Name : Sagar Patil
Roll No : 8061
BE Computer

Assignment No.2

Aim: To implement Decision Tree Classifier.

Objective:

- The Basic Concepts of Decision Tree Classifier.
- Implementation logic of Decision Tree Classifier.

Theory:

Introduction:

A decision tree is a flowchart-like structure in which internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

Decision trees are diagrams that attempt to display the range of possible outcomes and subsequent decisions made after an initial decision. For example, your original decision might be whether to attend college, and the tree might attempt to show how much time would be spent doing different activities and your earning power based on your decision. There are several notable pros and cons to using decision trees.

In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated. A decision tree consists of 3 types of nodes:

1. Decision nodes - commonly represented by squares
2. Chance nodes - represented by circles
3. End nodes - represented by triangles

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal. If in practice decisions have to be taken online with no recall under incomplete knowledge, a decision tree should be paralleled

by a probability model as a best choice model or online selection model algorithm. Another use of decision trees is as a descriptive means for calculating conditional probabilities.

Decision trees, influence diagrams, utility functions, and other decision analysis tools and methods are taught to undergraduate students in schools of business, health economics, and public health, and are examples of operations research or management science methods.

ALGORITHM USED:

In decision tree learning, ID3 (Iterative Dichotomiser 3) is an algorithm invented by Ross Quinlan used to generate a decision tree from a dataset.

The ID3 algorithm begins with the original set as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set and calculates the entropy (or information gain) of that attribute. It then selects the attribute which has the smallest entropy (or largest information gain) value. The set is then split by the selected attribute (e.g. age < 50, 50 <= age < 100, age >= 100) to produce subsets of the data. The algorithm continues to Recurse on each subset, considering only attributes never selected before. Recursion on a subset may stop in one of these cases:

- Every element in the subset belongs to the same class (+ or -), then the node is turned into a leaf and labelled with the class of the examples
- There are no more attributes to be selected, but the examples still do not belong to the same class (some are + and some are -), then the node is turned into a leaf and labelled with the most common class of the examples in the subset
- There are no examples in the subset, this happens when no example in the parent set was found to be matching a specific value of the selected attribute, for example if there was no example with age >= 100. Then a leaf is created, and labelled with the most common class of the examples in the parent set.

Throughout the algorithm, the decision tree is constructed with each non-terminal node representing the selected attribute on which the data was split, and terminal nodes representing the class label of the final subset of this branch.

Summary:

1. Calculate the entropy of every attribute using the data set

2. Split the set into subsets using the attribute for which entropy is minimum (or, equivalently, information gain is maximum) 3. Make a decision tree node containing that attribute.

4. Recurse on subsets using remaining attributes

Decision trees offer advantages over other methods of analyzing alternatives. They are:

Graphic

You can represent decision alternatives, possible outcomes, and chance events schematically. The visual approach is particularly helpful in comprehending sequential decisions and outcome dependencies.

Efficient

You can quickly express complex alternatives clearly. You can easily modify a decision tree as new information becomes available. Set up a decision tree to compare how changing input values affect various decision alternatives. Standard decision tree notation is easy to adopt.

Revealing

You can compare competing alternatives-even without complete information-in terms of risk and probable value. The Expected Value (EV) term combines relative investment costs, anticipated payoffs, and uncertainties into a single numerical value. The EV reveals the overall merits of competing alternatives.

Complementary

You can use decision trees in conjunction with other project management tools. For example, the decision tree method can help evaluate project schedules.

- Decision trees are self-explanatory and when compacted they are also easy to follow.

In other words if the decision trees has a reasonable number of leaves, it can be grasped by non-professional users. Furthermore decision trees can be converted to a set of rules. Thus, this representation is considered as comprehensible. • Decision trees can handle both nominal and numerical attributes.

- Decision trees representation is rich enough to represent any discrete-value classifier.
- Decision trees are capable of handling datasets that may have errors.
- Decision trees are capable of handling datasets that may have missing values.
- Decision trees are considered to be a nonparametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

On the other hand, decision trees have disadvantages such as:

- Most of the algorithms (like ID3 and C4.5) require that the target attribute will have only discrete values.
- As decision trees use the ―divide and conquer‖ method, they tend to perform well if a few highly relevant attributes exist, but less so if many complex interactions are present. One of the reasons for this is that other classifiers can compactly describe a classifier that would be very challenging to represent using a decision tree.
- The greedy characteristic of decision trees leads to another disadvantage that should be pointed out. This is its over-sensitivity to the training set, to irrelevant attributes and to noise.

Conclusion: We have studied the Decision tree classifier and also implemented successfully.

CODE:

```
import pandas as pd import numpy as np import tree
from sklearn.tree import DecisionTreeClassifier as DTC from
sklearn.preprocessing import LabelEncoder
```

In [2]:

```
class DecisionTreeClassifier:     def_init_(self, max_depth=None):
        self.max_depth = max_depth

    def fit(self, X, y):
        self.n_classes_ = len(set(y))
self.n_features_ = X.shape[1]         self.tree_
= self._grow_tree(X, y)
def predict(self, X):
        return [self._predict(inputs) for inputs in X]
def debug(self, feature_names, class_names,
show_details=True):         self.tree_.debug(feature_names, class_names,
show_details)
     def _gini(self, y):            m = y.size
return 1.0 - sum((np.sum(y == c)
/ m) ** 2 for c in range(self.n_classes_))
def _best_split(self, X, y):
        # Need at least two elements to split a node.
m = y.size
if m <= 1:
            return None, None

        # Count of each class in the current node.         num_parent =
[np.sum(y == c) for c in range(self.n_classes_)]

        # Gini of current node.         best_gini = 1.0 -
sum((n / m) ** 2 for n in num_parent)         best_idx,
best_thr = None, None

       # Loop through all features.         for idx in
range(self.n_features_):             # Sort data along
selected feature.             thresholds, classes =
zip(*sorted(zip(X[:, idx], y)))                  num_left =
[0] * self.n_classes_
num_right = num_parent.copy()             for i in range(1,
m):  # possible split positions                 c = classes[i -
1]                num_left[c] += 1
num_right[c] -= 1                gini_left = 1.0 - sum(
                    (num_left[x] / i) ** 2 for x in range(self.n_classes_)
                )
gini_right = 1.0 - sum(
                    (num_right[x] / (m - i)) ** 2 for x in
range(self.n_classes_)
                )

                gini = (i * gini_left + (m - i) * gini_right) / m
if thresholds[i] == thresholds[i - 1]:
                    continue
                if gini <
best_gini:
```

```
best_gini = gini                    best_idx
= idx                   best_thr =
(thresholds[i] + thresholds[i - 1]) / 2 #
midpoint
        return best_idx, best_thr
def _grow_tree(self, X, y,
depth=0):
        """Build a decision tree by recursively finding the best split."""
# Population for each class in current node. The predicted class is the one
with
        # largest population.
 num_samples_per_class = [np.sum(y == i) for i in range(self.n_classes_)]
 predicted_class = np.argmax(num_samples_per_class)         node =
 tree.Node(             gini=self._gini(y),
 num_samples=y.size,
 num_samples_per_class=num_samples_per_class,
 predicted_class=predicted_class,
         )

         # Split recursively until maximum depth is reached.          if
 depth < self.max_depth:
             idx, thr = self._best_split(X, y)
 if idx is not None:
                 indices_left = X[:, idx] < thr
                 X_left, y_left = X[indices_left], y[indices_left]
 X_right, y_right = X[~indices_left], y[~indices_left]
 node.feature_index = idx                  node.threshold = thr
 node.left = self._grow_tree(X_left, y_left, depth + 1)
 node.right = self._grow_tree(X_right, y_right, depth + 1)          return
 node
     def _predict(self,
 inputs):
         """Predict class for a single sample."""
 node = self.tree_          while node.left:             if
 inputs[node.feature_index] < node.threshold:
 node = node.left
 else:
                 node = node.right          return
 node.predicted_class
```

In [3]:

```
df = pd.read_csv("ml2.csv") print(df) # for col in df.columns:
#     df[col] = LabelEncoder().fit_transform(df[col])

df = df.apply(LabelEncoder().fit_transform)
```

In [5]:

```
print(df)
X = df.iloc[ : ,  : -1] y
= df['Buys']
```

In [7]:

```
X = np.asarray(X) print(X)  print(y)
clf = DecisionTreeClassifier(max_depth = 2) clf.fit(X, y)
```

In [10]:

```
clf.predict([[2, 0, 1, 0]])
# sklearn classifier
c =
DTC() c = c.fit(X, y)
c.predict([[2, 0, 1, 0]])
```

OUTPUT:

	AGE	INCOME	Gender	Maritial	Buys
0	<21	High	Male	Single	No
1	<21	High	Male	Married	No
2	21-35	High	Male	Single	Yes
3	>35	Medium	Male	Single	Yes
4	>35	Low	Female	Single	Yes
5	>35	Low	Female	Married	No
6	21-35	Low	Female	Married	Yes
7	<21	Medium	Male	Single	No
8	<21	Low	Female	Married	Yes
9	>35	Medium	Female	Single	Yes
10	<21	Medium	Female	Married	Yes
11	21-35	Medium	Male	Married	Yes
12	21-35	High	Female	Single	Yes
13	>35	Medium	Male	Married	No

```
[[1 0 1 1]
 [1 0 1 0]
 [0 0 1 1]
 [2 2 1 1]
 [2 1 0 1]
 [2 1 0 0]
 [0 1 0 0]
 [1 2 1 1]
 [1 1 0 0]
 [2 2 0 1]
 [1 2 0 0]
 [0 2 1 0]
 [0 0 0 1]
 [2 2 1 0]]
```

	AGE	INCOME	Gender	Maritial	Buys
0	1	0	1	1	0
1	1	0	1	0	0
2	0	0	1	1	1
3	2	2	1	1	1
4	2	1	0	1	1
5	2	1	0	0	0
6	0	1	0	0	1
7	1	2	1	1	0
8	1	1	0	0	1
9	2	2	0	1	1
10	1	2	0	0	1
11	0	2	1	0	1
12	0	0	0	1	1
13	2	2	1	0	0

```
0     0
1     0
2     1
3     1
4     1
5     0
6     1
7     0
8     1
9     1
10    1
11    1
12    1
13    0
Name: Buys, dtype: int64
```