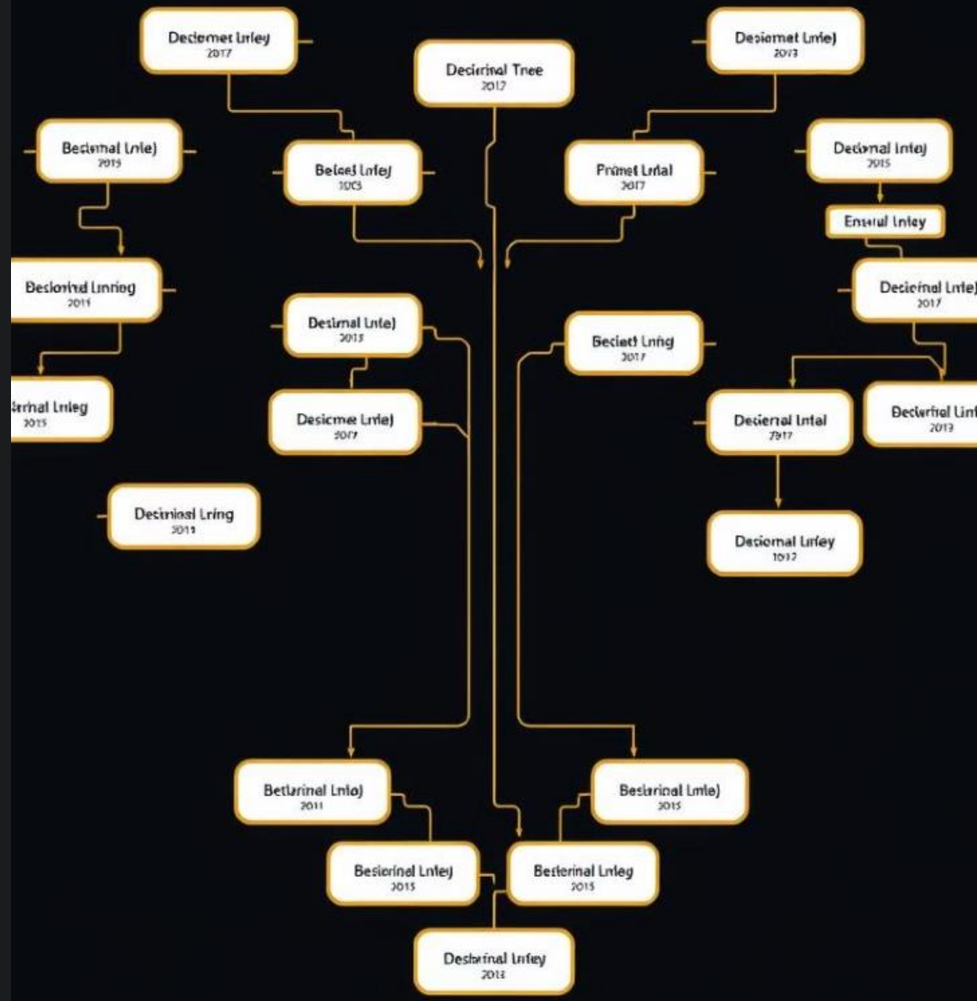
Classification and Regression Tree (CART) Algorithm

The Classification and Regression Tree (CART) algorithm is a powerful supervised learning model used for both classification and regression tasks in machine learning. It constructs a tree-like model of decisions based on feature values to make predictions.



Fundamentals of Decision Trees

1

Feature Selection

Decision trees recursively split the data based on the feature that provides the most information gain or decrease in impurity.

2

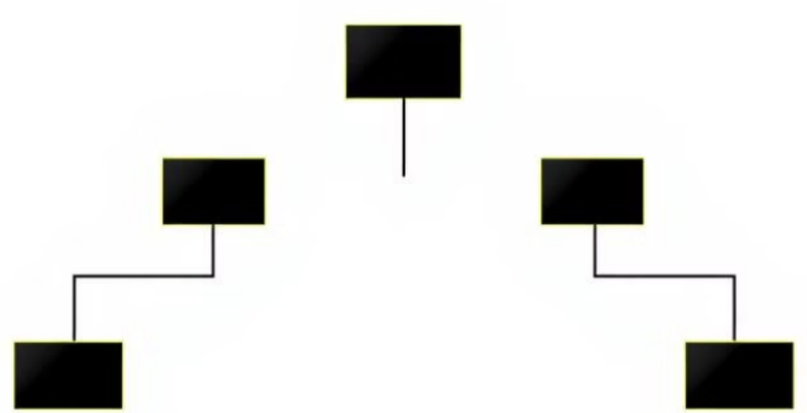
Tree Structure

The tree is composed of internal nodes (decision points), branches (feature splits), and leaf nodes (final predictions).

3

Tree Traversal

New data is classified by traversing the tree from the root, making decisions at each node until reaching a leaf node.



Key Characteristics of CART

Binary Splits

CART uses binary splits, meaning each internal node has exactly two child nodes.

Greedy Algorithm

CART employs a greedy algorithm, choosing the locally optimal split at each node without considering the global optimum.

Interpretability

The tree structure of CART models makes them easily interpretable and allows for visualization of the decision process.

Building a CART Model

1

Data Preparation

Preprocess the data by handling missing values, encoding categorical features, and splitting into training and testing sets.

2

Tree Growing

Recursively split the data by selecting the feature and threshold that maximizes the information gain or minimizes impurity.

3

Stopping Criteria

The tree-growing process continues until a stopping criterion is met, such as a maximum depth or minimum samples per leaf.

Advantages and Limitations of CART

1 **Advantages**

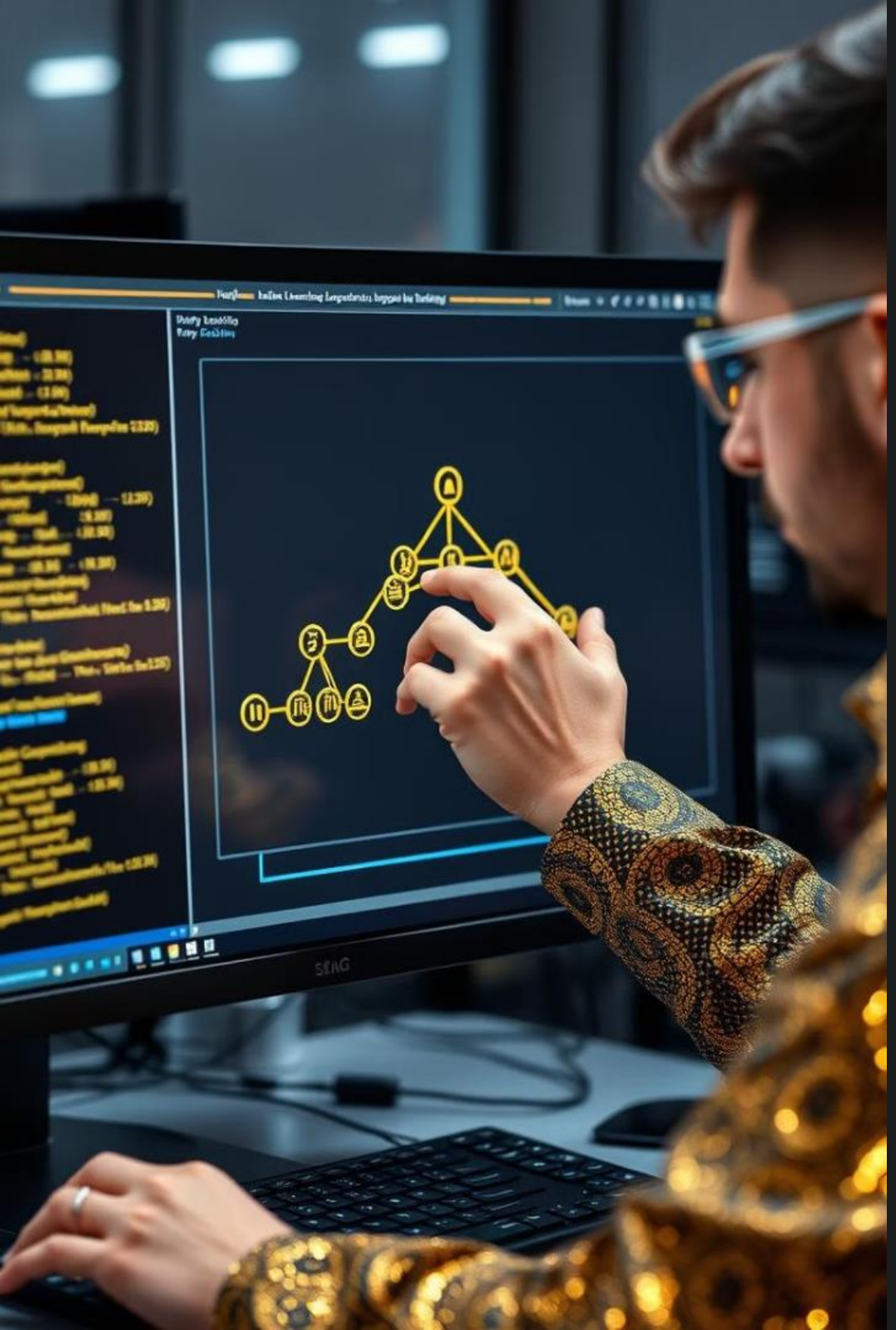
CART models are easy to interpret, handle missing data, and can capture non-linear relationships in the data.

2 **Limitations**

CART models can be unstable and prone to overfitting, especially with small datasets or high-dimensional features.

3 **Ensemble Methods**

Techniques like Random Forests and Gradient Boosting can be used to improve the performance of CART models.



Pruning and Tuning CART Models

1

Complexity Control

Pruning techniques, such as cost-complexity pruning, can be used to reduce the complexity of the tree and prevent overfitting.

2

Hyperparameter Tuning

Hyperparameters like maximum depth, minimum samples per leaf, and the splitting criterion can be optimized to improve model performance.

3

Cross-Validation

Cross-validation can be used to estimate the optimal hyperparameters and select the most appropriate model complexity.

Real-World Applications of CART

Healthcare

CART models are used for disease diagnosis, risk assessment, and patient outcome prediction.

Finance

CART models are applied to credit risk analysis, fraud detection, and investment decision-making.

Marketing

CART models are used for customer segmentation, churn prediction, and targeted advertising.

Retail

CART models are employed for inventory management, product recommendation, and customer behavior analysis

Sum

DEFINE THE DATASET

```
[4] data = {
        'CGPA': ['>=9', '>=8', '>=9', '<8', '>=8', '>=9', '<8', '>=9', '>=8', '<8'],
        'Interactive': ['Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes'],
        'Practical Knowledge': ['Very good', 'Good', 'Average', 'Average', 'Good', 'Good', 'Good', 'Very good', 'Average', 'Good'],
        'Comm Skills': ['Good', 'Moderate', 'Poor', 'Good', 'Moderate', 'Moderate', 'Poor', 'Good', 'Good', 'Poor'],
        'Job Offer': ['Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No']
    }
```

CONVERT THE DATA INTO DATAFRAME

```
[5] df = pd.DataFrame(data)
```

CONVERT THE NUMERICAL FEATURES INTO MUMERICAL FEATURES

```
[6] for col in df.columns:
        df[col] = pd.Categorical(df[col]).codes
```

SPLIT THE FEATURES AND TARGET

```
[7] X = df.drop('Job Offer', axis=1)  # Features
    y = df['Job Offer']               # Target
```

SPLIT TRAINING AND TEST DATASET

```
[8]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

PARAMETER FOR GRIDSEARCH

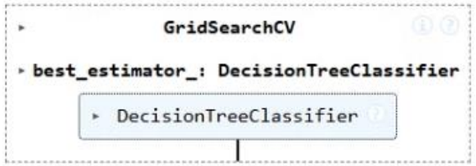
```
[9]  param_grid = {
         'max_depth': [3, 4, 5, None],
         'min_samples_split': [2, 5, 10],
         'min_samples_leaf': [1, 2, 4],
         'max_features': [None, 'sqrt', 'log2']
     }
```

INITIALIZE DECISION_TREE AND GRIDSEARCH

```
[12] clf = DecisionTreeClassifier(random_state=42)
     grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1)
```

FIT THE MODEL

```
[13] grid_search.fit(X_train, y_train)
```



GET THE BEST MODEL

```
[14] best_clf = grid_search.best_estimator_
     print("Best parameters found by GridSearchCV:", grid_search.best_params_)
```

```
Best parameters found by GridSearchCV: {'max_depth': 3, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2}
```

MAKE PREDICTIONS USING BEST MODEL

```
[15] y_pred = best_clf.predict(X_test)
```

ACCURACY

```
[16] accuracy = accuracy_score(y_test, y_pred)
     print(f"Improved Accuracy: {accuracy:.2f}")
```

```
Improved Accuracy: 1.00
```

CLASSIFICATION REPORT

```
[18] unique_classes = y_test.unique()
     if len(unique_classes) == 1:
         target_names = ['No'] if unique_classes[0] == 0 else ['Yes']
     else:
         target_names = ['No', 'Yes']


     print("Classification Report:")
     print(classification_report(y_test, y_pred, target_names=target_names))
```

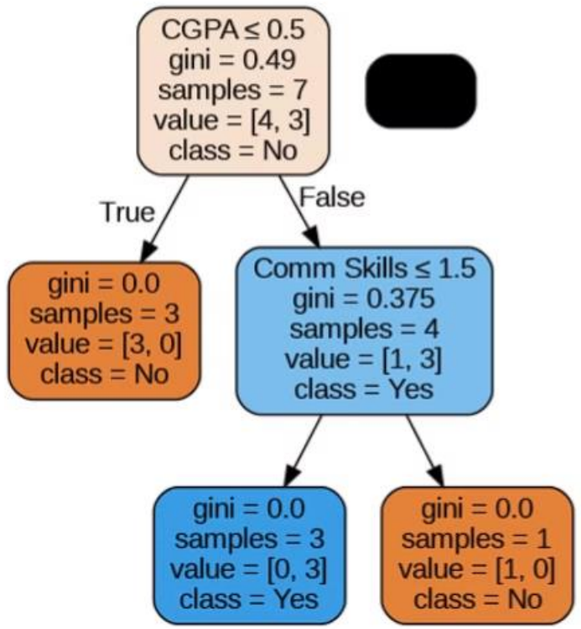
```
Classification Report:
              precision    recall  f1-score   support

         Yes       1.00      1.00      1.00         3

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```

VISUALISE THE TREE

```
[21] dot_data = StringIO()
     export_graphviz(best_clf, out_file=dot_data, feature_names=['CGPA', 'Interactive', 'Practical Knowledge', 'Comm Skills'],
                     class_names=['No', 'Yes'], filled=True, rounded=True, special_characters=True)
     graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
     Image(graph.create_png())
```



Done by,
Ajinesh - 22am007
Sarankumar.S - 22am055
Gobika.R - 22am069