Experiment 5

Aim:

A. To perform Decision Tree learning in WEKA.
B. To Implement Decision Tree classifier in python

Theory:

Decision tree induction is the learning of decision trees from class-labeled training tuples. A decision tree is a flowchart-like tree structure, where each internal node non leaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node.Given a tuple, X, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules.

ID3, C4.5, and CART adopt a greedy (i.e., non backtracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner. Most algorithms for decision tree induction also follow a top-down approach, which starts with a training set of tuples and their associated class labels. The training set is recursively partitioned into smaller subsets as the tree is being built.

An attribute selection measure is a heuristic for selecting the splitting criterion that "best" separates a given data partition, D, of class-labeled training tuples into individual classes. If we were to split D into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be pure (i.e., all the tuples that fall into a given partition would belong to the same class). Conceptually, the "best" splitting criterion is the one that most closely results in such a scenario. Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split

Strengths and Weakness of Decision Tree approach

Strengths:

- Decision trees are able to generate understandable rules.
- Decision trees perform classification without requiring much computation.
- Decision trees are able to handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

Weaknesses:

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many class and relatively small numbers of training examples.
- Decision trees can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.

Algorithm :

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, D.

Input:

- Data partition, D, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

Method:

(1) create a node N;
(2) **if** tuples in D are all of the same class, C, **then**
(3)   return N as a leaf node labeled with the class C;
(4) **if** *attribute_list* is empty **then**
(5)   return N as a leaf node labeled with the majority class in D; // majority voting
(6) apply **Attribute_selection_method**(D, *attribute_list*) to **find** the "best" *splitting_criterion*;
(7) label node N with *splitting_criterion*;
(8) **if** *splitting_attribute* is discrete-valued **and**
    multiway splits allowed **then** // not restricted to binary trees
(9)   *attribute_list* ← *attribute_list* − *splitting_attribute*; // remove *splitting_attribute*
(10) **for each** outcome j of *splitting_criterion*
    // partition the tuples and grow subtrees for each partition
(11)   let D_j be the set of data tuples in D satisfying outcome j; // a partition
(12)   **if** D_j is empty **then**
(13)   attach a leaf labeled with the majority class in D to node N;
(14)   **else** attach the node returned by **Generate_decision_tree**(D_j, *attribute_list*) to node N;
    endfor
(15) return N;

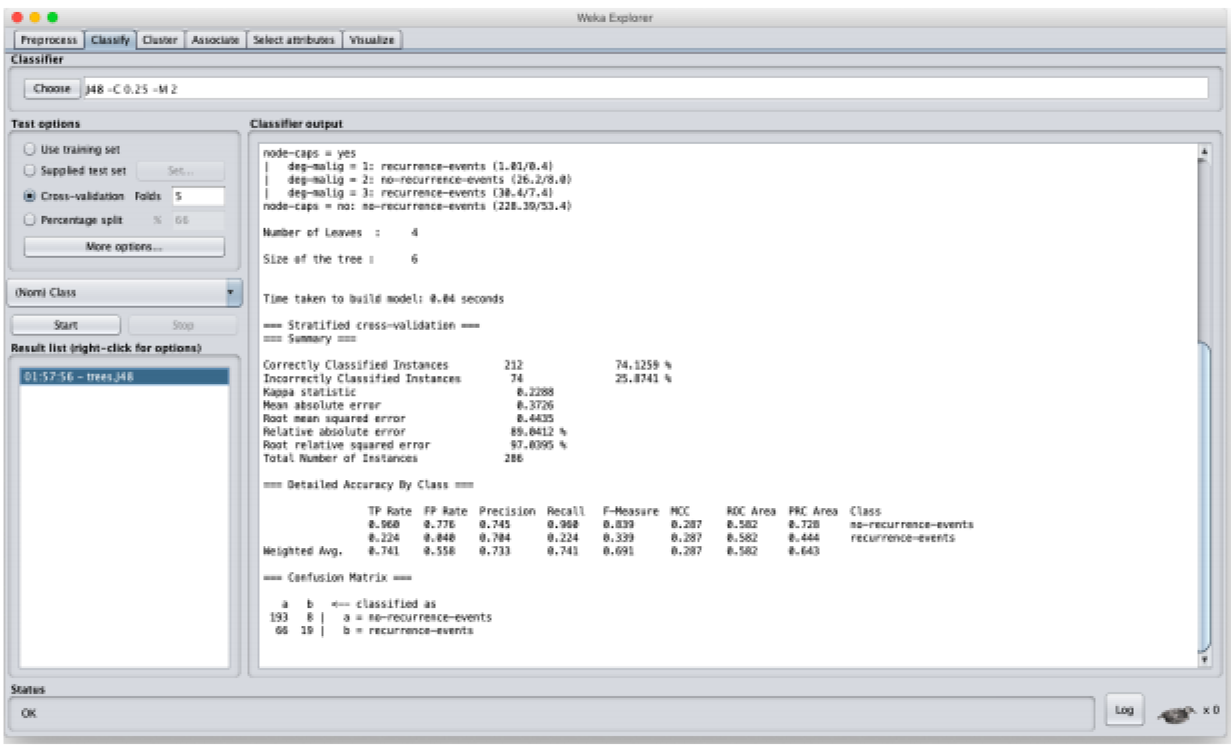
Part A

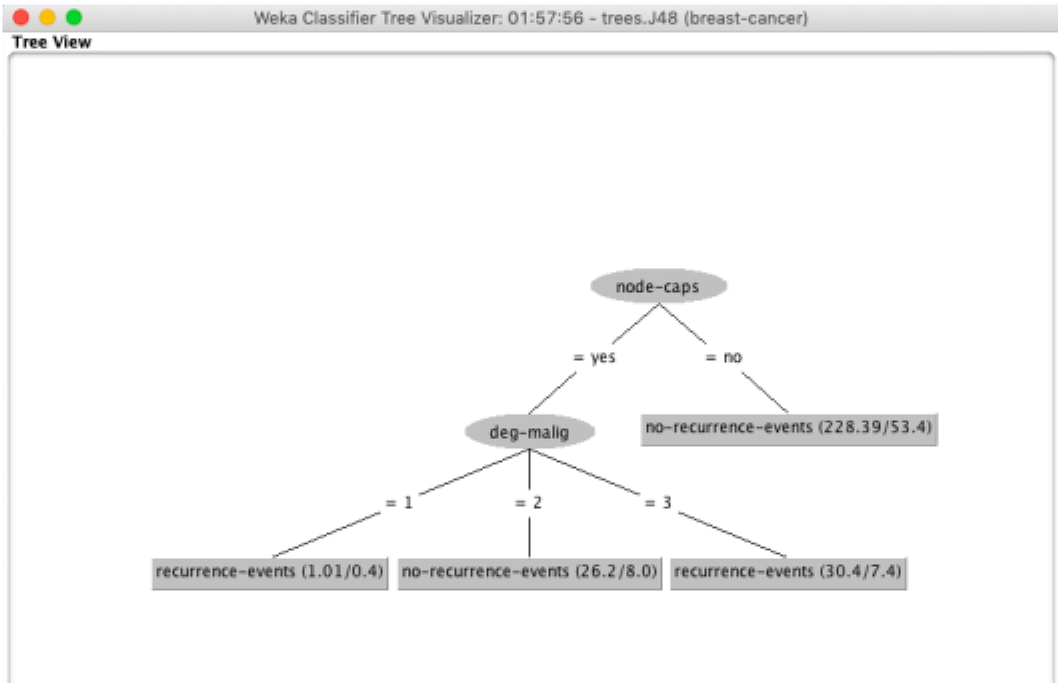
To perform Decision Tree learning in WEKA

Procedure :

1. Go to Weka Explorer.
2. Choose dataset in weka/data
3. Go to classify tab
4. Choose a classifier in trees/ID3 or any other.
5. Click start.
6. On the result, right click and visualize.

Output:





Findings and Learnings:

- Decision trees are the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.
- Weka software provides a good set of classification algorithms to be trained and tested on our dataset and makes it very simple to build a complex classifier using algorithms like decision trees.

Part B

To perform Decision Tree learning in Python

ID3 uses Information gain:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

$$Gain(A) = Info(D) - info_A(D)$$

C4.5 uses gain ratio:

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2 \frac{|D_j|}{|D|}$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

CART uses the GINI index:

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

Source Code:

```
import pandas as pd
import numpy as np
from pprint import pprint

def entropy(target_col):
    elements, counts = np.unique(target_col, return_counts=True)
    result = np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in
range(len(elements))])
    return result

def information_gain(data, split_attribute_name, target_name="class"):
    total_entropy = entropy(data[target_name])
```

```
    vals, counts = np.unique(data[split_attribute_name], return_counts=True)
    weighted_entropy = np.sum(
        [(counts[i] / np.sum(counts)) * entropy(data.where(data[split_attribute_name] ==
vals[i]).dropna()[target_name])
         for i in range(len(vals))])
    information_gain_ = total_entropy - weighted_entropy
    return information_gain_

def id3(data, original_data, features, target_attribute_name="class", parent_node_class=None):
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]
    elif len(data) == 0:
        return np.unique(original_data[target_attribute_name])[
            np.argmax(np.unique(original_data[target_attribute_name], return_counts=True)[1])]
    elif len(features) == 0:
        return parent_node_class
    else:
        parent_node_class = np.unique(data[target_attribute_name])[
            np.argmax(np.unique(data[target_attribute_name], return_counts=True)[1])]
        item_values = [information_gain(data, feature, target_attribute_name) for feature in
                       features]  # Return the information gain values for the features in the dataset
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]
        tree = {best_feature: {}}
        features = [i for i in features if i != best_feature]
        for value in np.unique(data[best_feature]):
            value = value
            sub_data = data.where(data[best_feature] == value).dropna()
            subtree = id3(sub_data, dataset, features, target_attribute_name, parent_node_class)
            tree[best_feature][value] = subtree
        return tree

def predict(query, tree, default=1):
    for key in list(query.keys()):
        if key in list(tree.keys()):
            try:
                result = tree[key][query[key]]
            except:
                return default
```

```
        result = tree[key][query[key]]
        if isinstance(result, dict):
            return predict(query, result)
        else:
            return result

def train_test_split(dataset):
    training_data = dataset.iloc[:80].reset_index(drop=True)
    testing_data = dataset.iloc[80:].reset_index(drop=True)
    return training_data, testing_data

def test(data, tree):
    queries = data.iloc[:, :-1].to_dict(orient="records")
    predicted = pd.DataFrame(columns=["predicted"])
    for i in range(len(data)):
        predicted.loc[i, "predicted"] = predict(queries[i], tree, 1.0)
    print('The prediction accuracy is: ', (np.sum(predicted["predicted"] == data["class"]) /
len(data)) * 100, '%')

if __name__ == '__main__':
    # loading the dataset
    dataset = pd.read_csv('zoo.csv', names=['animal_name', 'hair', 'feathers', 'eggs', 'milk',
                                            'airbone', 'aquatic', 'predator', 'toothed', 'backbone',
                                            'breathes', 'venomous', 'fins', 'legs', 'tail', 'domestic', 'catsize',
                                            'class'])

    print(dataset.head(10))

    dataset = dataset.drop('animal_name', axis=1)
    print(dataset.head(10))
    training_data = train_test_split(dataset)[0]
    testing_data = train_test_split(dataset)[1]

    tree = id3(training_data, training_data, training_data.columns[:-1])
    pprint(tree)
    test(testing_data, tree)
```

Output:

```
DWDM_LAB — -bash — 128×41

(ML) Anurags-MacBook-Air:DWDM_LAB jarvis$ python DecisionTree.py
  animal_name  hair  feathers  eggs  milk  airbone  aquatic  ...   venomous  fins  legs  tail  domestic  catsize  class
0    aardvark     1         0     0     1        0        0  ...          0     0     4     0         0        1      1
1    antelope     1         0     0     1        0        0  ...          0     0     4     1         0        1      1
2        bass     0         0     1     0        0        1  ...          0     1     0     1         0        0      4
3        bear     1         0     0     1        0        0  ...          0     0     4     0         0        1      1
4        boar     1         0     0     1        0        0  ...          0     0     4     1         0        1      1
5     buffalo     1         0     0     1        0        0  ...          0     0     4     1         0        1      1
6        calf     1         0     0     1        0        0  ...          0     0     4     1         1        1      1
7        carp     0         0     1     0        0        1  ...          0     1     0     1         1        0      4
8     catfish     0         0     1     0        0        1  ...          0     1     0     1         0        0      4
9        cavy     1         0     0     1        0        0  ...          0     0     4     0         1        0      1

[10 rows x 18 columns]
   hair  feathers  eggs  milk  airbone  aquatic  predator  ...   venomous  fins  legs  tail  domestic  catsize  class
0     1         0     0     1        0        0         1  ...          0     0     4     0         0        1      1
1     1         0     0     1        0        0         0  ...          0     0     4     1         0        1      1
2     0         0     1     0        0        1         1  ...          0     1     0     1         0        0      4
3     1         0     0     1        0        0         1  ...          0     0     4     0         0        1      1
4     1         0     0     1        0        0         1  ...          0     0     4     1         0        1      1
5     1         0     0     1        0        0         0  ...          0     0     4     1         0        1      1
6     1         0     0     1        0        0         0  ...          0     0     4     1         1        1      1
7     0         0     1     0        0        1         0  ...          0     1     0     1         1        0      4
8     0         0     1     0        0        1         1  ...          0     1     0     1         0        0      4
9     1         0     0     1        0        0         0  ...          0     0     4     0         1        0      1

[10 rows x 17 columns]
{'legs': {0: {'fins': {0.0: {'toothed': {0.0: 7.0, 1.0: 3.0}},
                       1.0: {'eggs': {0.0: 1.0, 1.0: 4.0}}}},
          2: {'hair': {0.0: 2.0, 1.0: 1.0}},
          4: {'hair': {0.0: {'toothed': {0.0: 7.0, 1.0: 5.0}}, 1.0: 1.0}},
          6: {'aquatic': {0.0: 6.0, 1.0: 7.0}},
          8: 7.0}}
The prediction accuracy is:  85.71428571428571 %
(ML) Anurags-MacBook-Air:DWDM_LAB jarvis$
(ML) Anurags-MacBook-Air:DWDM_LAB jarvis$
(ML) Anurags-MacBook-Air:DWDM_LAB jarvis$
(ML) Anurags-MacBook-Air:DWDM_LAB jarvis$
```

Findings and Learnings:

- We have Implemented Decision tree through the ID3 algorithm in python 3.
- We have learned the nuances of the Decision tree learning.
- We have learnt about the applications, strengths and weaknesses of Decision tree Learning