Practical No. 1

Code:

```
from collections import deque, defaultdict
class Graph:
    def __init__(self):
        self.graph = defaultdict(list)
    def add_edge(self,u,v):
        #For undirected graph
        self.graph[u].append(v)
        self.graph[v].append(u)
    def bfs(self,start):
        visited = set()
        queue = deque([start])
        traversal = []
        while queue:
            node = queue.popleft()
            if node not in visited:
                visited.add(node)
                traversal.append(node)
                for neighbor in self.graph[node]:
                    if neighbor not in visited:
                        queue.append(neighbor)
        return traversal
    def dfs(self,start):
        visited = set()
        stack = [start]
        traversal = []
        while stack:
```

```
            node = stack.pop()
            if node not in visited:
                visited.add(node)
                traversal.append(node)
                for neighbor in reversed(self.graph[node]):
                    if neighbor not in visited:
                        stack.append(neighbor)
        return traversal
g = Graph()
edges = [
    ('A','B'),('A','C'),
    ('B','D'),('B','E'),
    ('C','F'),('E','G'),
    ('F','H')
]
#Adding edges
for u,v in edges:
    g.add_edge(u,v)
start_node='A'
print("BFS Traversal: ",g.bfs(start_node))
print("DFS Traversal: ",g.dfs(start_node))
print("Narendra S")
```

OUTPUT :

```
BFS Traversal:  ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
DFS Traversal:  ['A', 'B', 'D', 'E', 'G', 'C', 'F', 'H']
Narendra S
```

Practical No. 2

Code:

```
import heapq
def a_star(graph, h, start, goal):
    open_list = []
    heapq.heappush(open_list, (h[start], 0, start, [start]))
    visited = set()
    while open_list:
        f, g, node, path = heapq.heappop(open_list)
        if node == goal:
            return path,g
        visited.add(node)
        for neighbor, cost in graph[node]:
            if neighbor not in visited:
                g_new = g + cost
                f_new = g_new + h[neighbor]
                heapq.heappush(open_list, (f_new, g_new, neighbor, path + [neighbor]))
    return None, float('inf')


def rbfs(graph, h, start, goal):
    def rbfs_helper(node, path, g, f_limit):
        if node == goal:
            return path, g
        successors = []
        for neighbor, cost in graph[node]:
            if neighbor not in path:
                g_new = g + cost
                f = max(g_new + h[neighbor] , f_limit)
```

```
            successors.append((f, neighbor, g_new, path + [neighbor]))
        if not successors:
            return None, float('inf')
        successors.sort()

        while successors:
            best = successors[0]
            alternative = successors[1][0] if len(successors)>1 else float('inf')
            result, f_new = rbfs_helper(best[1], best[3], best[2], min(f_limit, alternative))
            if result is not None:
                return result, f_new
            successors[0] = (f_new, best[1], best[2], best[3])
            successors.sort()
        return None, float('inf')
    return rbfs_helper(start, [start], 0, float('inf'))

graph = {
    'A':[('B',5), ('C', 10)],
    'B':[('A',5), ('D',4), ('E',3)],
    'C':[('A',10), ('G',2)],
    'D':[('B',4), ('E',6)],
    'E':[('B',3), ('D',6), ('G',2)],
    'G':[('C',2), ('E',2)]
}
heuristic = {
    'A':7,
    'B':6,
    'C':4,
    'D':3,
```

```
    'E':2,
    'G':0
}

start = 'A'
goal = 'G'
#Run A*
a_path, a_cost = a_star(graph, heuristic, start, goal)
print("A* Path: ", a_path, " Cost: ", a_cost)
#Run RBFS
rbfs_path, rbfs_cost = rbfs(graph, heuristic, start, goal)
print('RBFS Path: ', rbfs_path, ' Cost: ', rbfs_cost)
print(“Narendra S”)
```

OUTPUT :

```
A* Path:  ['A', 'B', 'E', 'G']  Cost:  10
RBFS Path:  ['A', 'B', 'D', 'E', 'G']  Cost:  17
Narendra S
```

Practical No. 3

Code:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
#Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
#Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3 , random_state = 42)
#Create and train the decision tree classifier
clf = DecisionTreeClassifier(criterion='entropy', random_state = 42)
clf.fit(X_train, y_train)
#Make Predictions
y_pred = clf.predict(X_test)
#Evaluate Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)
print("\nClassification Report: \n", classification_report(y_test, y_pred, target_names =
target_names))
print("Narendra S")
#Plot the decision tree
plt.figure(figsize=(12,8))
plot_tree(clf, feature_names = feature_names, class_names = target_names, filled = True)
```

plt.title("Decision Tree for Iris Dataset")

plt.show()

OUTPUT:

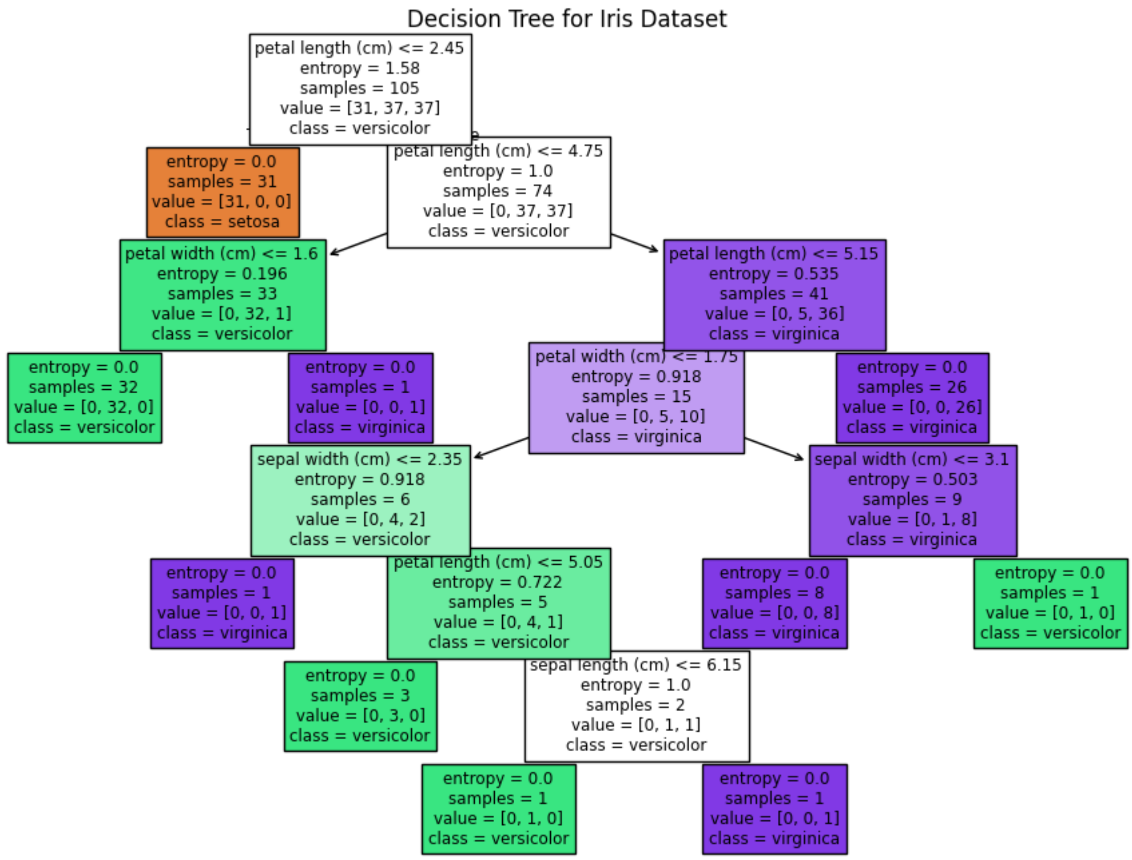
```
Accuracy:  0.9777777777777777

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       0.93      1.00      0.96        13
   virginica       1.00      0.92      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45

Narendra S
```



Practical No. 4

Code:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
#Load Binary classification dataset
data = load_breast_cancer()
X = data.data
y = data.target #Labels: 0= Malignant, 1 = Benign
#Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
#Split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.2, random_state =
42)
#Build SVM model
svm_model = SVC(kernel = 'rbf', C = 1.0, gamma = 'scale')
svm_model.fit(X_train, y_train)
#Make Predictions
y_pred = svm_model.predict(X_test)
#Evaluate Performance
print("Accuracy: ", accuracy_score(y_test, y_pred) * 100)
print("\nClassification Report: \n", classification_report(y_test, y_pred))
print("\nConfusion Matrix: \n", confusion_matrix(y_test, y_pred))
#Hyperparameter tning using GridSearchCV
param_grid = {
    'C' : [0.1, 1, 10],
```

```
    'gamma' : ['scale', 0.1, 1],
    'kernel' : ['rbf']
}
grid = GridSearchCV(SVC(), param_grid, cv = 5)
grid.fit(X_train, y_train)
print("\nBest Parameters: ", grid.best_params_)
print("Best Cross-validation Score: ", grid.best_score_)
print("Narendra S")
```

OUTPUT:

```
Accuracy:  97.36842105263158

Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114


Confusion Matrix:
 [[41  2]
 [ 1 70]]

Best Parameters:  {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
Best Cross-validation Score:  0.9736263736263737
Narendra S
```

Practical No. 5

Code:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
#Load dataset
data = load_breast_cancer()
X = data.data
y = data.target
#Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
#Weak classifier: Decision Stump (1-level Decision Tree)
weak_classifier = DecisionTreeClassifier(max_depth = 1)
#Train Adaboost model with 50 weak classifiers
adaboost_model = AdaBoostClassifier(estimator = weak_classifier, n_estimators = 50,
random_state = 42)
adaboost_model.fit(X_train, y_train)
#Predict and evaluate
y_pred_ada = adaboost_model.predict(X_test)
accuracy_ada = accuracy_score(y_test, y_pred_ada)
print(f"AdaBoost Accuracy: {accuracy_ada * 100:.2f}%")
print("\nClassification Report (Adaboost): \n", classification_report(y_test, y_pred_ada))
#Compare with a single weak classifier
weak_classifier.fit(X_train, y_train)
y_pred_weak = weak_classifier.predict(X_test)
accuracy_weak = accuracy_score(y_test, y_pred_weak)
print(f"\nSingle Weak Classifier Acccuracy: {accuracy_weak * 100:.2f}%")
```

print("Narendra S")

OUTPUT:

```
AdaBoost Accuracy: 96.49%

Classification Report (Adaboost):
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114


Single Weak Classifier Acccuracy: 89.47%
Narendra S
```

Practical No. 6

Code:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import numpy as np
#Load Dataset
data = load_iris()
X = data.data
y = data.target
class_names = data.target_names
#Split into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
#Train Naive Bayes Classifier
model = GaussianNB()
model.fit(X_train, y_train)
#Predict
y_pred = model.predict(X_test)
#Evaluate Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%\n")
#Class probabilities
probs = model.predict_proba(X_test)
print("Sample class probabilities (first 5 test samples): ")
print(np.round(probs[:5],3))
#Detailed Evaluation
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names =
class_names))
```

print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))

print("Narendra S")

OUTPUT:

```
Accuracy: 100.00%

l output actions class probabilities (first 5 test samples):
[[0.    0.996 0.004]
 [1.    0.    0.   ]
 [0.    0.    1.   ]
 [0.    0.978 0.022]
 [0.    0.87  0.13 ]]

Classification Report:
               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]]
Narendra S
```

Practical no. 7

Code:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import numpy as np
print("NarendraS_46")
data = load_iris()
X = data.data
y = data.target
class_names = data.target_names
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%\n")
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=class_names))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

OUTPUT:

```
NarendraS_46
Accuracy: 100.00%


Classification Report:
              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]]
```

Practical No. 8

Code:

```
import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules

dataset = [

['milk', 'bread', 'butter'],

['bread', 'butter'],

['milk', 'bread'],

['milk', 'butter'],

['bread'],

['milk', 'bread', 'butter'],

['butter'],

]

from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder()

tr_ary = te.fit(dataset).transform(dataset)

df = pd.DataFrame(tr_ary, columns=te.columns_)

frequent_itemsets = apriori(df, min_support=0.3, use_colnames=True)

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

print("Frequent Itemsets:\n", frequent_itemsets)

print("\nAssociation Rules:\n", rules[['antecedents', 'consequents', 'support', 'confidence',
'lift']])

print("NarendraS_46")
```

OUTPUT:

```
Frequent Itemsets:
     support          itemsets
0  0.714286          (bread)
1  0.714286         (butter)
2  0.571429           (milk)
3  0.428571  (bread, butter)
4  0.428571    (bread, milk)
5  0.428571   (milk, butter)

Association Rules:
   antecedents consequents   support  confidence  lift
0      (milk)     (bread)  0.428571        0.75  1.05
1      (milk)    (butter)  0.428571        0.75  1.05
NarendraS_46
```