ELL409 - Assignment 2 Report
Decision Tree Classifier

Priyal Jain 2021MT60949

October 2024

Contents

1 Objective

The objective of this report is to implement a Decision Tree Classifier from scratch to predict whether a client will subscribe to a term deposit based on various demographic and campaign-related features.

2 Data Preprocessing

2.1 Visualization of Class Distribution

To assess the balance of the dataset, the class distribution was visualized using a pie chart, which revealed significant class imbalance.

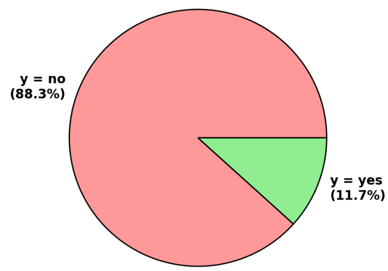

Figure 1: Class Distribution in the Dataset

2.2 Comparison of Balancing Techniques

To address the class imbalance, two techniques were considered:

- **Upsampling**:
  - Involves replicating instances of the minority class to achieve a more balanced dataset.
  - Simple and retains the original data distribution.
- **SMOTE (Synthetic Minority Over-sampling Technique)**:
  - Generates synthetic instances of the minority class.
  - Can introduce noise and may not represent the true distribution of the data.

Decision: **Upsampling** was chosen for its simplicity and effectiveness in preserving original samples, minimizing the risk of noise introduction.

2.3 Scaling Numeric Features

Numeric features were scaled to ensure uniform contribution to model performance. The following steps were applied:

- **Standardization**: Each feature was centered to have a mean of zero and a standard deviation of one.

This step is crucial for algorithms sensitive to the scale of input data.

2.4 Handling Categorical Features

Two encoding methods were considered for categorical features:

- **Label Encoding**:
  - Converts categorical values into integers.
  - Efficient in memory but introduces an ordinal relationship that may mislead the model.
- **One-Hot Encoding**:
  - Converts each category into a binary vector.
  - Eliminates ordinal relationships, ensuring accurate representation of categorical data.

Decision: **One-hot encoding** was chosen due to its ability to accurately represent the categorical nature of the data without misleading relationships, leading to improved model performance.

3 Decision Tree Criteria

3.1 Evaluation of Split Criteria

In constructing the Decision Tree Classifier, two criteria for evaluating splits were considered: Gini impurity and entropy.

- **Gini Impurity**:
  - Measures the impurity of a node.
  - Formula:
    $$Gini = 1 - \sum (p_i^2)$$
    where p_i is the probability of class i.
  - Benefits: Computationally efficient and performs well in binary classification tasks.
- **Entropy**:

- Measures the information gain of a node.
- Formula:

$$Entropy = -\sum(p_i \log_2 p_i)$$

where p_i is the probability of class i.
- Benefits: Provides a more thorough measure of uncertainty in the dataset.

3.2 Comparison of Criteria

Both criteria were evaluated to determine their effectiveness in creating splits:

- **Gini Impurity** was found to be:
  - Faster to compute.
  - Consistently provided high accuracy in binary classification tasks.
- **Entropy**:
  - While offering a deeper understanding of the dataset, it required more computation time.

3.3 Decision

Ultimately, **Gini impurity** was selected as the splitting criterion for the Decision Tree Classifier due to its computational efficiency and strong performance in the given classification task.

4 Metrics

To evaluate the performance of the Decision Tree Classifier, the following key metrics were utilized:

- **Accuracy**: Measures the proportion of correctly classified instances.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- **Precision**: Measures the proportion of true positives among all positive predictions.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall (Sensitivity)**: Measures the proportion of true positives among all actual positives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1 Score**: The harmonic mean of precision and recall, providing a balance between the two.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5 Pre-Pruning

5.1 Technique

During the pre-pruning with **k-fold cross validation**, various hyperparameters were evaluated to optimize the Decision Tree Classifier. The following hyperparameters were considered:

- **Max Depth**: Limits the maximum depth of the tree. Various depths were tested, including 8, 10, and 20
- **Min Samples Split**: Determines the minimum number of samples required to split an internal node. Values of 2, 5, and 10 were assessed.

5.2 Observation

- As the maximum depth increased, the model's training accuracy improved. However, beyond a certain point, further increases in max depth did not yield a significant improvement in validation accuracy.
- This observation indicated that deeper trees may not necessarily translate to better generalization, highlighting the importance of balancing model complexity with performance.

5.3 Decision

Overall, These findings suggested that a **max depth of 10 and min samples split of 5** provided a suitable trade-off between the complexity of the model and accuracy predicted.

6 Post-Pruning

6.1 Technique

- **Cost Complexity Pruning**: This technique was employed to reduce the size of the decision tree by removing branches that have little importance in predicting target outcomes. It involves finding the optimal trade-off between tree size and prediction accuracy.
- **Alpha Parameter**: The alpha parameter controls the trade-off between the tree's complexity and its accuracy. A higher alpha value will result in more pruning, simplifying the tree but potentially leading to underfitting.

- **Values Considered**:
  - **0.001**: Minimal pruning, retaining most of the tree structure.
  - **0.01**: Moderate pruning, striking a balance between complexity and accuracy.
  - **0.1**: Aggressive pruning, simplifying the tree significantly.

6.2 Observation

Increasing alpha generally reduced training accuracy but improved validation accuracy, indicating a better generalization to unseen data.

6.3 Decision

The optimal alpha value, **0.01**, was identified through these findings. This choice demonstrated improved performance in terms of validation accuracy while effectively controlling overfitting.

7 Comparison of Decision Trees

In this section, a comparative analysis of three different decision tree configurations is presented:

7.1 Results Summary

The following table summarizes the key metrics for each tree configuration:

Model Version	Number of Nodes	Accuracy	Precision	Recall	F1 Score
Base Tree	5,647	0.9233	0.9021	0.9509	0.9259
Pre Pruned	701	0.8599	0.8337	0.9017	0.8664
Pre + Post Pruned	435	0.8600	0.8356	0.8990	0.8661

Table 1: Comparison of Different Decision Tree Models

7.2 Observations

- The **initial Decision Tree without any Pruning** had a significantly higher accuracy (0.9233) but contained a large number of nodes (5647), clearly indicating overfitting. The complexity of the model resulted in high accuracy but may not generalize well to unseen data.
- The **Pre-Pruned Tree** demonstrated a slightly lower accuracy (0.8599) but drastically reduced complexity with only 701 nodes. This reduction in complexity reflects a more balanced model, effectively **managing overfitting** while maintaining reasonable predictive performance.

- **Post-Pruning the Pre-Pruned Tree** maintained a similar accuracy (0.8600) to the pre-pruned tree while further reducing the number of nodes to just 435. This indicates that the post-pruning technique effectively simplified and **generalized the model** without sacrificing predictive performance, making it the best model.

7.3 Decision

The **t**ree obtained with a **combination of Pre-Pruning and Post-Pruning** represents an optimal balance between model complexity and performance, achieving satisfactory accuracy with fewer nodes and better generalization.

8 Imported Libraries

- **pandas** for data manipulation and analysis.
- **numpy** for numerical operations and array handling
- **copy** for creating deep copies of objects
- **time** for tracking execution time
- **SMOTE** (from imblearn.over_sampling) for handling class imbalance
- **resample** (from sklearn.utils) for handling class imbalance
- **KFold** (from sklearn.model_selection) for cross-validation
- **train_test_split** (from sklearn.model_selection) for splitting data