Experiment No. 6: Understanding Accuracy Parameters

Aim

To understand the various accuracy parameters and write a Python program to read a dataset and apply decision tree classifier and measure various accuracy parameters.

Source Code

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve
import matplotlib.pyplot as plt

# Read data
data = pd.read_csv('data.csv')
label_encoder = LabelEncoder()
data['activity'] = label_encoder.fit_transform(data['activity'])

# Feature selection
X = data[['time', 'timestamp', 'x-acceleration', 'y-acceleration', 'z-acceleration']]
y = data['activity']

# Preprocessing
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model creation and training
decision_tree_model = DecisionTreeClassifier()
decision_tree_model.fit(X_train, y_train)

# Predictions
y_pred_decision_tree = decision_tree_model.predict(X_test)

# Evaluation metrics
accuracy_decision_tree = accuracy_score(y_test, y_pred_decision_tree)
print(f"Accuracy on the test set (Decision Tree): {accuracy_decision_tree}")

# Confusion matrix
confusion_decision_tree = confusion_matrix(y_test, y_pred_decision_tree)
print("Confusion Matrix for Decision Tree:")
print(classification_decision_tree)

# Cross-validation scores
cv_scores_decision_tree = cross_val_score(decision_tree_model, X, y, cv=5)
print("Cross-Validation Scores for Decision Tree:", cv_scores_decision_tree)

# ROC Curve
n_classes = len(label_encoder.classes_)
for i in range(n_classes):
    y_one_vs_all = (y_test == i)
    y_score = decision_tree_model.predict_proba(X_test)[:, i]
    fpr, tpr = roc_curve(y_one_vs_all, y_score)
    plt.plot(fpr, tpr, label=f'ROC curve (area={roc_auc[i]:.2f}) for class {label_encoder.classes_[i]}')
```

```
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Decision Tree (One-vs-All)')
plt.legend(loc='lower right')
plt.show()
```

Output

The experiment produced the following results:

1. Decision Tree Classification Report showing:
    - Precision, recall, and F1-score for each class
    - Support values
    - Overall accuracy of 0.99
    - Weighted average scores of 0.99
2. Cross-validation scores for Decision Tree: [0.34141859, 0.21639939, 0.33853196, 0.28504617, 0.16092307]
3. ROC Curves showing the performance for each class:
    - Downstairs (area = 0.99)
    - Jogging (area = 1.00)
    - Sitting (area = 1.00)
    - Standing (area = 1.00)
    - Upstairs (area = 0.99)
    - Walking (area = 1.00)

Implementation of Multi-Layer Neural Network for Number Comparison

Experiment 5 Documentation

Introduction

This experiment implements a multi-layer neural network designed to compare two numerical inputs (x_1 and x_2) and determine if $x_1 > x_2$. The network architecture consists of three distinct layers:

1. Input Layer: Accepts two inputs (x_1 and x_2)
2. Hidden Layer: Contains two neurons with non-linear activation (ReLU)
3. Output Layer: Produces a binary output (1 if $x_1 > x_2$, 0 otherwise)

Implementation

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

def create_comparison_network():
    # Initialize the sequential model
    model = Sequential([
        # Input layer (2 inputs)
        Dense(2, input_shape=(2,), activation='relu'),

        # Hidden layer with ReLU activation
        Dense(2, activation='relu'),

        # Output layer with sigmoid activation for binary output
        Dense(1, activation='sigmoid')
    ])

    # Compile the model
    model.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['accuracy'])

    return model

# Generate training data
def generate_training_data(num_samples=1000):
    x1 = np.random.uniform(0, 1, (num_samples, 1))
    x2 = np.random.uniform(0, 1, (num_samples, 1))
    X = np.hstack((x1, x2))
    y = (x1 > x2).astype(int)
    return X, y

# Create and train the model
model = create_comparison_network()
X_train, y_train = generate_training_data()
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

# Test the model
def test_model(model):
    test_cases = [
        [0.7, 0.3],
        [0.2, 0.8],
        [0.5, 0.5],
        [0.9, 0.1]
    ]

    print("\nTest Results:")
    print("x₁\tx₂\tPrediction")
    print("-" * 30)
```

```
    for test in test_cases:
        prediction = model.predict(np.array([test]), verbose=0)
        print(f"{test[0]:.1f}\t{test[1]:.1f}\t{prediction[0][0]:.3f}")

test_model(model)
```

Sample Output

```
Test Results:
x₁      x₂      Prediction
------------------------------
0.7     0.3     0.982
0.2     0.8     0.021
0.5     0.5     0.498
0.9     0.1     0.997
```

Analysis

The network successfully learns to compare two numbers using the following architecture:

- Input Layer: 2 neurons (x_1 and x_2)
- Hidden Layer: 2 neurons with ReLU activation
- Output Layer: 1 neuron with sigmoid activation

The output demonstrates that the network correctly:

1. Outputs values close to 1 when $x_1 > x_2$
2. Outputs values close to 0 when $x_1 < x_2$
3. Outputs values close to 0.5 when $x_1 = x_2$

The ReLU activation function in the hidden layer helps the network learn the non-linear decision boundary necessary for comparison operations, while the sigmoid activation in the output layer constrains the output to the range [0,1].