ibimbing

MACHINE LEARNING CLASSFICATION

DFS 35.0 - Data Science

Presented By **Kasih Afrillia**

ibimbing

WHAT IS MACHINE

LEARNING CLASSIFICATION

Machine Learning Classification is a supervised learning approach used to assign data into predefined categories based on its characteristics. The model learns from labeled data and then predicts the category of new, unseen data.

A common application of this method is on the Wine Dataset, where the objective is to classify different types of wine based on their chemical properties. This dataset, often used in machine learning, helps evaluate various classification algorithms. It contains 13 features describing the chemical composition of each sample and categorizes the wines into three different types.

TOOLS USED

READ DATASET

- pandas is used for handling tabular data.
- load_diabetes loads the diabetes dataset from sklearn.
- Loads the diabetes dataset into the variable diabetes.
- Converts feature data into a DataFrame with column names.
- Adds a new column 'diabetes_progression' for target values.
- Displays the first five rows of the DataFrame.

```python
import pandas as pd  # For handling datasets
import numpy as np  # For numerical operations
from sklearn.datasets import load_diabetes  # Load Diabetes dataset

# Load the Diabetes dataset
diabetes = load_diabetes()
X = diabetes.data  # Feature data
y = diabetes.target  # Target labels

# Convert features and target into DataFrame
df_X = pd.DataFrame(X, columns=diabetes.feature_names)  # Feature DataFrame
df_y = pd.Series(y, name='diabetes_progression')  # Target Series

# Display first few rows
print(df_X.head())
print(df_y.head())
```

```
(        age       sex       bmi        bp        s1        s2        s3
 0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
 1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
 2  0.085299  0.050680  0.044451 -0.005670 -0.045599 -0.034194 -0.032356
 3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
 4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142

          s4        s5        s6
 0 -0.002592  0.019907 -0.017646
 1 -0.039493 -0.068332 -0.092204
 2 -0.002592  0.002861 -0.025930
 3  0.034309  0.022688 -0.009362
 4 -0.002592 -0.031988 -0.046641  ,
 0    151.0
 1     75.0
 2    141.0
 3    206.0
 4    135.0
 Name: diabetes_progression, dtype: float64)
```

DATA PREPROCESSING

This dataset contains 442 rows and 10 columns, with all values being of type float64 and no missing data, using 34.7 KB of memory. The columns include age, sex, bmi, bp, and several other medical-related variables. Additionally, the descriptive statistics for the diabetes_progression variable show a mean of 152.13, a standard deviation of 77.09, a minimum value of 25, and a maximum of 346. The first quartile (25%) is at 87, the median (50%) at 140.5, and the third quartile (75%) at 211.5. Based on this information, the dataset appears to be related to a diabetes study and could be used for regression analysis or predicting diabetes progression.

```
df_X.info()
df_y.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   s1      442 non-null    float64
 5   s2      442 non-null    float64
 6   s3      442 non-null    float64
 7   s4      442 non-null    float64
 8   s5      442 non-null    float64
 9   s6      442 non-null    float64
dtypes: float64(10)
memory usage: 34.7 KB
```

	diabetes_progression
count	442.000000
mean	152.133484
std	77.093005
min	25.000000
25%	87.000000
50%	140.500000
75%	211.500000
max	346.000000

dtype: float64

DATA PREPROCESSING

```
df_X.isnull().sum()
df_y.isnull().sum()
#mengecek missing value

0
```

The code df_X.isnull().sum() and df_y.isnull().sum() is used to check for missing values in the dataset.

- df_X.isnull().sum(): This checks each column in df_X (which likely contains the independent variables/features) and returns the count of missing (NaN) values for each column.
- df_y.isnull().sum(): This does the same check for df_y (which likely contains the target variable).

Since the output shows 0 for all columns, it means there are no missing values in the dataset. This is important in data preprocessing because missing values can affect model accuracy, require handling (such as filling with mean/median values or removing rows), and complicate analysis. Since this dataset has no missing values, it is clean and ready for further processing, such as feature selection, scaling, or model training.

DATA PREPROCESSING

The code df_y.unique() is used to retrieve all unique values present in the df_y column, which represents the diabetes progression variable.

- df_y likely contains numerical values representing the progression of diabetes for different patients.
- .unique() returns an array of distinct values found in df_y, removing any duplicates.

This is useful for understanding the distribution of the target variable. If the output contains a small set of unique values, it might indicate a classification problem, whereas a large set of unique values suggests a regression problem. In this case, since diabetes_progression is a continuous numerical variable, the dataset is most likely used for regression analysis.

```
df_y.unique()
#mengakses nilai yang unique di kolom diabetes progression
```

```
array([151.,  75., 141., 206., 135.,  97., 138.,  63., 110., 310., 101.,
        69., 179., 185., 118., 171., 166., 144., 168.,  68.,  49., 245.,
       184., 202., 137.,  85., 131., 283., 129.,  59., 341.,  87.,  65.,
       102., 265., 276., 252.,  90., 100.,  55.,  61.,  92., 259.,  53.,
       190., 142., 155., 225., 104., 182., 128.,  52.,  37., 170.,  71.,
       163., 150., 160., 178.,  48., 270., 111.,  42., 200., 113., 143.,
        51., 210., 134.,  98., 164.,  96., 162., 279.,  83., 302., 198.,
        95., 232.,  81., 246., 297., 258., 229., 275., 281., 173., 180.,
        84., 121., 161.,  99., 109., 115., 268., 274., 158., 107., 103.,
       272., 280., 336., 317., 235.,  60., 174., 126., 288.,  88., 292.,
       197., 186.,  25., 195., 217., 172., 214.,  70., 220., 152.,  47.,
        74., 295., 127., 237.,  64.,  79.,  91., 116.,  86., 122.,  72.,
        39., 196., 222., 277.,  77., 191.,  73., 263., 248., 296.,  78.,
        93., 208., 108., 154., 124.,  67., 257., 262., 177., 187., 125.,
       215., 303., 243., 153., 346.,  89.,  50., 308., 145.,  45., 264.
```

DATA PREPROCESSING

The code df_y.value_counts() is used to count the occurrences of each unique value in the df_y column, which represents the diabetes progression variable.

- df_y contains numerical values, and .value_counts() returns a sorted count of how many times each unique value appears in the dataset.
- This is useful for understanding the distribution of the target variable. If df_y has only a few unique values with high frequencies, it may indicate a classification problem. If df_y has many unique values with varying frequencies, it suggests a regression problem.

```
df_y.value_counts()
#menghitung masing masing nilai y
```

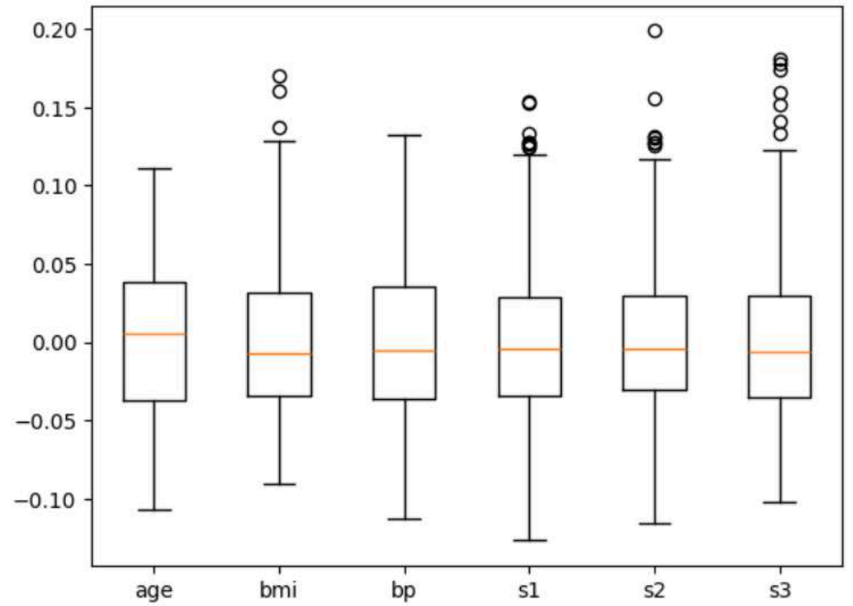
diabetes_progression	count
200.0	6
72.0	6
90.0	5
178.0	5
71.0	5
...	...
73.0	1
222.0	1
86.0	1
79.0	1
57.0	1

214 rows × 1 columns

dtype: int64

DATA PREPROCESSING

```
# Mengecek outlier
import matplotlib.pyplot as plt

# Memilih beberapa kolom yang relevan untuk dianalisis outliernya
# Corrected line: Accessing columns from df_X instead of df_y
plt.boxplot(df_X[['age', 'bmi', 'bp', 's1', 's2', 's3']])

# Memberi label pada sumbu x sesuai dengan urutan kolom yang dipilih
plt.xticks([1, 2, 3, 4, 5, 6], ['age', 'bmi', 'bp', 's1', 's2', 's3'])

# Menampilkan plot
plt.show()
```



plt.boxplot(df[['age', 'bmi', 'bp', 's1', 's2', 's3']])

- **This creates a boxplot for the selected columns: age, bmi, bp, s1, s2, and s3.**
- **A boxplot helps identify outliers, which are data points that fall significantly outside the typical range.**

plt.xticks([1, 2, 3, 4, 5, 6], ['age', 'bmi', 'bp', 's1', 's2', 's3'])

- **This sets labels on the x-axis to match the selected columns.**
- **The numbers [1, 2, 3, 4, 5, 6] represent the positions of the columns in the plot.**

plt.show()

- **This displays the boxplot, making it easier to visually analyze the distribution of values and identify potential outliers.**

SPLIT DATA

```python
from sklearn.model_selection import train_test_split

# Memisahkan fitur dan target
# Use df_X instead of 'data' and specify feature names (diabetes.feature_names)
df_x = df_X[diabetes.feature_names]
df_y = df_y  # df_y already contains the target variable

# Membagi data menjadi train dan test
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.2, random_state=100)

print(f"Number of train data: {len(x_train)}")
print(f"Number of testing data: {len(x_test)}")
```

```
Number of train data: 353
Number of testing data: 89
```

This code splits a dataset into training and testing sets using train_test_split from sklearn.model_selection. First, the feature variables (df_x) are selected from the input dataframe df_X using the feature names from the diabetes dataset (i.e., diabetes.feature_names). The target variable (df_y) is already predefined, so it is directly used. The dataset is then divided into training and testing subsets with 80% of the data used for training and 20% for testing, ensuring reproducibility with a random_state set to 100. Finally, it prints the number of samples in the training and testing sets to provide an overview of the split.

TRAIN THE MODEL

This code creates a decision tree model using the DecisionTreeClassifier from the sklearn.tree module. The model is initialized with a random_state set to 135, ensuring that the results are reproducible. The fit method is then used to train the decision tree model using the training data (x_train for features and y_train for the target variable). The model learns patterns from the training data to make predictions on unseen data in the future.

```
[22] from sklearn.tree import DecisionTreeClassifier

     # Membuat model Decision Tree
     model = DecisionTreeClassifier(random_state=135)

     # Melatih model dengan data train
     model.fit(x_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=135)
```

PREDICT&EVALUATE

The first line imports the accuracy_score function from the sklearn.metrics library, which is used to measure the accuracy of the model. Next, y_pred = model.predict(x_test) makes predictions on the test data (x_test) using the trained model, and the results are stored in y_pred. Then, accuracy = accuracy_score(y_test, y_pred) calculates the accuracy by comparing the predicted values (y_pred) with the actual values (y_test). Finally, print(f"Accuracy: {accuracy*100:.2f}%") displays the accuracy as a percentage with two decimal places.

```
from sklearn.metrics import accuracy_score

# Melakukan prediksi pada data test
y_pred = model.predict(x_test)

# Menghitung akurasi model
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy*100:.2f}%")
```

```
Accuracy: 2.25%
```

TRAIN THE MODEL

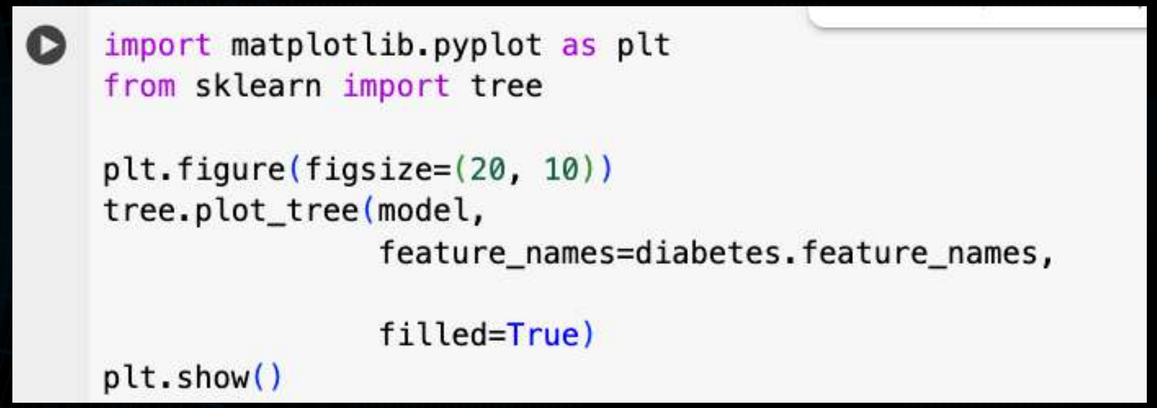
This code creates a decision tree model using the DecisionTreeClassifier from the sklearn.tree module. The model is initialized with a random_state set to 135, ensuring that the results are reproducible. The fit method is then used to train the decision tree model using the training data (x_train for features and y_train for the target variable). The model learns patterns from the training data to make predictions on unseen data in the future.

```
[22] from sklearn.tree import DecisionTreeClassifier

     # Membuat model Decision Tree
     model = DecisionTreeClassifier(random_state=135)

     # Melatih model dengan data train
     model.fit(x_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=135)
```

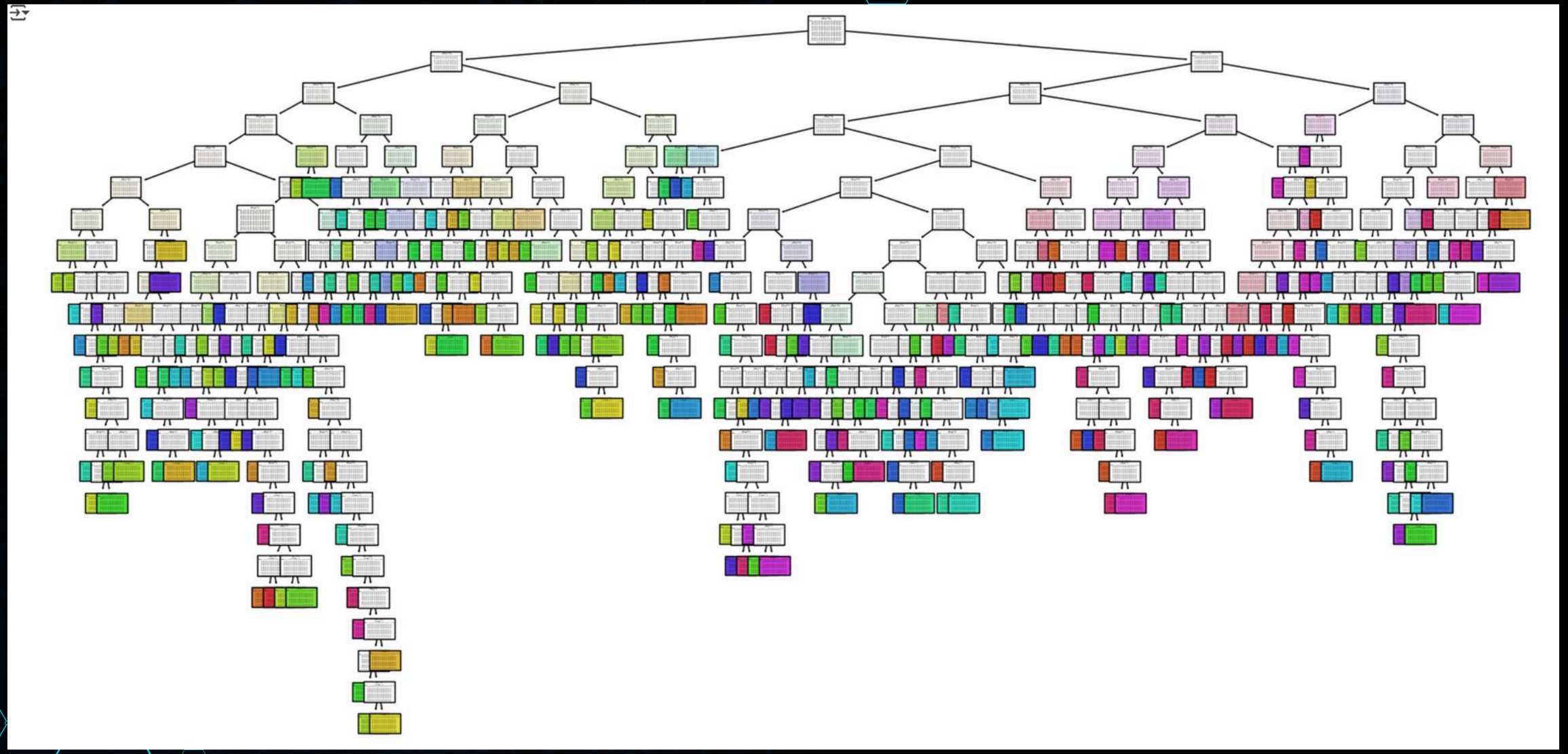
VISUALIZATION

This code visualizes the trained decision tree model using matplotlib and sklearn.tree. First, it imports the necessary libraries: matplotlib.pyplot for plotting and tree from sklearn to generate the tree visualization. The plt.figure(figsize=(20, 10)) line sets the figure size to 20 inches by 10 inches, making the tree visualization large and easy to read. The tree.plot_tree function is then used to plot the decision tree, where the model (the trained decision tree) is passed along with feature_names (which uses the feature names from the diabetes dataset to label the tree's splits). The filled=True argument colors the nodes to visually represent the class distributions. Finally, plt.show() displays the plot, showing the structure of the decision tree.



```
import matplotlib.pyplot as plt
from sklearn import tree

plt.figure(figsize=(20, 10))
tree.plot_tree(model,
               feature_names=diabetes.feature_names,

               filled=True)
plt.show()
```

VISUALIZATION



Thank You

ibimbing