Understanding Decision Tree Based Classification Algorithms

Lakavat Umesh Chandra

Department of Computer Science and Engineering Indian Institute of Technology Jodhpur, Rajasthan, India
Email :lumeshchandra18@gmail.com   Ph: 8008281740

Abstract: One of the techniques of machine learning is Decision Tree. Decision Trees are considered to be one of the most popular approaches for rep-resenting classifiers. Researchers from various disciplines such as statistics, ma-chine learning, pattern recognition, and Data Mining have dealt with the issue of growing a decision tree from available data. We discuss constructing decision tree classifiers from scratch and its implementation.

Keywords: DecisionTree,Information Gain, Gini Index, Pruning,Stopping Criteria

Introduction to Decision Tree

A decision tree is a machine learning classifier that recursively partitions data into subspaces to make predictions.It is tree like structure, It consists of a root node (the starting point with no incoming edges), internal nodes (which split data based on attribute conditions), and leaf nodes (which represent final predictions or probabilities). At each internal node, the data is divided based on a specific attribute. For categorical attributes, splits are based on distinct values, while for numerical attributes, splits are based on ranges. Instances are classified by traversing the tree from the root to a leaf, following the attribute-based conditions along the path. Each path represents a decision rule, such as "If (Age ≤ 30) AND (Gender = Male), then the customer will respond to the mail." Decision trees are valued for their interpretability, but tree complexity—measured by factors like depth, number of nodes, or attributes—affects both their accuracy and usability. Techniques like pruning are used to simplify trees and avoid overfitting, ensuring better generalization.

Splitting Nodes

Each decision outcome at a node is called a split, since it corresponds to splitting a subset of the training data.
The number of links descending from a node is sometimes called node's branching factor or branching ratio, denoted B.

Impurity of Node

Impurity of a node N denoted as i(N) = 0 if all of the patterns that reach the node bear the same category label, and to be large if the categories are equally represented.

There are many ways to calculate Impurity of Node

1.Entropy Impurity (or Information Impurity):

Measures randomness in node if more pure node less entropy

$$i(N) = -\sum_j P(\omega_j) log_2 P(w_j)$$

Where P(ωj) is the fraction of patterns at node N that are in category ωj

2.Gini Impurity:

$$i(N) = \sum_{i \neq j} P(\omega_i) P(w_j)$$

$$i(N) = 1 - \sum_j P^2(w_j)$$

Where P(ωj) is the fraction of patterns at node N that are in category ωj

3.Misclassification Impurity:

$$i(N) = 1 - \max_j P(\omega_j)$$

Where P(ωj) is the fraction of patterns at node N that are in category ωj

Drop in impurity or Information gain

When Node is split into subnodes based of **best_feature** and **threshold value** impurity of Node drops can be measured in following way

$$\Delta i(N) = i(N) - P_l i(N_l) - (1 - P_l) i(N_r)$$

Where Nl and Nr are the left and right descendent nodes, i(Nl) and i(Nr) their impurities, and Pl is the fraction of patterns at node N that will go to Nl when property test T is used.

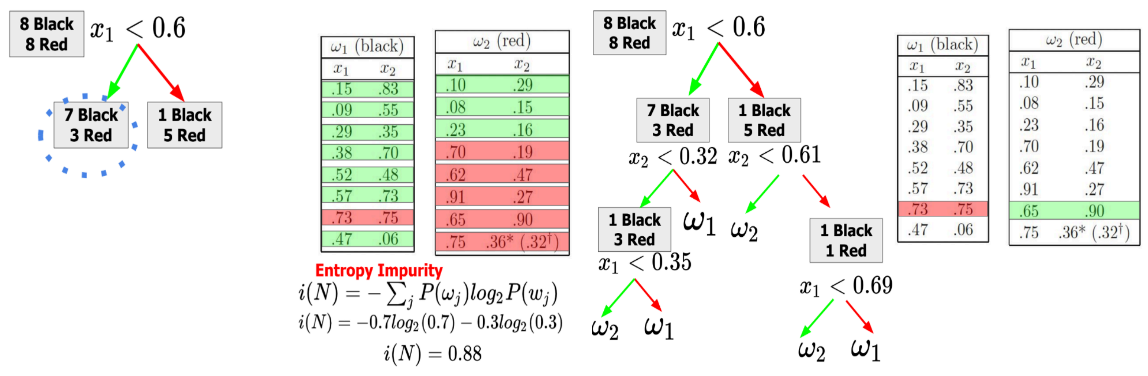
Information gain=Entopy(Parent)- Weighted .Entopy(Children)

Best_feature and Threshold

Node is split into children based on **Best_feature and threshold** to find best feature we consider all features available and we calculate information gain based on each feature and threshold which feature gives more information gain that is considered as best_feature and in same way for threshold

Feature	Information Gain
1.Gender	-0.5
2.Age	0.7

IG(Gender)<IG(Age) so Age is best feature to split node



ω_1 (black)		ω_2 (red)	
x_1	x_2	x_1	x_2
.15	.83	.10	.29
.09	.55	.08	.15
.29	.35	.23	.16
.38	.70	.70	.19
.52	.48	.62	.47
.57	.73	.91	.27
.73	.75	.65	.90
.47	.06	.75	.36* (.32†)

Entropy Impurity

$$i(N) = -\sum_j P(\omega_j) log_2 P(w_j)$$

$$i(N) = -0.7 log_2(0.7) - 0.3 log_2(0.3)$$

$$i(N) = 0.88$$

ω_1 (black)		ω_2 (red)	
x_1	x_2	x_1	x_2
.15	.83	.10	.29
.09	.55	.08	.15
.29	.35	.23	.16
.38	.70	.70	.19
.52	.48	.62	.47
.57	.73	.91	.27
.73	.75	.65	.90
.47	.06	.75	.36* (.32†)

Stopping Criteria

If we continue to grow the tree fully until each leaf node corresponds to the lowest impurity, then the data has typically been overfit. In the extreme but rare case, each leaf corresponds to a single training point and the full tree is merely a convenient implementation of a lookup table; it thus cannot be expected to generalize well.
Conversely, if splitting is stopped too early, then the error on the training data is not sufficiently low and hence performance may suffer.
Few stopping criterias we follow are max depth of tree,max splits, impurity

Pruning

In pruning, a tree is grown fully, that is, until leaf nodeshave minimum impurity.

Then, all pairs of neighboring leaf nodes are considered for elimination. Any pair whose elimination yields a satisfactory (small) increase in impurity is eliminated, and the common antecedent node declared a leaf.

Assignment of leaf node labels

In the more typical case, the leaf nodes have positive impurity, and each leaf should be labeled by the category that has most points represented.
An extremely small impurity is not necessarily desirable, since it may be an indication that the tree is overfitting the training data.

Learning decision trees

1. Start with an empty tree
2.Choose the next best attribute(feature) – for example,one that maximizes information gain
3.Split
4.Recurse

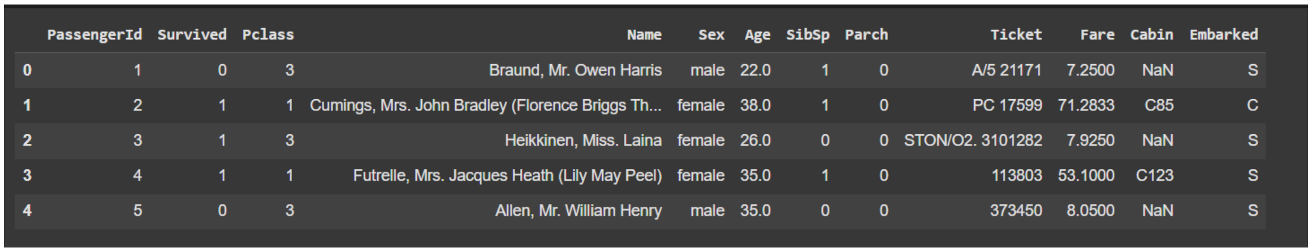
Implimentation of Decision Tree

Dataset on which Decision tree implemented is titanic.csv . It contains data to classify whether someone will survive in the Titanic wreck . Used the decision tree to classify it into one of the two classes, i.e., survived(1) or not survived (0). To calculate Impurity of Node Entropy imprity is used

$$i(N) = -\sum_j P(\omega_j)log_2 P(w_j)$$

Dataset

github: https://github.com/datasciencedojo/datasets/blob/master/titanic.csv



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Code: https://github.com/umesh-chandra-l/DecisionTree/blob/main/DecisionTree.ipynb

1.Node Class has attributes like feature,threshold,left and rightvalue

2.Decision Tree class structured by defining minimum no of splits that node can_go=2,max depth of tree=100,no of features to consider .

3.fit(X,y) This method adjusts the parameters of the model based on the provided data.
X: The feature matrix, where each row represents a sample and each column represents a feature.
y: The target vector, containing the labels or target values corresponding to the samples in X.

4._grow_tree(X, y, depth=0)**:**This method builds the decision tree step by step. It stops growing when: Maximum depth is reached.The number of samples is too small All labels in the node are the same.If it cannot split further, it creates a **leaf node** with the most common label.

5._best_split(X, y, feat_idxs):
Finds the best feature and the best value (threshold) for splitting the data.
Checks every unique value in the chosen features to calculate the **information gain**.
Chooses the split with the highest information gain.

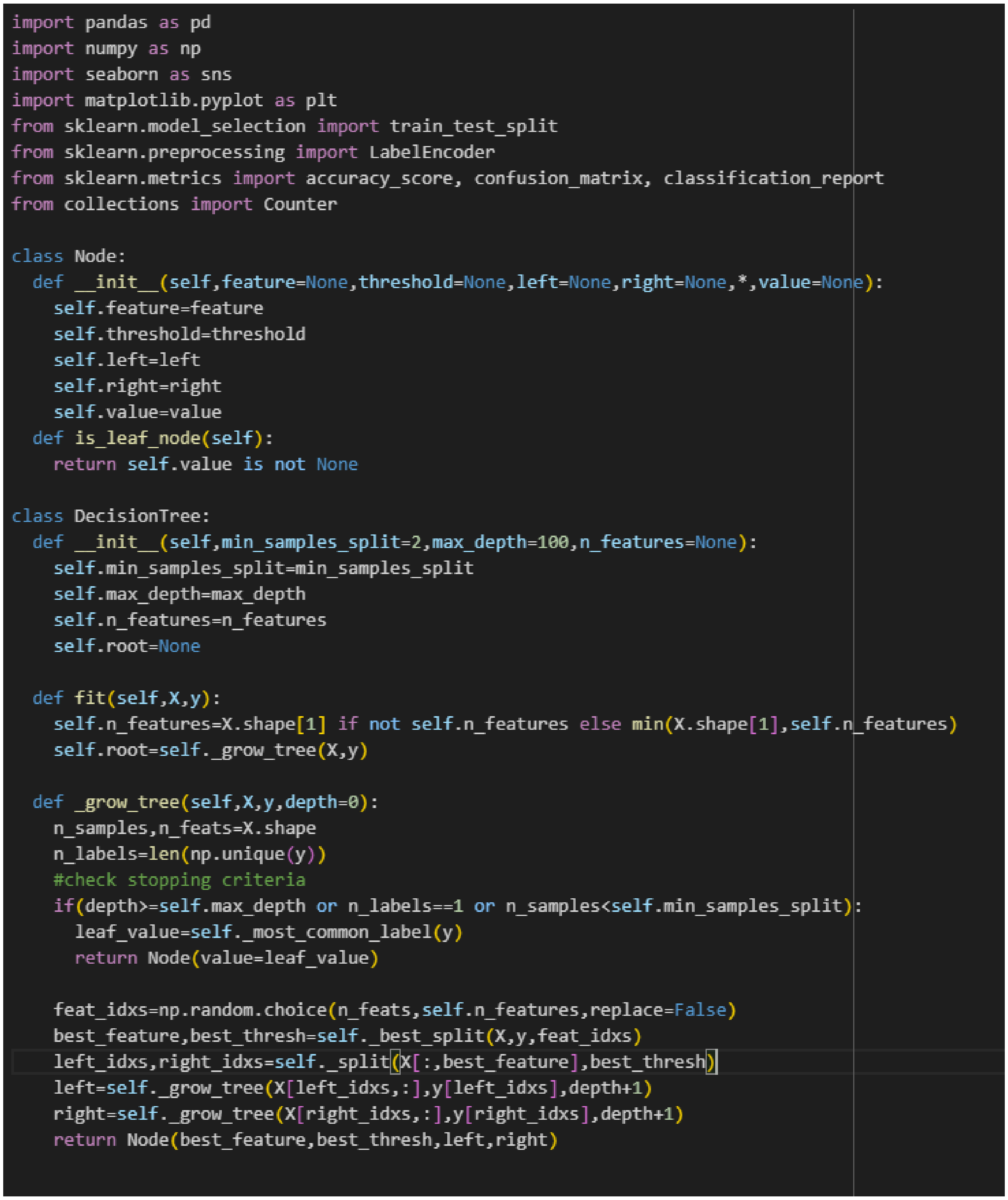
6**.**_information_gain(y, X_column, thr):
Measures how much splitting improves purity:
Parent entropy: Entropy of the current node before splitting.
Child entropy: Weighted entropy after the split (for left and right child nodes).
Information gain = Parent entropy - Child entropy.

7._split(X_column, split_thresh):
Divides the dataset into two groups:
Left group: Samples with values ≤ threshold.
Right group: Samples with values > threshold.

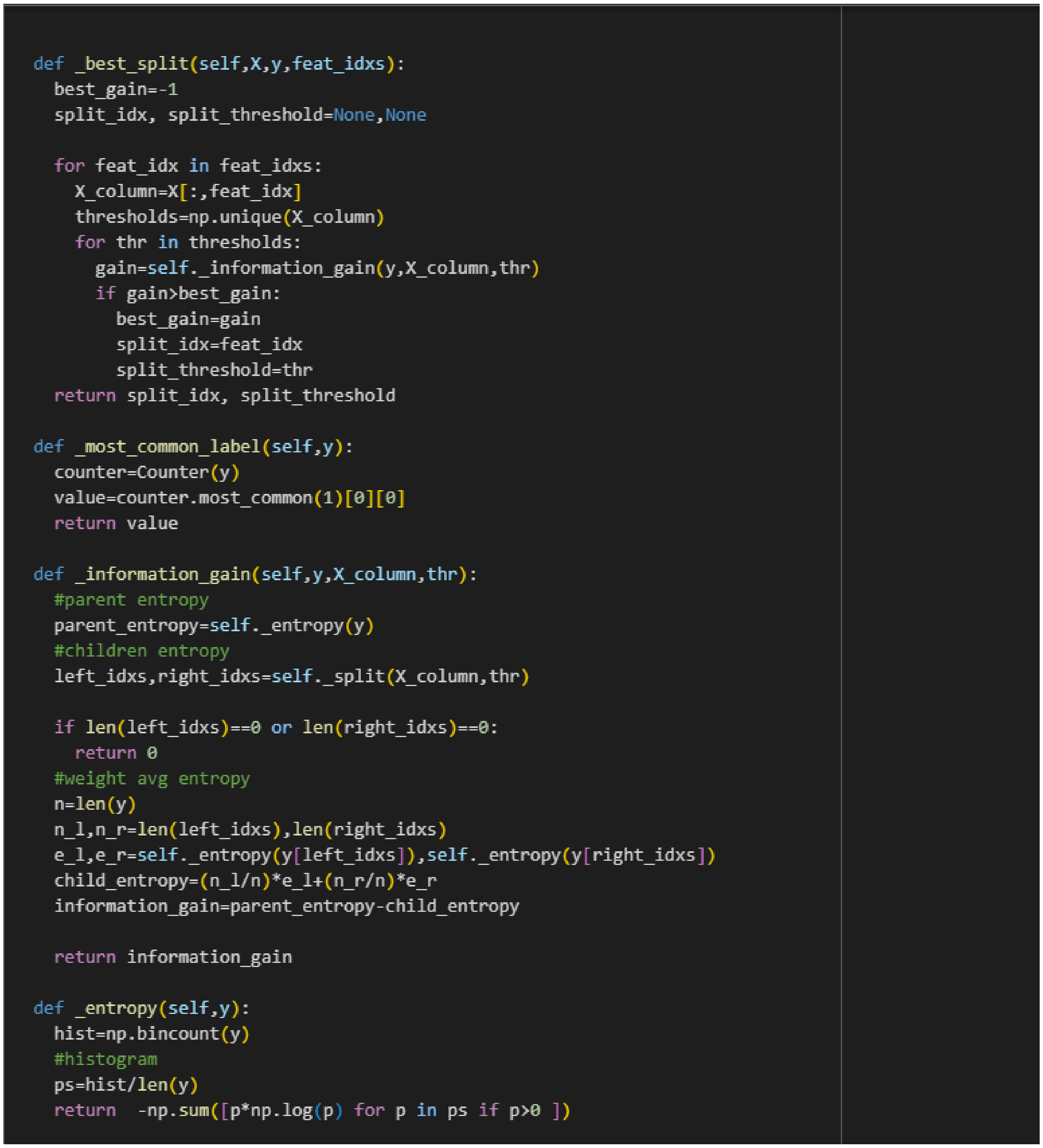
8._entropy(y):
Calculates how "impure" a group of labels.Entropy=−∑pi · log(pi)

9.predict(X):Predicts the label for each sample in `X` by going through the tree.

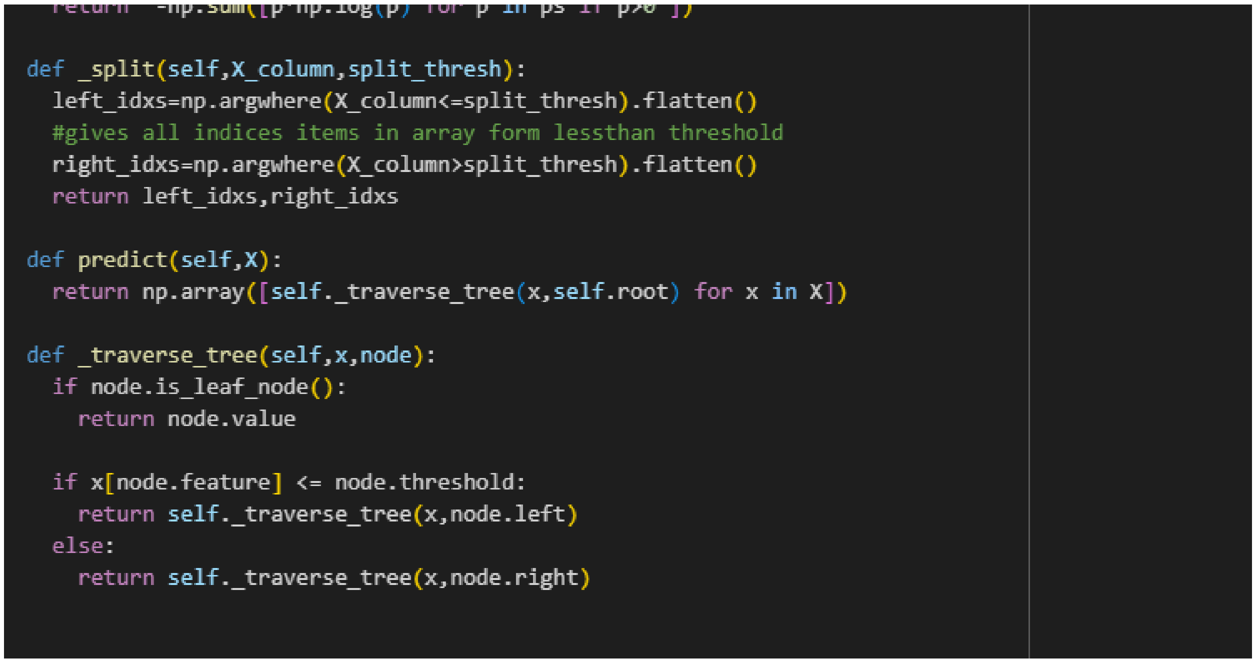
10._traverse_tree(x, node):
Recursively checks if a sample goes to the left or right child node until it reaches a **leaf node**.
Returns the label stored in the leaf.



```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from collections import Counter

class Node:
  def __init__(self,feature=None,threshold=None,left=None,right=None,*,value=None):
    self.feature=feature
    self.threshold=threshold
    self.left=left
    self.right=right
    self.value=value
  def is_leaf_node(self):
    return self.value is not None

class DecisionTree:
  def __init__(self,min_samples_split=2,max_depth=100,n_features=None):
    self.min_samples_split=min_samples_split
    self.max_depth=max_depth
    self.n_features=n_features
    self.root=None

  def fit(self,X,y):
    self.n_features=X.shape[1] if not self.n_features else min(X.shape[1],self.n_features)
    self.root=self._grow_tree(X,y)

  def _grow_tree(self,X,y,depth=0):
    n_samples,n_feats=X.shape
    n_labels=len(np.unique(y))
    #check stopping criteria
    if(depth>=self.max_depth or n_labels==1 or n_samples<self.min_samples_split):
      leaf_value=self._most_common_label(y)
      return Node(value=leaf_value)

    feat_idxs=np.random.choice(n_feats,self.n_features,replace=False)
    best_feature,best_thresh=self._best_split(X,y,feat_idxs)
    left_idxs,right_idxs=self._split(X[:,best_feature],best_thresh)
    left=self._grow_tree(X[left_idxs,:],y[left_idxs],depth+1)
    right=self._grow_tree(X[right_idxs,:],y[right_idxs],depth+1)
    return Node(best_feature,best_thresh,left,right)
```

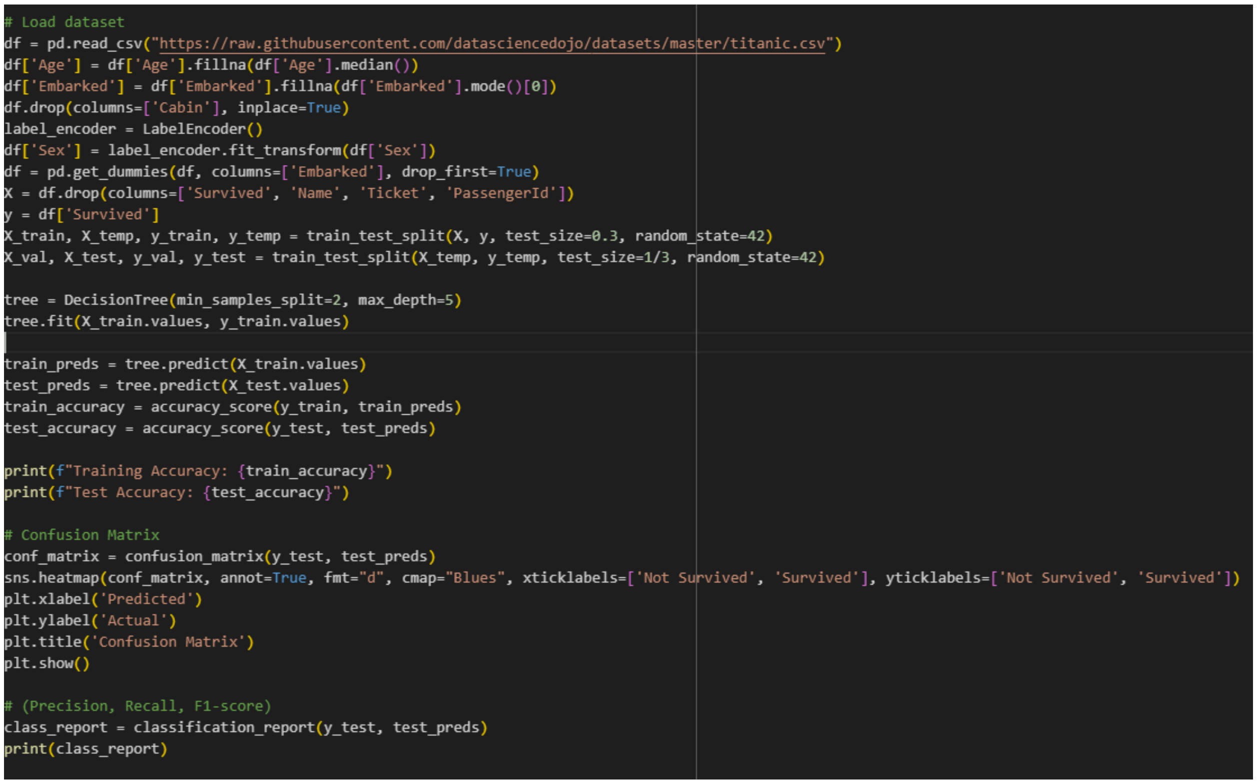


```
def _best_split(self,X,y,feat_idxs):
  best_gain=-1
  split_idx, split_threshold=None,None

  for feat_idx in feat_idxs:
    X_column=X[:,feat_idx]
    thresholds=np.unique(X_column)
    for thr in thresholds:
      gain=self._information_gain(y,X_column,thr)
      if gain>best_gain:
        best_gain=gain
        split_idx=feat_idx
        split_threshold=thr
  return split_idx, split_threshold

def _most_common_label(self,y):
  counter=Counter(y)
  value=counter.most_common(1)[0][0]
  return value

def _information_gain(self,y,X_column,thr):
  #parent entropy
  parent_entropy=self._entropy(y)
  #children entropy
  left_idxs,right_idxs=self._split(X_column,thr)

  if len(left_idxs)==0 or len(right_idxs)==0:
    return 0
  #weight avg entropy
  n=len(y)
  n_l,n_r=len(left_idxs),len(right_idxs)
  e_l,e_r=self._entropy(y[left_idxs]),self._entropy(y[right_idxs])
  child_entropy=(n_l/n)*e_l+(n_r/n)*e_r
  information_gain=parent_entropy-child_entropy

  return information_gain

def _entropy(self,y):
  hist=np.bincount(y)
  #histogram
  ps=hist/len(y)
  return  -np.sum([p*np.log(p) for p in ps if p>0 ])
```



```python
    return -np.sum([p*np.log(p) for p in ps if p>0])

  def _split(self,X_column,split_thresh):
    left_idxs=np.argwhere(X_column<=split_thresh).flatten()
    #gives all indices items in array form lessthan threshold
    right_idxs=np.argwhere(X_column>split_thresh).flatten()
    return left_idxs,right_idxs

  def predict(self,X):
    return np.array([self._traverse_tree(x,self.root) for x in X])

  def _traverse_tree(self,x,node):
    if node.is_leaf_node():
      return node.value

    if x[node.feature] <= node.threshold:
      return self._traverse_tree(x,node.left)
    else:
      return self._traverse_tree(x,node.right)
```

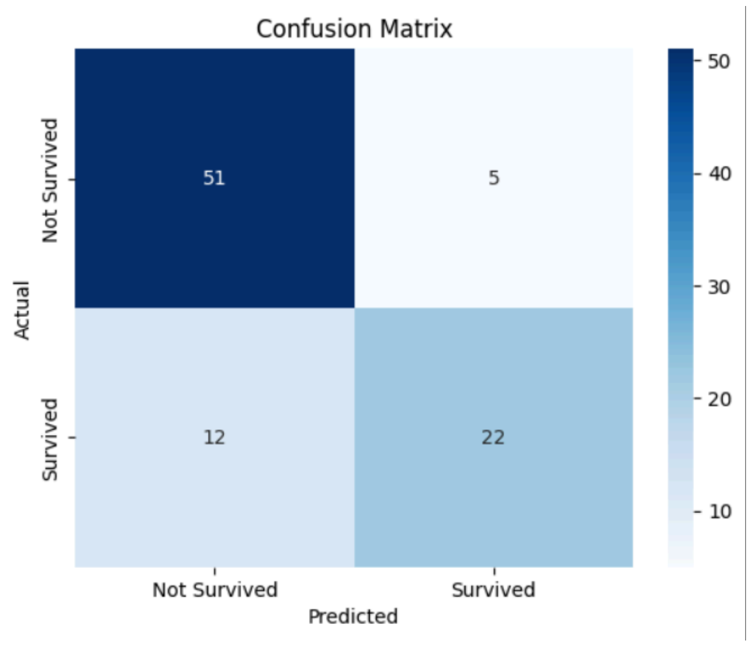
Dataset Preprocessing and Testing



```python
# Load dataset
df = pd.read_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])
df.drop(columns=['Cabin'], inplace=True)
label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])
df = pd.get_dummies(df, columns=['Embarked'], drop_first=True)
X = df.drop(columns=['Survived', 'Name', 'Ticket', 'PassengerId'])
y = df['Survived']
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=1/3, random_state=42)

tree = DecisionTree(min_samples_split=2, max_depth=5)
tree.fit(X_train.values, y_train.values)

train_preds = tree.predict(X_train.values)
test_preds = tree.predict(X_test.values)
train_accuracy = accuracy_score(y_train, train_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print(f"Training Accuracy: {train_accuracy}")
print(f"Test Accuracy: {test_accuracy}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, test_preds)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Not Survived', 'Survived'], yticklabels=['Not Survived', 'Survived'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# (Precision, Recall, F1-score)
class_report = classification_report(y_test, test_preds)
print(class_report)
```

Results:

Training Accuracy: 0.8475120385232745~84%
Test Accuracy: 0.8111111111111111-81%



	precision	recall	f1-score	support
0	0.81	0.91	0.86	56
1	0.81	0.65	0.72	34

0-Not survived
1-survived
This means that the model correctly predicted about 85% of the training data.
Test Accuracy: 0.8111 (approximately 81%)
This means that the model correctly predicted about 81% of the test data.
code

Code:https://colab.research.google.com/drive/1fmx4VQLW_b9Wvw9w8he7FNu6863hNou-?usp=sharing

References:

Pattern Classification
[DHS] Duda, Hart and Stroke:
https://github.com/rohinarora/EECE5644-Machine_Learning/blob/master/Richard%20O.%20Duda%2C%20Peter%20E.%20Hart%2C%20David%20G.%20Stork%20-%20Pattern%20classification%20(2001%2C%20Wiley).pdf