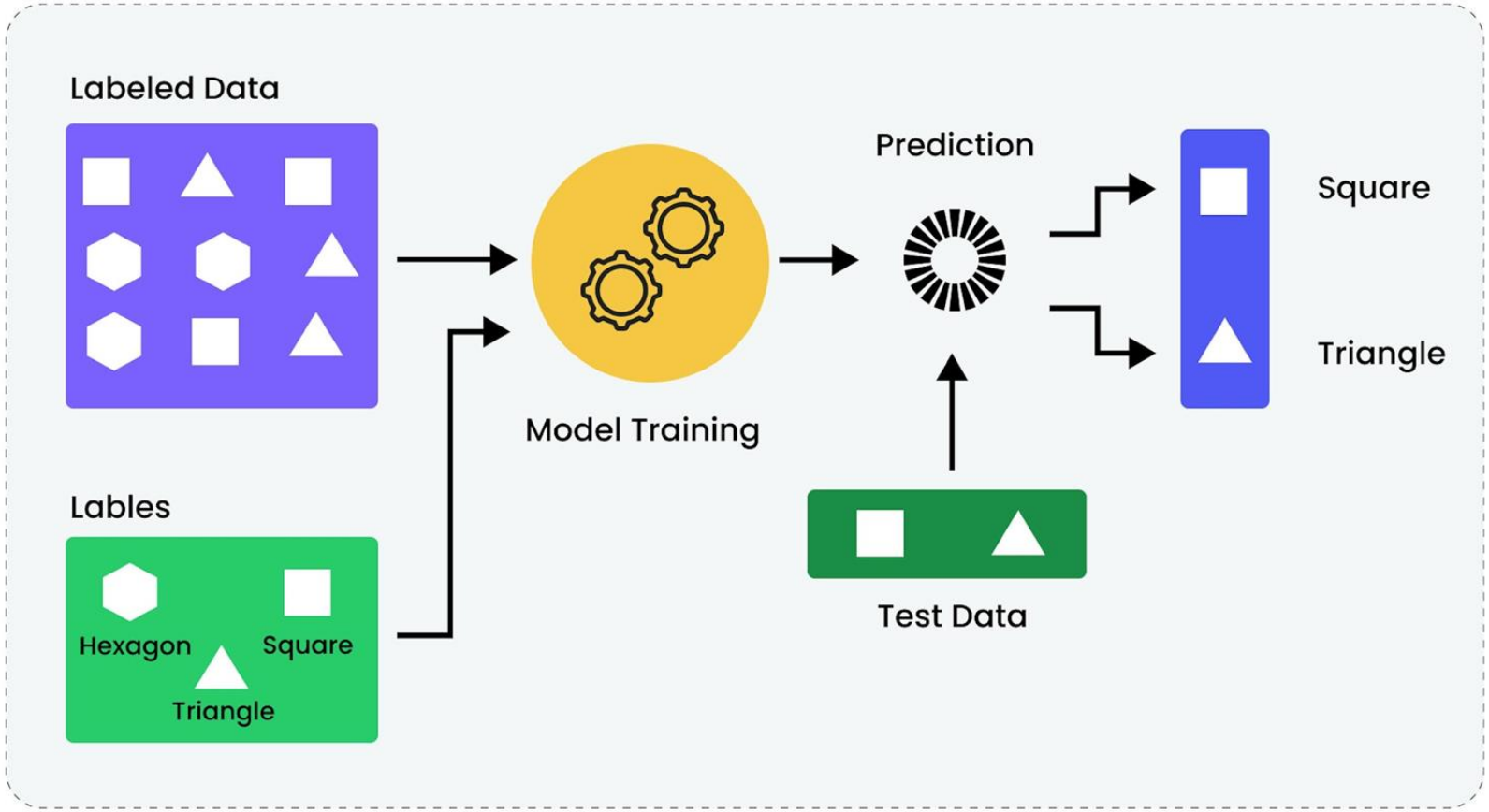
SUPERVISED LEARNING PART 1: CLASSIFICATION



K-Nearest Neighbors (KNN)

K-NEAREST NEIGHBORS (KNN)

Main Idea: When given a new data point, KNN looks at the *k* closest data points in the training set (neighbors) and makes predictions based on their labels.

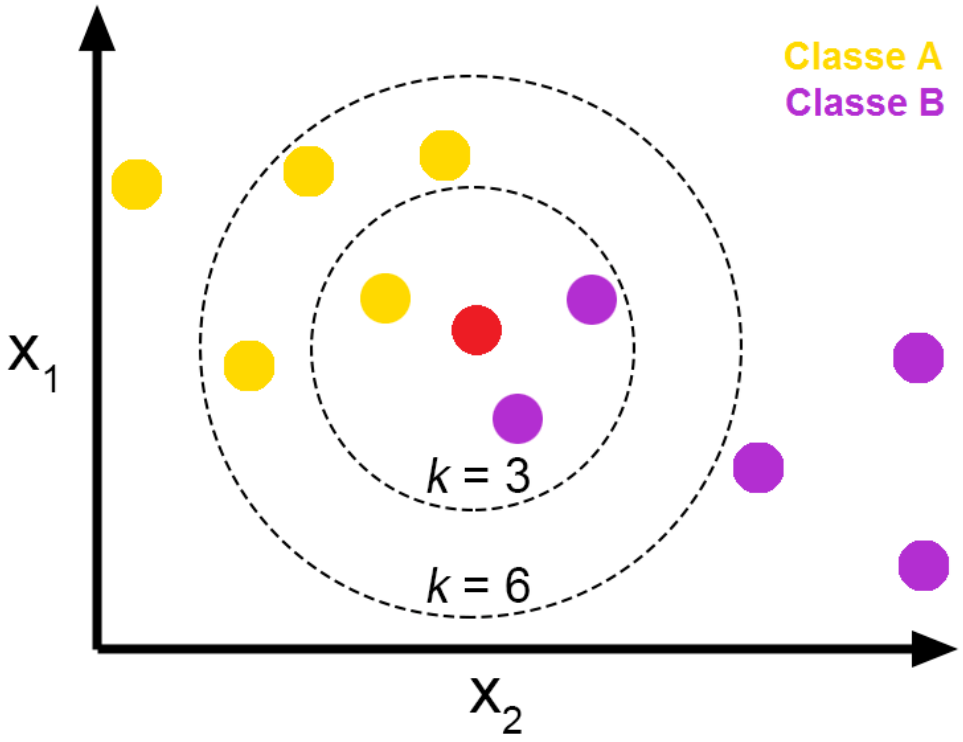
- For **classification**, the majority label among the neighbors is chosen.
- For **regression**, the average of the neighbors' values is used.

Steps:

1. Calculate the distance between the new point and all training points.
2. Sort the distances and find the k nearest neighbors.
3. Predict the label based on the neighbors.

The **K-nearest neighbor (KNN)** algorithm is considered a **supervised learning** algorithm because it uses labeled training data to make predictions about new, unseen data points.

Instead of building a mathematical model or decision boundary during training, KNN stores the labeled training data.



BYU

K-NN SUMMARY

Choosing k :

- Small k : Sensitive to noise, can overfit.
- Large k : Smooths predictions but may underfit.

Distance Metrics: Common ones are:

- Euclidean Distance (default in Scikit-learn)
- Manhattan Distance
- Minkowski Distance (generalization)

Scaling Features: Since KNN is distance-based, ensure features are on a similar scale (e.g., use Min-Max Scaling or Standardization).

Advantages and Disadvantages

Advantages:

- Simple to understand and implement.
- No explicit training phase; training data is directly used.

Disadvantages:

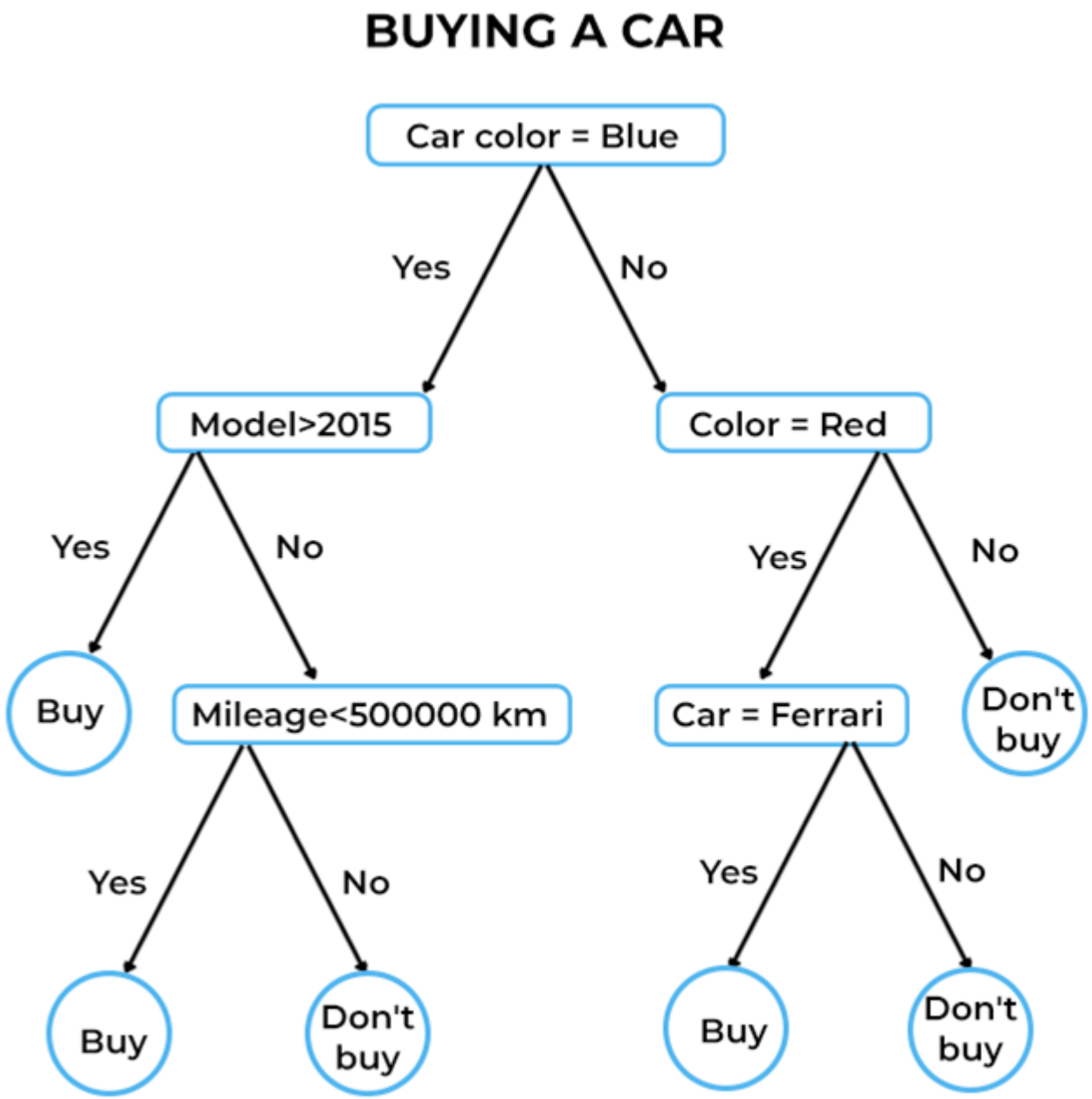
- Computationally expensive for large datasets.
- Sensitive to irrelevant features and feature scaling.

BYU

Decision Trees

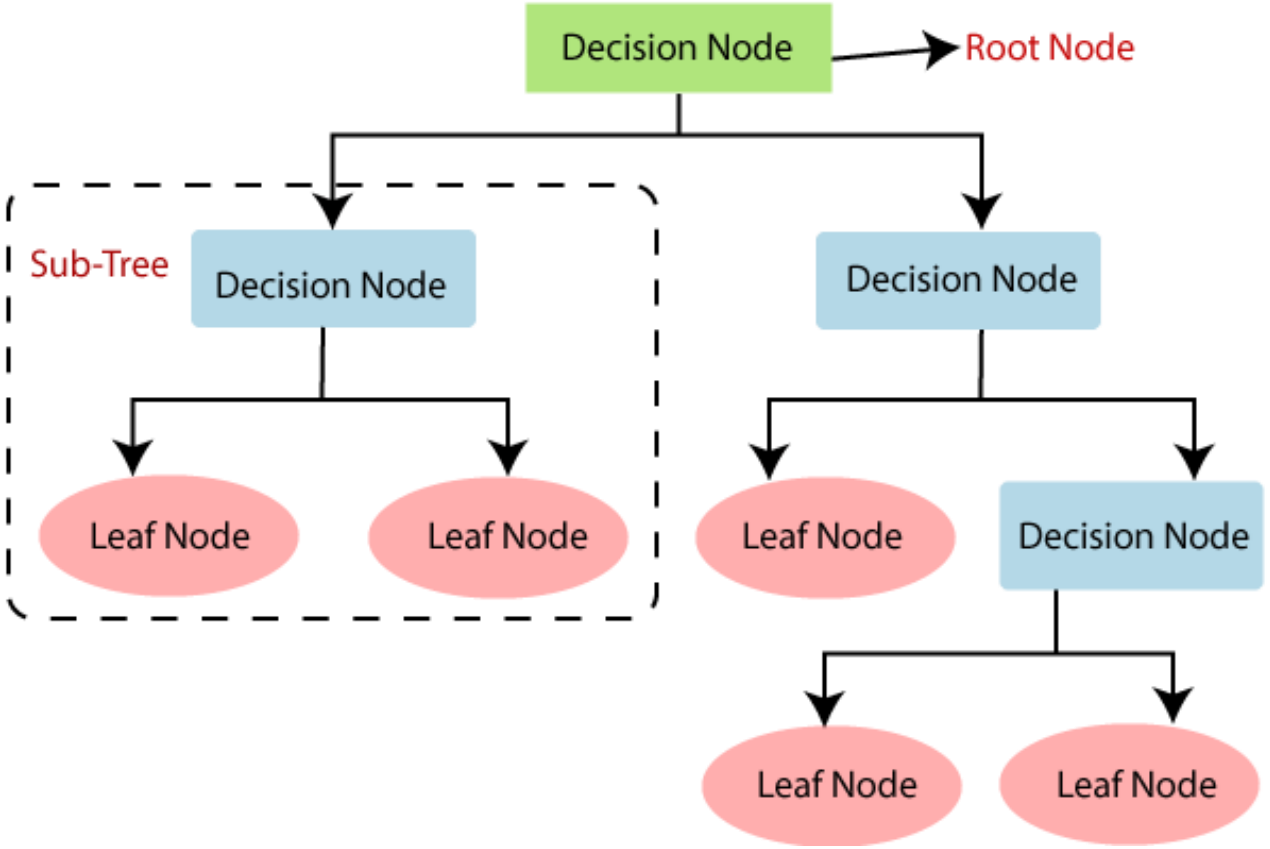
DECISION TREES

- **Decision Trees** are a powerful tool for both **classification** and **regression** tasks.
- They model data by splitting it based on feature values, creating a tree-like structure where each node represents a decision or split based on a feature.
- **They are explainable**
- **Can model non-linear relationships**
- **Handles both numeric and categorical data**
- **Works with missing data***
- **No feature scaling required**
- **Prone to data overfitting, instability, bias**



DECISION TREES DEFINITIONS

- **Root Node:** This attribute is used for dividing the data into two or more sets. The feature attribute in this node is selected based on Attribute Selection Techniques.
- **Branch or Sub-Tree:** A part of the entire decision tree is called a branch or sub-tree.
- **Splitting**: Dividing a node into two or more sub-nodes based on if-else conditions.
- **Decision Node:** After splitting the sub-nodes into further sub-nodes, then it is called the decision node.
- **Leaf or Terminal Node:** This is the end of the decision tree where it cannot be split into further sub-nodes.
- **Pruning:** Removing a sub-node from the tree is called pruning.



BYU

DECISION TREE HYPERPARAMETERS (Python)

- **max_depth**: The maximum depth of the tree. Limiting this helps prevent overfitting.
- **min_samples_split**: The minimum number of samples required to split an internal node.
- **min_samples_leaf**: The minimum number of samples required to be at a leaf node.
- **max_features**: The number of features to consider when looking for the best split.

BYU

TREE BUILDING PROCESS

1. **Start at the root**: Evaluate all possible splits based on features.

2. **Choose the best split** based on a criterion (e.g., Gini for classification, MSE for regression).

3. **Repeat** the process recursively until a stopping criterion is met (e.g., max depth, minimum samples per leaf).

Splitting Criteria:

1. **Gini Impurity** (used for classification):

$$Gini = 1 - \sum_{i=1}^{C} p_i^2$$

Where p_i is the proportion of class i in the node.

2. **Entropy** (used for classification):

$$Entropy = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

3. **Mean Squared Error (MSE)** (used for regression):

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

Where y_i is the true value and $\hat{y}_i$ is the predicted value.

GINI IMPURITY

- The **Gini Impurity** measure quantifies the "impurity" or **impurity score** of a node.
- A **lower Gini score** means a **purer** node.
- The algorithm will evaluate all possible splits and choose the one that results in the lowest Gini score.

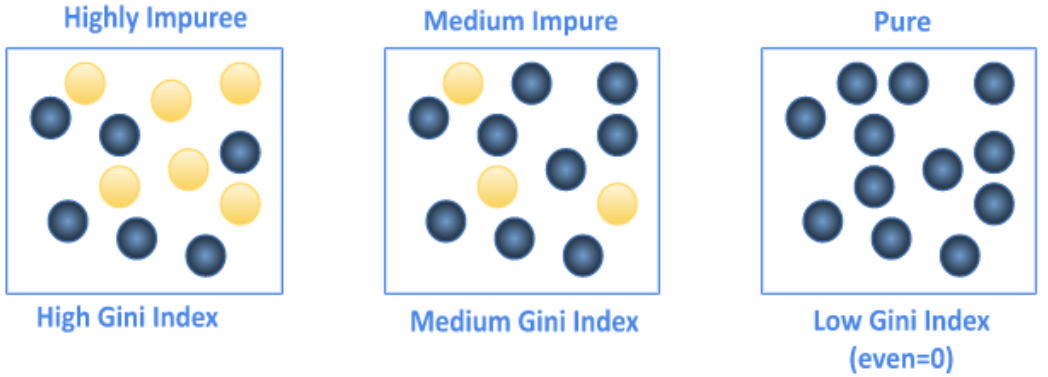
Formula:

For a set of classes, the Gini Impurity is calculated as:

$$Gini(D) = 1 - \sum_{i=1}^{C} p_i^2$$

Where:

- D is a dataset.
- C is the number of unique classes.
- p_i is the probability (or proportion) of class i in the dataset D.



If all data points in a node belong to a single class, the Gini Impurity is 0 (perfect purity).

• If the data points are evenly distributed among all classes, the Gini Impurity is at its maximum value.

Example Calculation:

Let's say we have a dataset of 10 samples belonging to two classes: Class 1 (6 samples) and Class 2 (4 samples)

1. **Proportions**:
   - $p_1 = \frac{6}{10} = 0.6$ (Class 1)
   - $p_2 = \frac{4}{10} = 0.4$ (Class 2)
2. **Gini Impurity**:

$$Gini(D) = 1 - (0.6^2 + 0.4^2) = 1 - (0.36 + 0.16) = 1 - 0.52 = 0.48$$

ENTROPY

- **Entropy** is another measure of impurity or disorder, derived from information theory. It is used in decision trees to decide the best split by evaluating how much uncertainty (or entropy) is present in the data.

The **Entropy** of a dataset is defined as:

$$Entropy(D) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

Where:

- D is a dataset.
- C is the number of unique classes.
- p_i is the proportion of class i in dataset D.

Just like with Gini, the goal in using Entropy is to minimize the uncertainty (entropy) of the child nodes after a split. The algorithm evaluates all possible splits and chooses the one that **maximizes information gain**, which is the **reduction in entropy**.

How to Interpret Entropy:

• If all samples in a node belong to the same class, entropy is 0 (pure node).

• If the samples are evenly distributed among all classes, entropy reaches its maximum, which is $\log_2(C)$ (where C is the number of classes).

Example Calculation:

Let's consider the same dataset of 10 samples with 6 samples of Class 1 and 4 samples of Class 2.

$$Entropy(D) = -(0.6 \log_2(0.6) + 0.4 \log_2(0.4))$$

Using a calculator:

$$Entropy(D) = -(0.6 \times -0.737 + 0.4 \times -1.322) = 0.442 + 0.528 = 0.970$$

Entropy vs. Gini Impurity

- Gini Impurity
  - Faster to compute (no logarithms).
  - Prefers pure splits, leading to shallower trees.
  - Default in CART (e.g., Scikit-learn).

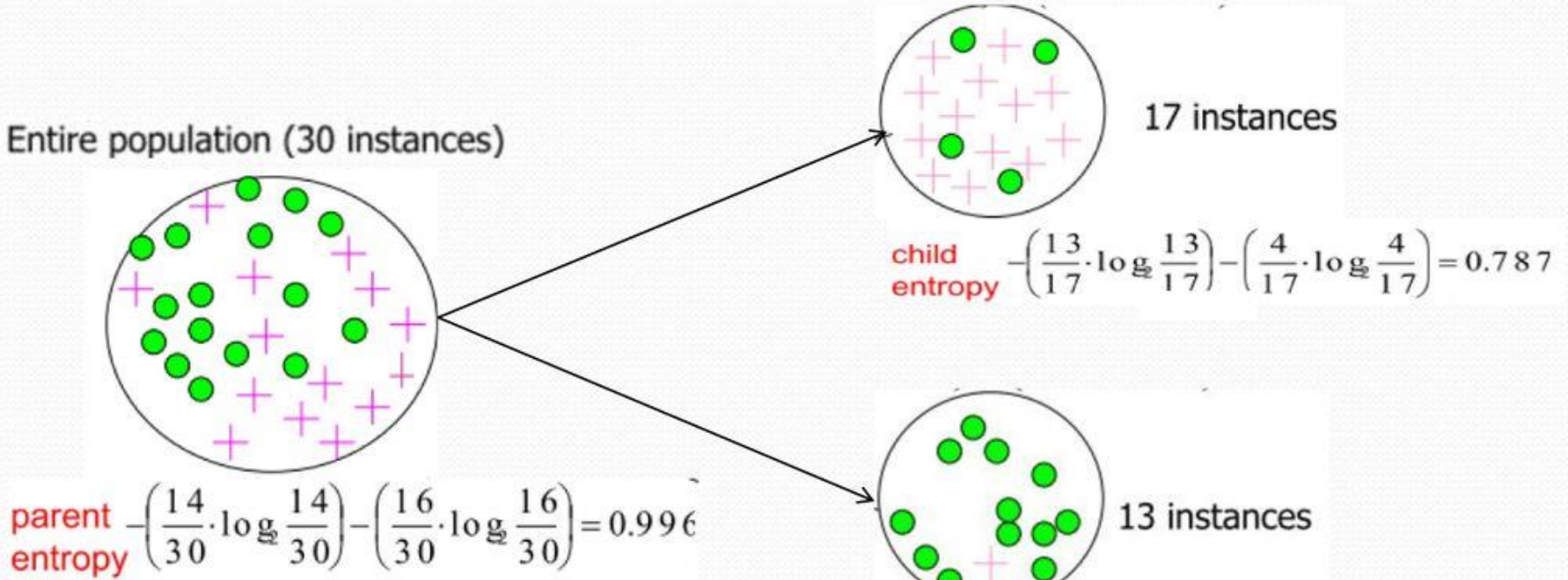
- Entropy
  - Slower (uses logarithms) but handles rare classes better.
  - Encourages more balanced splits, sometimes leading to deeper trees.
  - Used in ID3, C4.5 (Information Gain-based methods).

- Key Takeaways:
  - Use Gini when speed and simplicity matter.
  - Use Entropy for imbalanced datasets and finer splits.
  - Differences are usually small in practice.

BYU

Information gain calculation example

Information Gain = entropy(parent) – [average entropy(children)]

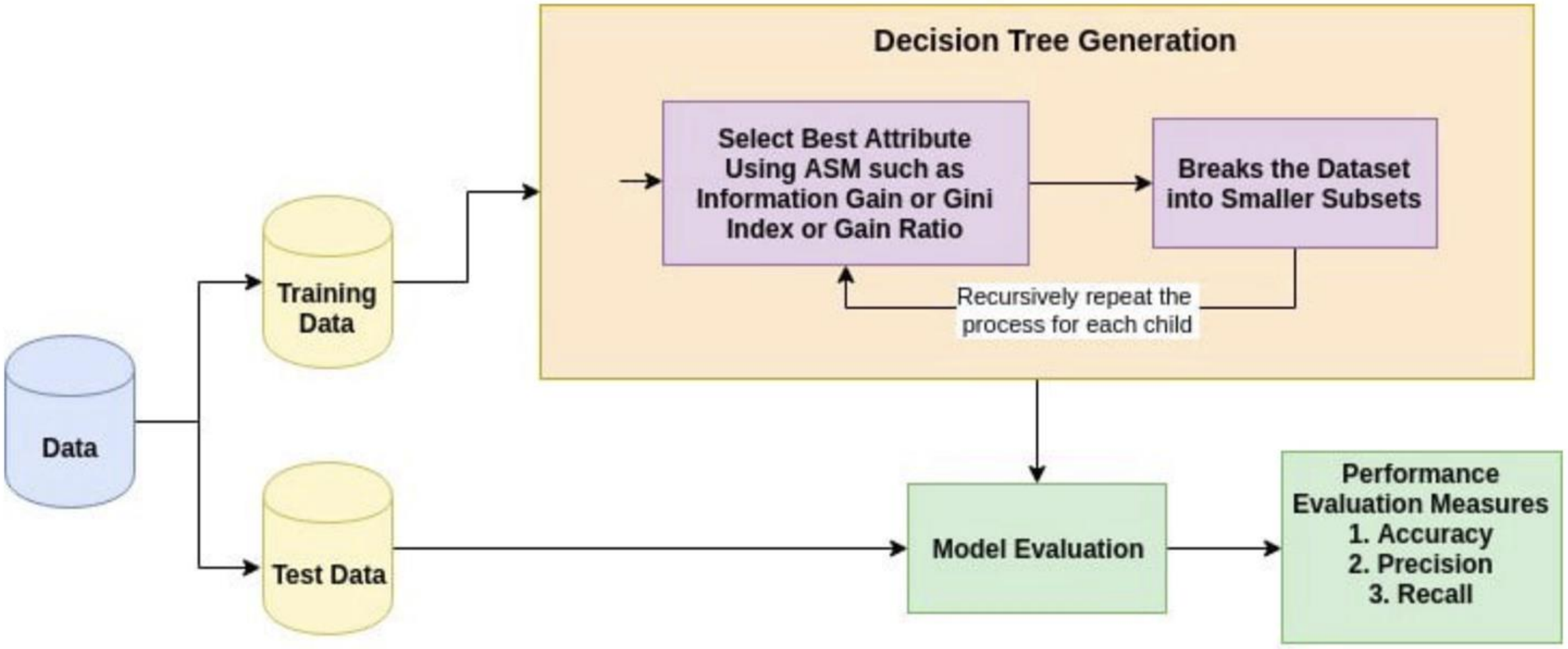


(Weighted) Average Entropy of Children = $\left(\frac{17}{30}\cdot 0.787\right)+\left(\frac{13}{30}\cdot 0.391\right)=0.615$

Information Gain= 0.996 - 0.615 = 0.38

BYU

HOW TO LEARN A DECISION TREE?



Source: DataCamp

Support Vector Machine (SVM)

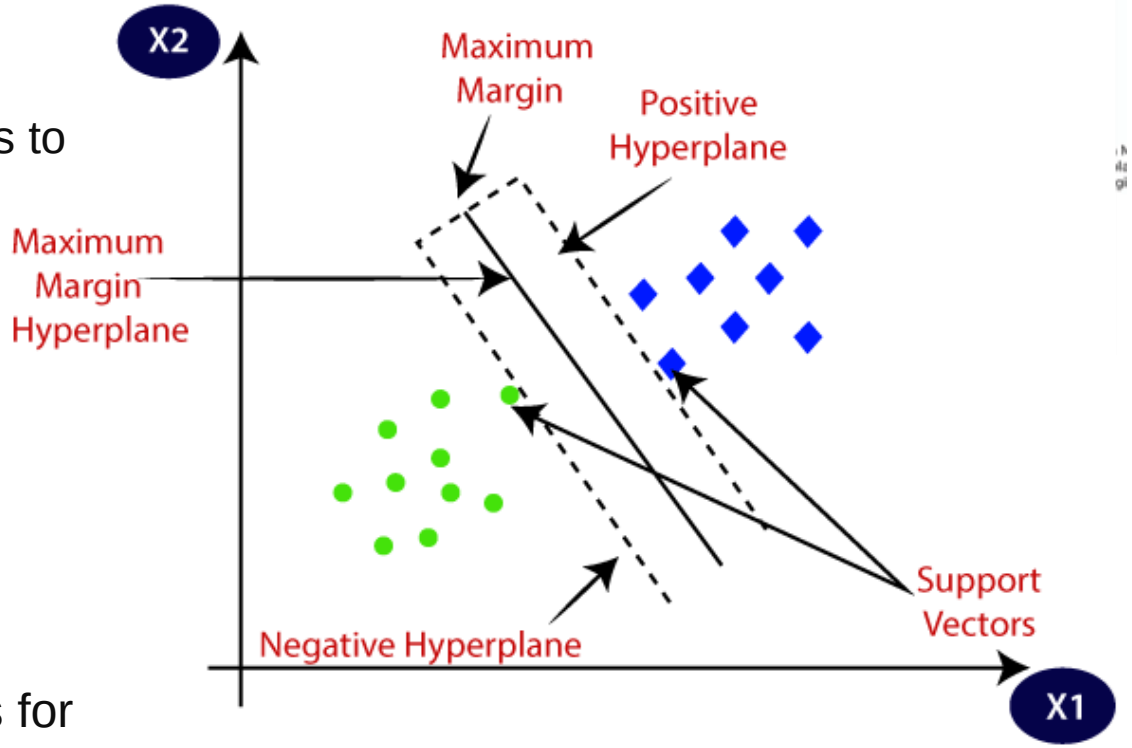
SUPPORT VECTOR MACHINE (SVM)

Core Idea: SVM finds the hyperplane that maximizes the (soft) margin between the closest data points (support vectors) of different classes.

- For non-linearly separable data, SVM uses kernel functions to project data into a higher-dimensional space where it becomes linearly separable.

Key Concepts:

- **Hyperplane**: A decision boundary.
- **Margin**: Distance between the hyperplane and the nearest data points from either class.
- **Kernel Trick**: Transforms data into higher dimensions for linear separability. (This is often used for non-linear problems)



BYU

SVM Kernel Functions

- Linear Kernel

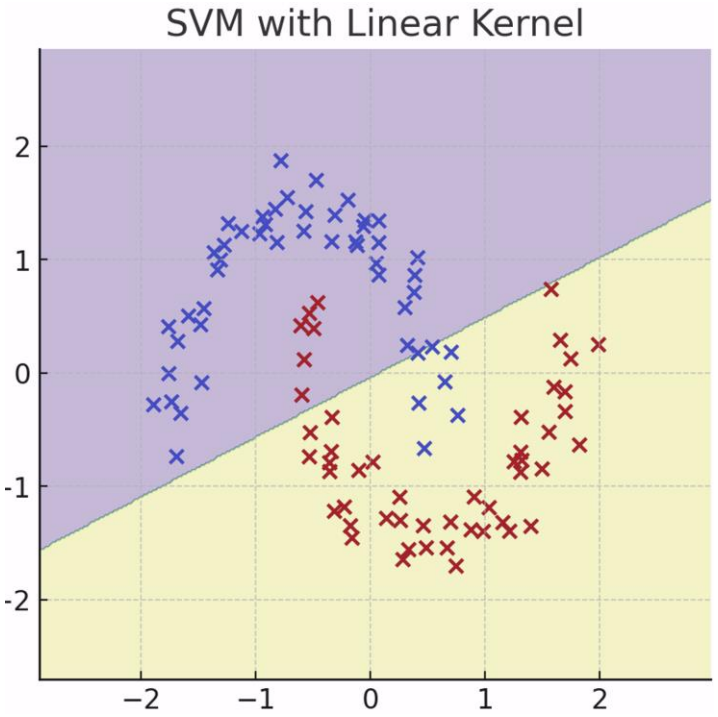
$$K(x_i, x_j) = x_i^T x_j$$

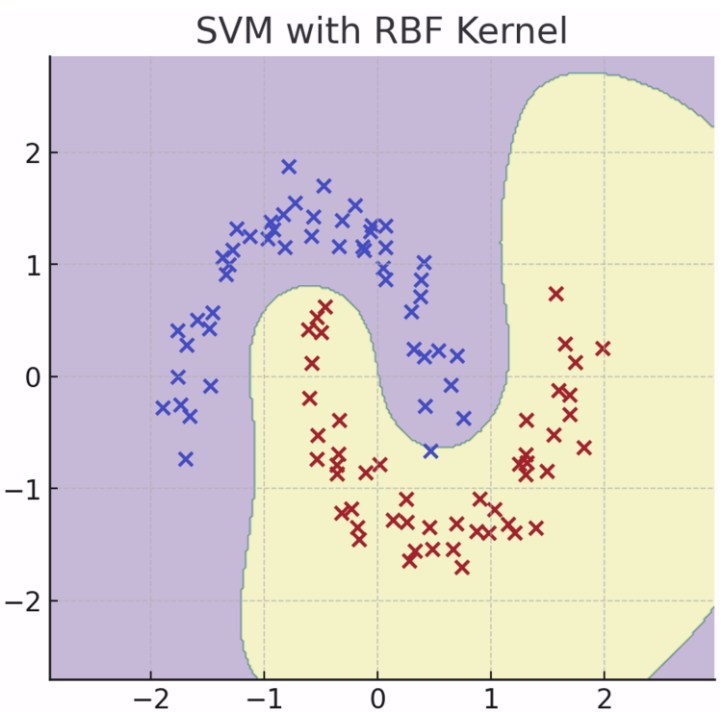
- Radial Basis Function (RBF)

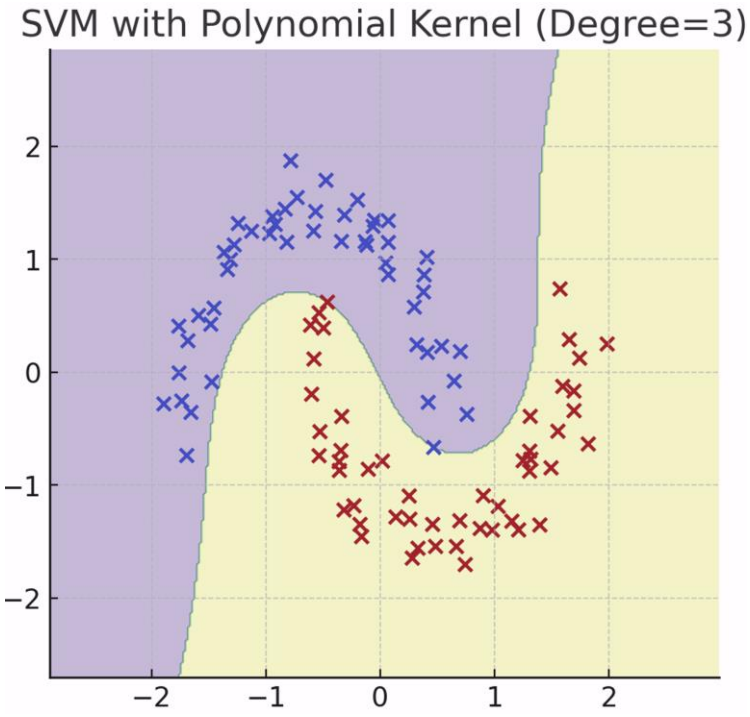
$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$$

- Polynomial Kernel

$$K(x_i, x_j) = \left(x_i^T x_j + c\right)^d$$







BYU

SVM KEY PARAMETERS

- **C**: Regularization parameter.
  - High **C** means the model tries to classify all points correctly (low margin), while
  - Low **C** allows more misclassifications (high margin).
- **kernel**: Defines the type of hyperplane (linear, rbf, poly, etc.).
- **gamma**: Defines the influence of a single training point (applicable for RBF and polynomial kernels).
- **degree**: Degree of the polynomial kernel.

Advantages:

• Effective for high-dimensional data.

• Works well with a clear margin of separation.

Disadvantages:

• Computationally expensive for large datasets.

• Choice of kernel and hyperparameters can be highly problem dependent.

BYU

Logistic Regression (Pull more from Deck 23)

LOGISTIC REGRESSION MATH BACKGROUND

- **LOGISTIC REGRESSION MATH BACKGROUND**
- Logistic Regression models the probability that an input belongs to a particular class.
- Unlike linear regression, which predicts continuous outcomes, logistic regression predicts probabilities constrained between 0 and 1.
- The logistic (sigmoid) function is defined as:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

- This maps any real-valued number into the (0, 1) interval, making it suitable for probability estimation.

- **Odds and Log-Odds:**
- Odds represent the ratio of the probability of an event occurring to it not occurring:

$$\text{Odds} = \frac{p}{1 - p}$$

- Taking the logarithm of the odds yields the log-odds (logit function):

$$\text{Logit}(p) = \log\left(\frac{p}{1 - p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n$$

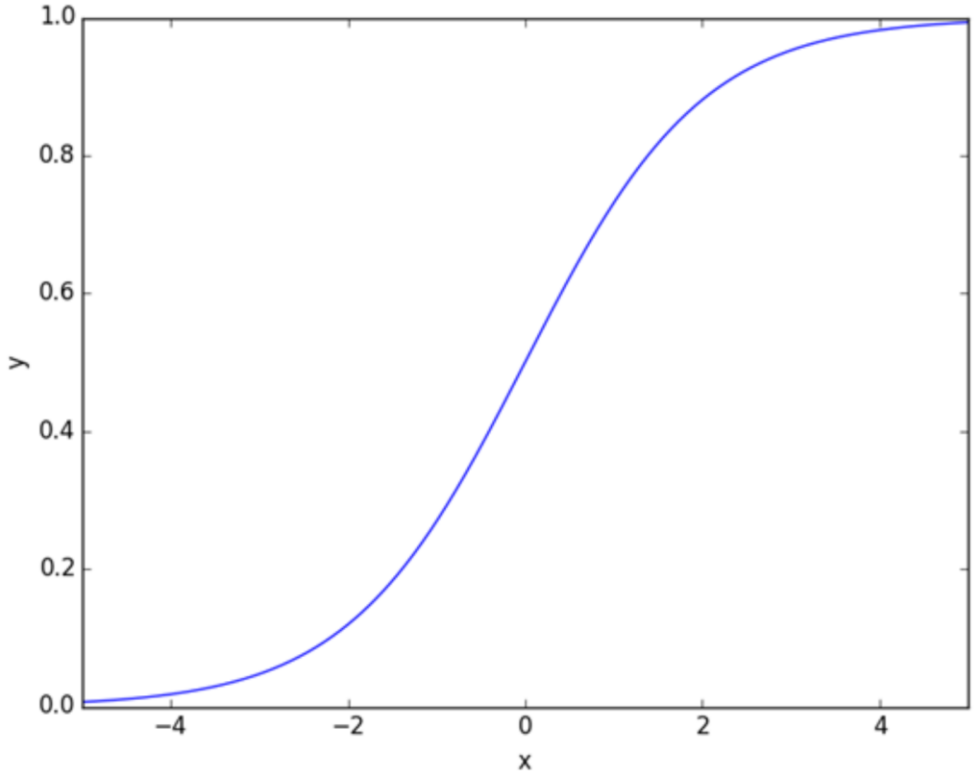
- This linear relationship between log-odds and input features forms the basis of logistic regression.

BYU

LOGISTIC REGRESSION MATH BACKGROUND

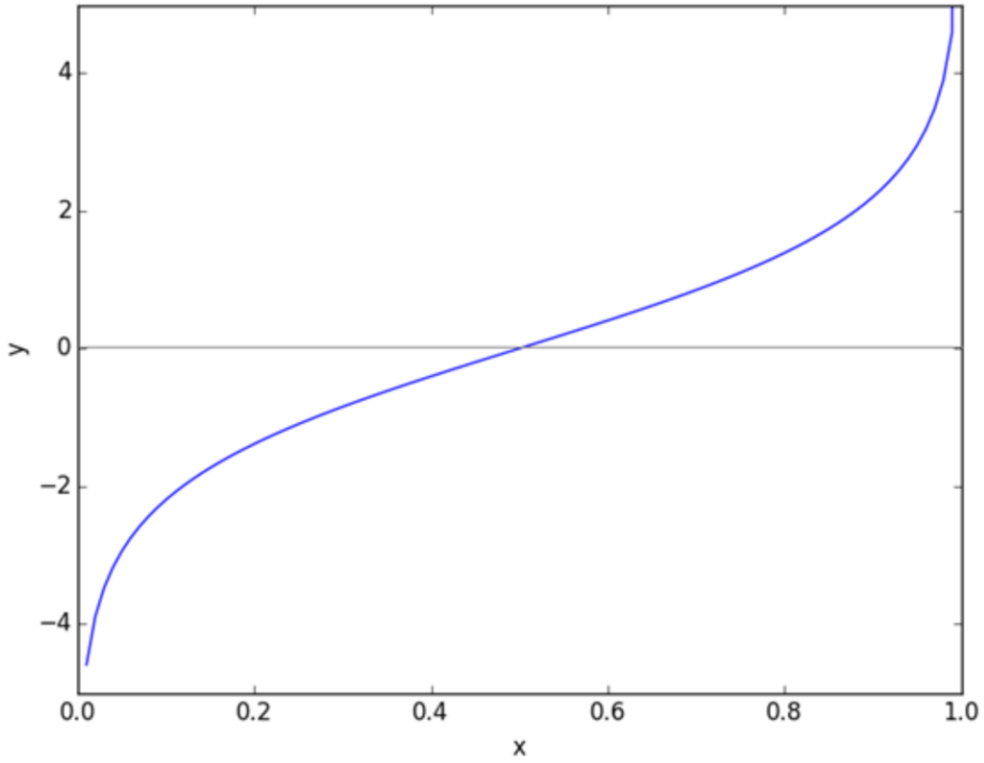
- **Sigmoid Function – Number to Probability**

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$



- **Logit Function – Probability to Number**

$$\text{Logit}(p) = \log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n$$



BYU

LOGISTIC REGRESSION

Problem Types:

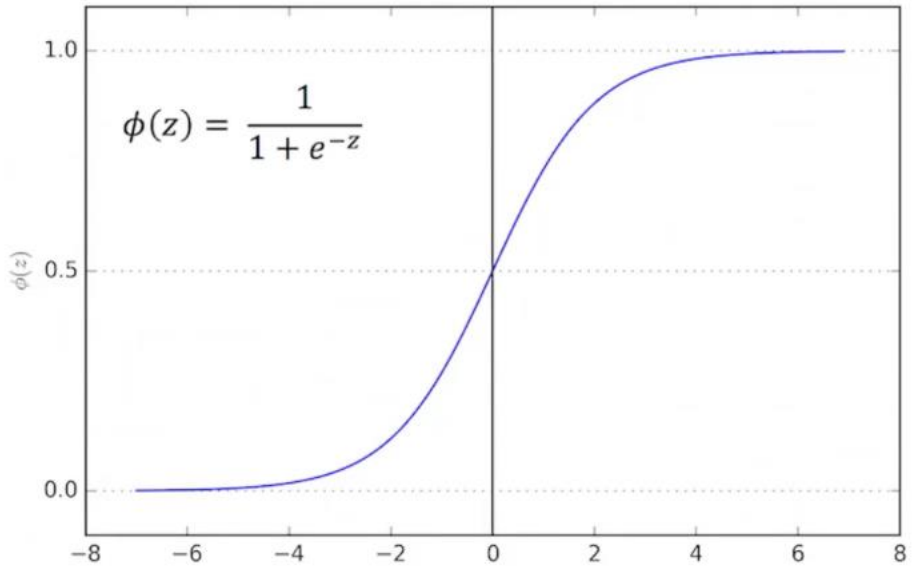
- Binary Classification: Predicts one of two possible outcomes (e.g., spam vs. non-spam emails).
- Multi-Class Classification: Extended by using techniques like One-vs-Rest or Softmax for handling multiple classes.

Advantages:

- **Simplicity**: Easy to understand and implement.
- **Interpretability**: Weights provide insights into feature importance.
- **Probabilistic Outputs**: Predicts probabilities for confidence estimation.

Limitations:

- **Linearity**: Assumes a linear relationship between input features and log-odds of the outcome.
- **Outliers**: Sensitive to outliers; regularization helps mitigate this.
- **Multicollinearity**: Correlated features can reduce interpretability.



Not suitable for predicting continuous values…ie “regression” problems.

BYU

Algorithm	Classification	Regression	Notes
Linear Regression	×	✔	Standard regression model; not suitable for classification.
Logistic Regression	✔	×	Despite its name, it's used only for classification.
Decision Trees	✔	✔	Versatile, handles both tasks by adapting the objective function.
Random Forest	✔	✔	Ensemble of decision trees; supports both tasks.
Support Vector Machines (SVM)	✔	✔	For regression, uses a variant called Support Vector Regression (SVR).
K-Nearest Neighbors (KNN)	✔	✔	Determines class (classification) or predicts value (regression) based on neighbors.
Naive Bayes	✔	×	Only suitable for classification due to probabilistic assumptions.
Gradient Boosting (e.g., XGBoost, LightGBM)	✔	✔	Boosting frameworks can handle both tasks efficiently.
Neural Networks (e.g., MLP)	✔	✔	Depending on the loss function and output layer configuration.
AdaBoost	✔	✔	Can perform both tasks but primarily used for classification.
Gaussian Processes	✔	✔	Provides probabilistic predictions for both tasks.
Linear Discriminant Analysis (LDA)	✔	×	Only designed for classification.
Ridge Regression	×	✔	A variant of linear regression with regularization.
Lasso Regression	×	✔	Regression method with feature selection through L1 regularization.
Polynomial Regression	×	✔	Extends linear regression for nonlinear relationships.
Bayesian Networks	✔	×	Primarily used for classification with probabilistic inference.