MACHINE LEARNING CLASSIFICATION

By: Deniis Naureen Alesha

DSF 35.0 - Data Science

What is Machine Learning Classification?

Machine learning classification is a supervised learning technique where an algorithm learns from labeled data to categorize new observations into predefined classes. It is widely used in various applications such as medical diagnosis, spam detection, and image recognition. Common classification algorithms include Logistic Regression, Decision Trees, Random Forest, and Neural Networks. The performance of a classification model is typically evaluated using metrics like accuracy, precision, recall, and the F1-score.

The Iris dataset in Scikit-learn consists of 150 samples from three Iris species: Setosa, Versicolor, and Virginica. It includes four features (sepal length, sepal width, petal length, and petal width) to help classify the species. The dataset is commonly used to demonstrate machine learning classification algorithms.

TOOLS USED


DATA ANALYSIS
MACHINE LEARNING
IN PYTHON
Jupyter Notebook
Pandas
Numpy
Matplotlib
Seaborn
Scikit learn
scikit
learn
matplotlib
NumPy
jupyter
pandas

Read Dataset

```python
import pandas as pd
from sklearn import datasets

iris = datasets.load_iris() # memuat dataset Iris dari scikit-learn
x = iris.data  # data fitur
y = iris.target  # data target (label)

df_x = pd.DataFrame(x, columns=iris.feature_names) # mengonversi data fitur dan target jadi DataFrame
df_y = pd.Series(y, name='iris_class')

df = pd.concat([df_x, df_y], axis=1) # gabungin fitur dan target dalam satu DataFrame
```

1. **import pandas as pd**: This imports the pandas library and gives it the alias pd. pandas is a powerful library in Python used for data manipulation and analysis. It provides structures like DataFrames and Series, which are useful for handling and analyzing structured data, such as CSV files or datasets like Iris.
2. **from sklearn import datasets**: This imports the datasets module from the sklearn (Scikit-learn) library, which contains a variety of preloaded datasets for machine learning tasks, including the Iris dataset.
3. **iris = datasets.load_iris()**: This loads the Iris dataset from sklearn.datasets and stores it in the variable iris. This dataset includes features like sepal and petal measurements, as well as class labels.
4. **x = iris.data**: This stores the feature data (sepal and petal measurements) from the iris dataset into the variable x.
5. **y = iris.target**: This stores the target labels (species of Iris) from the iris dataset into the variable y.
6. **df_x = pd.DataFrame(x, columns=iris.feature_names)**: This converts the feature data x into a pandas DataFrame, with column names specified by iris.feature_names (e.g., "sepal length", "sepal width", etc.).
7. **df_y = pd.Series(y, name='iris_class')**: This converts the target labels y into a pandas Series and names the column iris_class.
8. **df = pd.concat([df_x, df_y], axis=1)**: This combines the features (df_x) and target labels (df_y) into a single DataFrame along the horizontal axis (axis=1), so that both the features and target labels are in the same table.

Data Pre-Processing

```
[2]  # Data Pre-Processing

     df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   iris_class         150 non-null    int64  
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

The **df.info()** output shows that the Iris dataset has 150 entries with 5 columns. Four of the columns (sepal length, sepal width, petal length, petal width) are of type float64, and one (iris_class) is of type int64. All columns contain 150 non-null values, indicating no missing data. The dataset's memory usage is 6.0 KB. This summary confirms that the dataset is complete and well-structured for analysis.

Data Pre-Processing

```
df.isnull().sum() #untuk cek missing value
```

	0
sepal length (cm)	0
sepal width (cm)	0
petal length (cm)	0
petal width (cm)	0
iris_class	0

The **df.isnull().sum()** function checks for missing values in each column of the DataFrame. It will return the number of missing values (if any) for each column. If the result shows zeros for all columns, it indicates there are no missing values in the dataset. This is helpful to ensure the data is complete before analysis.

Data Pre-Processing

```
[4]  df['iris_class'].unique() # untuk mengakses kolom 'iris_class' yang unique

     array([0, 1, 2])
```

The **df['iris_class'].unique()** function returns the unique values in the iris_class column. In the Iris dataset, this will show the distinct class labels, which typically represent the different species of Iris (e.g., 0 for setosa, 1 for versicolor, and 2 for virginica). This helps to identify how many unique target classes are present in the dataset.

Data Pre-Processing

```
df['iris_class'].value_counts() # untuk meghitung masing" iris_class
```

iris_class	count
0	50
1	50
2	50

dtype: int64

The **df['iris_class'].value_counts()** function counts how many instances there are of each unique class in the iris_class column. It gives a breakdown of the distribution of the target labels, which represent the different Iris species. For example, it might show how many samples belong to each of the three species in the dataset.

Data Pre-Processing

```
[7]  df.describe() # statistik deskriptif
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	iris_class
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

The **df.describe()** function provides descriptive statistics for the numerical columns in the DataFrame.

Data Pre-Processing

```python
# cek outlier
import matplotlib.pyplot as plt
plt.boxplot(df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']]) # memilih kolom
plt.xticks([1, 2, 3, 4], ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']) # memberi label pada axis
plt.show()
```

The code creates a boxplot for the four feature columns: sepal length, sepal width, petal length, and petal width. A boxplot is useful for visualizing the distribution of the data, including the median, interquartile range (IQR), and any potential outliers (data points that lie outside the whiskers of the plot). The **plt.xticks()** function assigns appropriate labels to the x-axis for each feature.

Split Data

```
# Split Data

# misahin variabel prediktor dan variabel target
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split

# bagi data menjadi train dan test
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size = 0.4, random_state = 150)
print(f"Number of train data: {len(x_train)}")
print(f"Number of testing data: {len(x_test)}")
```

```
Number of train data: 90
Number of testing data: 60
```

The code splits the Iris dataset into training and testing sets using a 60-40 split. The training data (x_train, y_train) makes up 60% of the dataset, while the testing data (x_test, y_test) represents the remaining 40%. The random_state ensures that the split is reproducible. The print statements show how many data points are allocated to the training and testing sets. This split is important for training machine learning models and evaluating their performance.

Train Data

```
[2] # Train Data

    from sklearn.tree import DecisionTreeClassifier

    # bikin model Decision Tree
    model = DecisionTreeClassifier(random_state = 150)

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

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=150)
```

The code creates a Decision Tree model using DecisionTreeClassifier from sklearn.tree and fits it to the training data (x_train, y_train). The random_state = 150 ensures that the model's training process is reproducible. After this, the model is trained on the provided data and is ready to make predictions on new, unseen data.

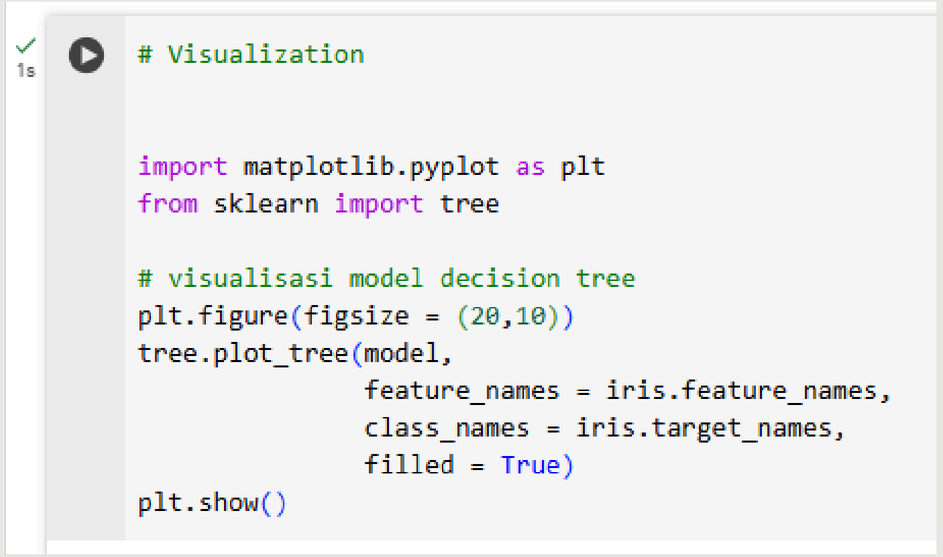
Predict & Evaluate

```
[15] # Predict & Evaluate


     from sklearn.metrics import accuracy_score

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

     # hitung akurasi model
     accuracy = accuracy_score(y_test, y_pred)
     print(f"Accuracy: {accuracy*100:.2f}%")
```

```
Accuracy: 100.00%
```

The code uses the trained Decision Tree model to predict the target values (y_pred) on the test data (x_test). Then, it evaluates the model's performance by calculating the accuracy score using accuracy_score from sklearn.metrics. The accuracy is printed as a percentage, indicating how well the model correctly classified the test data compared to the true labels (y_test). This is an important step in assessing the effectiveness of the model.

Visualization



The code visualizes the trained Decision Tree model using tree.plot_tree from sklearn. It generates a plot of the tree structure, displaying the decision rules, the features used at each node, and the class predictions at the leaf nodes. The filled=True option colors the nodes based on the predicted class, and the feature_names and class_names are used to label the tree with meaningful information. This helps in understanding how the model makes its predictions.

THANK YOU