DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GOVERNMENT ENGINEERING COLLEGE, THRISSUR

Practical Record

221LCS100 - COMPUTING LAB 1

AROMAL M B

TCR24CSC04

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GOVERNMENT ENGINEERING COLLEGE, THRISSUR

CERTIFICATE

This is to certify that this is a bonafide report of the work done in **221LCS100 - COMPUTING LAB 1** by **Mr. AROMAL M B (TCR24CSCE04)** under our guidance towards partial fulfillment of the requirement for the award of the Degree of Master of Technology in Computer Science and Engineering of APJ Abdul Kalam Technological University during the year 2024 - 2026.

Prof. Dr. Gilesh M P,
Professor
Dept. of CSE,
Govt Engineering College, Thrissur

Prof. Dr. Ajay James,
Associate Professor and HOD,
Dept. of CSE,
Govt Engineering College, Thrissur

Contents

Experiment 1
Decision Tree - ID3 Algorithm

Aim

The objective of this experiment is to implement the ID3 algorithm for constructing decision trees using information gain.

Algorithm

1. **Load Libraries:** Import necessary libraries: `pandas`, `numpy`, `matplotlib`, `LabelEncoder`, `DecisionTreeClassifier`, and `train_test_split`.
2. **Load Dataset:** Read the dataset `tennis.csv` into a DataFrame.
3. **Encode Data:** Convert categorical data to numeric using `LabelEncoder`.
4. **Split Data:** Separate features (`X`) and target (`Y`), then split into training and testing sets using `train_test_split`.
5. **Train Model:** Initialize and train a `DecisionTreeClassifier` with `entropy` criterion.
6. **Make Predictions:** Predict outcomes on the test data.
7. **Evaluate Accuracy:** Print the model's accuracy on the test set.
8. **Single Prediction:** Make a prediction for a sample input.
9. **Visualize Tree:** Plot the decision tree.

Implementation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split

# Load the dataset
df = pd.read_csv('/content/tennis.csv')

# Initialize LabelEncoder
encoder = LabelEncoder()

# Encode all categorical columns
for col in df.columns:
    df[col] = encoder.fit_transform(df[col])

# Split data into features and target
X = df.drop(columns=['play'])     # Features
Y = df['play']  # Target

# Split 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)

# Train the Decision Tree model
model = DecisionTreeClassifier(criterion='entropy', random_state=42)
model.fit(X_train, Y_train)

# Make predictions and evaluate accuracy
Y_pred = model.predict(X_test)
accuracy = model.score(X_test, Y_test)
print(f"\nModel Accuracy: {accuracy * 100:.2f}%")
Model Accuracy: 100.00%

# Example prediction
sample_input = [[2, 1, 0, 1]]  # Example input: [outlook, temp, humidity, windy]
prediction = model.predict(sample_input)
print(f"\nPrediction for input {sample_input}: {prediction} (1 = Play, 0 = Don't Play)")
Prediction for input [[2, 1, 0, 1]]: [0] (1 = Play, 0 = Don't Play)

# Visualize the Decision Tree
plt.figure(figsize=(9,9))
plot_tree(model, feature_names=X.columns, class_names=encoder.classes_, filled=True)
plt.show()
```

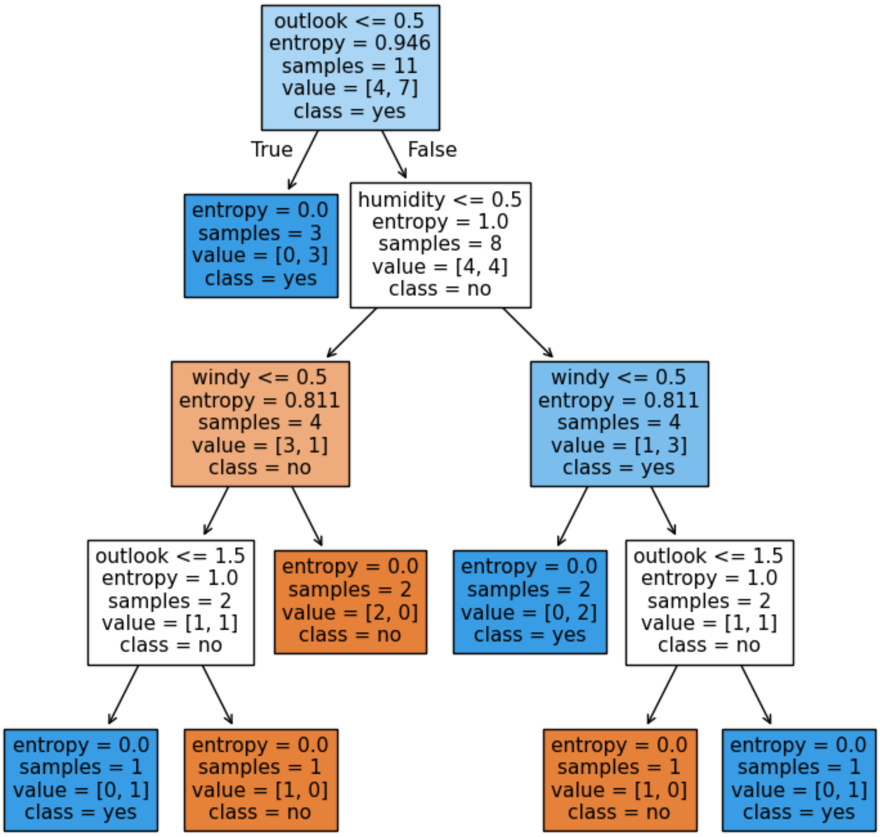


Figure 1: Decision Tree

Result

Successfully completed the program by implementing the Decision Tree (ID3) algorithm for classifying the tennis dataset.

Experiment 2
Naive Bayes Classifier on Iris Dataset

Aim

Write a program to implement the naïve bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Algorithm

1. **Load Libraries:** Import necessary libraries: `pandas`, `numpy`, and Naive Bayes from `sklearn`.
2. **Load Dataset:** Load the Iris dataset from `Iris.csv`.
3. **Preprocessing:** Drop the 'Id' column and ensure that all features are numerical.
4. **Split Data:** Split the dataset into features (`X`) and target (`y`), and then split the data into training and testing sets.
5. **Train Model:** Initialize and train a `GaussianNB` model using the training data.
6. **Make Predictions:** Use the trained model to predict the target for the test data.
7. **Evaluate Accuracy:** Calculate and print the accuracy of the model on the test set.
8. **Single Prediction:** Make predictions for a few sample inputs.

Implementation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = pd.read_csv('/content/Iris.csv')

# Drop the 'Id' column
iris.drop(columns="Id", inplace=True)

# Define features and target
X = iris.drop(columns='Species')
y = iris['Species']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
print("\nShapes of datasets:")
print(f"X_train: {X_train.shape}, X_test: {X_test.shape}")
print(f"y_train: {y_train.shape}, y_test: {y_test.shape}")
Shapes of datasets:
X_train: (105, 4), X_test: (45, 4)
y_train: (105,), y_test: (45,)

# Train the Gaussian Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy of the Naive Bayes model: {accuracy * 100:.2f}%")
Accuracy of the Naive Bayes model: 100.00%

# Example predictions
X_test_example = [
    [5.1, 3.5, 1.4, 0.2],
    [4.9, 3.0, 1.4, 0.2],]
y_pred_example = model.predict(X_test_example)
print("Predictions:", y_pred_example)
Predictions: ['Iris-setosa' 'Iris-setosa']
```

Result

Successfully implemented the Naive Bayes classifier on the Iris dataset.

Experiment 3
Naive Bayes Classifier on Iris Dataset (with Evaluation Metrics)

Aim

The objective of this experiment is to apply the Naive Bayes classifier on the Iris dataset, evaluate its performance using accuracy, precision, recall, and F1-score, and visualize class-wise metrics.

Algorithm

1. **Load Libraries:** Import the necessary libraries: `pandas`, `sklearn`, `matplotlib`, `seaborn`.
2. **Load Dataset:** Read the Iris dataset from `Iris.csv`.
3. **Preprocess Data:** Separate features (`X`) and target (`y`). Split the data into training and testing sets.
4. **Standardize Data:** Standardize the features using `StandardScaler`.
5. **Train Model:** Train a `GaussianNB` model using the training data.
6. **Make Predictions:** Predict the target values for the test set.
7. **Evaluate Model:** Calculate and print accuracy, precision, recall, and F1-score.
8. **Prediction for New Sample:** Make predictions for new sample data points.
9. **Visualize Metrics:** Visualize the precision, recall, and F1-score for each class.

Implementation

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score, \
        precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Iris dataset
file_path = '/content/Iris.csv'
data = pd.read_csv(file_path)

# Separate features and target
X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]  # Features
y = data['Species']  # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the Gaussian Naive Bayes model
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

# Make predictions
y_pred = nb_model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
Accuracy: 0.9777777777777777
Precision: 0.9793650793650793
Recall: 0.9777777777777777

# Print metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)

# Example prediction for a new sample
new_sample = [[5.0, 3.2, 4.0, 1.5]]
new_prediction = nb_model.predict(new_sample)
print("Prediction for new sample:", new_prediction[0])
Prediction for new sample: Iris-virginica

# Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

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

```
    Iris-setosa       1.00      1.00      1.00        19
Iris-versicolor       1.00      0.92      0.96        13
 Iris-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


# Calculate precision, recall, and F1-score by class
precision = precision_score(y_test, y_pred, average=None, labels=nb_model.classes_)
recall = recall_score(y_test, y_pred, average=None, labels=nb_model.classes_)
f1 = f1_score(y_test, y_pred, average=None, labels=nb_model.classes_)

# Dataframe for class-wise metrics
metrics = pd.DataFrame({
    'Class': nb_model.classes_,
    'Precision': precision,
    'Recall': recall,
    'F1-Score': f1
}).melt(id_vars='Class', var_name='Metric', value_name='Value')

# Plot the metrics
plt.figure(figsize=(10, 6))
sns.barplot(data=metrics, x='Class', y='Value', hue='Metric')
plt.title('Class-wise Precision, Recall, and F1-Score')
plt.ylim(0, 1.1)
plt.ylabel('Score')
plt.xlabel('Class')
plt.legend(loc='upper right')
plt.show()
```

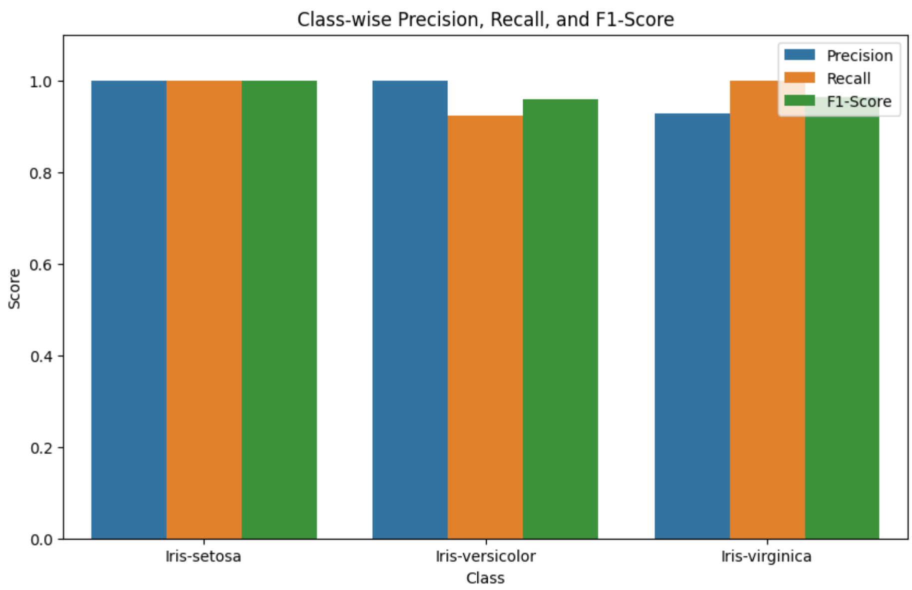


Figure 2: Class-wise Precision, Recall, and F1-Score

Result

The Naive Bayes classifier successfully classified the Iris dataset. The model's metrics, including accuracy, precision, recall, and F1-score, were evaluated, and class-wise metrics were visualized.

Experiment 4
Bayesian Network for Heart Disease Diagnosis

Aim

To construct a Bayesian Network using medical data and demonstrate the diagnosis of heart patients using the standard Heart Disease dataset.

Algorithm

1. Import the necessary libraries
2. Load the Heart Disease dataset and preprocess the data by handling missing values and encoding categorical features.
3. Define the structure of the Bayesian Network using domain knowledge or a structure-learning algorithm.
4. Train the Bayesian Network using Maximum Likelihood Estimation (MLE) or other suitable methods.
5. Perform inference on the network:
   - Query probabilities of the target variable (‘target‘) based on given evidence.
   - Predict the presence or absence of heart disease for individual patients.
6. Evaluate the model’s performance by comparing predictions with true labels and calculating accuracy.
7. Plot the accuracy of predictions to analyze model performance.

Implementation

```
!pip install pgmpy
import pandas as pd
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import ParameterEstimator,
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination

# Load the dataset
file_path = '/content/heart.csv'
data = pd.read_csv(file_path)

# Display dataset information
print("Dataset Head:\n", data.head())
print("\nDataset Info:\n", data.info())

# Preprocess the dataset
# Example: Assuming columns like 'sex', 'cp', and 'thal' are categorical
data[categorical_cols] = data[categorical_cols].astype('category')

# Define the Bayesian Network structure
# Example structure based on domain knowledge
model = BayesianNetwork([
    ('age', 'trestbps'),
    ('age', 'chol'),
    ('sex', 'thal'),
    ('chol', 'target'),
    ('thal', 'target'),
    ('trestbps', 'target'),])

# Train the model using Maximum Likelihood Estimation
model.fit(data, estimator=MaximumLikelihoodEstimator)

# Perform inference
inference = VariableElimination(model)

# Query: Probability of having heart disease given specific conditions
query_result = inference.query(
    variables=['target'],
    evidence={'sex': 1, 'age': 58, 'thal': 2}
print("\nQuery Result (Probability of Heart Disease):")
print(query_result)

# Predict for the entire dataset (optional)
predicted_target = []
for _, row in data.iterrows():
    evidence = row[['sex', 'age', 'thal']].to_dict()
    result = inference.map_query(variables=['target'], evidence=evidence)
    predicted_target.append(result['target'])

# Add predictions to the dataset
data['predicted_target'] = predicted_target
print("\nSample Predictions:")
print(data[['age', 'sex', 'thal', 'target', 'predicted_target']].head())

import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

```

```
# Assuming `data['target']` is the true target and
# `data['predicted_target']` is the predicted target
true_labels = data['target']
predicted_labels = data['predicted_target']

# Calculate overall accuracy
accuracy = accuracy_score(true_labels, predicted_labels)
print(f"Overall Accuracy: {accuracy:.2f}")

# Plotting accuracy
# Simulate accuracy over sample sizes for demonstration
sample_sizes = range(10, len(data), 10)
accuracies = [
    accuracy_score(true_labels[:size], predicted_labels[:size])
    for size in sample_sizes]

# Plot accuracy vs. sample size
plt.figure(figsize=(10, 6))
plt.plot(sample_sizes, accuracies, marker='o', linestyle='-', \
    color='blue', label='Accuracy')
plt.axhline(y=accuracy, color='red', linestyle='--', \
    label=f'Overall Accuracy: {accuracy:.2f}')
plt.title("Accuracy of Bayesian Network Predictions")
plt.xlabel("Number of Samples")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()

#Ouput
Query Result (Probability of Heart Disease):
+-----------+---------------+
| target    |   phi(target) |
+===========+===============+
| target(0) |        0.4821 |
+-----------+---------------+
| target(1) |        0.5179 |
+-----------+---------------+

Sample Inputs
   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0

Prediction on Sample Inputs
Sample Predictions:
   age sex thal  target  predicted_target
0   52   1    3       0                 0
1   53   1    3       0                 0
2   70   1    3       0                 0
3   61   1    3       0                 0
4   62   0    2       0                 1
Overall Accuracy: 0.79
```

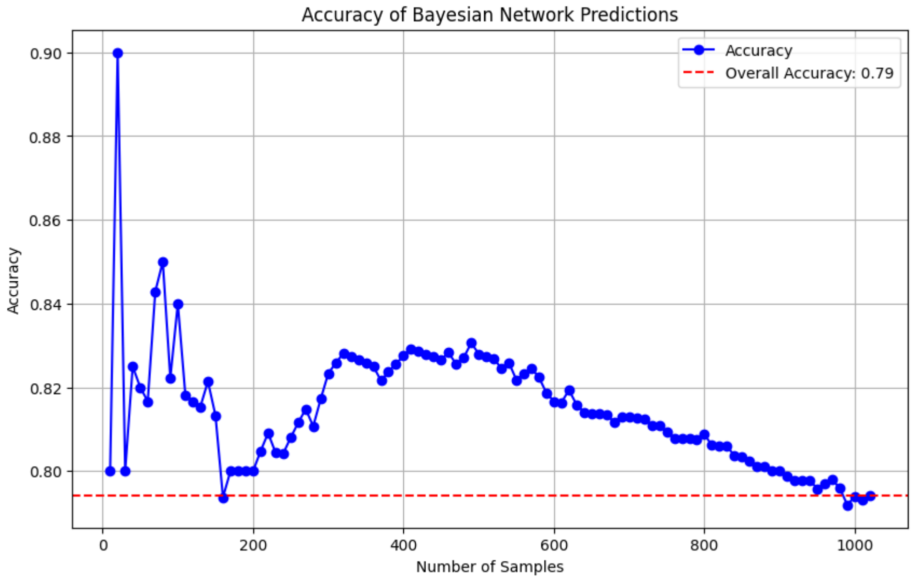


Figure 3: Accuracy Curve

Result

The Bayesian Network was successfully constructed and trained using the Heart Disease dataset. The network accurately predicted the presence or absence of heart disease.

Experiment 5
Clustering Using EM Algorithm and K-Means

Aim

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering.

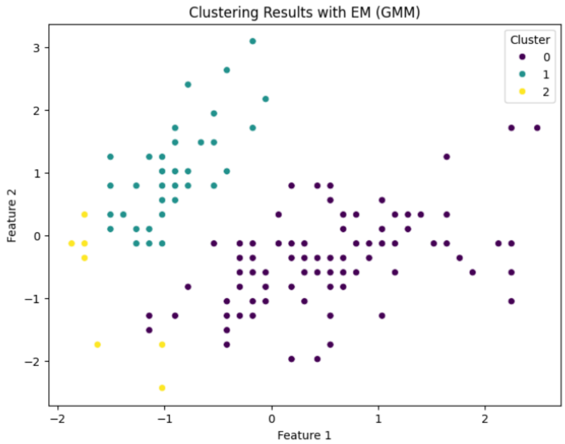
Algorithm

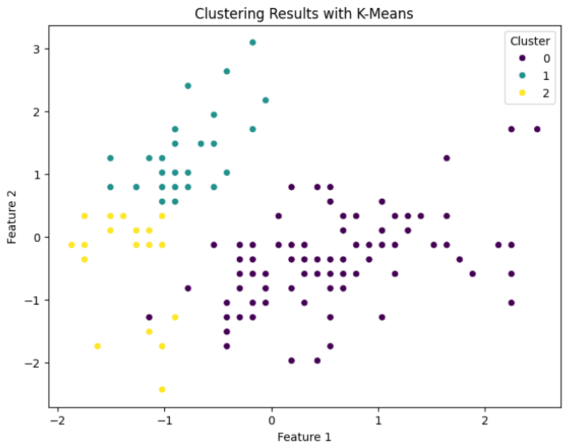
1. **Import Libraries:** Import necessary libraries
2. **Load Data:** Load the Iris dataset and extract the features.
3. **Preprocessing:** Standardize the dataset to scale the features.
4. **Gaussian Mixture Model (EM Algorithm):**
    - Apply the Gaussian Mixture Model (GMM) with the number of clusters set to 3.
    - Fit the model to the dataset and predict the cluster labels.
5. **K-Means Clustering:**
    - Apply K-Means clustering with 3 clusters.
    - Fit the model and predict the cluster labels.
6. **Evaluate and Compare:** Compare the clustering results using silhouette scores and visualizations.
7. **Visualize Clusters:** Visualize the clusters obtained from both algorithms using scatter plots.

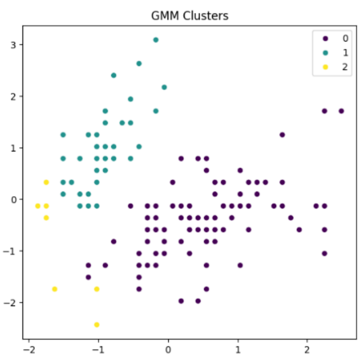
Implementation

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score, accuracy_score

# Load the Iris dataset from sklearn
from sklearn.datasets import load_iris

# Load the data into a pandas DataFrame
iris = load_iris()
data = pd.DataFrame(iris.data, columns=iris.feature_names)

# Display the first few rows of the dataset
data.head()

# Standardize the data to ensure better performance of clustering algorithms
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled[:5]
array([[-0.90068117,  1.01900435, -1.34022653, -1.3154443 ],
       [-1.14301691, -0.13197948, -1.34022653, -1.3154443 ],
       [-1.38535265,  0.32841405, -1.39706395, -1.3154443 ],
       [-1.50652052,  0.09821729, -1.2833891 , -1.3154443 ],
       [-1.02184904,  1.24920112, -1.34022653, -1.3154443 ]])

# Apply the EM algorithm using GaussianMixture model (GMM)
gmm = GaussianMixture(n_components=3, random_state=42)
gmm.fit(X_scaled)
gmm_labels = gmm.predict(X_scaled)
data['GMM_Cluster'] = gmm_labels

# Visualize the GMM Clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_scaled[:, 0], y=X_scaled[:, 1], hue=gmm_labels, palette='viridis')
plt.title('Clustering Results with EM (GMM)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend(title='Cluster')
plt.show()

# Apply k-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)
kmeans_labels = kmeans.predict(X_scaled)
data['KMeans_Cluster'] = kmeans_labels

# Visualize the KMeans Clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_scaled[:, 0], y=X_scaled[:, 1], hue=kmeans_labels, palette='viridis')
plt.title('Clustering Results with K-Means')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

```
plt.legend(title='Cluster')
plt.show()

# Evaluating the clustering performance using silhouette score
gmm_silhouette = silhouette_score(X_scaled, gmm_labels)
kmeans_silhouette = silhouette_score(X_scaled, kmeans_labels)
Number of clusters in GMM: 3
Number of clusters in KMeans: 3

print(f'Silhouette Score for GMM: {gmm_silhouette:.4f}')
print(f'Silhouette Score for KMeans: {kmeans_silhouette:.4f}')
Silhouette Score for GMM: 0.4751
Silhouette Score for KMeans: 0.4799
```


Clustering Results with EM (GMM)
Cluster
0
1
2
Feature 2
Feature 1
−2
−1
0
1
2
3


Clustering Results with K-Means
Cluster
0
1
2
Feature 2
Feature 1
−2
−1
0
1
2
3


GMM Clusters
0
1
2


KMeans Clusters
0
1
2

Result

Successfully implemented the Gaussian Mixture Model (EM algorithm) and K-Means clustering algorithms on the Iris dataset. The silhouette scores were similar for both.

Experiment 6
Classification Using k-Nearest Neighbour Algorithm

Aim

Implement the k-Nearest Neighbour (k-NN) algorithm to classify the Iris dataset and print both correct and wrong predictions.

Algorithm

1. **Import Libraries:** Import necessary libraries
2. **Load Data:** Load the Iris dataset from ‘sklearn‘.
3. **Split Data:** Divide the dataset into training and testing sets with an 80-20 split.
4. **Preprocessing:** Standardize the dataset using ‘StandardScaler‘ for better performance.
5. **Train k-NN Model:**
   - Initialize the k-NN classifier with the desired value of ‘k‘.
   - Train the model using the scaled training data.
6. **Make Predictions:**
   - Predict the labels for the test dataset.
   - Compare predicted labels with actual labels to identify correct and wrong predictions.
7. **Evaluate:** Calculate classification metrics, accuracy, and display predictions.

Implementation

```
# Importing necessary libraries
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# 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, stratify=y)

# Standardize the feature values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the k-NN classifier
k = 3
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the predictions
correct_predictions = [
    (i, true, pred) for i, (true, pred) in enumerate(zip(y_test, y_pred)) if true == pred]
wrong_predictions = [
    (i, true, pred) for i, (true, pred) in enumerate(zip(y_test, y_pred)) if true != pred]

# Print correct predictions
print("\nCorrect Predictions:")
for index, true, pred in correct_predictions:
    print(f"Index: {index}, True Label: {true}, Predicted Label: {pred}")
Correct Predictions:
Index: 0, True Label: 1, Predicted Label: 1
Index: 2, True Label: 0, Predicted Label: 0
Index: 3, True Label: 2, Predicted Label: 2
Index: 4, True Label: 1, Predicted Label: 1
Index: 5, True Label: 1, Predicted Label: 1
Index: 6, True Label: 1, Predicted Label: 1
Index: 7, True Label: 0, Predicted Label: 0
Index: 8, True Label: 0, Predicted Label: 0
Index: 9, True Label: 0, Predicted Label: 0
Index: 10, True Label: 1, Predicted Label: 1
Index: 11, True Label: 0, Predicted Label: 0
Index: 12, True Label: 0, Predicted Label: 0
Index: 13, True Label: 2, Predicted Label: 2
Index: 14, True Label: 2, Predicted Label: 2
Index: 15, True Label: 2, Predicted Label: 2
Index: 16, True Label: 2, Predicted Label: 2
Index: 17, True Label: 2, Predicted Label: 2
```

```
Index: 18, True Label: 1, Predicted Label: 1
Index: 19, True Label: 0, Predicted Label: 0
Index: 20, True Label: 1, Predicted Label: 1
Index: 21, True Label: 2, Predicted Label: 2
Index: 22, True Label: 1, Predicted Label: 1
Index: 23, True Label: 2, Predicted Label: 2
...
Index: 26, True Label: 0, Predicted Label: 0
Index: 27, True Label: 1, Predicted Label: 1
Index: 28, True Label: 2, Predicted Label: 2
Index: 29, True Label: 0, Predicted Label: 0

# Print wrong predictions
print("\nWrong Predictions:")
for index, true, pred in wrong_predictions:
    print(f"Index: {index}, True Label: {true}, Predicted Label: {pred}")
Wrong Predictions:
Index: 1, True Label: 2, Predicted Label: 1

# Print classification report and accuracy
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.91      1.00      0.95        10
   virginica       1.00      0.90      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

Accuracy: 0.97

# Test the algorithm with new examples
def test_knn(example):
    example_scaled = scaler.transform([example])
    prediction = knn.predict(example_scaled)
    predicted_class = iris.target_names[prediction[0]]
    return predicted_class
example_1 = [5.1, 3.5, 1.4, 0.2]
example_2 = [6.7, 3.0, 5.2, 2.3]
print("\nTesting new examples:")
print(f"Example 1: {example_1} -> Predicted Class: {test_knn(example_1)}")
print(f"Example 2: {example_2} -> Predicted Class: {test_knn(example_2)}")

Testing new examples:
Example 1: [5.1, 3.5, 1.4, 0.2] -> Predicted Class: setosa
Example 2: [6.7, 3.0, 5.2, 2.3] -> Predicted Class: virginica
```

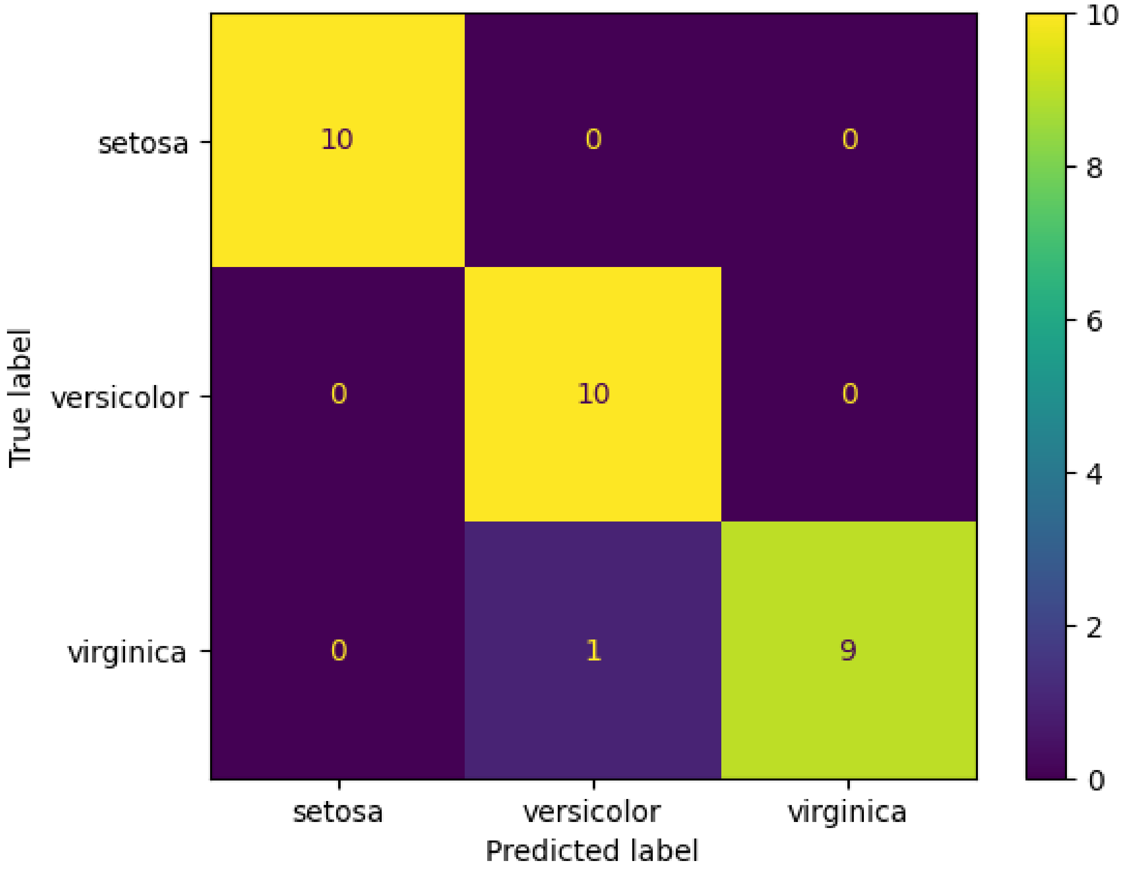


Figure 4: k-NN Confusion Matrix

Result

Successfully implemented the k-Nearest Neighbour (k-NN) algorithm on the Iris dataset. The correct and wrong predictions were printed, and the classification report showed the precision, recall, and F1-scores for all classes.

Experiment 7
Locally Weighted Regression (LWR)

Aim

Implement the Locally Weighted Regression (LWR) algorithm to fit data points using the ‘tips‘ dataset. Visualize the fitted curve and analyze the results.

Algorithm

1. **Import Required Libraries:** Import the necessary libraries
2. **Load the Dataset:** Load the ‘tips‘ dataset, and extract the independent variable (‘total_bill‘) and dependent variable (‘tip‘).
3. **Define Kernel Function:** Define a kernel function (e.g., Gaussian kernel) that will compute weights for data points based on their distance from the query point.
4. **Weighted Linear Regression:** For each query point:
   - Compute the weights using the kernel function.
   - Perform weighted linear regression to calculate the predicted value for the query point.
5. **Make Predictions:** Repeat the prediction for a range of query points, which will create a smooth fitted curve.
6. **Visualize Results:** Visualize the original data points along with the fitted curve on a scatter plot.

Implementation

```python
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# Load the tips dataset
tips = sns.load_dataset('tips')
X = tips['total_bill'].values
y = tips['tip'].values
X = X.reshape(-1, 1)

# Define Gaussian kernel
def kernel(x, x_point, tau):
    return np.exp(-np.sum((x - x_point) ** 2, axis=1) / (2 * tau ** 2))

# Locally Weighted Regression function
def locally_weighted_regression(x_query, X, y, tau):
    m = len(X)
    weights = kernel(X, x_query, tau)
    W = np.diag(weights)
    X_aug = np.hstack((np.ones((m, 1)), X))
    theta = np.linalg.pinv(X_aug.T @ W @ X_aug) @ (X_aug.T @ W @ y)
    return np.dot([1, x_query], theta)

# Perform predictions for a range of x values
tau = 5
x_values = np.linspace(X.min(), X.max(), 200)
y_pred = np.array([locally_weighted_regression(np.array([x]), X, y, tau) \
    for x in x_values])

# Plot original data and fitted curve
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', alpha=0.5, label='Original Data')
plt.plot(x_values, y_pred, color='red', label='LWR Fit (tau=5)', linewidth=2)
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.legend()
plt.show()
```

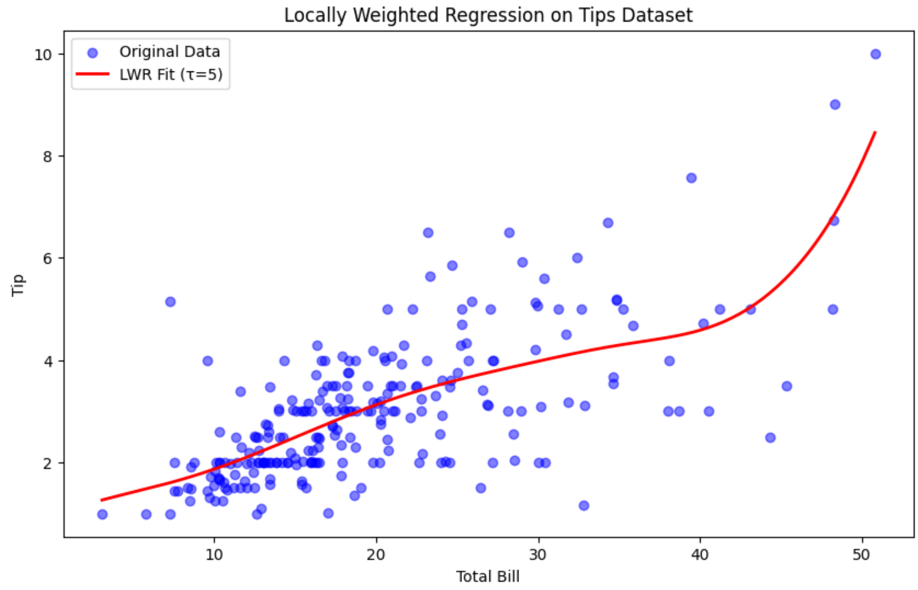


Figure 5: Fitted Curve for Locally Weighted Regression (LWR) on Tips Dataset

Result

Successfully implemented the Locally Weighted Regression (LWR) algorithm on the ‘tips‘ dataset.

Experiment 8
5-Fold Cross Validation

Aim

The aim of this experiment is to perform 5-fold cross-validation on the Breast Cancer dataset using Logistic Regression. Evaluate the model's performance based on the metrics of accuracy, precision, recall, and F1-score.

Algorithm

1. **Import Required Libraries:** Import the necessary libraries
2. **Load the Dataset:** Load the Breast Cancer dataset from the 'sklearn.datasets' module.
3. **Initialize the Classifier:** Use Logistic Regression as the classifier.
4. **Set Up Cross-Validation:** Implement 5-fold cross-validation using the 'StratifiedKFold' method to preserve the percentage of samples for each class in each fold.
5. **Model Evaluation:** Use cross-validation to evaluate the classifier's performance based on accuracy, precision, recall, and F1-score for each fold.
6. **Plot the Results:** Plot the evaluation metrics (accuracy, precision, recall, F1-score) for each fold to visualize the performance.

Implementation

```python
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_validate, StratifiedKFold
from sklearn.linear_model import LogisticRegression
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Initialize the classifier (you can change it to any classifier)
model = LogisticRegression(max_iter=10000, random_state=42)

# Set up 5-fold cross-validation
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Cross-validate the model using 5-fold cross-validation
cv_results = cross_validate(
    model, X, y, cv=kf, scoring=['accuracy', 'precision', 'recall', 'f1'], \
        return_train_score=False
)

# Extract metrics for each fold
accuracy = cv_results['test_accuracy']
precision = cv_results['test_precision']
recall = cv_results['test_recall']
f1 = cv_results['test_f1']

# Display the results for each fold
results_df = pd.DataFrame({
    'Fold': np.arange(1, 6),
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F-Score': f1
})

print("Cross-validation results for each fold:")
print(results_df)

# Display average performance metrics across all folds
print("\nAverage Performance across all 5 folds:")
print(f"Average Accuracy: {accuracy.mean():.4f}")
print(f"Average Precision: {precision.mean():.4f}")
print(f"Average Recall: {recall.mean():.4f}")
print(f"Average F-Score: {f1.mean():.4f}")

# Plotting the metrics for each fold
fig, ax = plt.subplots(figsize=(10, 6))

# Define the positions of the bars for each fold
x = np.arange(5)

# Bar width
width = 0.2
```

```

# Plotting each metric
ax.bar(x - width*1.5, accuracy, width, label='Accuracy', color='blue')
ax.bar(x - width*0.5, precision, width, label='Precision', color='green')
ax.bar(x + width*0.5, recall, width, label='Recall', color='orange')
ax.bar(x + width*1.5, f1, width, label='F1 Score', color='red')

# Adding labels and title
ax.set_xlabel('Fold')
ax.set_ylabel('Score')
ax.set_title('Evaluation Metrics for Each Fold (5-Fold Cross Validation)')
ax.set_xticks(x)
ax.set_xticklabels([f'Fold {i}' for i in range(1, 6)])
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```

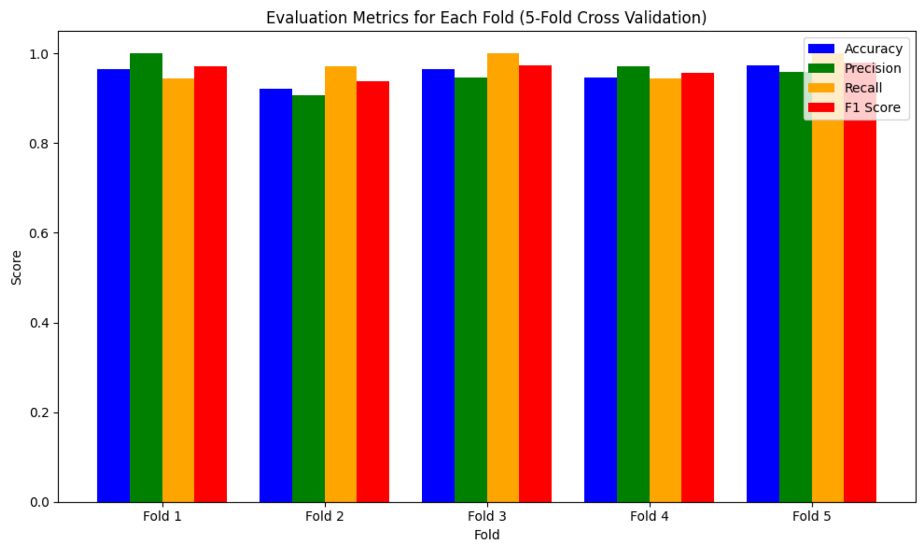


Figure 6: Evaluation Metrics for Each Fold (5-Fold Cross Validation)

Result

The 5-fold cross-validation results for the Breast Cancer dataset are shown below, detailing the accuracy, precision, recall, and F1-score for each fold. The average performance metrics across all folds are also presented.

Experiment 9
Support Vector Machine for CIFAR-10 Classification

Aim

The aim of this experiment is to implement Support Vector Machine (SVM) classifier on the CIFAR-10 dataset, using KNN and a 3-layer neural network.

Algorithm

1. **Import Libraries:** Import necessary libraries (e.g., `numpy`, `sklearn`, `tensorflow`) for data handling and model building.
2. **Load CIFAR-10 Dataset:** Load the CIFAR-10 dataset, normalize pixel values to [0, 1], and flatten images for KNN.
3. Train KNN Classifier: Train a KNN classifier with a subset of the dataset.
4. **Predict Using KNN:** Predict labels for a subset of the test data using the trained KNN classifier.
5. **Evaluate KNN:** Evaluate the KNN classifier by calculating accuracy and generating a classification report.
6. **Define Neural Network:** Define a 3-layer neural network with ReLU activations and softmax output for 10 classes.
7. **Train Neural Network:** Train the neural network on a subset of the CIFAR-10 dataset with validation monitoring.
8. **Predict Using Neural Network:** Predict labels for a subset of the test data using the trained neural network.

9. **Evaluate Neural Network:** Evaluate the neural network on the test set and generate a classification report.

10. **Compare Results:** Compare the accuracy and performance of KNN and the neural network.

Implementation

```python
import numpy as np
from tensorflow.keras.datasets import cifar10
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from sklearn.neighbors import KNeighborsClassifier

# Load CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Normalize pixel values to [0, 1]
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Flatten images for KNN and scaling
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)

# Reshape labels to 1D
y_train = y_train.ravel()
y_test = y_test.ravel()

print("Data loaded and preprocessed.")

# Initialize KNN with k=15 and Manhattan distance
knn = KNeighborsClassifier(n_neighbors=15, metric='manhattan')

# Train KNN on the CIFAR-10 training data
print("Training KNN...")
knn.fit(X_train_flat[:20000], y_train[:20000]) # Use a subset for computational efficiency
print("KNN training complete.")

# Predict on the test set
y_pred_knn = knn.predict(X_test_flat[:5000])  # Use a subset for faster testing

# Evaluate KNN
knn_accuracy = accuracy_score(y_test[:5000], y_pred_knn)
print(f"KNN Accuracy: {knn_accuracy:.4f}")
print("\nClassification Report:")
print(classification_report(y_test[:5000], y_pred_knn))

# Define the 3-layer neural network
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(32, 32, 3)),  # Input layer
    tf.keras.layers.Dense(256, activation='relu'),  # First hidden layer
    tf.keras.layers.Dense(128, activation='relu'),  # Second hidden layer
    tf.keras.layers.Dense(10, activation='softmax')  # Output layer for 10 classes
```

```
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
print("Training 3-layer neural network...")
history = model.fit(X_train[:20000], y_train[:20000], epochs=10,
                    validation_data=(X_test[:5000], y_test[:5000]),
                    batch_size=64)
print("Neural network training complete.")

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=2)
print(f"Neural Network Test Accuracy: {test_accuracy:.4f}")

# Predict on the test set
y_pred_nn = np.argmax(model.predict(X_test), axis=1)

# Classification report for Neural Network
print("\nClassification Report for Neural Network:")
print(classification_report(y_test, y_pred_nn))
```

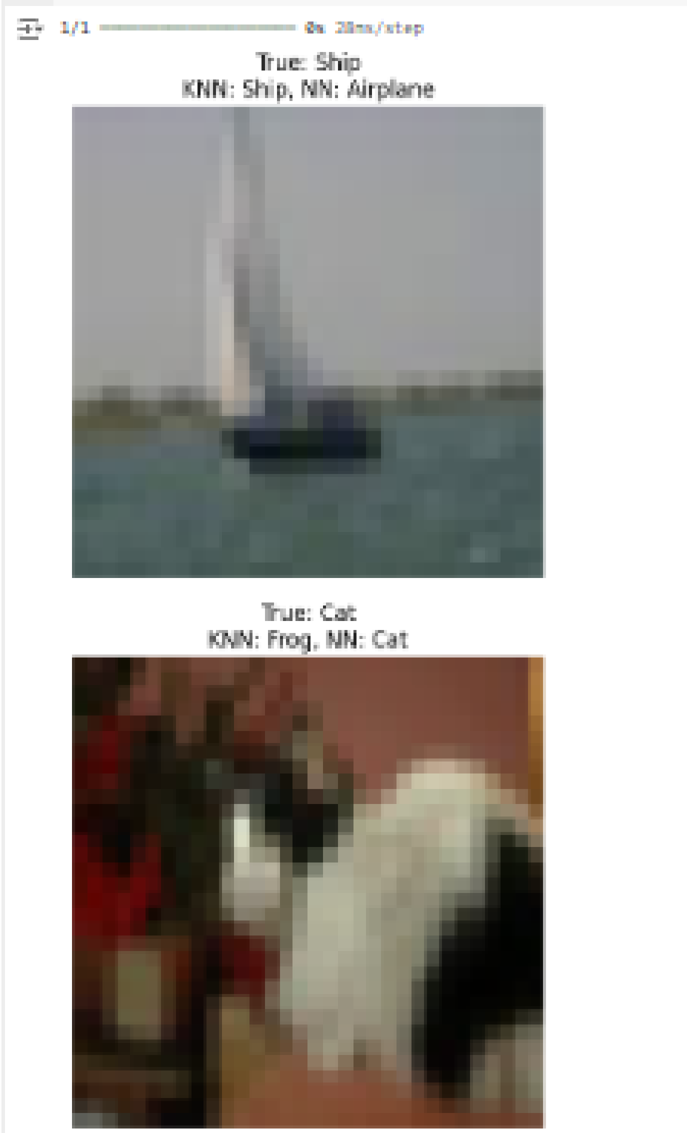


Figure 7: Prediction result

Result

Successfully implemented SVM using two machine learning models, a K-Nearest Neighbors (KNN) classifier and a 3-layer neural network, to classify images from the CIFAR-10 dataset.

Experiment 10
Artificial Neural Network with Backpropagation

Aim

To build an Artificial Neural Network (ANN) using the Backpropagation algorithm and test it with appropriate datasets.

Algorithm

1. Import necessary libraries
2. Initialize the network parameters, including weights, biases, and hyperparameters (learning rate, number of epochs, etc.).
3. Define the activation function and its derivative (e.g., sigmoid).
4. Perform forward propagation:
    - Compute the output of each layer using the activation function.
    - Calculate the final output.
5. Compute the error using a loss function (e.g., Mean Squared Error).
6. Perform backpropagation:
    - Calculate the gradient of the loss with respect to each parameter (weights and biases).
    - Update weights and biases using gradient descent.
7. Repeat steps 3 to 5 for the specified number of epochs.
8. Test the network with unseen data and analyze its performance.

Implementation

```python
# Python code to implement the Artificial Neural Network with Backpropagation
import numpy as np
import matplotlib.pyplot as plt

# Sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Dataset (XOR Problem)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

# Initialize hyperparameters
input_neurons = 2      # Number of input features
hidden_neurons = 4     # Number of hidden layer neurons
output_neurons = 1     # Number of output neurons
learning_rate = 0.5    # Learning rate
epochs = 10000         # Number of iterations

# Initialize weights and biases
np.random.seed(42)  # For reproducibility
weights_input_hidden = np.random.uniform(size=(input_neurons, hidden_neurons))
weights_hidden_output = np.random.uniform(size=(hidden_neurons, output_neurons))
bias_hidden = np.random.uniform(size=(1, hidden_neurons))
bias_output = np.random.uniform(size=(1, output_neurons))

# Store loss for plotting
loss_history = []

# Training the ANN
for epoch in range(epochs):
    # Forward propagation
    hidden_input = np.dot(X, weights_input_hidden) + bias_hidden
    hidden_output = sigmoid(hidden_input)
    final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
    final_output = sigmoid(final_input)

# Compute error
    error = y - final_output
    loss = np.mean(np.square(error))  # Mean squared error
    loss_history.append(loss)

    # Backpropagation
    d_output = error * sigmoid_derivative(final_output)
    error_hidden_layer = d_output.dot(weights_hidden_output.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_output)

    # Update weights and biases
    weights_hidden_output += hidden_output.T.dot(d_output) * learning_rate
    weights_input_hidden += X.T.dot(d_hidden_layer) * learning_rate
    bias_hidden += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate
    bias_output += np.sum(d_output, axis=0, keepdims=True) * learning_rate

    # Print loss every 1000 epochs
```

```
    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}")

# Testing the ANN
print("\nTesting the ANN:")
for x, target in zip(X, y):
    hidden_input = np.dot(x, weights_input_hidden) + bias_hidden
    hidden_output = sigmoid(hidden_input)
    final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
    final_output = sigmoid(final_input)
    print(f"Input: {x}, Predicted: {final_output.round(2)}, Target: {target}")

# Plotting the loss curve
plt.figure(figsize=(10, 5))
plt.plot(loss_history, label="Training Loss", color="blue")
plt.title("Loss Curve")
plt.xlabel("Epochs")
plt.ylabel("Mean Squared Error")
plt.legend()
plt.grid()
plt.show()

# Plotting the decision boundary
def plot_decision_boundary(X, y, model_predict):
    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, 0.1),
                         np.arange(y_min, y_max, 0.1))
    grid = np.c_[xx.ravel(), yy.ravel()]
    predictions = model_predict(grid)
    predictions = predictions.reshape(xx.shape)
    plt.contourf(xx, yy, predictions, alpha=0.8, cmap=plt.cm.Spectral)
    plt.scatter(X[:, 0], X[:, 1], c=y.flatten(), edgecolor='k', cmap=plt.cm.Spectral)
    plt.title("Decision Boundary")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()

# Define model prediction function
def model_predict(data):
    hidden_input = np.dot(data, weights_input_hidden) + bias_hidden
    hidden_output = sigmoid(hidden_input)
    final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
    return (sigmoid(final_input) > 0.5).astype(int)

# Plot the decision boundary
plot_decision_boundary(X, y, model_predict)

#output
Epoch 0, Loss: 0.3855
Epoch 1000, Loss: 0.0168
Epoch 2000, Loss: 0.0025
Epoch 3000, Loss: 0.0012
Epoch 4000, Loss: 0.0008
Epoch 5000, Loss: 0.0006
Epoch 6000, Loss: 0.0004
Epoch 7000, Loss: 0.0004
Epoch 8000, Loss: 0.0003
Epoch 9000, Loss: 0.0003

Testing the ANN:
```

```
Input: [0 0], Predicted: [[0.02]], Target: [0]
Input: [0 1], Predicted: [[0.98]], Target: [1]
Input: [1 0], Predicted: [[0.99]], Target: [1]
Input: [1 1], Predicted: [[0.01]], Target: [0]
```

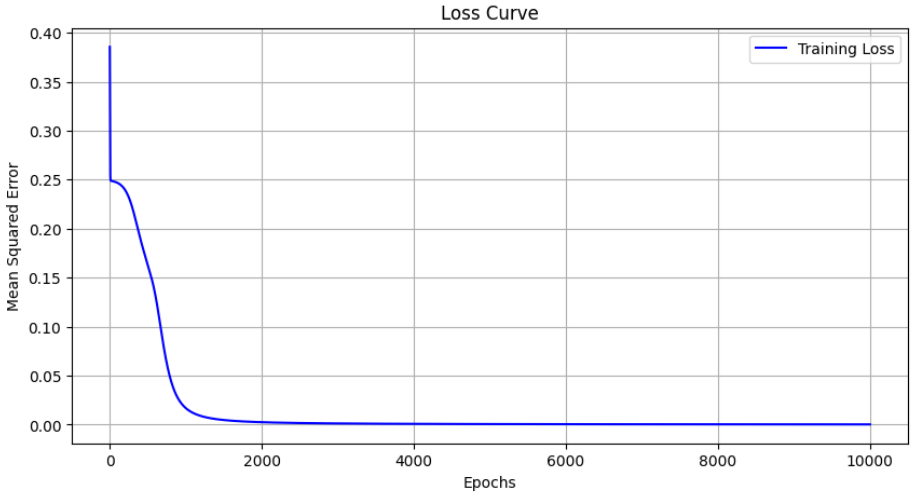


Figure 8: Loss Curve

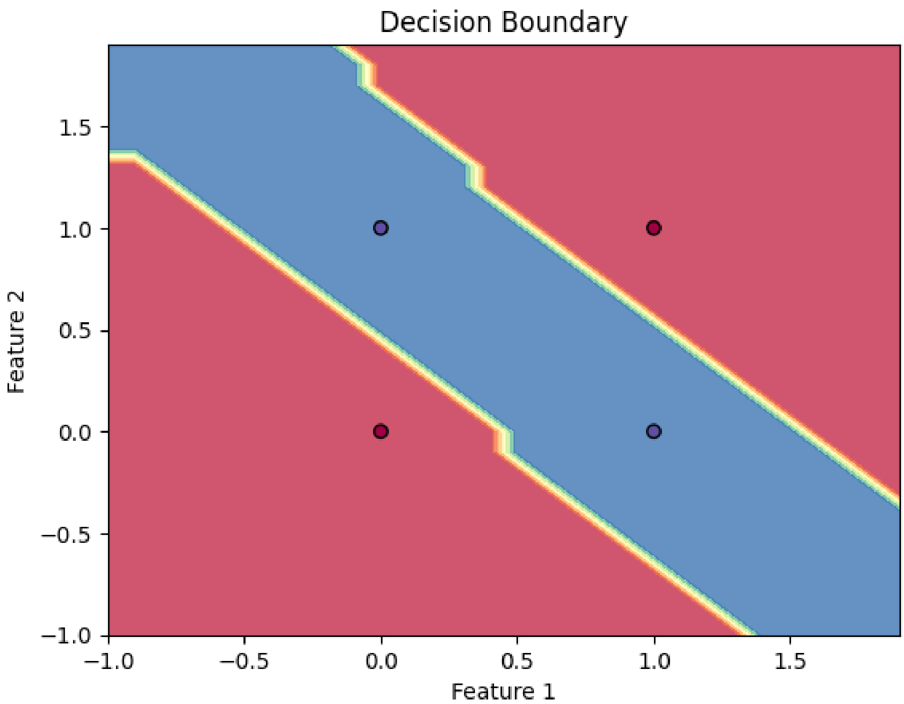


Figure 9: Loss Curve

Result

The Artificial Neural Network was successfully implemented using the Backpropagation algorithm. The network was trained on the XOR dataset and achieved accurate predictions.