Supervised Learning Algorithms

A comprehensive guide to supervised learning algorithms for MSBTE K-Scheme diploma students. This presentation covers decision trees, KNN, SVM, linear regression, logistic regression, and model evaluation techniques with practical Python implementations.

What is Supervised Learning?

Core Concept

Supervised learning is a type of machine learning where algorithms learn from labeled training data to make predictions on new, unseen data. The algorithm learns patterns by studying examples with known outputs.

- Labeled Data: Training data with known outputs
- Features: Independent variables (X)
- Target: Dependent variable (Y)
- Training Phase: Learning patterns
- Prediction Phase: Applying learned patterns

Types of Supervised Learning

Regression

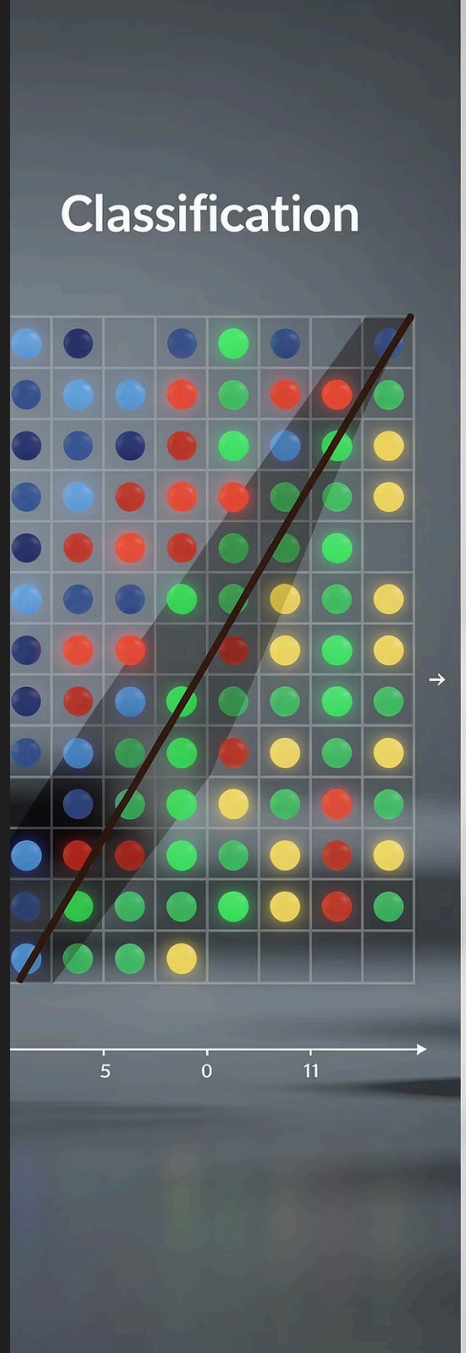
Predicts continuous numerical values

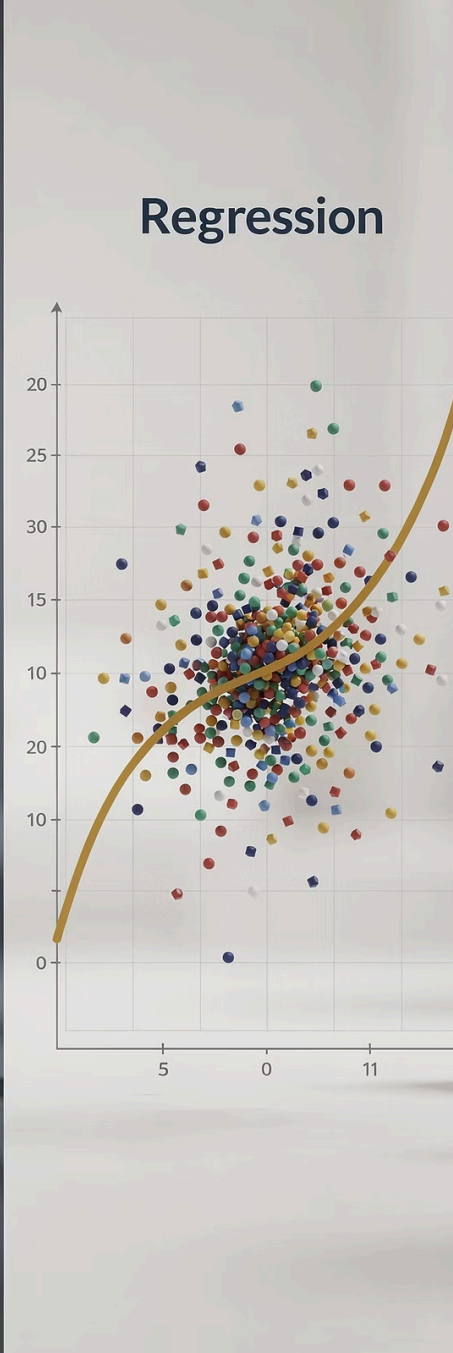
- House price prediction
- Stock price forecasting
- Temperature prediction
- Sales forecasting

Classification

Predicts discrete categories or classes

- Email spam detection
- Disease diagnosis
- Customer churn prediction
- Image recognition





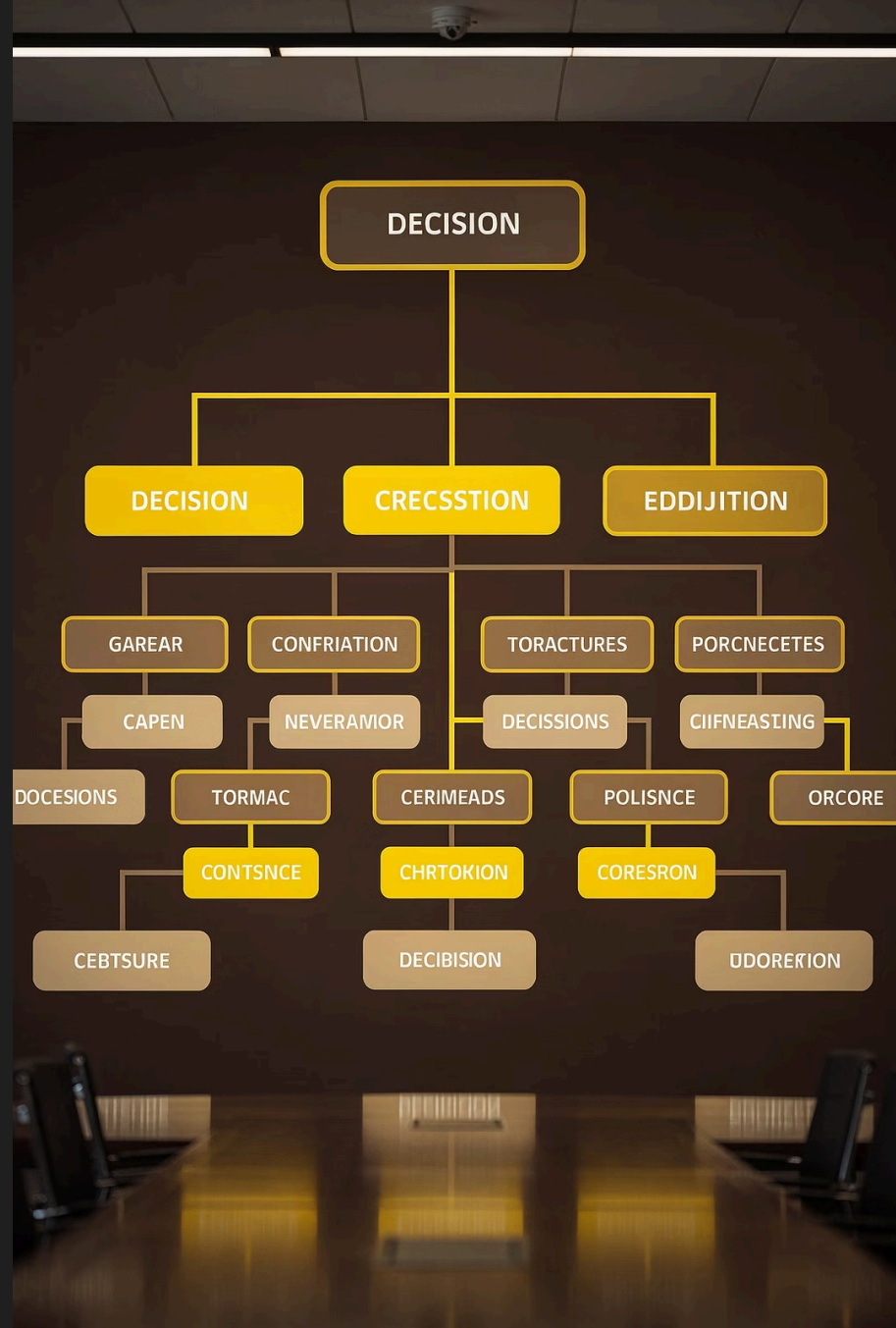
Supervised Learning Workflow

1. **Data Collection**
   Gather relevant labeled datasets for training
2. **Data Preprocessing**
   Clean, normalize, and prepare data for analysis
3. **Feature Selection**
   Identify most important variables for prediction
4. **Algorithm Selection**
   Choose appropriate model for the problem
5. **Model Training**
   Train algorithm on prepared dataset
6. **Model Evaluation**
   Test performance using validation metrics
7. **Model Deployment**
   Deploy model for real-world predictions

Decision Tree Algorithm

Decision trees are versatile algorithms that perform both classification and regression by recursively splitting data based on feature values. They create a flowchart-like structure where internal nodes represent features, branches represent decision rules, and leaf nodes represent outcomes.

The algorithm works by finding the best feature to split on at each node, partitioning data based on feature values, and repeating until stopping criteria are met. This creates an intuitive, interpretable model that mimics human decision-making.



How Decision Trees Work

01

Start at Root

Begin with all training data at the root node

02

Select Best Feature

Choose feature that best separates the classes

03

Partition Data

Split data based on selected feature values

04

Repeat Recursively

Apply same process to each subset

05

Stop When Criteria Met

End when all instances belong to same class or maximum depth reached

Splitting Criteria for Decision Trees

Classification Metrics

Gini Impurity

Measures node purity. Formula: Gini = 1 - Σ(p_i^2) where p_i is proportion of class i. Lower values indicate purer nodes.

Entropy

Measures disorder. Formula: Entropy = -Σ($p_i * \log_2(p_i)$). Used to calculate information gain.

Information Gain

Reduction in entropy after split. IG = Entropy(parent) - Σ(weighted_entropy(children))

Regression Metrics

Mean Squared Error

Measures average squared difference between predicted and actual values. Commonly used for regression trees.

Mean Absolute Error

Measures average absolute difference. More robust to outliers than MSE.

Pruning Techniques

Pre-pruning (Early Stopping)

Stop growing tree before it becomes too complex by setting constraints during construction.

- Maximum tree depth
- Minimum samples per split
- Minimum samples per leaf
- Maximum number of leaf nodes

Post-pruning

Grow full tree first, then remove branches that don't improve performance.

- Reduced Error Pruning: Remove branches if accuracy improves
- Cost Complexity Pruning: Balance accuracy vs complexity
- Prevents overfitting on training data

K-Nearest Neighbors (KNN)

KNN is a simple, instance-based learning algorithm that classifies new instances based on similarity to training instances. It's a "lazy learner" that stores all training data and makes decisions at prediction time by finding the K closest neighbors.

The algorithm calculates distances between the new point and all training points, selects K nearest neighbors, and uses majority voting (classification) or averaging (regression) to make predictions. No explicit training phase is required.

How KNN Works

Store Training Data

Memorize all training examples with their labels

Find K Neighbors

Select K closest training instances based on calculated distances

Calculate Distance

Compute distance between new instance and all training instances using chosen metric

Make Prediction

Classification: majority vote among neighbors. Regression: average of neighbor values

Distance Metrics in KNN

Euclidean Distance (L2)

Most common metric. Calculates straight-line distance: $d = \sqrt{(\Sigma(x_i - y_i)^2)}$

Manhattan Distance (L1)

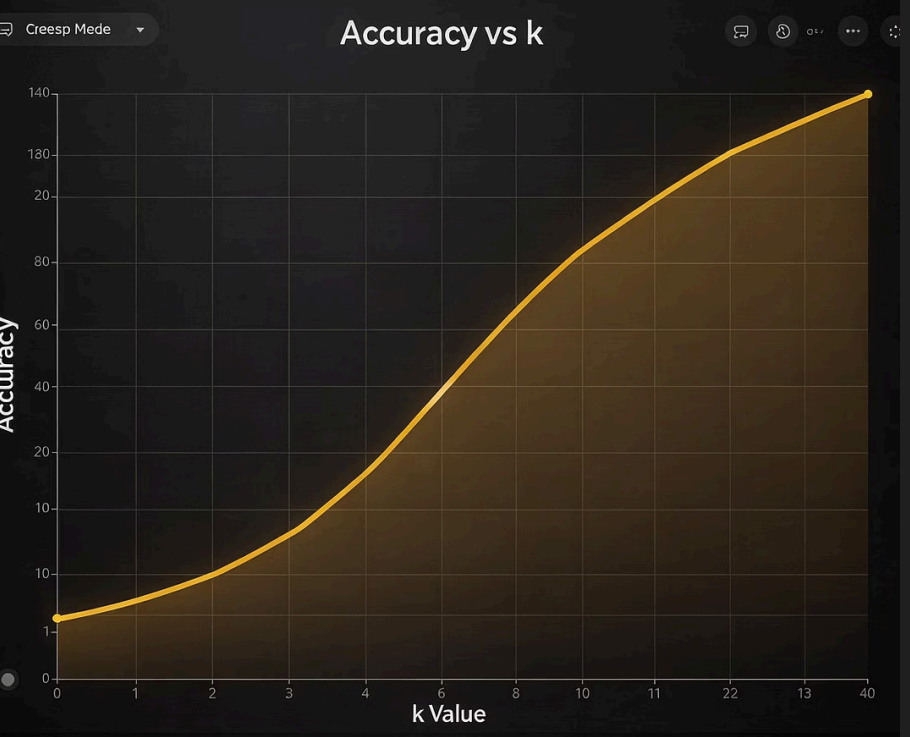
Sum of absolute differences: $d = \Sigma|x_i - y_i|$. Useful for grid-like paths.

Minkowski Distance (Lp)

Generalization of Euclidean and Manhattan: $d = (\Sigma|x_i - y_i|^p)^{(1/p)}$

Hamming Distance

For categorical variables. Counts positions where values differ.



Choosing the Right K Value

Small K (e.g., K=1)

Advantages:

- Captures local patterns
- Low bias
- Flexible boundaries

Disadvantages:

- High variance
- Sensitive to noise
- Overfitting risk

Large K (e.g., K=20)

Advantages:

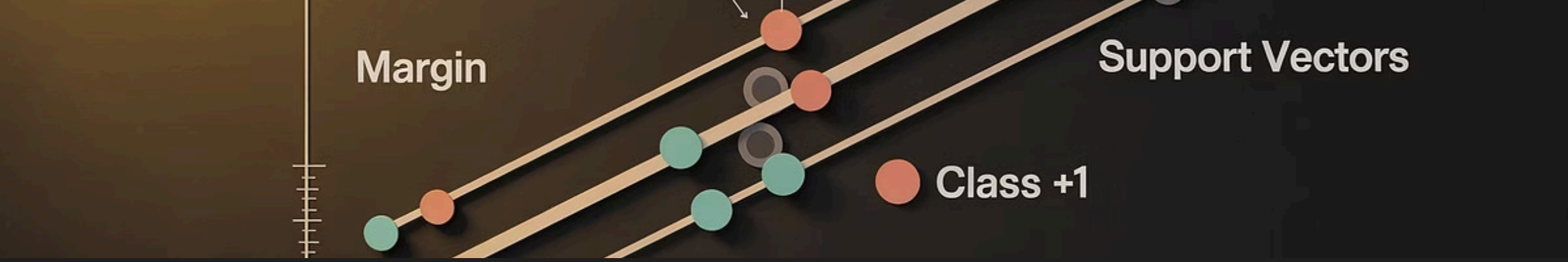
- More stable predictions
- Less sensitive to noise
- Smoother boundaries

Disadvantages:

- May miss local patterns
- Higher bias
- Underfitting risk

Finding Optimal K

- Use odd K to avoid ties in binary classification
- Apply cross-validation to test different K values
- Rule of thumb: K = √(n) where n is sample size
- Plot accuracy vs K to visualize performance



Support Vector Machine (SVM)

SVM is a powerful supervised learning algorithm that finds the optimal hyperplane to separate classes in feature space. It maximizes the margin between classes, making it robust and effective for complex classification tasks.

Support vectors are the data points closest to the hyperplane that define the margin. The algorithm focuses on these critical points rather than all training data, making it memory-efficient and effective even in high-dimensional spaces.

Linear vs Non-Linear SVM

Linear SVM

Used when data is linearly separable. Finds a straight hyperplane that maximizes margin between classes.

Mathematical Formulation:

Minimize: $(1/2)||w||^2$ subject to: $y_i(w \cdot x_i + b) \geq 1$

Best for: Linearly separable data, text classification, simple boundaries

Non-Linear SVM

Uses kernel trick to transform data into higher-dimensional space where linear separation is possible.

Kernel Functions:

- Polynomial: $K(x,y) = (x \cdot y + c)^d$
- RBF: $K(x,y) = \exp(-\gamma||x-y||^2)$
- Sigmoid: $K(x,y) = \tanh(\alpha(x \cdot y) + c)$

Best for: Complex non-linear boundaries, image recognition

SVM Kernel Functions

Linear Kernel

No transformation. Best for linearly separable data. Fast and interpretable. No additional parameters needed.

Polynomial Kernel

Creates polynomial decision boundaries. Parameters: degree and coef0. Good for non-linear but smooth boundaries.

RBF (Radial Basis Function)

Most popular for complex boundaries. Parameter: gamma controls influence. Smaller gamma = smoother, larger = more complex.

Sigmoid Kernel

Neural network-like behavior. Parameters: coef0. Similar to neural network activation functions.

SVM Hyperparameters

C Parameter

Regularization parameter that controls trade-off between margin maximization and misclassification.

- **Small C:** Wider margin, more misclassification tolerance, simpler model
- **Large C:** Narrower margin, less misclassification tolerance, complex model
- Default: C = 1.0

Gamma Parameter

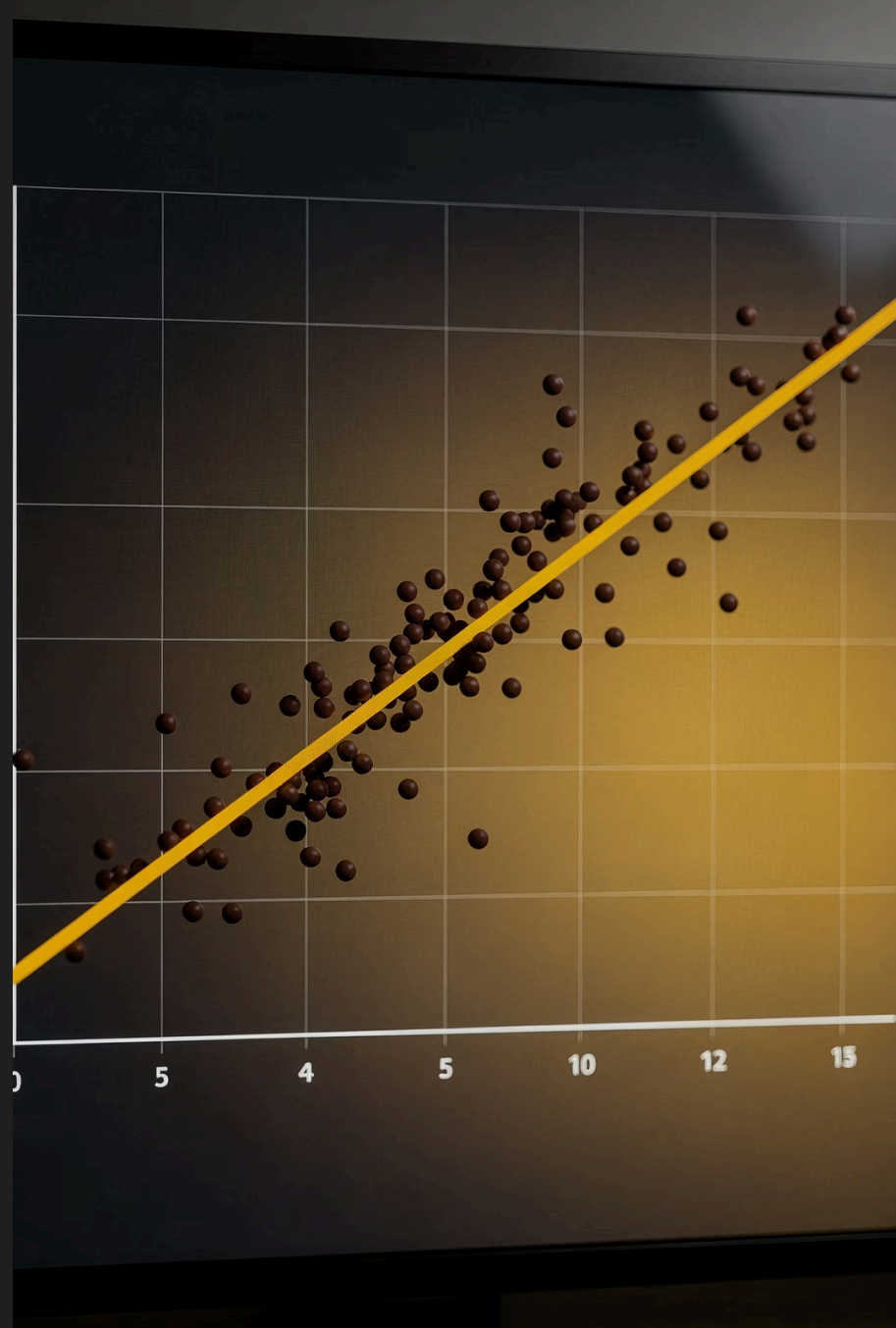
Kernel coefficient for RBF, polynomial, and sigmoid kernels. Defines influence of single training example.

- **Small gamma:** Far reach, smoother decision boundary
- **Large gamma:** Close reach, more complex boundary
- Options: 'scale' (default) or 'auto'

Linear Regression

Linear regression models the relationship between dependent and independent variables by fitting a linear equation to observed data. It's the foundation of predictive modeling for continuous outcomes.

The algorithm finds the best-fitting straight line through data points by minimizing the sum of squared differences between predicted and actual values. Simple linear regression uses one independent variable, while multiple linear regression uses several.



Linear Regression Fundamentals

Simple Linear Regression

Model: $y = mx + c$

- y: dependent variable (target)
- x: independent variable (feature)
- m: slope/coefficient
- c: intercept

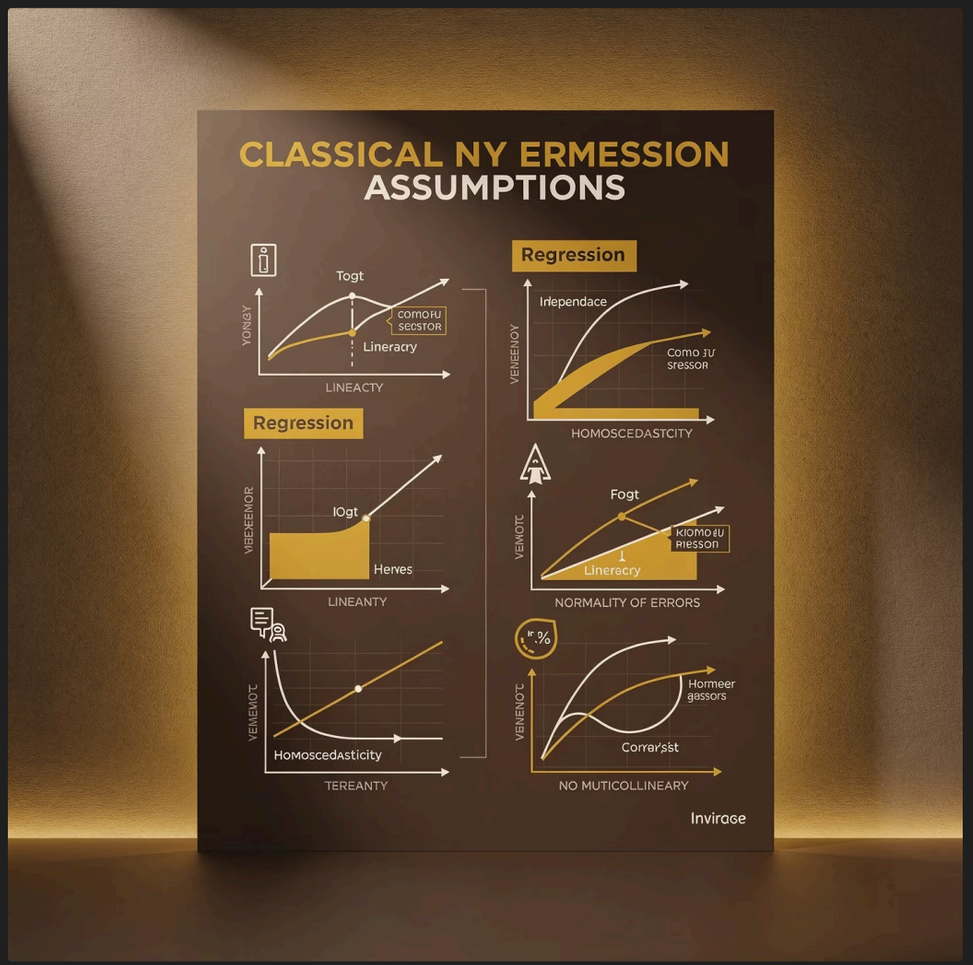
Multiple Linear Regression

Model: $y = b_0 + b_1x_1 + b_2x_2 + ... + b_nx_n + \varepsilon$

Matrix form: $y = X\beta + \varepsilon$

Key Assumptions

1. **Linearity:** Linear relationship between variables
2. **Independence:** Observations are independent
3. **Homoscedasticity:** Constant variance of residuals
4. **Normality:** Residuals are normally distributed



Cost Function and Optimization

01

Define Cost Function

Mean Squared Error (MSE) = $(1/n) \Sigma(y_i - \hat{y}_i)^2$. Measures average squared difference between predictions and actual values.

02

Initialize Parameters

Start with random or zero values for coefficients (θ). These will be optimized during training.

03

Compute Predictions

Calculate predicted values ($\hat{y}$) using current parameters: $\hat{y} = X\theta$

04

Calculate Gradient

Compute partial derivatives of cost function with respect to each parameter: $\partial J/\partial \theta$

05

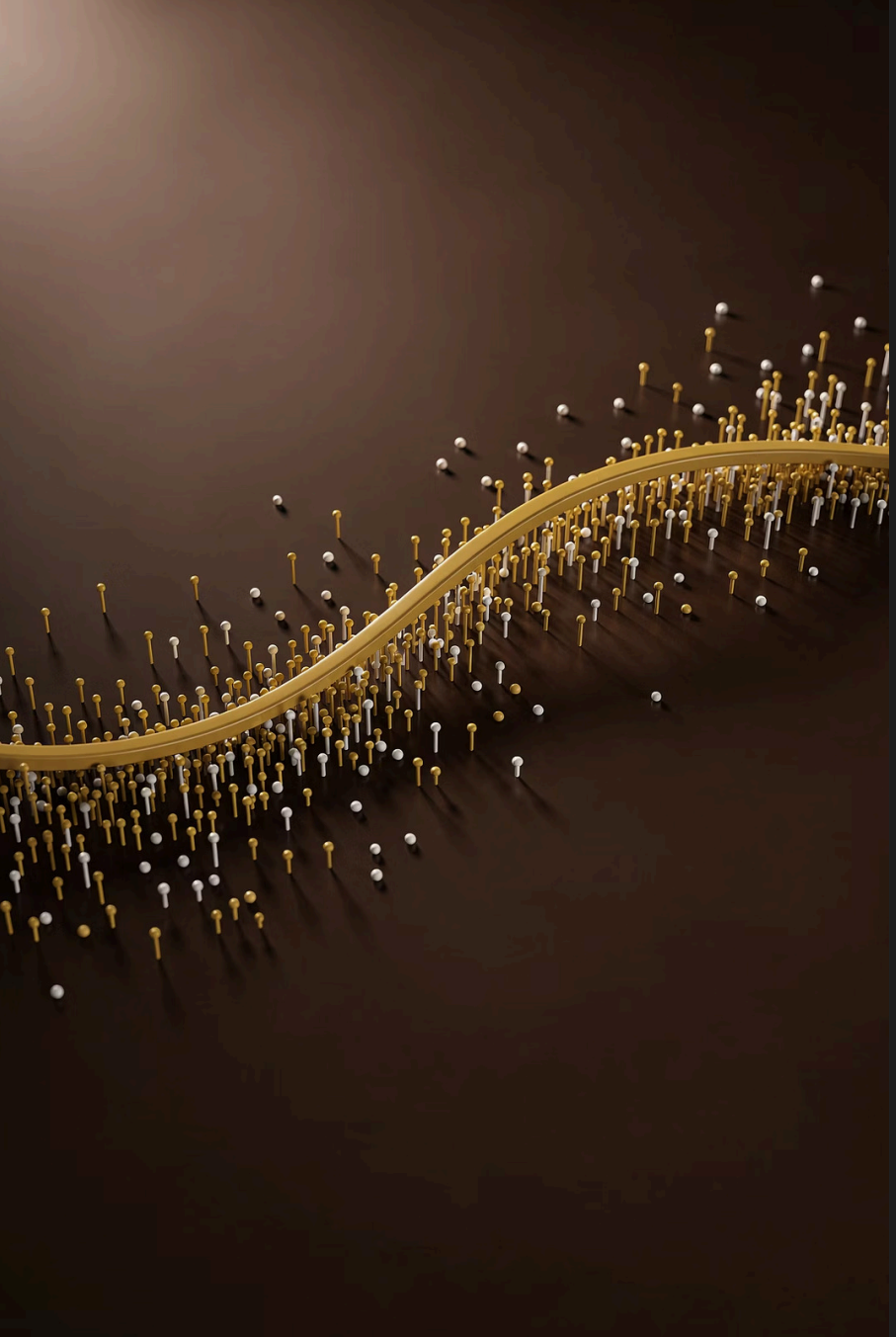
Update Parameters

Adjust parameters in direction of steepest descent: $\theta = \theta - \alpha * \partial J/\partial \theta$ where α is learning rate

06

Repeat Until Convergence

Continue iterations until cost function stops decreasing significantly or maximum iterations reached



Logistic Regression

Logistic regression is a classification algorithm that predicts the probability of a binary outcome using the sigmoid function. Despite its name, it's used for classification, not regression.

The algorithm transforms linear combinations of features through the sigmoid function to produce probabilities between 0 and 1. A threshold (typically 0.5) converts probabilities to class predictions, making it ideal for binary classification tasks like spam detection or disease diagnosis.

Sigmoid Function and Cost Function

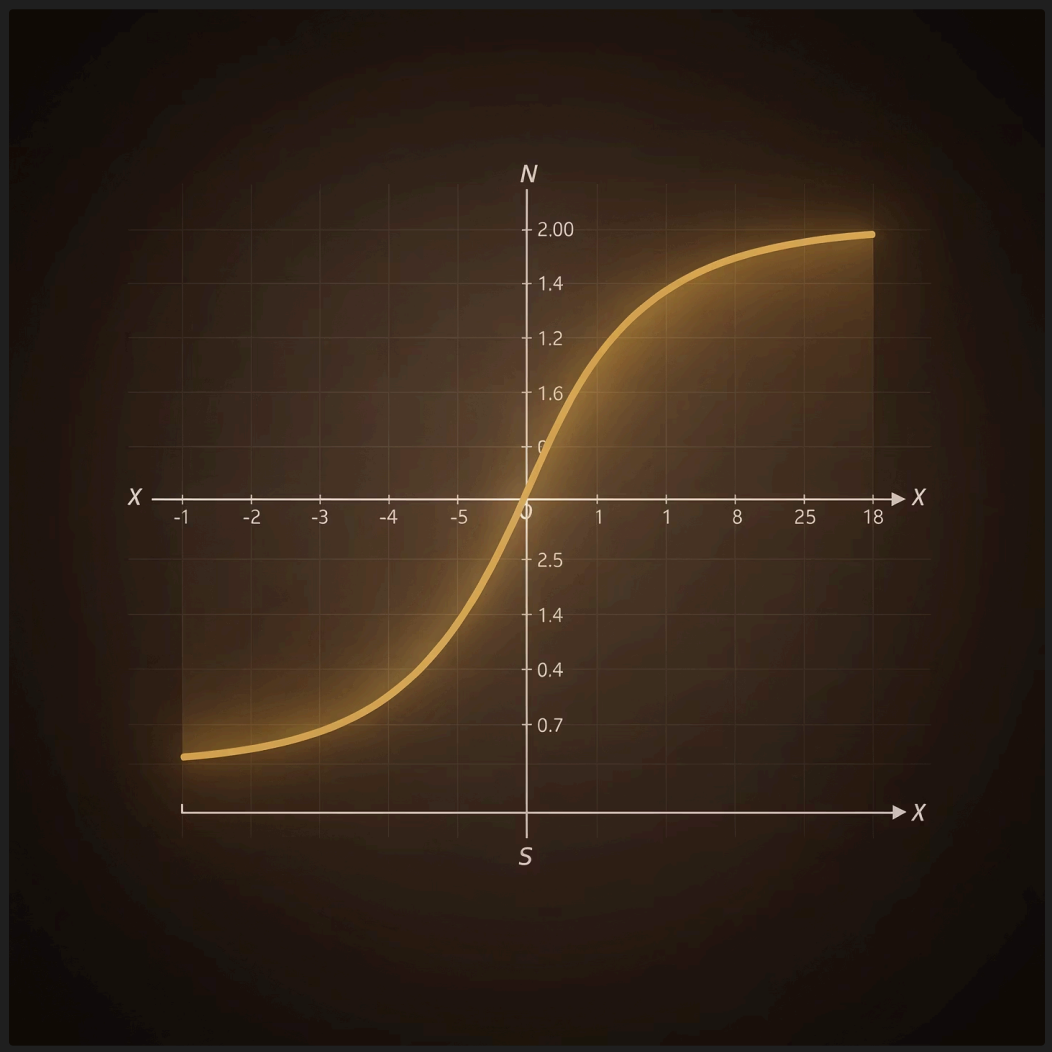
Sigmoid Function

Formula: σ(z) = 1 / (1 + e^(-z))

Where z = w·x + b

Properties

- Output range: (0, 1)
- S-shaped curve
- Differentiable everywhere
- Maps any real value to probability



Log Loss (Binary Cross-Entropy)

Formula: J = -(1/n) Σ[y_i log(ŷ_i) + (1-y_i) log(1-ŷ_i)]

Why Not MSE?

- MSE creates non-convex optimization landscape
- Multiple local minima make optimization difficult
- Log loss is convex for logistic regression
- Better suited for probabilistic interpretation

Multi-class Classification

1

One-vs-Rest (OvR)

Train n binary classifiers for n classes. Each classifier predicts probability of belonging to one class vs all others. Choose class with highest probability.

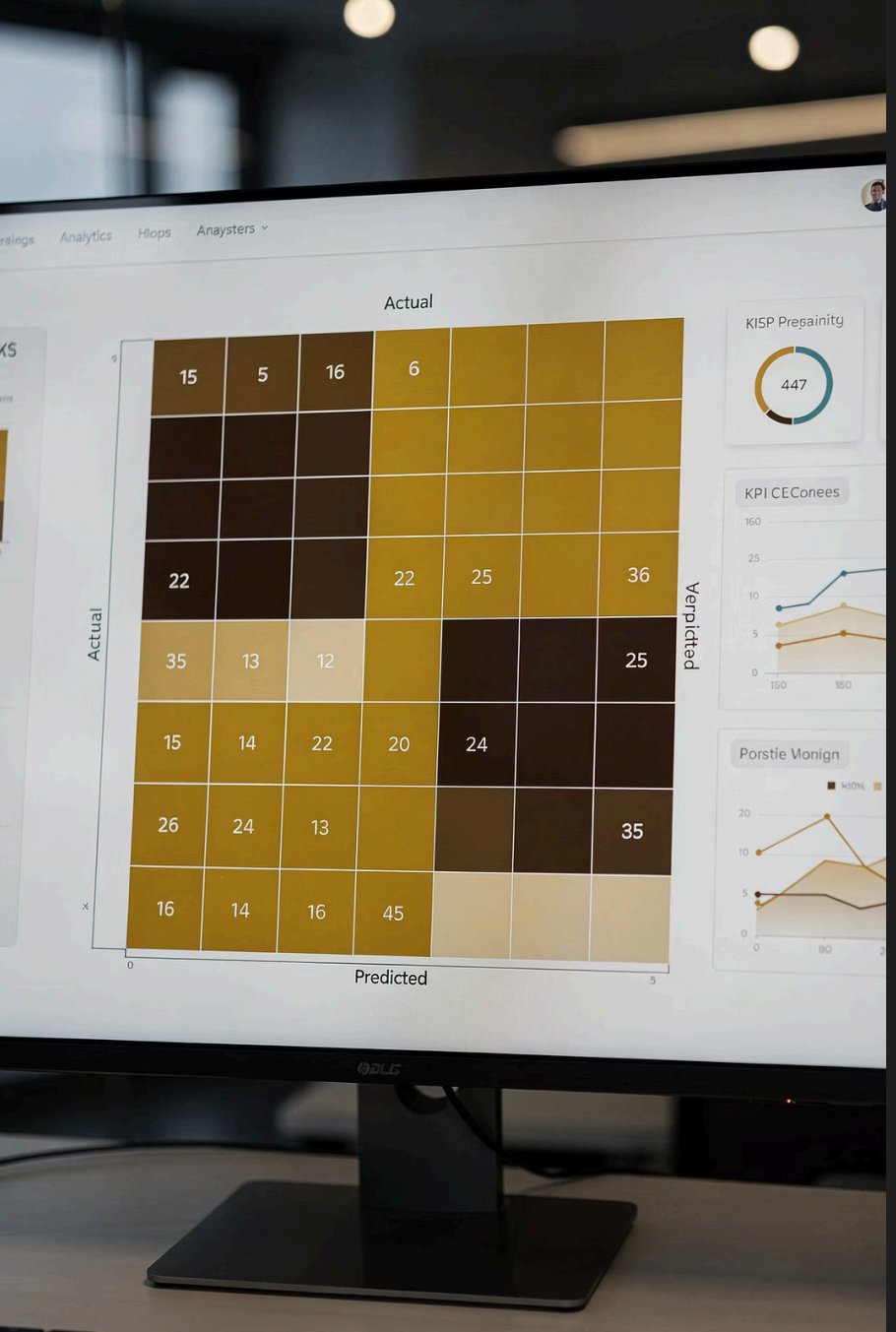
Example: For 3 classes (A, B, C), train classifiers: A vs (B,C), B vs (A,C), C vs (A,B)

2

Softmax Regression

Generalization of logistic regression for multiple classes. Uses softmax function instead of sigmoid to produce probability distribution over all classes.

Formula: $P(y=k|x) = \exp(z_k) / \Sigma\exp(z_j)$ where $z = Wx + b$



Model Evaluation Metrics

Evaluating model performance requires appropriate metrics based on the problem type. Classification and regression tasks use different evaluation approaches to measure accuracy, precision, and overall effectiveness.

Classification Metrics

Accuracy

Proportion of correct predictions: (TP + TN) / (TP + TN + FP + FN)

Best for balanced datasets

Precision

Proportion of positive predictions that are correct: TP / (TP + FP)

Important when false positives are costly

Recall (Sensitivity)

Proportion of actual positives correctly identified: TP / (TP + FN)

Important when false negatives are costly

F1-Score

Harmonic mean of precision and recall: 2 * (Precision * Recall) / (Precision + Recall)

Balances precision and recall

Regression Metrics

Mean Absolute Error (MAE)

Average absolute difference between predictions and actual values: $MAE = (1/n) \Sigma |y_i - \hat{y}_i|$

Easy to interpret, same units as target variable. Robust to outliers.

Mean Squared Error (MSE)

Average squared difference: $MSE = (1/n) \Sigma (y_i - \hat{y}_i)^2$

Penalizes large errors more heavily. Commonly used for optimization.

Root Mean Squared Error (RMSE)

Square root of MSE: $RMSE = \sqrt{MSE}$

Same units as target variable. More interpretable than MSE.

R^2 Score

Coefficient of determination: $R^2 = 1 - (SS_{res} / SS_{tot})$

Proportion of variance explained. Range: 0 to 1 (higher is better).

Confusion Matrix and ROC Curve

Confusion Matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Visualizes classification performance across all classes. Shows types of errors made by model.

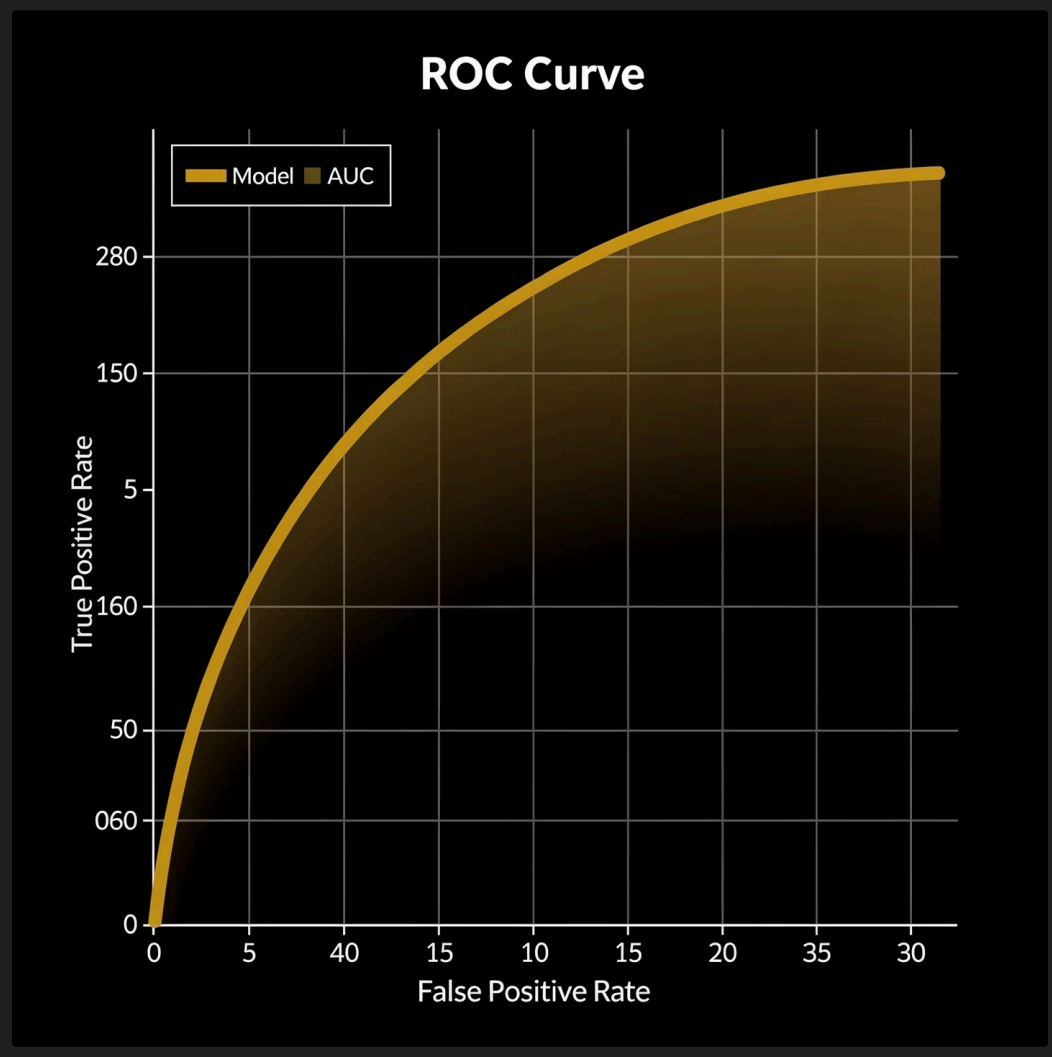
ROC Curve and AUC

ROC Curve: Plots True Positive Rate vs False Positive Rate at various thresholds

- Diagonal line = random classifier
- Curve closer to top-left = better

AUC (Area Under Curve):

- AUC = 1: Perfect classifier
- AUC = 0.5: Random classifier
- AUC = 0: Worst classifier



Algorithm Comparison

Algorithm	Type	Advantages	Disadvantages	Best Use
Decision Tree	Both	Interpretable, handles mixed data, no scaling needed	Prone to overfitting, unstable with small changes	Interpretability needed
KNN	Both	Simple, no training phase, non-parametric	Slow prediction, memory intensive, sensitive to scale	Small datasets
SVM	Both	Effective in high dimensions, memory efficient	Slow training, kernel selection challenging	Text classification
Linear Regression	Regression	Simple, fast, interpretable	Assumes linearity, sensitive to outliers	Linear relationships
Logistic Regression	Classification	Probabilistic output, fast, interpretable	Assumes linear decision boundary	Binary classification

Real-World Applications

Decision Trees

Customer churn prediction in telecom, credit risk assessment in banking, medical diagnosis systems, fraud detection in finance, and loan approval decisions.

K-Nearest Neighbors

Recommender systems for e-commerce, image recognition and classification, pattern recognition in biometrics, anomaly detection in cybersecurity.

Support Vector Machine

Text classification and sentiment analysis, face detection in security systems, bioinformatics for protein classification, handwriting recognition in OCR.

Linear Regression

House price prediction in real estate, sales forecasting for businesses, stock price analysis, weather temperature prediction.

Logistic Regression

Email spam filtering, disease prediction in healthcare, customer conversion prediction in marketing, credit default prediction in finance.

Python Implementation Tips

1

Import Libraries

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LinearRegression,
LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
mean_squared_error
```

2

Prepare Data

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
# Scale features if needed
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

3

Train Model

```
model = DecisionTreeClassifier(max_depth=5)
# or KNeighborsClassifier(n_neighbors=5)
# or SVC(kernel='rbf', C=1.0)
model.fit(X_train, y_train)
```

4

Evaluate Performance

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.3f}")
# Use appropriate metrics for your task
```

Key Takeaways

Decision Trees

Intuitive and interpretable but prone to overfitting. Use pruning techniques and ensemble methods for better performance.

KNN

Simple lazy learner sensitive to K value and feature scaling. Best for small datasets with clear patterns.

SVM

Powerful for complex boundaries using kernel trick. Requires careful hyperparameter tuning for optimal results.

Linear Regression

Foundation for regression tasks assuming linear relationships. Fast and interpretable for continuous predictions.

Logistic Regression

Go-to algorithm for binary classification providing probability outputs. Simple yet effective for many tasks.

Evaluation

Choose metrics based on problem type and business requirements. Balance multiple metrics for comprehensive assessment.