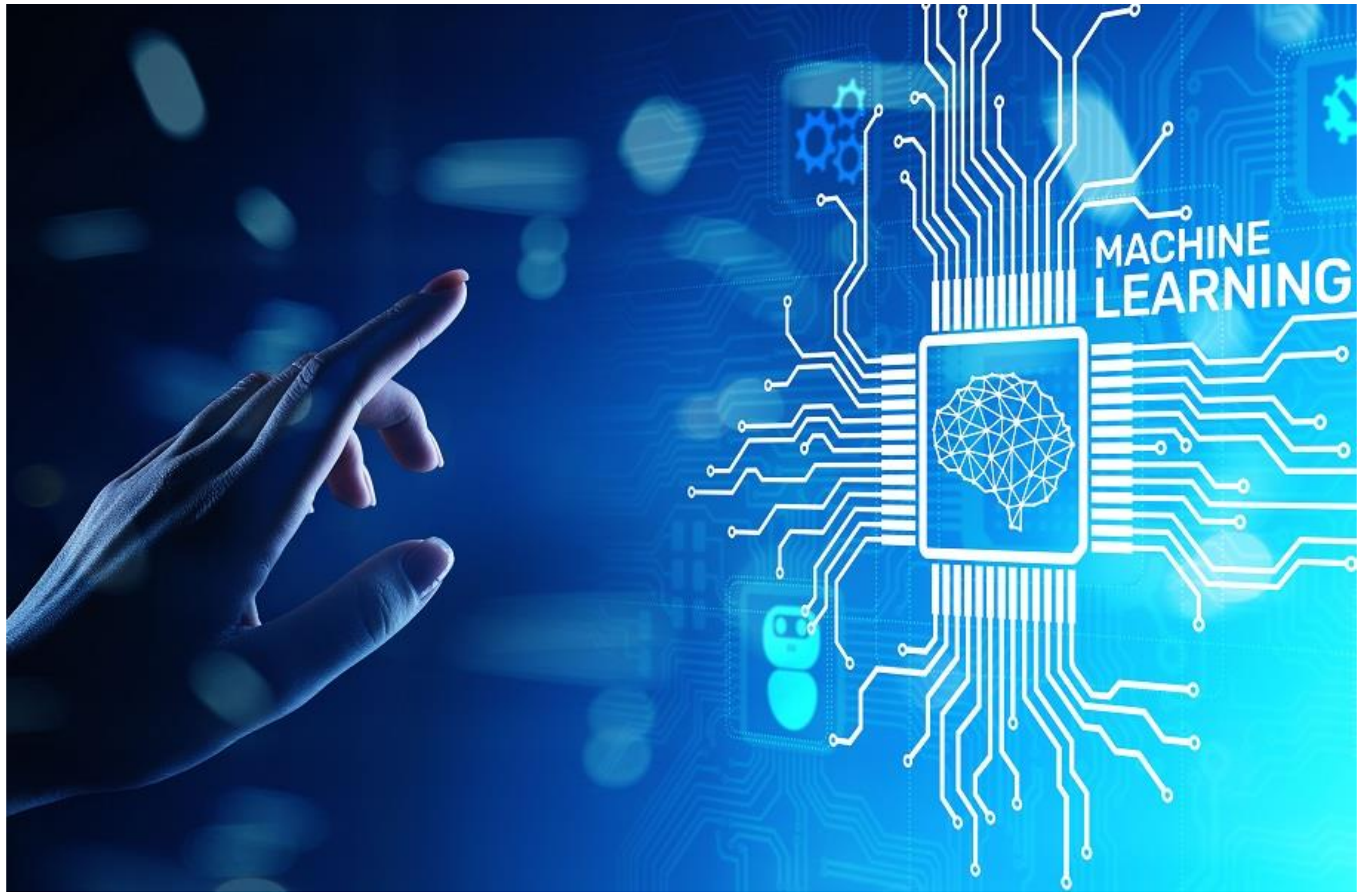
Duration: 100 min | BCSE424P | ML for Robotics Lab | Dr. Harini S

Decision Tree – ID3 and CART



Learning for Today (13.1.2026)

- Data Preprocessing
  - Handling missing data
  - Some Visualization Techniques
- Decision Tree
  - ID3
  - CART

Handling Missing Data

- Why Missing Data?
  - Missing Completely At Random
    - Example: A sensor fails randomly for some readings.
  - MAR – Missing At Random
    - The missingness depends on other observed features.
    - Example: DOB is missing in a data set that results in Age also missing
  - MNAR – Missing Not At Random
    - The missingness depends on the missing value itself.
    - Example: Some People with high income do not report income for Income Tax

Basic Approaches for Handling Missing Data

- ***Removing Missing Data***

import pandas as pd

df.dropna()   # drop rows with any missing value

df.dropna(axis=1)   # drop columns with any missing value

- **Imputation (Filling Missing Data)**

Mean / Median / Mode Imputation

df['Age'].fillna(df['Age'].mean(), inplace=True)

df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)

Forward/Backward Fill

df.fillna(method='ffill', inplace=True)

df.fillna(method='bfill', inplace=True)

Basic Approaches for Handling Missing Data

- ***Removing Missing Data***

import pandas as pd

df.dropna()   # drop rows with any missing value

df.dropna(axis=1)   # drop columns with any missing value

- **Imputation (Filling Missing Data)**

Mean / Median / Mode Imputation

df['Age'].fillna(df['Age'].mean(), inplace=True)

df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)

Forward/Backward Fill

df.fillna(method='ffill', inplace=True)

df.fillna(method='bfill', inplace=True)

Some Visualization Techniques

- ***Bar plot of missing values***
  - ***X –axis: Features***
  - ***Y-Axis: Number of missing values or percentage missing***

```
import pandas as pd
import matplotlib.pyplot as plt

# Example: before and after missing data handling
before = df.isna().sum()
after = df_filled.isna().sum()

comparison = pd.DataFrame({'Before': before, 'After': after})
comparison.plot(kind='bar', figsize=(10,5))
plt.ylabel('Number of missing values')
plt.title('Missing Values: Before vs After Imputation')
plt.show()
```

- **Heatmap of missing data**
  - ***Use seaborn or missingno library***
  - ***Each cell shows if a value is missing***
  - ***Compare before vs after visually.***

```
import seaborn as sns

plt.figure(figsize=(12,4))
sns.heatmap(df.isna(), cbar=False)
plt.title('Before Handling Missing Data')
plt.show()

plt.figure(figsize=(12,4))
sns.heatmap(df_filled.isna(), cbar=False)
plt.title('After Handling Missing Data')
plt.show()
```

Some Visualization Techniques

- ***Histograms / Density plots***

```
plt.figure(figsize=(8,4))
df['Age'].hist(alpha=0.5, bins=20, label='Before')
df_filled['Age'].hist(alpha=0.5, bins=20,
label='After')
plt.legend()
plt.title('Age Distribution Before vs After
Imputation')
plt.show()
```

- **Summary Table / Percentage Plot**

```
missing_perc = pd.DataFrame({
    'Before': df.isna().mean()*100,
    'After': df_filled.isna().mean()*100
})
missing_perc.plot(kind='bar', figsize=(10,5))
plt.ylabel('% Missing')
plt.title('Percentage of Missing Values Before vs After')
plt.show()
```

Exercise

1. Apply the missing data handling (shared in previous slide) to Titanic Dataset from Kaggle. Visualize the results using Bar Plot, Heat Map, Histogram/Density Plot, Box Plot

- Age – Apply Median Imputation
- Cabin – Fill missing category as “Unknown”
- Embarked - Remove the rows where this feature is missing

Classification Tree (Iris Dataset) – Data load

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
```

```
iris = load_iris()
X = iris.data
y = iris.target

print("Feature names:", iris.feature_names)
print("Target classes:", iris.target_names)
print("Shape of X:", X.shape)

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

Example 1: ID3

```
dt_entropy = DecisionTreeClassifier(criterion='entropy', max_depth=3)

dt_entropy.fit(X_train, y_train)

y_pred_entropy = dt_entropy.predict(X_test)

print("Entropy Accuracy:", accuracy_score(y_test, y_pred_entropy))
```

```
plt.figure(figsize=(12,6))
plot_tree(dt_entropy,
          feature_names=iris.feature_names,
          class_names=iris.target_names,
          filled=True)
plt.show()
```

Example 1: CART

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

dt_gini = DecisionTreeClassifier(criterion='gini',
max_depth=3)
dt_gini.fit(X_train, y_train)
```

```
print("Accuracy:", accuracy_score(y_test,
y_pred))
print("Confusion Matrix:\n",
confusion_matrix(y_test, y_pred))
```

```
plt.figure(figsize=(12,6))
plot_tree(dt_gini,
        feature_names=iris.feature_names,
        class_names=iris.target_names,
        filled=True)
plt.show()
```

What is the role of ”max_depth” ?

- Depth of a decision tree is the maximum number of splits (levels) from the root node to a leaf node.
- Control parameter for the depth of decision tree in python libraries – max_depth
- By giving various values here – we perform pre-pruning

What is the role of ”max_depth” ?

- Depth of a decision tree is the maximum number of splits (levels) from the root node to a leaf node.

- Control parameter for the depth of decision tree in python libraries – max_depth

- By giving various values here – we perform pre-pruning

Overfitting and Pre-Pruning

Apply the below code to Example 1 CART Tree

```
depths = [1, 3, 5, None]

for d in depths:
    model = DecisionTreeClassifier(max_depth=d, random_state=42)
    model.fit(X_train, y_train)

    train_acc = model.score(X_train, y_train)
    test_acc = model.score(X_test, y_test)

    print(f"Depth={d}, Train Acc={train_acc:.2f}, Test Acc={test_acc:.2f}")
```

```
plt.figure(figsize=(12,6))
plot_tree(model,
        feature_names=iris.feature_names,
        class_names=iris.target_names,
        filled=True)
plt.show()
```

Exercise

2. In titanic data set, build CART tree and find the optimal max_depth value

CART Post Prining with ccp_alpha

- Cost-complexity pruning parameter
- Apply the below code to iris dataset

```
path = clf.cost_complexity_pruning_path(X_train, y_train)

ccp_alphas = path.ccp_alphas

impurities = path.impurities

train_acc = []

test_acc = []

for alpha in ccp_alphas:
    clf_alpha = DecisionTreeClassifier(random_state=42, ccp_alpha=alpha)
    clf_alpha.fit(X_train, y_train)


    train_acc.append(accuracy_score(y_train, clf_alpha.predict(X_train)))

    test_acc.append(accuracy_score(y_test, clf_alpha.predict(X_test)))
```

CART Post Prining with ccp_alpha

```
plt.figure(figsize=(8,5))
plt.plot(ccp_alphas, train_acc, marker='o', label='Train Accuracy')
plt.plot(ccp_alphas, test_acc, marker='s', label='Test Accuracy')
plt.xlabel('ccp_alpha (Cost Complexity Pruning Parameter)')
plt.ylabel('Accuracy')
plt.title('Post-Pruning on Iris Dataset')
plt.legend()
plt.grid(True)
plt.show()
```

```
best_alpha = ccp_alphas[test_acc.index(max(test_acc))]
clf_pruned = DecisionTreeClassifier(random_state=42, ccp_alpha=best_alpha)
clf_pruned.fit(X_train, y_train)
print(f"Best ccp_alpha: {best_alpha}")
print(f"Test Accuracy (pruned tree): {accuracy_score(y_test, clf_pruned.predict(X_test))}")
```

Exercise

3. Apply Post pruning to Titanic Dataset