CMSC 5724 Project #1: Decision Tree

1155215526 Ding Luxiao 1155215527 Chen Chen
1155215555 Chen Yubin 1155215807 Zhang Jiarui

Abstract

In this report, we made a predictive analysis using the Adult dataset. We employed Hung's algorithm to construct a decision tree, leveraging the Gini index as the criterion for optimal splits. The dataset was processed through a series of utility functions designed for data reading, frequency counting, label encoding, and model evaluation. Key optimizations reduced the time complexity for calculating ordered features to O(nlogn), enhancing efficiency in identifying the best split points.

The experimental setup involved training on a dataset of 30,162 records and evaluating on a separate set of 15,060 records, with specified parameters for minimum samples per split and maximum depth 10. Results indicated an accuracy of 86.718% on the training set and 85.604% on the evaluation set, with precision, recall, and F1 scores illustrating the model's performance. The decision tree's structure was documented, providing insights into the predictive relationships within the dataset.

Dataset

We will use the Adult dataset to predict whether an individual's annual income exceeds $50K/year based on census data, also known as the "Census Income" dataset. The training set is adult.data, and the evaluation set is adult.test.

It's description is available https://archive.ics.uci.edu/dataset/2/adult

It can be downloaded in https://archive.ics.uci.edu/static/public/2/adult.zip.

Algorithm

Using the optimization logic from ex1 Problem 3, we reduced the time complexity for calculating ordered features to O(n log n).

Hung's Algorithm for Decision Tree

/* S is the training set; the function returns the root of a decision tree */

1. if all the objects in S belong to the same class
2. return a leaf node with the value of this class

3. if (all the objects in S have the same attribute values) or (|S| is too small)
4. return a leaf node whose class value is the majority one in S
5. find the “best” split attribute A* and predicate P* using GINI
6. S1 ← the set of objects in R satisfying P*; S2 ← S \ S1
7. u1 ← Hunt(R1); u2 ← Hunt(R2)
8. create a root u with left child u1 and right child u2
9. set Au ← A* and Pu ← P*
10. return u

1. Define the Dataset: Let S be the set of records in the table. Each record r is in the form (rA, rB), representing its values on attribute A and class label B, respectively.
2. Sort the Dataset: Sort the dataset S in ascending order by the values of attribute A to efficiently calculate the Gini index for each possible split point.
3. Initialize a Counter: Set up a counter c to track the number of positive (yes) records up to and including the current split point.
4. Traverse the Records: After sorting, traverse the records of S in ascending order of A.
5. Update the Counter:

If the class label B of the record r is positive (yes), increment the counter c by 1.

For each record r, consider its A attribute value r.A as a potential split point and set c1y(a) to the current value of the counter c, representing the number of positive class records up to and including the current record r.

6. Calculate the Gini Index for Each Split: Using the counts obtained in the previous steps, calculate the Gini index for each possible split point. The Gini index measures the impurity of a dataset; the lower the value, the higher the purity of the resulting subsets.
7. Select the Best Split Point: Go through all possible split points and choose the one that results in the smallest Gini index as the best split point.

The key to this algorithm is performing a single sorting operation (O(n log n) time complexity) and a single traversal (O(n) time complexity), allowing efficient calculation of the Gini index for all possible split points, thus identifying the best split point. This method avoids calculating the Gini index for each split point individually, significantly improving efficiency.

Experimental Process

Utility Functions

Includes functions for data reading, frequency counting, label encoding, and model evaluation.

read_data: Reads data files and converts each line into a dictionary format, stored in a list.

count_frequencies: Counts the frequency of each attribute in the data.

get_sorted_list_value: Sorts attribute frequencies by value.

get_sorted_list_key: Sorts attribute frequencies by key.

label_encode: Encodes nonnumeric labels using numeric substitutes.

evaluate_model: Computes model evaluation metrics, including accuracy, precision, recall, and F1 score.

print_model_evaluation_result: Prints the model's evaluation results.

TreePrinter: Used to print the structure of the decision tree.

Decision Tree Construction

Uses the Gini index as the splitting criterion. Below is a detailed explanation of each part:

1. Data Preparation: Use the split_dataset method to divide the dataset into two subsets based on attributes and split points.

```
  def split_dataset(self, features, labels, feature_index, split) ->
tuple[list]:
    left_features, left_labels = [], []
    right_features, right_labels = [], []
    for i in range(len(features)):
      if feature_index in self.numeric_feature_index: # For numeric
        if features[i][feature_index] <= split:
          left_features.append(features[i])
          left_labels.append(labels[i])
        else:
          right_features.append(features[i])
          right_labels.append(labels[i])
      else: # For non-numeric
        if features[i][feature_index] == split:
          left_features.append(features[i])
          left_labels.append(labels[i])
        else:
          right_features.append(features[i])
          right_labels.append(labels[i])
    return left_features, left_labels, right_features, right_labels
```

2. Gini Index Calculation: Use the gini and gini_split methods to calculate the Gini index for labels, assessing the quality of the splits.

```
  def gini(slef, labels: list) -> float:
    total = len(labels)
    if total == 0:
      return 0
    return 1 - ((labels.count(0) / total) ** 2 + (labels.count(1) /
total) ** 2)
```

3. Best Split Point Selection: The find_best_split_gini method traverses all features and possible split points to find the feature and split value that minimize the Gini index.

```
  def find_best_split_gini(self, features, labels) -> tuple:
    best_feature_index = None
    best_split = None
    best_gini = float('inf')
    n_features = len(features[0])
    for feature_index in range(n_features):

      # 如果该特征是连续的，使用数值分割点
      if feature_index in self.numeric_feature_index:

        # 对数据按特征值排序
        feature_and_labels = sorted(zip(features, labels), key=lambda x:
x[0][feature_index])
        sorted_features = [f for f, _ in feature_and_labels]
        sorted_labels = [l for _, l in feature_and_labels]

        # 累积统计左右子集的 "yes" 和 "no" 计数
        left_yes_count = 0
        left_no_count = 0
        right_yes_count = sorted_labels.count(1)
        right_no_count = len(sorted_labels) - right_yes_count

        # 遍历所有可能的分割点，更新左右子集计数并计算基尼指数
        for i in range(1, len(sorted_features)):
          if sorted_labels[i - 1] == 1:
            left_yes_count += 1
            right_yes_count -= 1
          else:
            left_no_count += 1
            right_no_count -= 1
```

```
            # 确保每个分割点都是有效的
            if sorted_features[i][feature_index] == sorted_features[i -
1][feature_index]:
              continue

            # 计算当前分割点的基尼指数
            total_left = left_yes_count + left_no_count
            total_right = right_yes_count + right_no_count
            gini_left = 1 - (left_yes_count / total_left) ** 2 -
(left_no_count / total_left) ** 2
            gini_right = 1 - (right_yes_count / total_right) ** 2 -
(right_no_count / total_right) ** 2
            weighted_gini = (total_left / len(labels)) * gini_left +
(total_right / len(labels)) * gini_right

            # 更新最佳分割点
            if weighted_gini < best_gini:
              best_gini = weighted_gini
              best_feature_index = feature_index
              best_split = (sorted_features[i - 1][feature_index] +
sorted_features[i][feature_index]) / 2

        # 如果是非连续变量（分类特征），则逐个类别进行分割
        else:
          unique_values = set([row[feature_index] for row in features])
          for value in unique_values:
            _, left_labels, _, right_labels = self.split_dataset(features,
labels, feature_index, value)

            # 如果划分的子集为空，则跳过该分割
            if len(left_labels) == 0 or len(right_labels) == 0:
              continue

            # 计算当前分割的基尼指数
            current_gini = self.gini_split(left_labels, right_labels)

            # 更新最佳分割点
            if current_gini < best_gini:
              best_gini = current_gini
              best_feature_index = feature_index
              best_split = value

    return best_feature_index, best_split
```

4. Tree Construction: The fit method recursively builds the decision tree until stop conditions are met (e.g., all labels are the same, sample size is less than the minimum split size, or maximum depth is reached).

```
 def fit(self, features: list, labels: list, depth=0) -> tuple:

    n_samples = len(labels)
    # 如果所有标签相同，返回该标签
    if len(set(labels)) == 1:
      return labels[0]

    # 检查是否没有特征或样本数小于最小分割样本数，或超过了最大深度
    if len(features) == 0 or \
            n_samples < self.min_samples_split or \
            (self.max_depth is not None and depth >= self.max_depth):
      # 确保 labels 不为空，否则返回 None 或一个默认值
      return max(set(labels), key=labels.count) if labels else None

    # 找到最佳的特征和分割点
    best_feature_index, best_split = self.find_best_split_gini(features,
labels)

    # 如果没有找到合适的分割点，返回当前节点的多数类
    if best_feature_index is None:
      return max(set(labels), key=labels.count) if labels else None

    # 根据最佳分割点进行划分
    left_features, left_labels, right_features, right_labels =
self.split_dataset(features, labels, best_feature_index,

      best_split)

    # 避免递归到空子集
    if len(left_labels) == 0 or len(right_labels) == 0:
      return max(set(labels), key=labels.count) if labels else None

    # 递归生成左子树和右子树
    left_subtree = self.fit(left_features, left_labels, depth + 1)
    right_subtree = self.fit(right_features, right_labels, depth + 1)

    return (best_feature_index, best_split, left_subtree, right_subtree)
```

5. Prediction: The predict method uses the constructed decision tree to make predictions on new datasets.

```
def _predict(self, sample, tree):
  if not isinstance(tree, tuple):
    return tree
  feature_index, split, left_subtree, right_subtree = tree
  if feature_index in self.numeric_feature_index:
    if sample[feature_index] <= split:  # For numeric
      return self._predict(sample, left_subtree)
    else:
      return self._predict(sample, right_subtree)
  else:  # For non-numeric
    if sample[feature_index] == split:
      return self._predict(sample, left_subtree)
    else:
      return self._predict(sample, right_subtree)
```

Experimental Result

We conducted evaluations on the training set and evaluation set, with training data size: 30,162 and evaluation data size: 15,060, using min_samples_split=10 and max_depth=10. Results for the evaluation set are generated and partially displayed.

Table1: Experimental Result in Training SET and Evaluation SET

Dataset	Accuracy	Precision	Recall	F1 Score
Training SET	0.86718	0.79779	0.62480	0.70078
Evaluation SET	0.85604	0.76505	0.59757	0.67102

We printed the complete structure of the decision tree and the evaluation set selection in res.txt.

Appendix:

1. github：https://github.com/Chenchen2001/5724Proj1
2. Evaluation SET Result

```
traing data size: 30162  evaluation data set: 15060
min_samples_split=10, max_depth=5
========== TRAINING MODEL on Training SET ==========
========== TRAINING MODEL FINISHED ==========
========== EVALUATION on Training SET ==========
Accuracy: 0.84275
Precision: 0.79062
Recall: 0.50093
F1 Score: 0.61329
========== EVALUATION on Evaluation SET ==========
Accuracy: 0.84064
Precision: 0.77993
Recall: 0.48946
F1 Score: 0.60146
========== DECISION TREE RESULT ==========
|- martial-status is Married-civ-spouse
L   |- education-num <= 12
L   L   |- capital-gain <= 5013
L   L   L   |- education-num <= 8
L   L   L   L   |- capital-loss <= 1735
L   L   L   R   |- age <= 35
L   L   R   |- age <= 60
L   L   R   L   |- education is Preschool
L   L   R   L   L   |- INCOME <=50k
L   L   R   L   R   |- INCOME >50k
L   L   R   R   |- workclass is Local-gov
L   L   R   R   L   |- INCOME <=50k
L   L   R   R   R   |- INCOME >50k
L   R   |- capital-gain <= 5013
L   R   L   |- capital-loss <= 1740
L   R   L   L   |- hours-per-week <= 30
L   R   L   L   L   |- INCOME <=50k
L   R   L   L   R   |- INCOME >50k
L   R   L   R   |- capital-loss <= 1977
L   R   L   R   L   |- INCOME >50k
L   R   L   R   R   |- INCOME >50k
L   R   R   |- age <= 79
L   R   R   L   |- occupation is Farming-fishing
L   R   R   L   L   |- INCOME >50k
L   R   R   L   R   |- INCOME >50k
L   R   R   R   |- INCOME >50k
R   |- capital-gain <= 6849
```