

Practical No 1

AIM: Breadth First Search & Iterative Depth First Search

- Implement the Breadth First Search algorithm to solve a given problem.
- Implement the Iterative Depth First Search algorithm to solve the same problem.
- Compare the performance and efficiency of both algorithms.

Program 1: Breadth First Search.

```
from collections import deque
graph = {
    "S": [("A", 2), ("B", 4)],
    "A": [("S", 2), ("B", 2), ("C", 3), ("D", 7)],
    "B": [("S", 4), ("A", 2), ("C", 1)],
    "C": [("A", 3), ("B", 1), ("G", 5)],
    "D": [("A", 7), ("G", 2)],
    "G": [("C", 5), ("D", 2)]
}
def bfs(start, goal):
    queue = deque([(start, [start])])
    visited = set()
    while queue:
        node, path = queue.popleft()
        if node == goal:
            return path
        if node not in visited:
            visited.add(node)
            for neighbor, _ in graph[node]:
                queue.append((neighbor, path + [neighbor]))
    return None
print("BFS Path:", bfs("S", "G"))
```

Output:

BFS Path: ['S', 'A', 'C', 'G']

Program 2: Iterative Depth First Search.

```
from collections import deque
graph = { "S": [("A", 2), ("B", 4)],
          "A": [("S", 2), ("B", 2), ("C", 3), ("D", 7)],
          "B": [("S", 4), ("A", 2), ("C", 1)],
          "C": [("A", 3), ("B", 1), ("G", 5)],
          "D": [("A", 7), ("G", 2)],
          "G": [("C", 5), ("D", 2)]
}

def dfs_iter(start, goal):
    stack = [(start, [start])]
    visited = set()
    while stack:
        node, path = stack.pop()
        if node == goal:
            return path
        if node not in visited:
            visited.add(node)
            for neighbor, _ in graph[node]:
                stack.append((neighbor, path + [neighbor]))
    return None

print("DFS Iterative Path:", dfs_iter("S", "G"))
```

Output:

DFS Iterative Path: ['S', 'B', 'C', 'G']

Practical No 2

AIM: A* Search and Recursive Best-First Search.

- Implement the A* Search algorithm for solving a pathfinding problem.
- Implement the Recursive Best-First Search algorithm for the same problem.
- Compare the performance and effectiveness of both algorithms.

Program 1: A* Search

```
import heapq
graph = {
    "S": [("A", 2), ("B", 4)],
    "A": [("S", 2), ("B", 2), ("C", 3), ("D", 7)],
    "B": [("S", 4), ("A", 2), ("C", 1)],
    "C": [("A", 3), ("B", 1), ("G", 5)],
    "D": [("A", 7), ("G", 2)],
    "G": [("C", 5), ("D", 2)]
}
heuristic = {
    "S": 7, "A": 6, "B": 2, "C": 1, "D": 1, "G": 0
}
def astar(start, goal):
    pq = [(heuristic[start], 0, start, [start])] # (f, g, node, path)
    visited = set()
    while pq:
        f, g, node, path = heapq.heappop(pq)
        if node == goal:
            return path, g
        if node not in visited:
            visited.add(node)
            for neighbor, cost in graph[node]:
                new_g = g + cost
                new_f = new_g + heuristic[neighbor]
                heapq.heappush(pq, (new_f, new_g, neighbor, path + [neighbor]))
    return None, float("inf")
path, cost = astar("S", "G")
print("A* Path:", path, "Cost:", cost)
```

Output:

A* Path: ['S', 'B', 'C', 'G'] Cost: 10

Program 2: Recursive Best-First Search.

```
graph = {
    "S": [("A", 2), ("B", 4)],
    "A": [("S", 2), ("B", 2), ("C", 3), ("D", 7)],
    "B": [("S", 4), ("A", 2), ("C", 1)],
    "C": [("A", 3), ("B", 1), ("G", 5)],
    "D": [("A", 7), ("G", 2)],
    "G": [("C", 5), ("D", 2)]
}
heuristic = {
    "S": 7, "A": 6, "B": 2, "C": 1, "D": 1, "G": 0
}
def rbfs(node, path, g, f_limit, goal):
    if node == goal:
        return path, g
    successors = []
    for neighbor, cost in graph[node]:
        new_g = g + cost
        new_f = new_g + heuristic[neighbor]
        successors.append([new_f, neighbor, path + [neighbor], new_g])
    if not successors:
        return None, float("inf")
    while True:
        successors.sort(key=lambda x: x[0])
        best = successors[0]
        if best[0] > f_limit:
            return None, best[0]
        alternative = successors[1][0] if len(successors) > 1 else float("inf")
        result, best[0] = rbfs(
            best[1], best[2], best[3], min(f_limit, alternative), goal
        )
        successors[0] = best
        if result is not None:
            return result, best[0]
def recursive_best_first_search(start, goal):
    path, cost = rbfs(start, [start], 0, float("inf"), goal)
    return path, cost
path, cost = recursive_best_first_search("S", "G")
print("RBFS Path:", path, "Cost:", cost)
```

Output:

RBFS Path: ['S', 'A', 'B', 'C', 'G'] Cost: 10

Practical No 3

AIM: Decision Tree Learning-

- **Implement the Decision Tree Learning algorithm to build a decision tree for a given dataset.**
- **Evaluate the accuracy and effectiveness of the decision tree on test data.**
- **Visualize and interpret the generated decision tree.**

Install Libraries:

pip install pandas matplotlib seaborn scikit-learn

Program:

#imports

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
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, confusion_matrix
```

Load dataset

```
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target, name="species")
```

Train-Test Split (70% train, 30% test)

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
```

Train Decision Tree (using entropy & max_depth=3)

```
dtree = DecisionTreeClassifier(criterion="entropy", max_depth=3, random_state=42)
dtree.fit(X_train, y_train)
```

Predictions

```
y_pred = dtree.predict(X_test)
```

Evaluation

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Confusion Matrix Heatmap

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(
    cm, annot=True, fmt="d", cmap="Blues",
```

```

    xticklabels=iris.target_names,
    yticklabels=iris.target_names
)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# Visualize the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(
    dtree,
    feature_names=iris.feature_names,
    class_names=iris.target_names,
    filled=True,
    rounded=True,
    fontsize=10
)
plt.show()

```

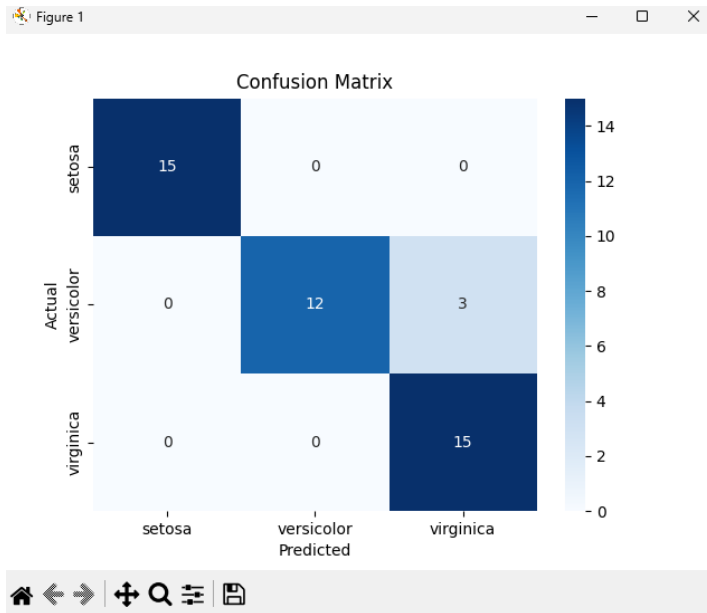
Output:

Accuracy: 0.9333333333333333

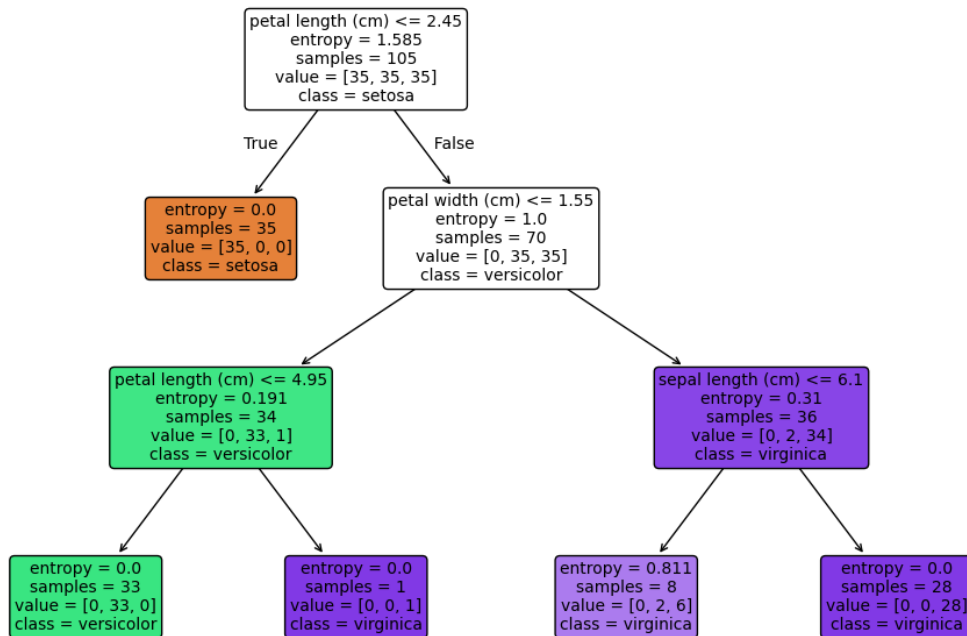
Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	15
1	1.00	0.80	0.89	15
2	0.83	1.00	0.91	15
accuracy			0.93	45
macro avg	0.94	0.93	0.93	45
weighted avg	0.94	0.93	0.93	45

Confusion Matrix-



Visual Representation-



Practical No 4

AIM: Feed Forward Backpropagation Neural Network

- Implement the Feed Forward Backpropagation algorithm to train a neural network.
- Use a given dataset to train the neural network for a specific task.
- Evaluate the performance of the trained network on test data.

Program:

Step 1: Import Libraries

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
```

Step 2: Load and Preprocess the Dataset

```
iris = load_iris()
X, y = iris.data, iris.target.reshape(-1, 1)
encoder = OneHotEncoder(sparse_output=False)
y = encoder.fit_transform(y)
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 3: Define the Neural Network Architecture and Initialize Weights

```
input_size = X_train.shape[1] # Number of features (4)
hidden_size = 8               # Number of neurons in the hidden layer
output_size = y_train.shape[1] # Number of output classes (3)
lr = 0.1                      # Learning rate for weight updates
epochs = 500                  # Number of passes through the entire training dataset
```

```
np.random.seed(42)
W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros((1, output_size))
```

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_deriv(x):
    return x * (1 - x)
def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)
```


Step 4: Training with Forward and Backward Propagation

for epoch in range(epochs):

 z1 = np.dot(X_train, W1) + b1

 a1 = sigmoid(z1)

 z2 = np.dot(a1, W2) + b2

 a2 = softmax(z2)

 loss = -np.mean(np.sum(y_train * np.log(a2 + 1e-9), axis=1))

 dz2 = a2 - y_train

 dW2 = np.dot(a1.T, dz2) / X_train.shape[0]

 db2 = np.sum(dz2, axis=0, keepdims=True) / X_train.shape[0]

 dz1 = np.dot(dz2, W2.T) * sigmoid_deriv(a1)

 dW1 = np.dot(X_train.T, dz1) / X_train.shape[0]

 db1 = np.sum(dz1, axis=0, keepdims=True) / X_train.shape[0]

 W1 -= lr * dW1

 b1 -= lr * db1

 W2 -= lr * dW2

 b2 -= lr * db2

if epoch % 100 == 0:

 print(f"Epoch {epoch}, Loss: {loss:.4f}")

Step 5: Evaluate the Model on the Test Set

z1_test = np.dot(X_test, W1) + b1

a1_test = sigmoid(z1_test)

z2_test = np.dot(a1_test, W2) + b2

a2_test = softmax(z2_test)

y_pred = np.argmax(a2_test, axis=1)

y_true = np.argmax(y_test, axis=1)

accuracy = np.mean(y_pred == y_true)

print(f"\nTest Accuracy: {accuracy:.4f}")

Output:

Epoch 0, Loss: 1.6243

Epoch 100, Loss: 0.4853

Epoch 200, Loss: 0.3677

Epoch 300, Loss: 0.3071

Epoch 400, Loss: 0.2612

Test Accuracy: 0.9667

Practical No 5

AIM: Support Vector Machines (SVM)

- Implement the SVM algorithm for binary classification.
- Train an SVM model using a given dataset and optimize its parameters.
- Evaluate the performance of the SVM model on test data and analyze the Results

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Generate synthetic dataset
X, y = make_classification(n_samples=300, n_features=2, n_redundant=0,
                          n_informative=2, n_clusters_per_class=1,
                          random_state=42)

# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
print("Training samples:", X_train.shape[0])
print("Test samples:", X_test.shape[0])

svm = SVC(kernel="linear", C=1.0, random_state=42) # Initialize with linear kernel
svm.fit(X_train, y_train) # Train
y_pred = svm.predict(X_test) # Predict
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Define parameter grid
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto']
}

# Grid Search with 5-fold CV
grid = GridSearchCV(SVC(random_state=42), param_grid, cv=5, scoring='accuracy')
```

```

grid.fit(X_train, y_train)
print("Best Parameters:", grid.best_params_)
print("Best Cross-Validation Score:", grid.best_score_)

# Best model
best_svm = grid.best_estimator_
# Predictions
y_best_pred = best_svm.predict(X_test)
# Accuracy
print("Test Accuracy:", accuracy_score(y_test, y_best_pred))
print("\nClassification Report:\n", classification_report(y_test, y_best_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_best_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

def plot_decision_boundary(model, X, y):
    h = 0.02
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                          np.arange(y_min, y_max, h))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=40, edgecolors='k', cmap=plt.cm.coolwarm)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title("SVM Decision Boundary")
    plt.show()

plot_decision_boundary(best_svm, X, y)

```

Output:

Training samples: 210

Test samples: 90

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.91	0.94	45
1	0.92	0.98	0.95	45
accuracy			0.94	90
macro avg	0.95	0.94	0.94	90
weighted avg	0.95	0.94	0.94	90

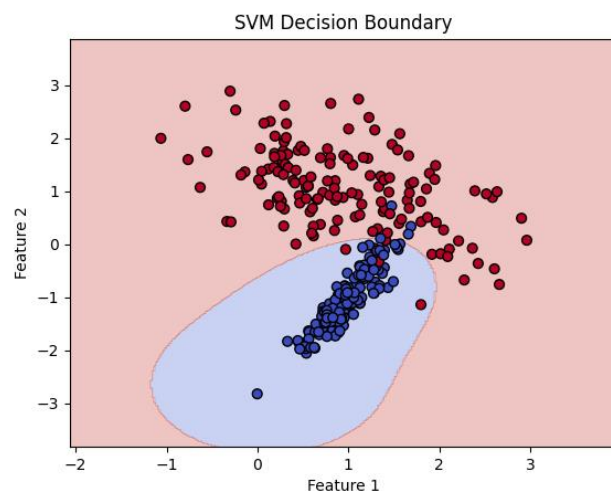
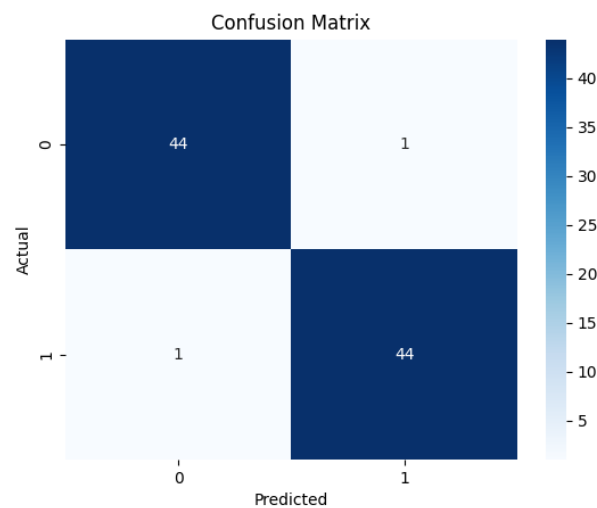
Best Parameters: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}

Best Cross-Validation Score: 0.9523809523809523

Test Accuracy: 0.9777777777777777

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.98	0.98	45
1	0.98	0.98	0.98	45
accuracy			0.98	90
macro avg	0.98	0.98	0.98	90
weighted avg	0.98	0.98	0.98	90



Practical No 6

AIM: Adaboost Ensemble Learning

- Implement the Adaboost algorithm to create an ensemble of weak classifiers.
- Train the ensemble model on a given dataset and evaluate its performance.
- Compare the results with individual weak classifiers.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier

# 1. Create a toy dataset
X, y = make_classification(
    n_samples=500, n_features=2, n_informative=2, n_redundant=0,
    n_clusters_per_class=1, flip_y=0.1, class_sep=1.5, random_state=42
)
y_mod = np.where(y == 0, -1, 1) # Convert labels from {0,1} to {-1,1} for AdaBoost from scratch
# Split train/test
X_train, X_test, y_train, y_test = train_test_split(X, y_mod, test_size=0.3, random_state=42)

# 2. Implement AdaBoost from Scratch
class AdaBoostScratch:
    def __init__(self, n_estimators=50):
        self.n_estimators = n_estimators
        self.alphas = []
        self.models = []

    def fit(self, X, y):
        n_samples, _ = X.shape
        # Initialize weights
        w = np.ones(n_samples) / n_samples
        for _ in range(self.n_estimators):
            # Train weak classifier (Decision stump)
            stump = DecisionTreeClassifier(max_depth=1, random_state=42)
            stump.fit(X, y, sample_weight=w)
            y_pred = stump.predict(X)
```

```

# Compute weighted error
err = np.sum(w * (y_pred != y)) / np.sum(w)
# Compute alpha
alpha = 0.5 * np.log((1 - err) / (err + 1e-10))
# Update weights
w *= np.exp(-alpha * y * y_pred)
w /= np.sum(w)
# Save
self.models.append(stump)
self.alphas.append(alpha)

```

```

def predict(self, X):
    # Weighted vote
    final_pred = np.zeros(X.shape[0])
    for alpha, model in zip(self.alphas, self.models):
        final_pred += alpha * model.predict(X)
    return np.sign(final_pred)

```

3. Train AdaBoost (Scratch)

```

ada_scratch = AdaBoostScratch(n_estimators=50)
ada_scratch.fit(X_train, y_train)
y_pred_scratch = ada_scratch.predict(X_test)

acc_scratch = accuracy_score(y_test, y_pred_scratch)
print("AdaBoost (Scratch) Accuracy:", acc_scratch)

```

4. Train AdaBoost (Sklearn)

```

ada_sklearn = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(max_depth=1),
    n_estimators=50,
    random_state=42
)
ada_sklearn.fit(X_train, y_train)
y_pred_sklearn = ada_sklearn.predict(X_test)

acc_sklearn = accuracy_score(y_test, y_pred_sklearn)
print("AdaBoost (Sklearn) Accuracy:", acc_sklearn)

```

5. Compare with Weak Classifier Alone

```

weak_clf = DecisionTreeClassifier(max_depth=1)
weak_clf.fit(X_train, y_train)

```

```

y_pred_weak = weak_clf.predict(X_test)
acc_weak = accuracy_score(y_test, y_pred_weak)
print("Weak Classifier Accuracy:", acc_weak)

```

6. Visualization (Decision Boundaries)

```

def plot_decision_boundary(model, X, y, title):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                        np.linspace(y_min, y_max, 200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k')
    plt.title(title)
    plt.show()

```

```

plot_decision_boundary(weak_clf, X_test, y_test, "Weak Classifier")
plot_decision_boundary(ada_sklearn, X_test, y_test, "AdaBoost (Sklearn)")

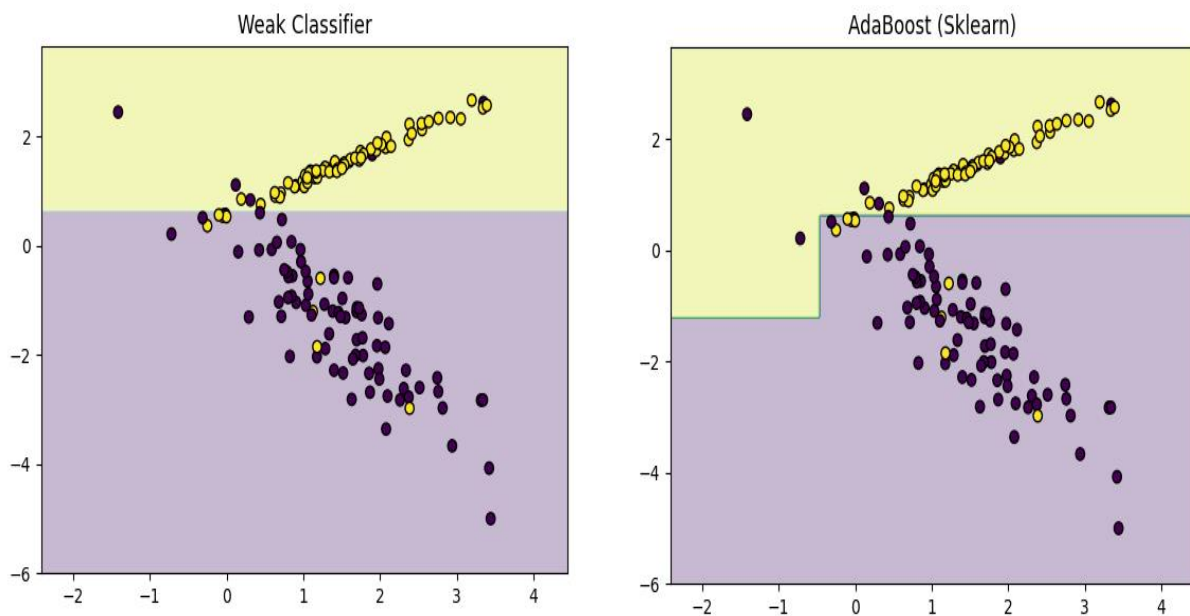
```

Output:

AdaBoost (Scratch) Accuracy: 0.9

AdaBoost (Sklearn) Accuracy: 0.9

Weak Classifier Accuracy: 0.9066666666666666



Practical No 7

AIM: Naive Bayes' Classifier

- **Implement the Naive Bayes' algorithm for classification.**
- **Train a Naive Bayes' model using a given dataset and calculate class probabilities.**
- **Evaluate the accuracy of the model on test data and analyze the results.**

Program:

```
import numpy as np
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

# 1. Load Dataset
iris = load_iris()
X, y = iris.data, iris.target
# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# 2. Train Naive Bayes Model
nb = GaussianNB()
nb.fit(X_train, y_train)

# 3. Predictions & Probabilities
y_pred = nb.predict(X_test)
y_prob = nb.predict_proba(X_test) # class probabilities

# 4. Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=iris.target_names))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

# 5. Show some probabilities
for i in range(5):
    print(f"Sample {i+1} True={iris.target_names[y_test[i]]}, Pred={iris.target_names[y_pred[i]]}")
    print("Probabilities:", y_prob[i])
    print("-" * 40)
```


Output:

Accuracy: 0.9777777777777777

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	19
versicolor	1.00	0.92	0.96	13
virginica	0.93	1.00	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

Confusion Matrix:

```
[[19  0  0]
```

```
[ 0 12  1]
```

```
[ 0  0 13]]
```

Sample 1 True=versicolor, Pred=versicolor

Probabilities: [4.15880005e-88 9.95527834e-01 4.47216606e-03]

Sample 2 True=setosa, Pred=setosa

Probabilities: [1.00000000e+00 1.31031235e-13 2.21772205e-20]

Sample 3 True=virginica, Pred=virginica

Probabilities: [9.83170191e-285 2.70138564e-012 1.00000000e+000]

Sample 4 True=versicolor, Pred=versicolor

Probabilities: [9.54745274e-92 9.74861431e-01 2.51385686e-02]

Sample 5 True=versicolor, Pred=versicolor

Probabilities: [1.0867956e-103 8.3191070e-001 1.6808930e-001]

|

Practical No 8

AIM: K-Nearest Neighbors (K-NN)

- **Implement the K-NN algorithm for classification or regression.**
- **Apply the K-NN algorithm to a given dataset and predict the class or value for test data.**
- **Evaluate the accuracy or error of the predictions and analyze the results.**

Program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.datasets import load_iris, fetch_california_housing

iris = load_iris()
X_cls, y_cls = iris.data, iris.target
housing = fetch_california_housing()
X_reg, y_reg = housing.data, housing.target
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(X_cls, y_cls, test_size=0.2,
random_state=42)
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_reg, y_reg, test_size=0.2,
random_state=42)

scaler_cls = StandardScaler()
scaler_reg = StandardScaler()
X_train_cls = scaler_cls.fit_transform(X_train_cls)
X_test_cls = scaler_cls.transform(X_test_cls)
X_train_reg = scaler_reg.fit_transform(X_train_reg)
X_test_reg = scaler_reg.transform(X_test_reg)

print("--- KNN Classification ---")
knn_cls = KNeighborsClassifier(n_neighbors=5)
knn_cls.fit(X_train_cls, y_train_cls)
y_pred_cls = knn_cls.predict(X_test_cls)
acc = accuracy_score(y_test_cls, y_pred_cls)
print(f"Classification Accuracy: {acc:.4f}\n")
print("--- KNN Regression ---")
knn_reg = KNeighborsRegressor(n_neighbors=5)
knn_reg.fit(X_train_reg, y_train_reg)
y_pred_reg = knn_reg.predict(X_test_reg)
```

```
mse = mean_squared_error(y_test_reg, y_pred_reg)
print(f"Regression MSE: {mse:.4f}")
```

Output:

--- KNN Classification ---
Classification Accuracy: 1.0000

--- KNN Regression ---
Regression MSE: 0.4324

Practical No 9

AIM: Association Rule Mining

- Implement the Association Rule Mining algorithm (e.g., Apriori) to find frequent itemsets.
- Generate association rules from the frequent itemsets and calculate their support and confidence.

Interpret and analyze the discovered association rules.

Install:

pip install pandas mlxtend

```
C:\Users\Tasmiya Shaikh>pip install pandas mlxtend
Requirement already satisfied: pandas in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (2.3.2)
Collecting mlxtend
  Downloading mlxtend-0.23.4-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: numpy>=1.26.0 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from pandas) (2.1.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.2.1 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from mlxtend) (1.16.2)
Requirement already satisfied: scikit-learn>=1.3.1 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from mlxtend) (1.7.2)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from mlxtend) (3.9.2)
Requirement already satisfied: joblib>=0.13.2 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from mlxtend) (1.5.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (4.55.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (3.2.0)
Requirement already satisfied: six>=1.5 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\tasmiya shaikh\appdata\local\programs\python\python312\lib\site-packages (from scikit-learn>=1.3.1->mlxtend) (3.6.0)
Downloading mlxtend-0.23.4-py3-none-any.whl (1.4 MB)
Installing collected packages: mlxtend
Successfully installed mlxtend-0.23.4
```

Program:

```
import pandas as pd
```

```
from mlxtend.frequent_patterns import apriori, association_rules
```

```
from mlxtend.preprocessing import TransactionEncoder
```

```
dataset = [
    ['Milk', 'Bread', 'Eggs'],
    ['Milk', 'Bread'],
    ['Milk', 'Eggs'],
    ['Bread', 'Eggs'],
    ['Milk', 'Bread', 'Butter'],
    ['Bread', 'Butter']
]
```

```
te = TransactionEncoder()
te_array = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_array, columns=te.columns_)
```

```
print("--- One-Hot Encoded DataFrame ---")
print(df)
print("\n" + "="*30 + "\n")
```

```
frequent_itemsets = apriori(df, min_support=0.3, use_colnames=True)
print("--- Frequent Itemsets (Support >= 0.3) ---")
print(frequent_itemsets)
```

```
print("\n" + "="*30 + "\n")
```

```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)
print("--- Association Rules (Confidence >= 0.6) ---")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

Output:

```
--- One-Hot Encoded DataFrame ---
  Bread Butter Eggs Milk
0  True  False  True  True
1  True  False False  True
2  False False  True  True
3  True  False  True False
4  True   True False  True
5  True   True False False

=====

--- Frequent Itemsets (Support >= 0.3) ---
  support  itemsets
0  0.833333  (Bread)
1  0.333333  (Butter)
2  0.500000  (Eggs)
3  0.666667  (Milk)
4  0.333333 (Butter, Bread)
5  0.333333 (Bread, Eggs)
6  0.500000 (Bread, Milk)
7  0.333333 (Milk, Eggs)

=====

--- Association Rules (Confidence >= 0.6) ---
  antecedents consequents  support  confidence  lift
0  (Butter)  (Bread)  0.333333  1.000000  1.2
1  (Eggs)   (Bread)  0.333333  0.666667  0.8
2  (Bread)  (Milk)   0.500000  0.600000  0.9
3  (Milk)   (Bread)  0.500000  0.750000  0.9
4  (Eggs)   (Milk)   0.333333  0.666667  1.0
```

Practical No 10

AIM: Demo of OpenAI/TensorFlow Tools:

- Explore and experiment with OpenAI or TensorFlow tools and libraries.
- Perform a demonstration or mini-project showcasing the capabilities of the tools.
- Discuss and present the findings and potential applications.

Program 1:

```
import google.generativeai as genai
import os
from dotenv import load_dotenv
```

```
load_dotenv()
```

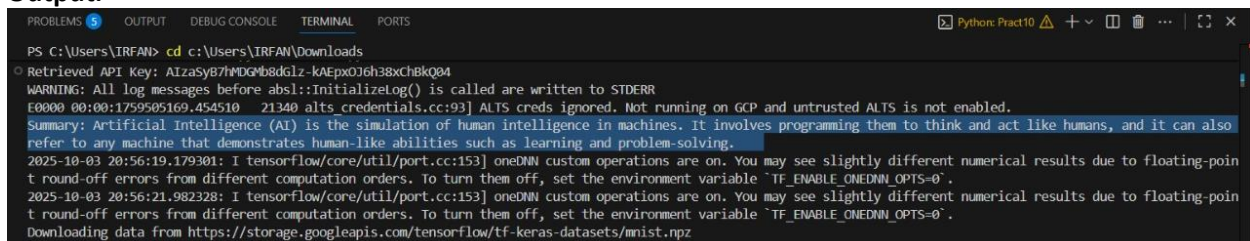
```
GOOGLE_API_KEY=os.getenv('GOOGLE_API_KEY')
print(f"Retrieved API Key: {GOOGLE_API_KEY}")
genai.configure(api_key=GOOGLE_API_KEY)
```

```
prompt = """Artificial Intelligence is the simulation of human intelligence
in machines that are programmed to think like humans and mimic their actions.
The term may also be applied to any machine that exhibits traits associated
with a human mind such as learning and problem-solving."""
```

```
try:
```

```
    gemini_model = genai.GenerativeModel('models/gemini-2.5-pro')
    response = gemini_model.generate_content('Summarize the following text:\n\n' + prompt)
    print("Summary:", response.text.strip())
except Exception as e:
    print(f"An error occurred with the Gemini API call: {e}")
```

Output:



```
PS C:\Users\IRFAN> cd c:\Users\IRFAN\Downloads
Retrieved API Key: AIzaSyB7hMDGmb8dG1z-kAEpxOJ6h38xChBkQ04
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1759505169.454510 21340 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untrusted ALTS is not enabled.
Summary: Artificial Intelligence (AI) is the simulation of human intelligence in machines. It involves programming them to think and act like humans, and it can also refer to any machine that demonstrates human-like abilities such as learning and problem-solving.
2025-10-03 20:56:19.179301: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-10-03 20:56:21.982328: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
```



```
# Evaluate the model on the test dataset
print("\nEvaluating the model on the test data...")
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print("\nTest Accuracy:", test_acc)
```

```
# --- Step 5: Visualize a Prediction ---
# Make a prediction on the first image in the test set
predictions = model.predict(x_test)
predicted_label = np.argmax(predictions[0])
true_label = y_test[0]
```

```
# Display the image and the prediction
plt.figure()
plt.imshow(x_test[0], cmap=plt.cm.binary)
plt.title(f'Predicted: {predicted_label}, True: {true_label}')
plt.colorbar()
plt.grid(False)
plt.show()
```

Output:

```
Loading MNIST dataset...
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 0s 0us/step
Dataset loaded and normalized.
Building the model...
Model built and compiled.
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape` / `input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object.
  super().__init__(**kwargs)
Model: "sequential"

Layer (type)                Output Shape                Param #
-----
flatten (Flatten)           (None, 784)                 0
dense (Dense)                (None, 10)                  7840
dense_1 (Dense)              (None, 10)                  1000

Total params: 8840 (397.54 KB)
Trainable params: 8840 (397.54 KB)
Non-trainable params: 0 (0.00 B)

Training the model...
Epoch 1/5
1688/1688 - 7s - 4ms/step - accuracy: 0.9205 - loss: 0.2763 - val_accuracy: 0.9625 - val_loss: 0.1314
Epoch 2/5
1688/1688 - 7s - 4ms/step - accuracy: 0.9638 - loss: 0.1233 - val_accuracy: 0.9748 - val_loss: 0.0932
Epoch 3/5
1688/1688 - 6s - 3ms/step - accuracy: 0.9749 - loss: 0.0832 - val_accuracy: 0.9758 - val_loss: 0.0825
Epoch 4/5
1688/1688 - 8s - 5ms/step - accuracy: 0.9803 - loss: 0.0628 - val_accuracy: 0.9747 - val_loss: 0.0867
Epoch 5/5
1688/1688 - 6s - 4ms/step - accuracy: 0.9849 - loss: 0.0482 - val_accuracy: 0.9780 - val_loss: 0.0761
Training finished.

Evaluating the model on the test data...
313/313 - 1s - 3ms/step - accuracy: 0.9761 - loss: 0.0749
```

