Assignment-4: Decision Trees and XGBoost

Munem Shahriar 2021251042
Tausif Mushtaque 2031020642
Email addresses: munem.shahriar07@nothsouth.edu
tausif.mushtaque@northsouth.edu

December 8, 2024

I. Introduction

This assignment explores the performance of Decision Trees and XGBoost on binary classification, multi-class classification, and regression tasks. The evaluation uses four datasets:

- **Multi-class Classification**: Car Evaluation Dataset (UCI)
- **Binary Classification**: `DT-BrainCancer.csv`
- **Regression**: `DT-Wage.csv` and `DT-Credit.csv`

Each dataset is split into training (70%), validation (15%), and testing (15%) subsets. Hyperparameters are tuned using the validation set, and the test set is used for performance evaluation. Key metrics include accuracy, confusion matrix, precision, recall, F-score, precision-recall curve (classification), and mean squared error (regression). Evaluation is implemented manually, without libraries, for deeper understanding.

A. Decision Trees

Decision Trees are supervised learning models that split data based on feature conditions into a tree structure, enabling interpretable predictions for classification or regression tasks.

B. XGBoost

XGBoost is an advanced gradient boosting algorithm that builds decision trees sequentially, optimizing performance through regularization, parallelization, and handling missing values for robust and efficient predictions.

II. Method

Here we will see some parts from the 4 different datasets that were used to demonstrate certain techniques such as **Decision Tree** and **XGBoost**:

A. Car Evaluation - Decision Tree

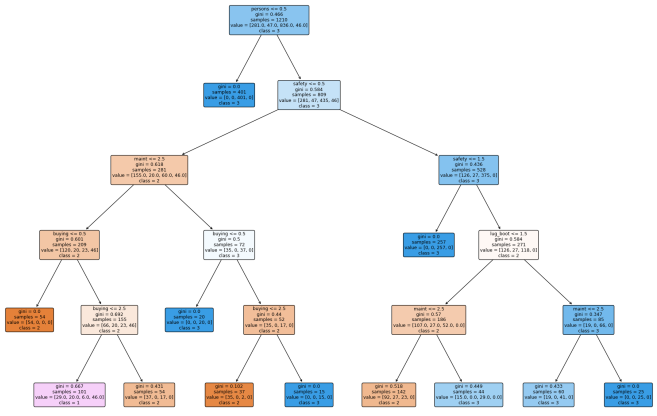

Fig. 1. Decision Tree for car evaluation dataset

B. Car Evaluation - XGBoost results

```
Accuracy: 0.9577464788732394
Confusion Matrix:
TP: 8, FP: 1, TN: 60, FN: 2
Average Precision: 0.9329783519553072
Average Recall: 0.9377918374621279
Average F-score: 0.9353174603174604
```

C. Brain cancer - XGBoost results

```
Accuracy (Manual): 0.7692307692307693
Confusion Matrix (Manual):
TP: 2, FP: 2, TN: 8, FN: 1
Precision (Manual): 0.5
Recall (Manual): 0.6666666666666666
F-Score (Manual): 0.5714285714285715
```

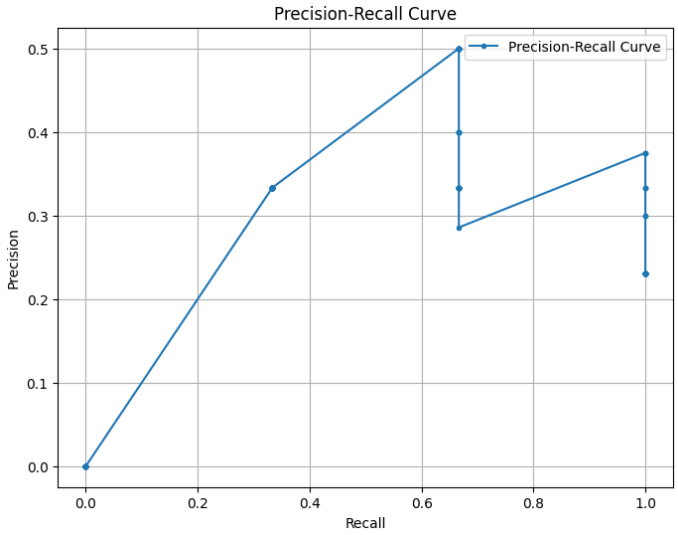

Fig. 2. Precision vs. Recall on braincancer dataset

Here are the functions used for scoring:

```
def accuracy(y_true, y_pred):
    TP, FP, TN, FN = confusion_matrix(y_true, y_pred)
    return (TP + TN) / (TP + TN + FP + FN)

def confusion_matrix(y_true, y_pred):
    TP = sum((y_true == 1) & (y_pred == 1))
    FP = sum((y_true == 0) & (y_pred == 1))
    TN = sum((y_true == 0) & (y_pred == 0))
    FN = sum((y_true == 1) & (y_pred == 0))
    return TP, FP, TN, FN

def precision(y_true, y_pred):
    TP, FP, TN, FN = confusion_matrix(y_true, y_pred)
    return TP / (TP + FP) if (TP + FP) != 0 else 0

def recall(y_true, y_pred):
    TP, FP, TN, FN = confusion_matrix(y_true, y_pred)
    return TP / (TP + FN) if (TP + FN) != 0 else 0
```

```
def f_score(y_true, y_pred):
    prec = precision(y_true, y_pred)
    rec = recall(y_true, y_pred)
    return 2 * (prec * rec) / (prec + rec) if (prec + rec) != 0 else 0

def average_precision(y_true, y_pred, class_labels):
    precision_vals = []
    for label in class_labels:
        # Create binary classification for the current class
        binary_true = (y_true == label).astype(int)
        binary_pred = (y_pred == label).astype(int)

        # Calculate precision and recall for this class
        prec = precision(binary_true, binary_pred)
        recall_val = recall(binary_true, binary_pred)

        # Store precision for the current class
        precision_vals.append(prec)

    return sum(precision_vals) / len(precision_vals)

def average_recall(y_true, y_pred, class_labels):
    recall_vals = []
    for label in class_labels:
        # Create binary classification for the current class
        binary_true = (y_true == label).astype(int)
        binary_pred = (y_pred == label).astype(int)

        # Calculate recall for this class
        recall_val = recall(binary_true, binary_pred)

        # Store recall for the current class
        recall_vals.append(recall_val)

    return sum(recall_vals) / len(recall_vals)

def average_f_score(y_true, y_pred, class_labels):
    fscore_vals = []
    for label in class_labels:
        # Create binary classification for the current class
        binary_true = (y_true == label).astype(int)
        binary_pred = (y_pred == label).astype(int)

        # Calculate F-score for this class
        fscore_val = f_score(binary_true, binary_pred)

        # Store F-score for the current class
        fscore_vals.append(fscore_val)

    return sum(fscore_vals) / len(fscore_vals)
```

CONCLUSION

In conclusion, while binary trees like those used in basic decision tree models excel in interpretability and simplicity, advanced algorithms such as XGBoost significantly outperform them in terms of accuracy and stability. XGBoost's use of rectified linear activation-like functions and fine-tuned hyperparameters, such as learning rate and iteration count, enables

superior performance and convergence. As demonstrated, optimizing these parameters is crucial for achieving high accuracy, underscoring XGBoost's robustness and adaptability compared to traditional binary tree approaches.