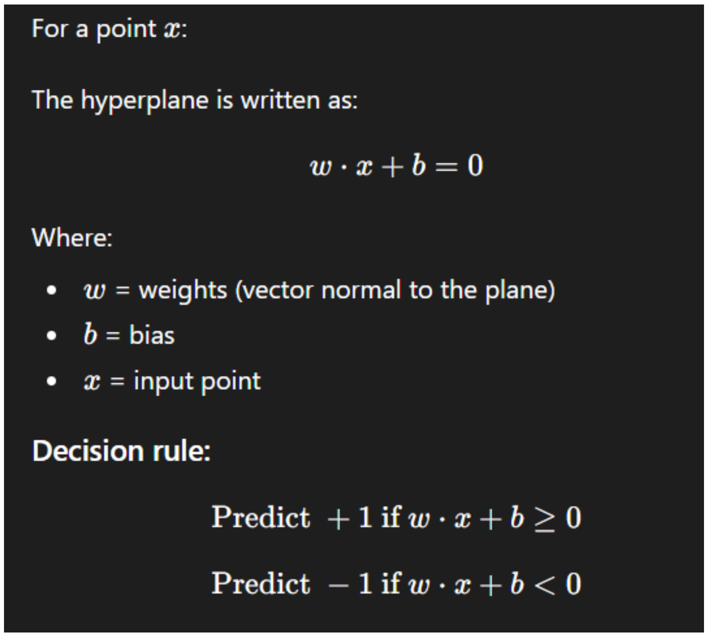
- Map data into higher dimensions without actually computing the transformation.

- In the new space, classes may become separable.

Common kernels:

- Linear
- Polynomial
- Radial Basis Function (RBF)
- Sigmoid

Math :



For a point x:

The hyperplane is written as:

$$w \cdot x + b = 0$$

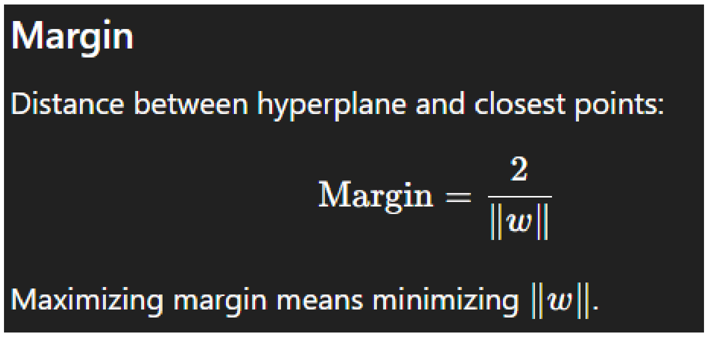
Where:

- w = weights (vector normal to the plane)
- b = bias
- x = input point

Decision rule:

$$\text{Predict } +1 \text{ if } w \cdot x + b \geq 0$$

$$\text{Predict } -1 \text{ if } w \cdot x + b < 0$$



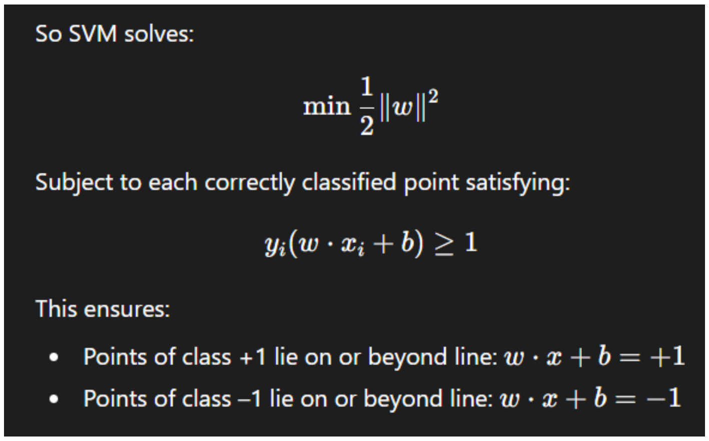
Margin

Distance between hyperplane and closest points:

$$\text{Margin} = \frac{2}{\|w\|}$$

Maximizing margin means minimizing $\|w\|$.

Hard Margin :


So SVM solves:

$$\min \frac{1}{2}\|w\|^2$$

Subject to each correctly classified point satisfying:

$$y_i(w \cdot x_i + b) \geq 1$$

This ensures:

- Points of class +1 lie on or beyond line: $w \cdot x + b = +1$
- Points of class –1 lie on or beyond line: $w \cdot x + b = -1$

Why margin matters?

A larger margin →

• less overfitting

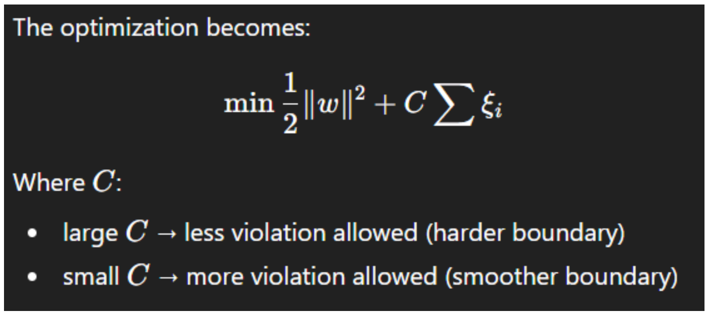
• better generalization

• more stable classifier

Soft Margin :

Real-world data has noise.

So SVM allows some points to violate the margin using slack variables ξi.


The optimization becomes:

$$\min \frac{1}{2}\|w\|^2 + C\sum \xi_i$$

Where C:

- large C → less violation allowed (harder boundary)
- small C → more violation allowed (smoother boundary)

Some datasets cannot be separated by a straight line.

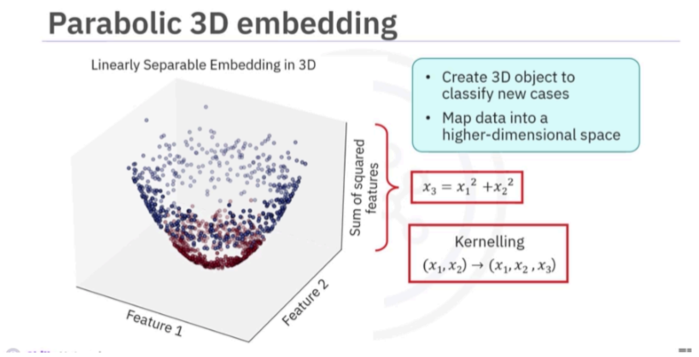
SVM solves this using **kernel functions**:

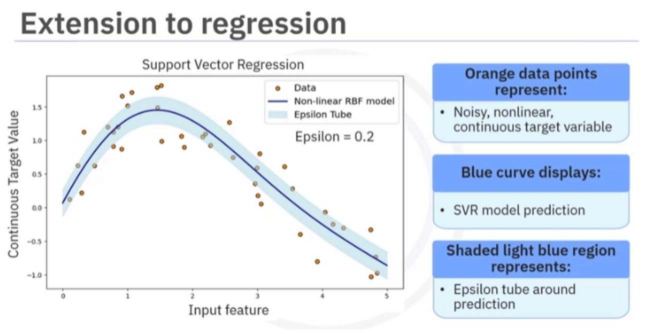
- Map data into higher dimensions without actually computing the transformation.

- In the new space, classes may become separable.

Common kernels:

- Linear (Default)
- Polynomial
- Radial Basis Function (RBF)
- Sigmoid


Parabolic 3D embedding
Linearly Separable Embedding in 3D
Create 3D object to classify new cases
Map data into a higher-dimensional space
Sum of squared features
$x_3 = x_1^2 + x_2^2$
Kernelling
$(x_1, x_2) \rightarrow (x_1, x_2, x_3)$
Feature 1
Feature 2



Regression version of SVM uses an **epsilon-tube** around predictions. Errors inside the tube are ignored; errors outside are penalized.

Advantages

- Works well in high-dimensional spaces
- Effective with clear margin/separation
- Can model non-linear boundaries using kernels
- Resistant to overfitting if C and kernel are selected well

Limitations

- Slow for large datasets
- Sensitive to noise
- Requires tuning of C, kernel, gamma, degree, etc.

SVM is good for image analysis tasks, such as image classification and handwritten digit recognition.
It's also highly effective for parsing, spam detection, and sentiment analysis. SVM can be used for machine learning problems, such as speech recognition, anomaly detection, and noise filtering.

KNN

K-NN is a supervised ML algorithm that uses labeled data to predict the label of new data based on *similarity* (distance).

It can be used for:

- **Classification** (predict category/class)
- **Regression** (predict continuous value)

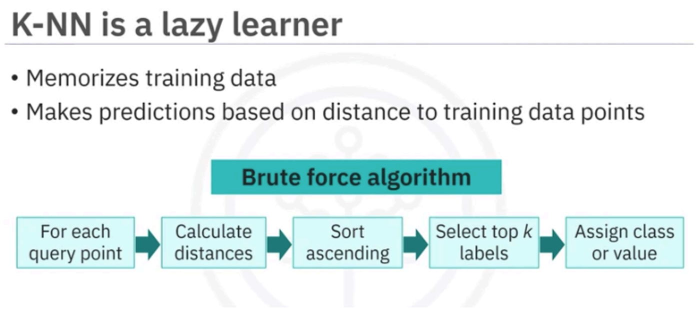
1. Intuition

Points that are **close to each other** usually **belong to the same class** or have **similar values**.

So K-NN:

1. Stores all training data
2. For a new point → finds the **K nearest neighbors**
3. Classification → **majority vote**
4. Regression → **average/median**

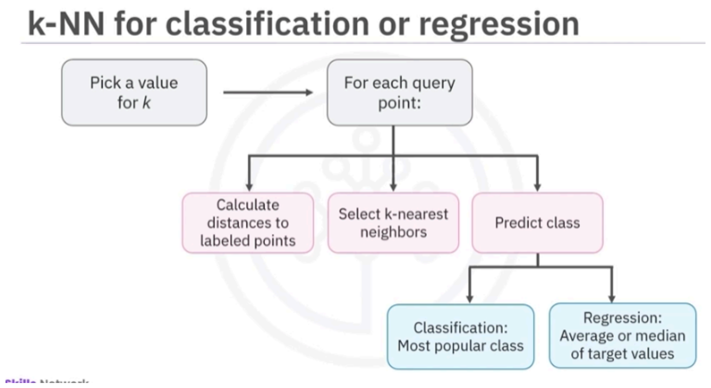
K-NN is called a **lazy learner** because it does *not* learn a model—it waits until prediction time.


K-NN is a lazy learner

- Memorizes training data
- Makes predictions based on distance to training data points

2. Basic Math: Distance Calculation

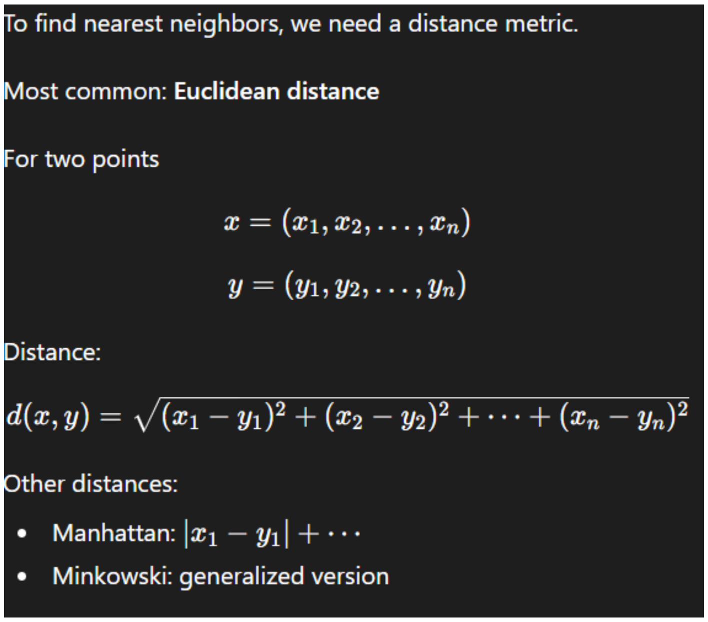
To find nearest neighbors, we need a distance metric.


k-NN for classification or regression

Pick a value for k
For each query point:
Calculate distances to labeled points
Select k-nearest neighbors
Predict class
Classification: Most popular class
Regression: Average or median of target values

Most common: **Euclidean distance**

For two points



To find nearest neighbors, we need a distance metric.

Most common: **Euclidean distance**

For two points

$$x = (x_1, x_2, \ldots, x_n)$$

$$y = (y_1, y_2, \ldots, y_n)$$

Distance:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

Other distances:

- Manhattan: $|x_1 - y_1| + \cdots$
- Minkowski: generalized version

---

3. How K-NN works (step-by-step)

For classification:

1. Choose **K**
2. Compute distance from query point to every training point
3. Pick **K smallest distances**
4. Look at the labels of those K points
5. Output the **majority class**

For regression:

- Output **mean/median** of K nearest neighbors' values

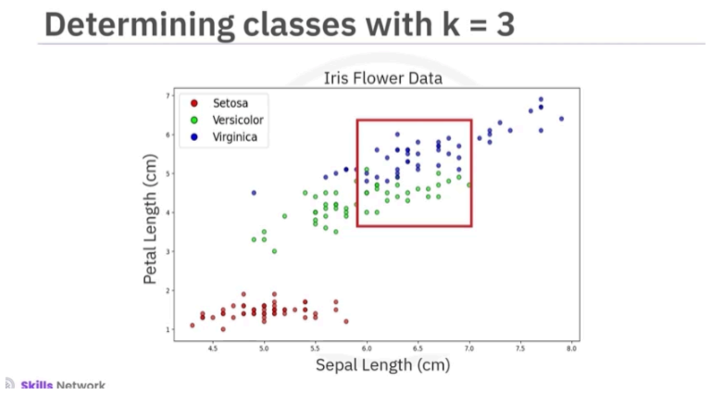
---

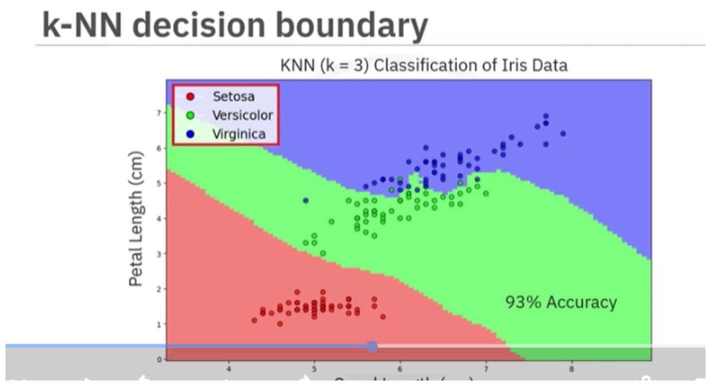
4. Example (conceptual)

If K = 3 and the nearest neighbors have labels:

- Class A
- Class A
- Class B

Prediction = **Class A** (majority vote)





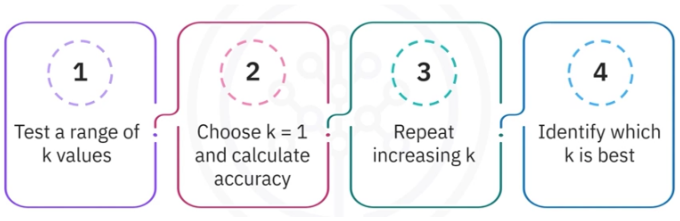
5. Choosing K

Effect of K:

- **Small K (e.g., K=1)**
  - Very sensitive to noise
  - High variance → **overfitting**
- **Large K (e.g., K=30)**
  - Too smooth
  - High bias → **underfitting**

Solution:

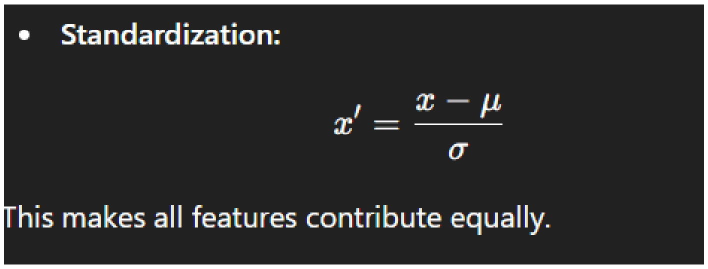
Try different K values and pick the one giving highest accuracy on test data.



6. Why scaling is important

If one feature has large numeric values (e.g., height in cm) and another has small values (e.g., petal width), then the large feature dominates the distance.

So you must apply:



- **Standardization:**

$$x' = \frac{x - \mu}{\sigma}$$

This makes all features contribute equally.

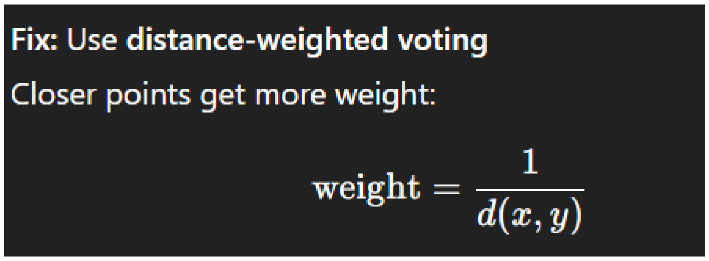
7. Problems & Improvements

1. Skewed class distribution

If one class is much more frequent, majority voting becomes biased.

Fix: Use **distance-weighted voting**

Closer points get more weight:



Fix: Use **distance-weighted voting**

Closer points get more weight:

$$\text{weight} = \frac{1}{d(x, y)}$$

2. Irrelevant or redundant features

- Increase noise

- Require larger K
- Hurt accuracy and speed

Fix: Keep only relevant features (feature selection).

---

8. Pros and Cons

Advantages

- Simple
- No training time
- Works for classification & regression

Disadvantages

- Slow for large datasets (must compute distance for every point)
- Sensitive to irrelevant features
- Needs scaling
- Hard with high-dimensional data

- K-NN predicts the label of a point based on its **K nearest neighbors**.
- Distance is usually **Euclidean**.
- Classification = **majority vote**; regression = **average**.
- Small K → overfitting; large K → underfitting.
- Needs feature scaling.
- Use cross-validation to choose best K.
- Weighted K-NN helps when class distribution is imbalanced.

Bias, Variance & Ensemble Models

---

1. Bias and Variance Concepts

Bias

- Measures **how far predictions are from the true target**.
- High bias → model predictions are consistently wrong (off-target).
- Low bias → predictions are close to the true values (accurate).

Example:

If model predicts always too high/low, bias is large.

Variance

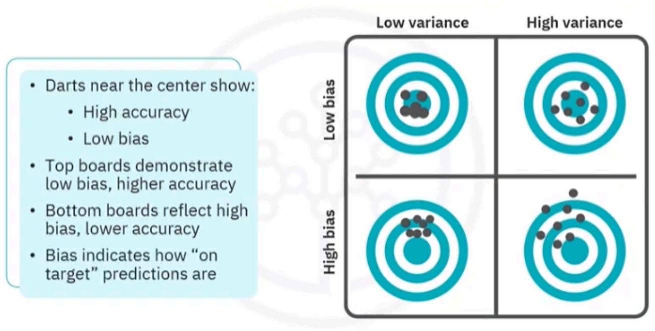
- Measures **how much predictions change** if trained on different subsets of data.
- High variance → model is very sensitive to training data (unstable).

  tracks noise in training data
- Low variance → predictions remain similar across samples.

  Less sensitive to noise

---

2. Dartboard Analogy

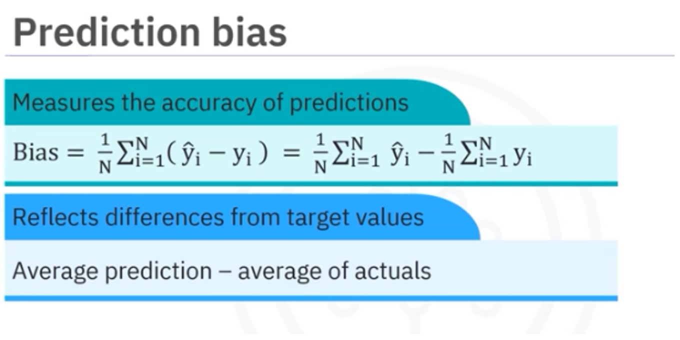


- Darts near the center show:
  - High accuracy
  - Low bias
- Top boards demonstrate low bias, higher accuracy
- Bottom boards reflect high bias, lower accuracy
- Bias indicates how "on target" predictions are

- **Top boards** = low bias (predictions near the center).
- **Left boards** = low variance (grouped tightly).
- Best = **low bias + low variance**.

---

3. Mathematical View



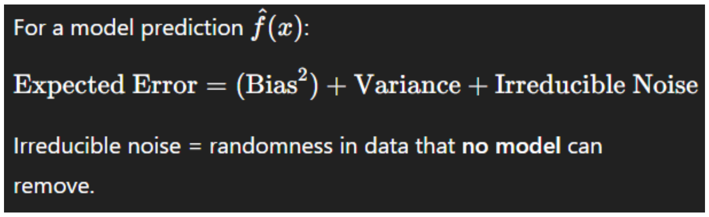
Prediction bias

Measures the accuracy of predictions

$$\text{Bias} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i) = \frac{1}{N}\sum_{i=1}^{N}\hat{y}_i - \frac{1}{N}\sum_{i=1}^{N}y_i$$

Reflects differences from target values

Average prediction – average of actuals



For a model prediction $\hat{f}(x)$:

$$\text{Expected Error} = (\text{Bias}^2) + \text{Variance} + \text{Irreducible Noise}$$

Irreducible noise = randomness in data that **no model** can remove.

4. Bias–Variance Tradeoff

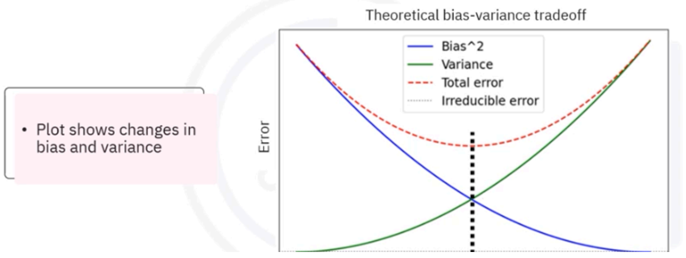
As model complexity increases:

- **Bias ↓** (model fits training data better)
- **Variance ↑** (model becomes sensitive / may overfit)

Two extremes:

- **Underfitting** → high bias, low variance
- **Overfitting** → low bias, high variance

There exists an **optimal complexity** where error is minimum.



• Plot shows changes in bias and variance

5. Weak vs Strong Learners

Weak Learner

- Slightly better than random guessing.
- **High bias**, low variance (underfits).
- Example: a shallow decision tree (depth = 1 or 2).

Strong Learner

- Very flexible model.
- **Low bias**, high variance (overfits).
- Example: a deep decision tree.

---

6. Ensemble Methods

Ensembles combine many models to solve the bias–variance tradeoff.

A. Bagging (Bootstrap Aggregating)

Goal: Reduce Variance

How it works:

1. Create many bootstrap samples (sampling with replacement).
2. Train a model (usually a tree) on each.
3. Average their predictions.

Effect:

- Averaging cancels out high variance.
- Bias may slightly increase, but variance drops a lot.

Random Forest = Bagging + randomness in feature selection

Use when:

- Base learner has **high variance** (deep trees).
- You want to **prevent overfitting**.

B. Boosting

Goal: Reduce Bias

How it works:

1. Train a weak model.
2. Give more weight to misclassified points.
3. Train next model to fix previous errors.
4. Final model = weighted sum of all learners.

Effect:

- Each model removes bias of previous one → total **bias decreases**.
- Complexity increases → variance may increase.

Popular Algorithms:

- AdaBoost
- Gradient Boosting
- XGBoost
- LightGBM

Use when:

- Base learner has **high bias** (simple tree).
- You want to **improve accuracy** by decreasing bias.

---

7. Summary Table

Concept	High Bias	High Variance
Model behavior	Oversimplifies	Too sensitive to noise
Problem	Underfitting	Overfitting
Fix	Boosting	Bagging

Ensemble	Addresses	How
Bagging (Random Forest)	Overfitting (high variance)	Averages many high-variance trees
Boosting (XGBoost etc.)	Underfitting (high bias)	Sequentially learns from mistakes

---

8. Key Takeaways

- **Bias** = error from wrong assumptions.
- **Variance** = error from sensitivity to training data.
- Increasing model complexity → bias ↓, variance ↑.
- **Bagging** reduces variance (helps with overfitting).
- **Boosting** reduces bias (helps with underfitting).
- Decision trees are commonly used as base learners.