Since 2011

RIG
Realistic
Infotech
Group

Our outcomes are over 5000 trainees.

Get IT Right from RIG

Artificial Intelligence Engineering (Level-1)

Learn, Create, and Shine

Digital Space

Level-1

- Module 1: Introduction to AI and Machine Learning
- Module 2: Linear Algebra, Statistics and Probability for AI
- Module 3: Neural Network Architecture
- Module 4: Building Machine Learning Models
- Module 5: Deep Learning Concepts
- Module 6: Python Data Structure
- Module 7: Data Handling with Pandas and NumPy
- Module 8: Python for AI
- Module 9: Classification AI Project
- Module 10: Prediction AI Project

Artificial Intelligence Engineering (Level-1)

Module 9: Classification AI Project

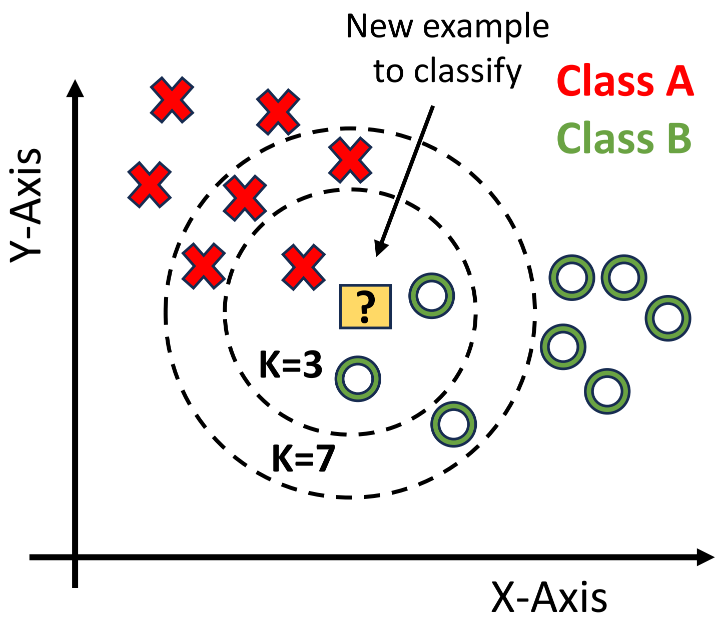
Content

- K-Nearest Neighbors (KNN)
- Decision Tree Classifier
- Implementation Steps

K-Nearest Neighbors (KNN)

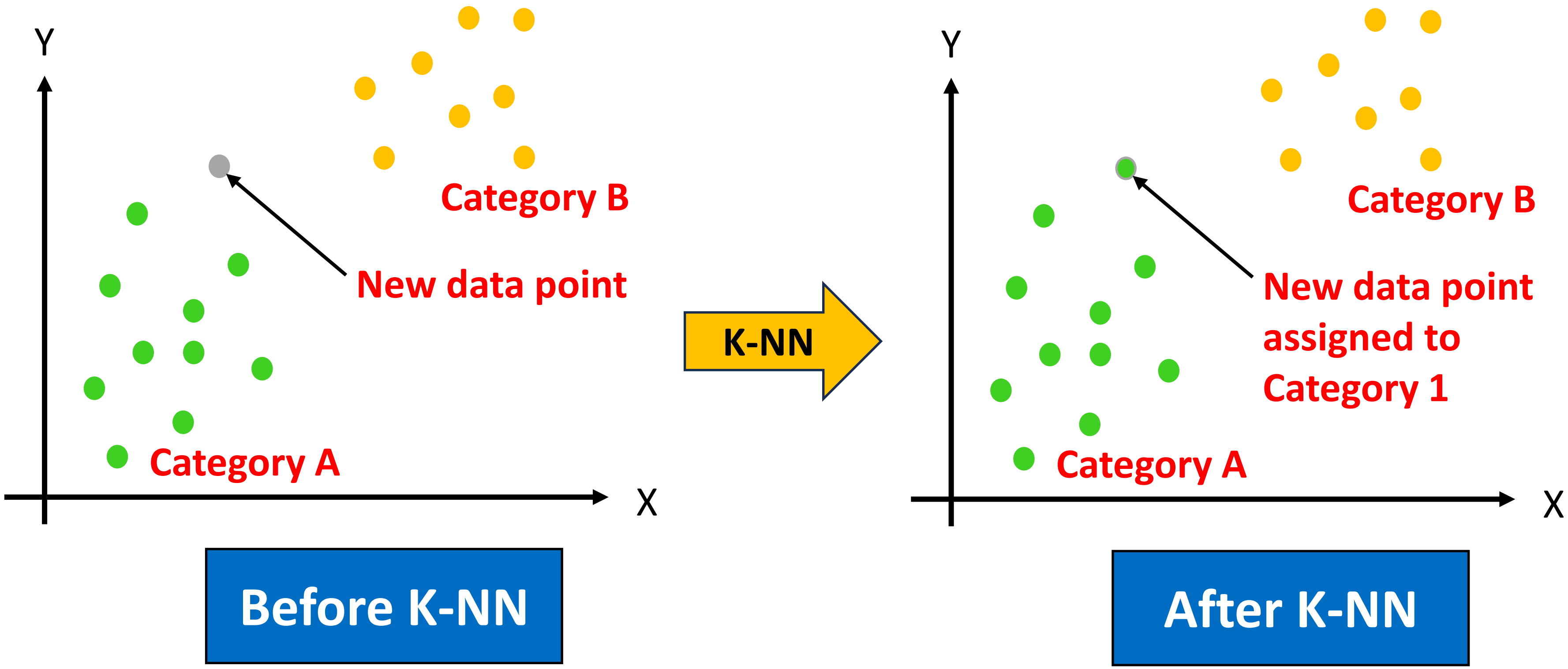
What is K-Nearest Neighbors?

A simple, instance-based, supervised learning algorithm used for both classification and regression.



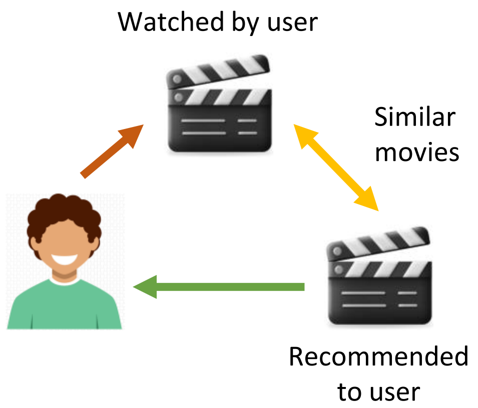
How KNN work?

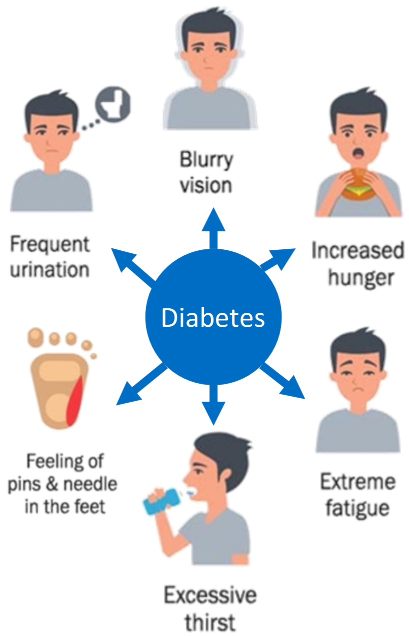
- Calculates the distance between data points.
- Finds the 'K' closest neighbors to the query point.
- Classifies the point based on the majority class of its neighbors (for classification).

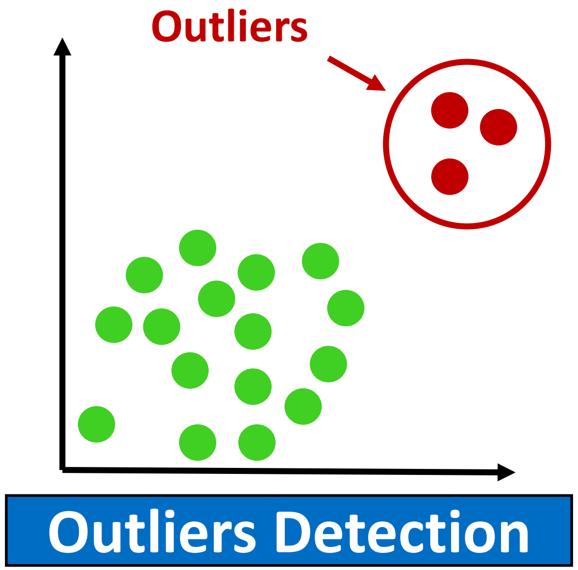


Use Cases of KNN work

- **Real-world applications**:
  - **Recommendation Systems**: Suggest items based on user similarity.
  - **Medical Diagnosis**: Classify diseases based on symptoms.
  - **Image Recognition**: Classify images based on pixel intensity.
  - **Anomaly Detection**: Detect outliers in financial data or network security.





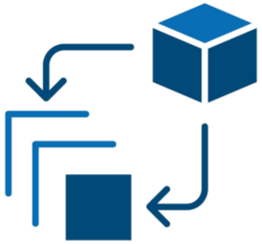


Data Preprocessing for KNN

- **Feature scaling**:
    - Importance of normalizing/standardizing features.
    - Techniques: MinMaxScaler, StandardScaler.

- **Handling missing values:**
    - Filling missing data or removing rows/columns.

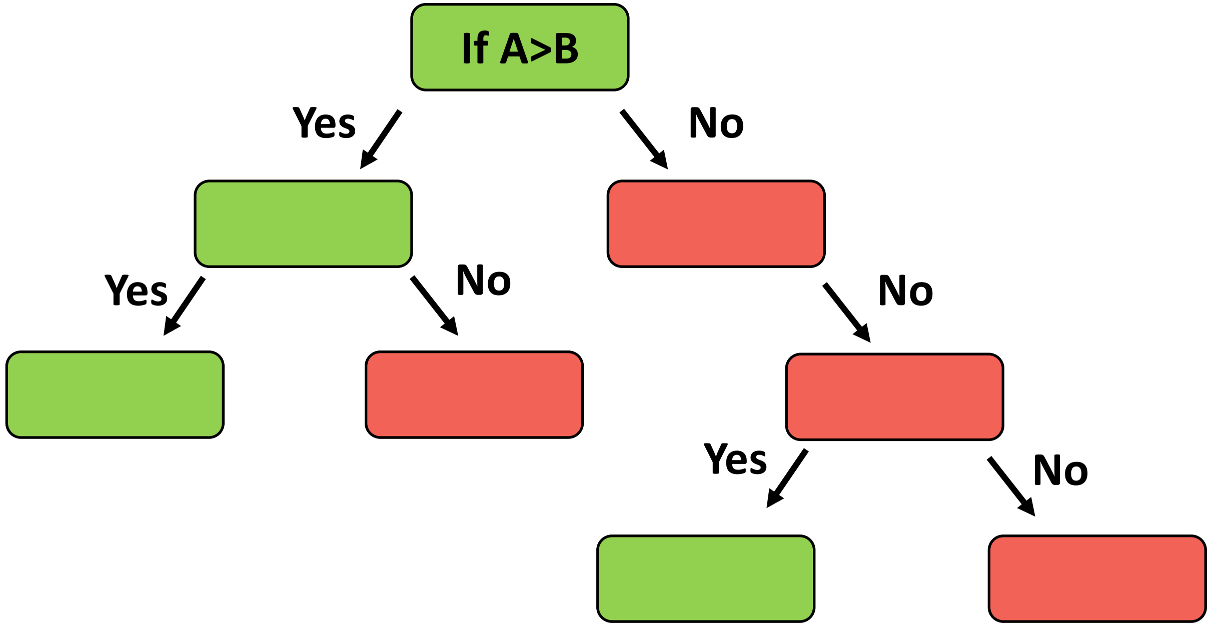
- **Feature selection:**
    - Reducing irrelevant features to improve accuracy.



Decision Tree Classifier

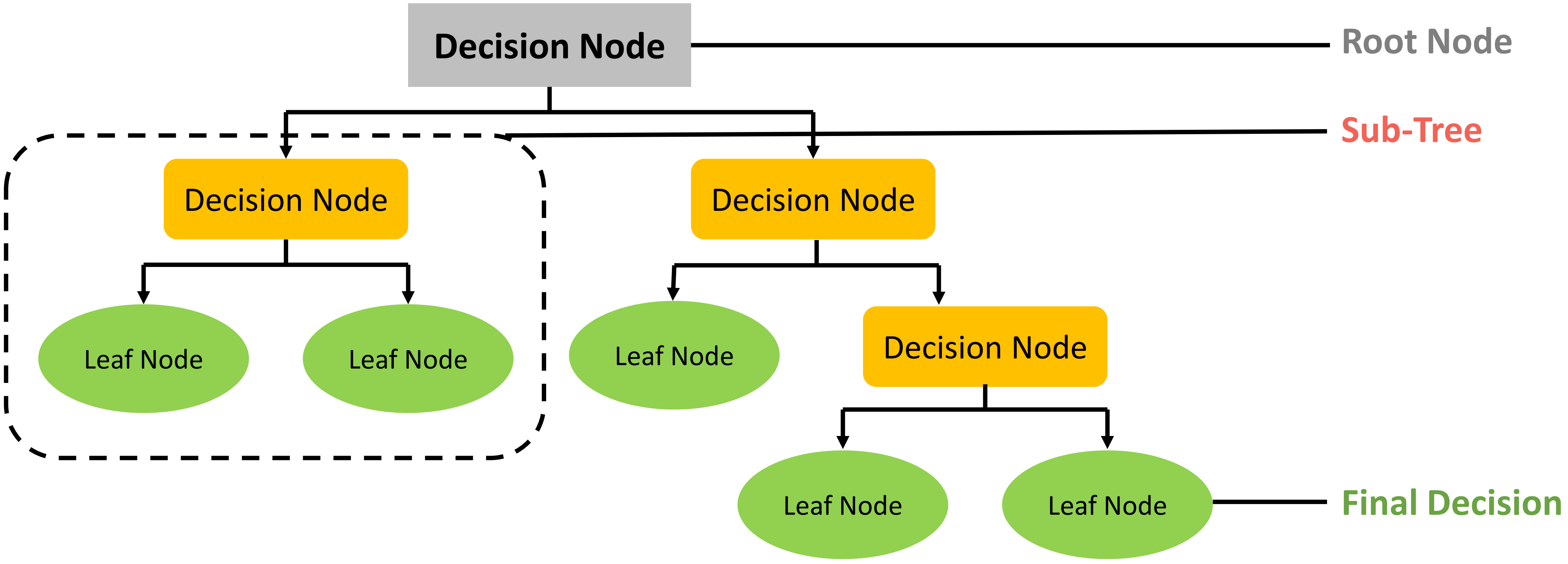
What is Decision Tree?

A tree-like model used for decision-making and classification tasks.



How does a Decision Tree work?

- Splits data into subsets based on feature values.
- Uses conditions (nodes) to classify data until reaching a final decision (leaf).



Pros and Cons of Decision Tree

Advantages

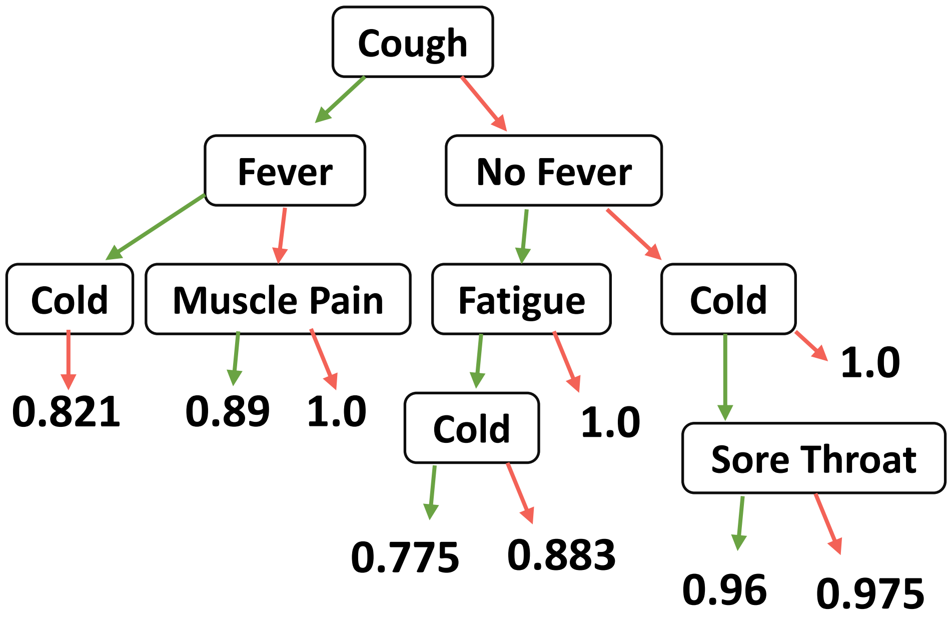
- Easy to understand and interpret.
- Can handle both numerical and categorical data.
- Requires little data preprocessing (no scaling needed).

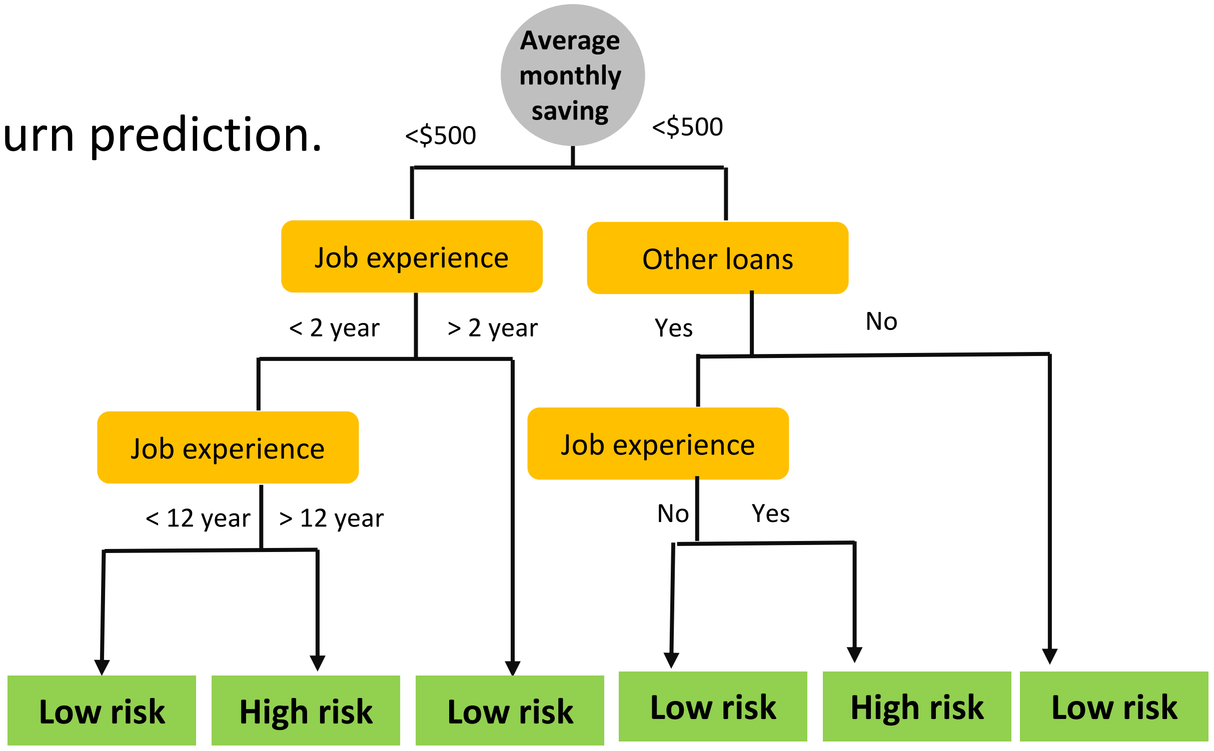
Disadvantages

- Prone to overfitting, especially on small datasets.
- Sensitive to noisy data.
- Can be biased towards features with more levels.

Use Cases of Decision Trees

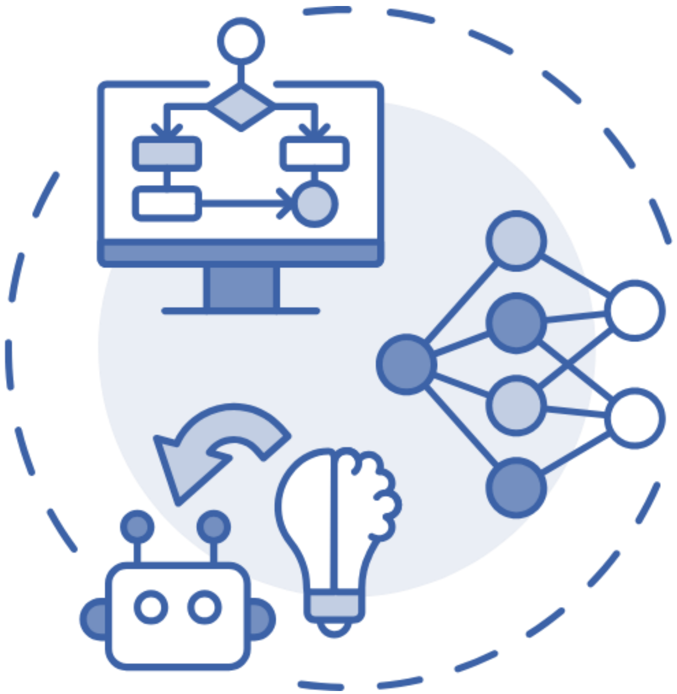
- **Real-world applications**:
  - **Healthcare:** Disease diagnosis based on symptoms.
  - **Finance:** Credit risk assessment.
  - **Marketing:** Customer segmentation and churn prediction.
  - **Retail:** Product recommendation systems.





Data Preprocessing for Decision Tree

- **Data preparation:**
  - Handling missing values.
  - Encoding categorical variables (if needed).
  - Splitting data into training and testing sets.
- **Feature selection:**
  - Removing irrelevant or highly correlated features.



Implementation Steps

- Step 1: Import Libraries
- Step 2: Load and Explore the Dataset
- Step 3: Preprocess the Data
- Step 4: Visualize Relationships between Features
- Step 5: Train the Model
- Step 6: Evaluate the Model Performance

Step 1: Import Libraries

First, we'll import all the necessary libraries.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```

Step 2: Load and Explore the Dataset

We'll load the Iris dataset and explore its structure.

```
from sklearn.datasets import load_iris

# Load the dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target
```

Step 2: Load and Explore the Dataset

```
# Map target values to actual species names
df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor',
2: 'virginica'})

# Display the first 5 rows
print(df.head())
```

Step 2: Load and Explore the Dataset

```
# Display basic information about the dataset
print("\nDataset Information:")
print(df.info())

# Summary statistics of the dataset
print("\nSummary Statistics:")
print(df.describe())
```

Step 3: Preprocess the Data

Here, we'll check for missing values and normalize the features.

```
# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())

# Separate features (X) and target (y)
X = df.drop('species', axis=1)
y = df['species']
```

Step 3: Preprocess the Data

```
# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features (important for algorithms like KNN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print("\nData Preprocessing Completed!")
```

Step 4: Visualize Relationships between Features

We'll use **Seaborn** to visualize the relationships between the features.

```
# Pairplot to visualize relationships between features
sns.pairplot(df, hue='species')
plt.title("Pairplot of Iris Dataset")
plt.show()

# Heatmap to visualize correlation between features
plt.figure(figsize=(8, 5))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

Step 5: Train the Model

Option 1: K-Nearest Neighbors (KNN)

We'll train a KNN classifier and evaluate its performance.

```
# Train the KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Make predictions on the test set
knn_predictions = knn.predict(X_test)
```

Step 5: Train the Model

```
# Evaluate the model
print("\nKNN Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, knn_predictions))
print("\nConfusion Matrix:\n", confusion_matrix(y_test,
knn_predictions))
print("\nClassification Report:\n", classification_report(y_test,
knn_predictions))
```

Step 5: Train the Model

Option 2: Decision Tree Classifier

Alternatively, we can use a Decision Tree classifier.

```
# Train the Decision Tree model
tree = DecisionTreeClassifier(random_state=42)
tree.fit(X_train, y_train)

# Make predictions on the test set
tree_predictions = tree.predict(X_test)
```

Step 5: Train the Model

```
# Evaluate the model
print("\nDecision Tree Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, tree_predictions))
print("\nConfusion Matrix:\n", confusion_matrix(y_test,
tree_predictions))
print("\nClassification Report:\n", classification_report(y_test,
tree_predictions))
```

Step 6: Evaluate the Model Performance

We'll evaluate the models using metrics such as accuracy, precision, recall, and confusion matrix.

```
import seaborn as sns

def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=iris.target_names, yticklabels=iris.target_names)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.show()
```

Step 6: Evaluate the Model Performance

```
# Plot confusion matrices for both models
plot_confusion_matrix(y_test, knn_predictions, "KNN")
plot_confusion_matrix(y_test, tree_predictions, "Decision Tree")
```

Complete Project Summary

- **Objective**: Classify iris species using petal and sepal measurements.
- **Dataset**: Iris dataset from scikit-learn.
- **Models Used**: KNN and Decision Tree classifiers.
- **Evaluation**: Accuracy, confusion matrix, classification report.

How to develop AI Classification Project

How to develop AI Classification Project

1. Define the Prediction Problem → 2. Data Collection & Preparation → 3. Model Selection & Training → 4. Model Evaluation → 5. Model Optimization → 6. Model Deployment → 7. Model Monitoring & Maintenance

1. Define the Prediction Problem

- Identify what needs to classify(spam vs non-spam emails).
- Define the input data and expected output. (images, text, tabular data)

2. Data Collection & Preparation

2.1 Collect Data

- Obtain a dataset that represents all classes well.Ensure a balanced dataset (if possible) to prevent bias.

2.2 Annotate & Label Data

- Use tools like LabelImg (for images) or Pandas (for tabular data).
- Store labels in a structured format (CSV, JSON, or XML).

2. Data Collection & Preparation

2.3 Data Preprocessing

- For images: Resize, normalize, augment (flip, rotate, etc.).
- For text: Tokenization, stopword removal, word embeddings.
- For tabular data: Handle missing values, normalize features.

2.4 Split Dataset

- Training Set (70-80%): Used to train the model.
- Validation Set (10-15%): Used for hyperparameter tuning.
- Test Set (10-15%): Used to evaluate the final model.

3. Model Selection & Training

3.1 Choose the Right Model

- Deep Learning Models (for images & text)
  - ✓ CNNs (ResNet, EfficientNet) for images.
  - ✓ RNNs, Transformers (BERT) for text.
- Machine Learning Models (for tabular data)
  - ✓ Decision Trees, Random Forest, SVM, XGBoost

3. Model Selection & Training

3.2 Define Model Architecture

- Use TensorFlow/Keras or PyTorch for deep learning.
- Adjust layers, activations, dropout, and batch normalization.

3.3 Compile Model

- Choose optimizer (Adam, RMSprop, SGD).
- Define loss function (MSE for regression, cross-entropy for classification).
- Select metrics (accuracy, F1-score)

3. Model Selection & Training

3.4 Train the Model

- Use GPU/TPU for faster training.
- Implement early stopping to prevent overfitting.
- Monitor validation loss to fine-tune hyperparameters..

4. Model Evaluation

- Evaluate on the test dataset.
- Use metrics
  - Accuracy: Overall correctness.
  - Precision-Recall: For imbalanced datasets.
  - Confusion Matrix: For understanding misclassifications.

5. Model Optimization

- Hyperparameter tuning (Grid Search, Random Search, Bayesian Optimization).
- Reduce overfitting: use dropout, data augmentation, weight regularization.
- Improve Performance: Transfer learning from pre-trained models.

6. Model Deployment

- Convert model to a deployable format (TF SavedModel, ONNX).
- Deploy as a REST API using Flask/FastAPI.
- Use Cloud Platforms (AWS SageMaker, GCP AI Platform).
- Deploy on Edge Devices (Raspberry Pi, TensorFlow Lite).

7. Model Monitoring & Maintenance

- Collect real-world feedback
- Retrain with new data periodically.
- Implement MLOps for continuous monitoring.

Realistic Infotech Group
IT Training & Services
No.79/A, First Floor
Corner of Insein Road and
Damaryon Street
Quarter (9), Hlaing Township
Near Thukha Bus Station
09256675642, 09953933826
http://www.rig-info.com