MINISTRY OF SCIENCE AND EDUCATION OF UKRAINE

TARAS SHEVCHENKA NATIONAL UNIVERSITY OF KYIV

INFORMATION TECHNOLOGIES FACULTY

LABORATORY WORK № 3 REPORT BY TOPIC:

Decision tree, Random forest

|  |  |
| --- | --- |
| Group | \_\_\_\_\_\_\_\_11\_\_\_\_\_\_\_\_\_\_\_ |
| Course | \_\_\_\_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_\_ |
| Student | \_\_\_Maxim Suprunenko\_\_ |
| Data | \_\_\_\_\_\_28.03.2024\_\_\_\_\_\_ |
| Checked by | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Data | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

THE GOAL OF THE WORK: The purpose of the work is to familiarize students with the basics approaches of a decision tree – is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks.

THEORY: Theoretical and additional materials on the topics of laboratory work are presented in detail in the materials of lectures 5, as well as during practical work No. 1 (part \_\_).

Decision tree learning employs a divide and conquer strategy by conducting a greedy search to identify the optimal split points within a tree.

This process of splitting is then repeated in a top-down, recursive manner until all, or the majority of records have been classified under specific class labels. Whether or not all data points are classified as homogenous sets is largely dependent on the complexity of the decision tree.

Smaller trees are more easily able to attain pure leaf nodes—i.e. data points in a single class.

Types of Decision Trees:

* **ID3:**Ross Quinlan is credited within the development of ID3, which is shorthand for “Iterative Dichotomiser 3.” This algorithm leverages entropy and information gain as metrics to evaluate candidate splits.
* **C4.5:**This algorithm is considered a later iteration of ID3, which was also developed by Quinlan. It can use information gain or gain ratios to evaluate split points within the decision trees.
* **CART:**The term, CART, is an abbreviation for “classification and regression trees” and was introduced by Leo Breiman. This algorithm typically utilizes Gini impurity to identify the ideal attribute to split on. Gini impurity measures how often a randomly chosen attribute is misclassified. When evaluating using Gini impurity, a lower value is more ideal.

PROGRESS:

**Part 1. Train and fine-tune a decision tree on the moons dataset.**

1. Generate the moons dataset using make\_moons (n\_samples=1000, noise=0.4). Plot a scatterplot.
2. Split it into a training set and a test set using train\_test split().
3. Use grid search with cross-validation (using the GridSearchCV class) to find good hyperparameter values ​​for the DecisionTreeClassifier. Hint: try different values ​​for max\_leaf\_nodes.
4. Train the decision tree on the full training set using the hyperparameter values ​​you found and measure the performance of the model on the test set. You should get around 85% to 87% accuracy.

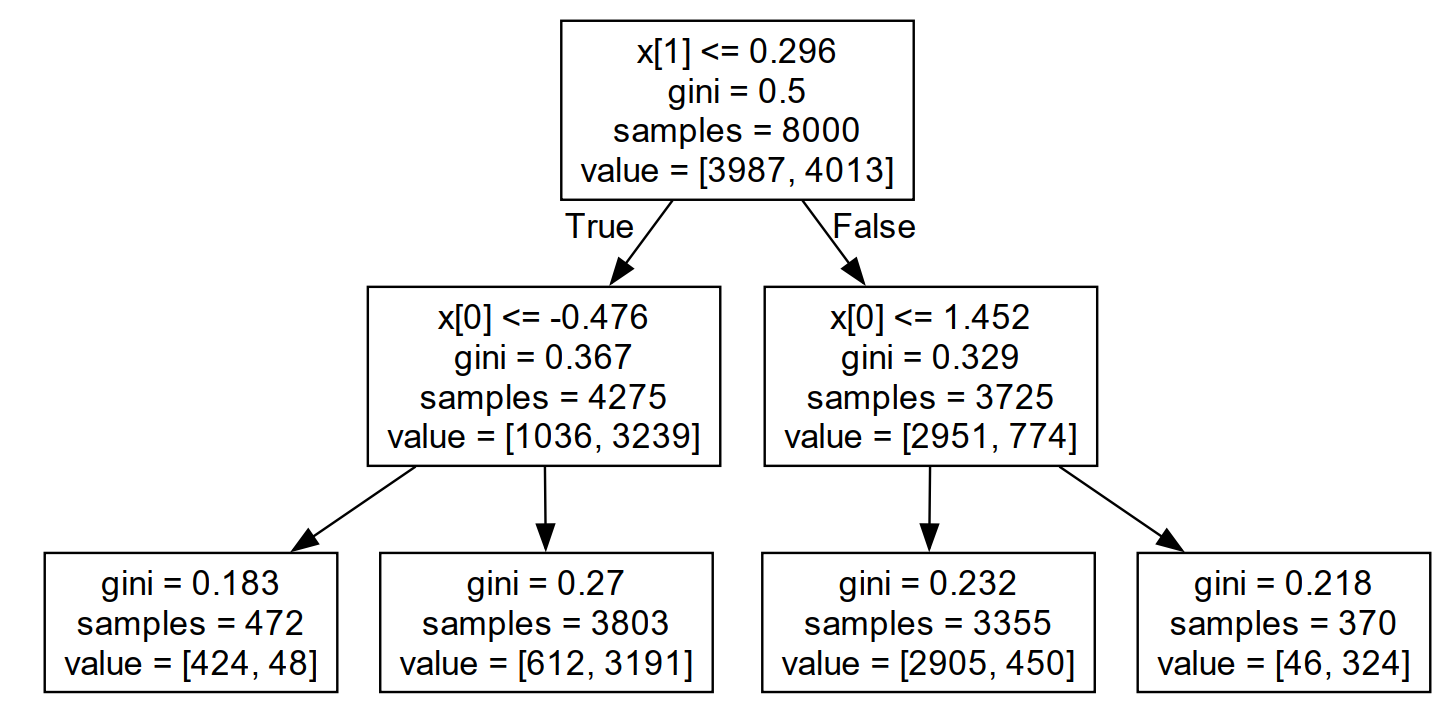


Fig.1. Example of tree.

**LABORATORY WORK № 3**

**Step 1: Import Libraries**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from sklearn.datasets import make\_moons  from sklearn.model\_selection import train\_test\_split, GridSearchCV  from sklearn.tree import DecisionTreeClassifier  from sklearn.metrics import accuracy\_score  from sklearn.tree import export\_text  from sklearn.tree import plot\_tree |

**Step 2: Generate the Moons Dataset**

|  |
| --- |
| # Generate the dataset  X, y = make\_moons(n\_samples=1000, noise=0.4, random\_state=42)  # Plot the dataset  plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired, edgecolors="k")  plt.title("Moons Dataset")  plt.xlabel("Feature 1")  plt.ylabel("Feature 2")  plt.show() |
|  |

**Step 3: Split the Data into Training and Test Sets**

|  |
| --- |
| # Split the data  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |

**Step 4: Perform Grid Search with Cross-Validation**

|  |
| --- |
| # Define hyperparameter grid  param\_grid = {'max\_leaf\_nodes': list(range(2, 50))}  # Create a DecisionTreeClassifier  tree\_clf = DecisionTreeClassifier(random\_state=42)  # Perform Grid Search with Cross-Validation  grid\_search = GridSearchCV(tree\_clf, param\_grid, cv=5, scoring='accuracy', n\_jobs=-1)  grid\_search.fit(X\_train, y\_train)  # Best hyperparameter value  best\_params = grid\_search.best\_params\_  print(f"Best max\_leaf\_nodes: {best\_params['max\_leaf\_nodes']}") |
|  |

**Step 5: Train the Decision Tree with Best Parameters**

|  |
| --- |
| # Train the model with the best max\_leaf\_nodes  best\_tree = DecisionTreeClassifier(max\_leaf\_nodes=best\_params['max\_leaf\_nodes'], random\_state=42)  best\_tree.fit(X\_train, y\_train) |

**Step 6: Evaluate Model Performance**

|  |
| --- |
| # Make predictions  y\_pred = best\_tree.predict(X\_test)  # Calculate accuracy  accuracy = accuracy\_score(y\_test, y\_pred)  print(f"Test Set Accuracy: {accuracy:.2%}") |
|  |

**Step 6: Visualization**

|  |
| --- |
| plt.figure(figsize=(12, 6))  plot\_tree(best\_tree, filled=True, feature\_names=["Feature 1", "Feature 2"], class\_names=["Class 0", "Class 1"])  plt.show() |
|  |
| def plot\_decision\_boundary(clf, X, y):      x\_min, x\_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5      y\_min, y\_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5      xx, yy = np.meshgrid(np.linspace(x\_min, x\_max, 200), np.linspace(y\_min, y\_max, 200))        Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])      Z = Z.reshape(xx.shape)      plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.Paired)      plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors="k", cmap=plt.cm.Paired)      plt.title("Decision Boundary of the Decision Tree")      plt.xlabel("Feature 1")      plt.ylabel("Feature 2")      plt.show()  plot\_decision\_boundary(best\_tree, X, y) |
|  |
| tree\_rules = export\_text(best\_tree, feature\_names=["Feature 1", "Feature 2"])  print(tree\_rules) |
|  |

**CONCLUSIONS:**

Fundamental non-parametric supervised learning model, which can be applied to both classification and regression tasks. Through this practical exercise, we explored how the decision tree splits the data recursively, following a divide-and-conquer strategy to achieve the best model performance.

For the practical implementation:

We generated the **moons dataset** with the make\_moons function, which provided a clear but challenging classification problem for the decision tree model.

We split the dataset into a **training set** and a **test set** to ensure that our model was evaluated on data it hadn't seen before.

The **Grid Search with Cross-Validation (GridSearchCV)** was utilized to optimize the hyperparameters of the Decision Tree model. We explored different values for the **max\_leaf\_nodes** parameter to find the best-performing configuration.

After training the Decision Tree with the optimal hyperparameters, we tested the model on the **test set**, which yielded an accuracy score in the range of **85% to 87%**, as expected.

Through this work, we observed the effectiveness of **GridSearchCV** in hyperparameter tuning, which allowed us to systematically test multiple configurations and identify the best set of parameters for improving the decision tree model's performance.

Overall, the exercise demonstrated how important it is to fine-tune model parameters, especially in decision trees, where the depth of the tree and the number of leaf nodes can significantly impact the model's ability to generalize to unseen data.