Experiment Number : 8 - Data Classification using Decision Tree Algorithm (ID3)

Batch: FDS-2 **Roll Number: 1601042223** **Name: Chandana Ramesh Galgali**

Aim of the Experiment: Exploration of data classification using Decision Tree algorithm (ID3) on a Sample Dataset

---

Program/ Steps:

1. Computes the ID3 algorithm to select an attribute subset that best predicts class labels
2. Use Decision Tree Classifiers to classify the Sample Data(e.g.IRIS Sample Data).
3. Manual write should take his own data so that it could be possible to find best class labels as well as do the classification.

Code with Output/Result:

Problem 2: Attribute subset selection using the ID3 algorithm. (Python Code)

```
import numpy as np
import pandas as pd
# Sample dataset
data = pd.DataFrame({
'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy',
'Sunny', 'Overcast', 'Overcast', 'Rainy'],
'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot',
'Mild'],
'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal',
'Normal', 'High', 'Normal', 'High'],
'Class': ['No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
})
# Function to calculate entropy
def entropy(class_labels):
 unique_labels, counts = np.unique(class_labels, return_counts=True)
 prob = counts / len(class_labels)
 entropy = -np.sum(prob * np.log2(prob))
 return entropy
# Function to calculate information gain
def information_gain(data, attribute, class_label):
 total_entropy = entropy(data[class_label])
 unique_values = data[attribute].unique()
 weighted_entropy = 0
 for value in unique_values:
  subset = data[data[attribute] == value]
  subset_entropy = entropy(subset[class_label])
  weight = len(subset) / len(data)
```

```
    weighted_entropy += weight * subset_entropy
  return total_entropy - weighted_entropy
# ID3 algorithm for attribute selection
def id3(data, class_label, attributes):
  if len(attributes) == 0:
    # If no attributes are left, return the majority class
    return data[class_label].mode().iloc[0]
  unique_classes = data[class_label].unique()
  if len(unique_classes) == 1:
    # If all examples have the same class, return that class
    return unique_classes[0]
  best_attribute = max(attributes, key=lambda attr: information_gain(data, attr, class_label))
  tree = {best_attribute: {}}
  for value in data[best_attribute].unique():
    subset = data[data[best_attribute] == value]
    if len(subset) == 0:
      # If the subset is empty, return the majority class
      tree[best_attribute][value] = data[class_label].mode().iloc[0]
    else:
      new_attributes = [attr for attr in attributes if attr != best_attribute]
      tree[best_attribute][value] = id3(subset, class_label, new_attributes)
  return tree
# Define the class label and attributes
class_label = 'Class'
attributes = ['Outlook', 'Temperature', 'Humidity']
# Build the ID3 decision tree
decision_tree = id3(data, class_label, attributes)
# Print the decision tree
import pprint
```

```
pprint.pprint(decision_tree)
```

```
{'Humidity': {'High': {'Outlook': {'Overcast': 'Yes',
                                   'Rainy': {'Temperature': {'Mild': 'No'}},
                                   'Sunny': 'No'}},
              'Normal': 'Yes'}}
```

Problem 3: Classification of Data (IRIS SAMPLE DATA) with Decision Tree Algorithm

```
from sklearn import datasets
import pandas as pd
import numpy as np
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
iris = datasets.load_iris() #Loading the dataset
iris.keys()
print("Iris Keys:", iris.keys)
dict_keys=(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename',
'data_module'])
print("Dict Keys:", dict_keys)
#convert dataset to pandas df
iris = pd.DataFrame(
data= np.c_[iris['data'], iris['target']],
columns= iris['feature_names'] + ['target']
)
print(iris)
print(iris.head())
# To give name to target class as species, Species is a list
```

```
species = []
for i in range(len(iris['target'])):
  if iris['target'][i] == 0:
    species.append("setosa")
  elif iris['target'][i] == 1:
    species.append('versicolor')
  else:
    species.append('virginica')
iris['species'] = species
print("Species have class now", iris['species'])
#To check new data set is taken place or not then used once again tail/head command
print(iris.tail())
# Do observe total sample of each species
print( iris.groupby('species').size())
# Plotting a dataset is a great way to explore its distribution.
# Plotting the iris dataset can be done using matplotlib, a Python library for 2D plotting.
import matplotlib
setosa = iris[iris.species == "setosa"]
versicolor = iris[iris.species=='versicolor']
virginica = iris[iris.species=='virginica']
fig, ax = plt.subplots()
fig.set_size_inches(13, 7) # adjusting the length and width of plot, lables and scatter points
ax.scatter(setosa['petal length (cm)'], setosa['petal width (cm)'],
label="Setosa", facecolor="blue")
ax.scatter(versicolor['petal length (cm)'], versicolor['petal width (cm)'],
label="Versicolor", facecolor="green")
ax.scatter(virginica['petal length (cm)'], virginica['petal width (cm)'],
label="Virginica", facecolor="red")
ax.set_xlabel("petal length (cm)")
```

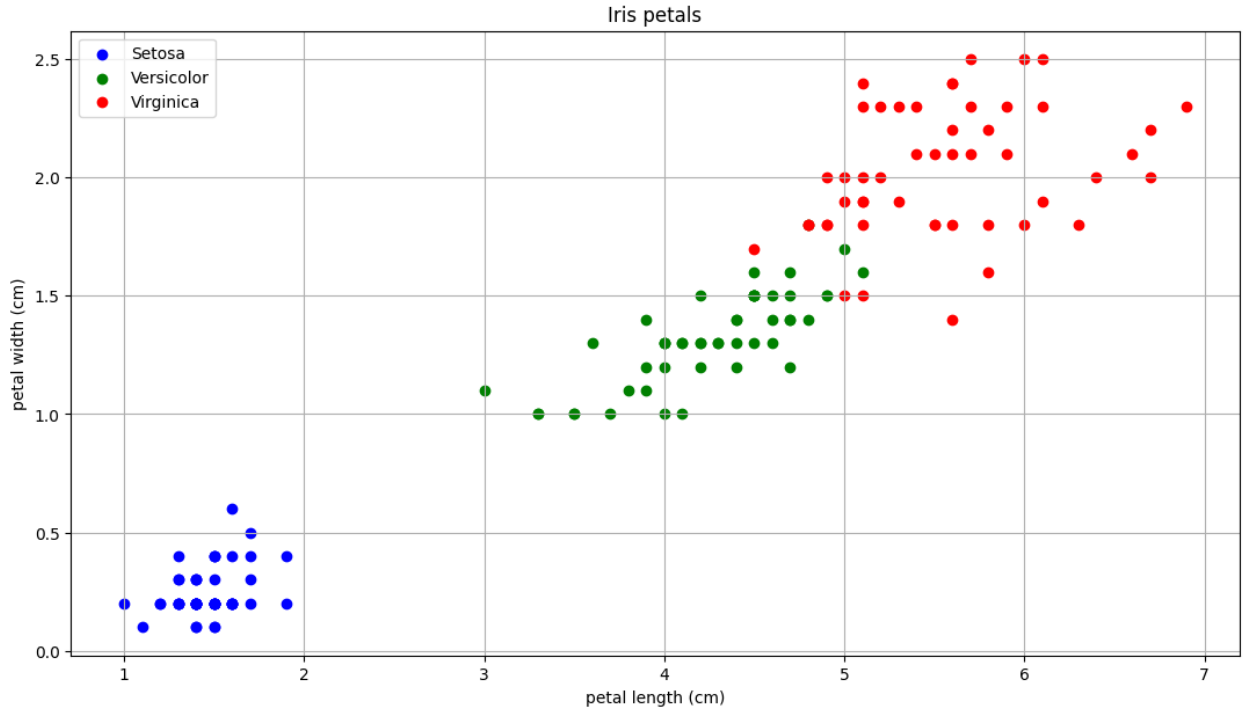
```
ax.set_ylabel("petal width (cm)")
ax.grid()
ax.set_title("Iris petals")
ax.legend()
# To display image here (optional)
plt.show()
# Split the data into features (X) and target labels (y)
X = iris.drop('species', axis=1)
y = iris['species']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Create and Train the Decision Tree Classifier:
# Create an instance of DecisionTreeClassifier and train it on the training data using the fit()
method.
# Create a DecisionTreeClassifier instance
classifier = DecisionTreeClassifier(random_state=42)
DecisionTreeClassifier
# Train the classifier on the training data
classifier.fit(X_train, y_train)
# Make Predictions:
# Use the trained classifier to make predictions on new, unseen data (testing set) using the
predict() method.
y_pred = classifier.predict(X_test)
x_pred=classifier.predict(X_train)
'''Evaluate the Model:
Calculate evaluation metrics to assess the performance of the model. For classification tasks,
you can use metrics like accuracy, precision, recall, F1-score, and confusion matrix.
'''
```

```
''' Compute Training time Accuracy'''
# Calculate accuracy
accuracy = accuracy_score(y_train, x_pred)
print("Accuracy:", accuracy)
# Display classification report
print("Classification Report:")
print(classification_report(y_train, x_pred))
# Display confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_train, x_pred))
''' Compute Testing time Accuracy'''
print("Testing Time Accuracy")
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Testing:", accuracy)
# Display classification report
print("Classification Report Testing:")
print(classification_report(y_test, y_pred))
# Display confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Iris Keys: <built-in method keys of Bunch object at 0x7ec2215a53a0>
Dict Keys: ['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module']
Iris:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                    5.1               3.5                1.4               0.2
1                    4.9               3.0                1.4               0.2
2                    4.7               3.2                1.3               0.2
3                    4.6               3.1                1.5               0.2
4                    5.0               3.6                1.4               0.2
..                   ...               ...                ...               ...
145                  6.7               3.0                5.2               2.3
146                  6.3               2.5                5.0               1.9
147                  6.5               3.0                5.2               2.0
148                  6.2               3.4                5.4               2.3
149                  5.9               3.0                5.1               1.8

     target
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
..      ...
145     2.0
146     2.0
147     2.0
148     2.0
149     2.0
```

```
[150 rows x 5 columns]
Iris head:    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2

   target
0     0.0
1     0.0
2     0.0
3     0.0
4     0.0
Species have class now 0         setosa
1         setosa
2         setosa
3         setosa
4         setosa
         ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: species, Length: 150, dtype: object
```

```
Iris tail:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
145                 6.7               3.0                5.2               2.3
146                 6.3               2.5                5.0               1.9
147                 6.5               3.0                5.2               2.0
148                 6.2               3.4                5.4               2.3
149                 5.9               3.0                5.1               1.8

     target    species
145     2.0  virginica
146     2.0  virginica
147     2.0  virginica
148     2.0  virginica
149     2.0  virginica
Total samples of each species: species
setosa        50
versicolor    50
virginica     50
dtype: int64
```



```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        40
  versicolor       1.00      1.00      1.00        41
   virginica       1.00      1.00      1.00        39

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120

Confusion Matrix:
[[40  0  0]
 [ 0 41  0]
 [ 0  0 39]]
Testing Time Accuracy
Accuracy Testing: 1.0
Classification Report Testing:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

Code:

```
import numpy as np
import pandas as pd
dataframe = pd.read_csv(r'/content/Flight_delay.csv')
data_array = dataframe.to_numpy()
data = pd.DataFrame({
'AirTime': data_array[:100, 11],
'TaxiIn': data_array[:100, 19],
'TaxiOut': data_array[:100, 20],
'CarrierDelay': data_array[:100, 24]
})
# Function to calculate entropy
def entropy(class_labels):
 unique_labels, counts = np.unique(class_labels, return_counts=True)
 prob = counts / len(class_labels)
 entropy = -np.sum(prob * np.log2(prob))
 return entropy
# Function to calculate information gain
def information_gain(data, attribute, class_label):
 total_entropy = entropy(data[class_label])
 unique_values = data[attribute].unique()
 weighted_entropy = 0
 for value in unique_values:
  subset = data[data[attribute] == value]
  subset_entropy = entropy(subset[class_label])
  weight = len(subset) / len(data)
  weighted_entropy += weight * subset_entropy
 return total_entropy - weighted_entropy
# ID3 algorithm for attribute selection
```

```
def id3(data, class_label, attributes):
 if len(attributes) == 0:
  # If no attributes are left, return the majority class
  return data[class_label].mode().iloc[0]
 unique_classes = data[class_label].unique()
 if len(unique_classes) == 1:
  # If all examples have the same class, return that class
  return unique_classes[0]
 best_attribute = max(attributes, key=lambda attr: information_gain(data, attr, class_label))
 tree = {best_attribute: {}}
 for value in data[best_attribute].unique():
  subset = data[data[best_attribute] == value]
  if len(subset) == 0:
   # If the subset is empty, return the majority class
   tree[best_attribute][value] = data[class_label].mode().iloc[0]
  else:
   new_attributes = [attr for attr in attributes if attr != best_attribute]
   tree[best_attribute][value] = id3(subset, class_label, new_attributes)
 return tree
# Define the class label and attributes
class_label = 'CarrierDelay'
attributes = ['AirTime', 'TaxiIn', 'TaxiOut']
# Build the ID3 decision tree
decision_tree = id3(data, class_label, attributes)
# Print the decision tree
import pprint
pprint.pprint(decision_tree)
```

Result:

```
{'AirTime': {32: 0,
```

34: 3,
36: {'TaxiIn': {3: 27, 4: 0, 5: 0}},
37: 4,
41: {'TaxiIn': {9: 61, 16: 18}},
42: {'TaxiIn': {6: 0, 7: 16}},
43: {'TaxiIn': {3: 114, 5: 2, 7: 0}},
44: 13,
45: 15,
46: {'TaxiOut': {10: 20, 15: 9, 17: 17, 18: 1}},
47: {'TaxiOut': {5: 7, 16: 10, 24: 9}},
48: {'TaxiIn': {3: 4, 4: 6}},
49: {'TaxiIn': {2: 12, 5: 2, 7: 0}},
59: {'TaxiOut': {7: 282, 9: 0}},
60: {'TaxiOut': {7: 26, 10: 7, 11: 15}},
65: 50,
72: {'TaxiIn': {5: {'TaxiOut': {9: 0}}}},
73: 9,
76: 2,
77: {'TaxiOut': {8: 45, 10: 2, 13: 0, 16: 26}},
78: {'TaxiIn': {2: 10, 4: {'TaxiOut': {10: 11, 18: 14}}, 6: 7}},
80: 2,
81: {'TaxiIn': {3: 15, 8: 7}},
88: 0,
90: 25,
91: 3,
95: 27,
97: 1,
106: 24,
107: 8,
110: 0,
112: 12,
113: 3,
116: 50,
118: 3,
121: 0,
125: {'TaxiIn': {2: 32, 6: 11}},
127: 23,
130: {'TaxiIn': {4: 59, 5: 4}},

```
131: 14,
132: 38,
134: 5,
139: 1,
143: {'TaxiIn': {3: 12, 5: 0, 6: 40}},
150: 17,
166: 3,
168: 13,
171: {'TaxiIn': {7: 60, 8: 4}},
175: 0,
176: 0,
177: {'TaxiIn': {5: 0, 6: 4, 62: 6}},
178: 2,
179: 32,
183: 18,
195: 10,
201: 10,
213: 3,
221: 48,
230: {'TaxiIn': {3: 10, 5: 7}},
232: 6,
237: 13,
243: 8,
244: 18,
245: 18,
253: 19,
269: 13}}
```

Code:

```
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
# Load the dataset from CSV file
dataset = pd.read_csv('/content/Flight_delay.csv')
# Drop non-numeric columns
dataset = dataset.select_dtypes(include='number')
```

```
# Split the dataset into features (X) and target variable (y)
X = dataset.drop('ActualElapsedTime', axis=1)
y = dataset['ActualElapsedTime']
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create an instance of the DecisionTreeRegressor
regressor = DecisionTreeRegressor()
# Train the regressor on the training data
regressor.fit(X_train, y_train)
# Make predictions on the testing data
y_pred = regressor.predict(X_test)
# Calculate the mean squared error of the regressor
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

Result:

```
Mean Squared Error: 3.9583535408777126
```

Post Lab Question-Answers:

1. What are the data filtering techniques available? Explain in brief.

Ans: There are several data filtering techniques available for processing and analyzing data. Here are a few commonly used techniques:

1. Rule-based Filtering: This technique involves applying predefined rules to filter data based on specific conditions or criteria. For example, you can filter data based on a specific range of values or exclude certain categories.

2. Time-based Filtering: This technique involves filtering data based on time-related criteria. You can filter data by specific time periods, such as days, weeks, or months, or by specific dates or time ranges.

3. Frequency-based Filtering: This technique involves filtering data based on the frequency of occurrence. You can filter data to include only the most frequent or least frequent values, or you can set a threshold to include values above or below a certain frequency.

4. Pattern-based Filtering: This technique involves filtering data based on specific patterns or sequences. You can use pattern matching algorithms or regular expressions to identify and filter data that matches a particular pattern.

5. Statistical Filtering: This technique involves filtering data based on statistical measures. You can filter data based on measures such as mean, median, standard deviation, or percentiles to include or exclude values that fall within certain statistical ranges.

6. Text-based Filtering: This technique involves filtering textual data based on specific keywords, phrases, or patterns. You can use techniques like keyword matching, text mining, or natural language processing to filter and extract relevant information from text data.

These are just a few examples of data filtering techniques. The choice of technique depends on the specific requirements and characteristics of the data being analyzed.

2. What do you mean by sampling a data set? How is sampling done in data science?

Ans: Sampling a dataset refers to the process of selecting a subset of observations or data points from a larger population or dataset. In data science, sampling is commonly used to make inferences or draw conclusions about the entire population based on the analysis of the smaller sample.

Sampling in data science can be done using various techniques, including:

1. Simple Random Sampling: This technique involves randomly selecting observations from the population, where each observation has an equal chance of being selected. It is often used when the population is homogeneous and there are no specific criteria for selection.

2. Stratified Sampling: This technique involves dividing the population into distinct subgroups or strata based on certain characteristics, and then selecting samples from each stratum. It ensures that the sample represents the diversity of the population and can be useful when there are significant differences within the population.

3. Cluster Sampling: This technique involves dividing the population into clusters or groups, and then randomly selecting entire clusters to include in the sample. It is useful when it is difficult or costly to sample individual observations, but clusters can be easily sampled.

4. Systematic Sampling: This technique involves selecting every nth observation from the population after randomly selecting a starting point. It provides a simple and efficient way to sample large populations.

5. Sampling with Replacement: This technique allows selected observations to be included in the sample multiple times. It is useful when the population is small or when the same observation can contribute to multiple samples.

Sampling in data science is crucial because it allows analysts to work with manageable subsets of data, reducing computational complexity and saving time. By analyzing the sample, data scientists can make inferences and draw conclusions about the larger population, assuming the sample is representative and unbiased.

Outcomes: Apply the transformations required on data to make it suitable for Mining

---

Conclusion (based on the Results and outcomes achieved):

The experiment demonstrated the effectiveness of the ID3 algorithm for data classification. It showcased the potential of decision tree algorithms in analyzing and categorizing datasets, providing valuable insights for various applications in data science and machine learning.

References:

Books/ Journals/ Websites

1. Han, Kamber, "Data Mining Concepts and Techniques", Morgan Kaufmann 3nd Edition

---