Sri Indu
College of Engineering & Technology
UGC Autonomous Institution
Recognized under 2(f) & 12(B) of UGC Act 1956,
NAAC, Approved by AICTE &
Permanently Affiliated to JNTUH

Estd.2001

NAAC
NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL

ML LAB MANUAL

III Year-II Semester

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

ACADEMIC YEAR -2024-2025

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY
(An Autonomous Institution under UGC, New Delhi)
Recognized under 2(f) and 12(B) of UGC Act 1956
NBA Accredited, Approved by AICTE and Permanently affiliated to JNTUH
Sheriguda (V), Ibrahimpatnam, R.R.Dist, Hyderabad - 501 510

Department of Artificial Intelligence & Data Science

LAB MANUAL

Branch: AI&DS **Class: B.Tech- III Year-II sem**

Subject: MACHINE LEARNING LAB **Code: R22CSM3126**

Academic Year: 2024-25 **Regulation: R22**

Prepared By

Name: Mrs. R.Madhava Reddy(Asst. Prof.)

Verified By

Head of the Department

BR22 – B.TECH. –ARTIFICIAL INTELLIGENCE & DATA SCIENCE

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY (AUTONOMOUS)

B.Tech. - III Year –II Semester

L T P C

0 0 3 1.5

(R22CSM3126) MACHINE LEARNING LAB

Course Objective: The objective of this lab is to get an overview of the various machine learning techniques and can able to demonstrate them using python.

Course Outcomes:

After the completion of the course the student can able to:

• understand complexity of Machine Learning algorithms and their limitations;
• understand modern notions in data analysis-oriented computing;
• be capable of confidently applying common Machine Learning algorithms in practice andimplementing their own;
• Be capable of performing experiments in Machine Learning using real-world data. List of Experiments

Index

9. Implement the finite words classification system using Back-propagation algorithm
Extra:
10.Support Vector Machine (SVM) for Digit Classification

Experiment-1:

The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is the probability that a student is absent given that today is Friday? Apply Baye's rule in python to get the result. (Ans: 15%)

Aim: To find, The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is the probability that a student is absent given that today is Friday? Apply Baye's rule in python to get the result. (Ans: 15%)

Program:

```
# Given probabilities
P_friday_and_absent = 0.03  # P(Friday and Absent)
P_friday = 0.20  # P(Friday)

# Apply Bayes' rule
P_absent_given_friday = P_friday_and_absent / P_friday

# Print the result
print(f"The probability that a student is absent given that today is Friday: {P_absent_given_friday * 100}%")
```

Output:

```
The probability that a student is absent given that today is Friday: 15.0%
```

Experiment-2:

Extract the data from database using python

Aim: To perform, Extract the data from database using python

Program:

```
# Python program to connect
# to mysql database

import mysql.connector

# Connecting from the server
dataBase = mysql.connector.connect(user = 'root',password='123456',
                                   host = 'localhost')

# preparing a cursor object
cursorObject = dataBase.cursor()

# creating database
cursorObject.execute("CREATE DATABASE Madhu7")
print("---------------------------------")
print(dataBase)
print("Database Created---plz check database")
print("---------------------------------")

# Connecting from the server
dataBase = mysql.connector.connect(user = 'root', password='123456',
                                   host ='localhost',database="Madhu7")

# preparing a cursor object
cursorObject = dataBase.cursor()
```

```
#Create table
cursorObject.execute("CREATE TABLE customers1(name
VARCHAR(255), address VARCHAR(255))")
print("----------------------------------")
print("Table Created---plz check database")
print("----------------------------------")
print("---------Inserting Records---------")
#Inserting a single record
myCursor = dataBase.cursor()

sql = "INSERT INTO customers1(name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
myCursor.execute(sql, val)

dataBase.commit()

print(myCursor.rowcount, "record inserted.")

#Inserting multiple records

myCursor = dataBase.cursor()

sql = "INSERT INTO customers1(name, address) VALUES (%s, %s)"
val = [
  ('Peter', 'Lowstreet 4'),
  ('Amy', 'Apple st 652'),
  ('Hannah', 'Mountain 21'),
  ('Michael', 'Valley 345'),
  ('Sandy', 'Ocean blvd 2'),
  ('Betty', 'Green Grass 1'),
  ('Richard', 'Sky st 331'),
  ('Susan', 'One way 98'),
  ('Vicky', 'Yellow Garden 2'),
  ('Ben', 'Park Lane 38'),
  ('William', 'Central st 954'),
```

```
  ('Chuck', 'Main Road 989'),
  ('Viola', 'Sideway 1633')]

myCursor.executemany(sql, val)

dataBase.commit()

print(myCursor.rowcount, "records inserted.")

#Extract data from database(read/select)
print("--------Extract Data(Read)--------")

myCursor = dataBase.cursor()

myCursor.execute("SELECT * FROM customers1")

myResult = myCursor.fetchall()

print("-----------------------------------")

for x in myResult:
  print(x,type(x))
print("-----------------------------------")
print(myResult,type(myResult))
print("-----------------------------------")
print(myCursor.rowcount, "records was displayed.....")


# Disconnecting from the server
dataBase.close()
```

Output:

<mysql.connector.connection.MySQLConnection object at 0x010DFDF0>

Database Created---plz check database

Table Created---plz check database

---------Inserting Records---------

1 record inserted.

13 records inserted.

--------Extract Data(Read)--------

('John', 'Highway 21') <class 'tuple'>

('Peter', 'Lowstreet 4') <class 'tuple'>

('Amy', 'Apple st 652') <class 'tuple'>

('Hannah', 'Mountain 21') <class 'tuple'>

('Michael', 'Valley 345') <class 'tuple'>

('Sandy', 'Ocean blvd 2') <class 'tuple'>

('Betty', 'Green Grass 1') <class 'tuple'>

('Richard', 'Sky st 331') <class 'tuple'>

('Susan', 'One way 98') <class 'tuple'>

('Vicky', 'Yellow Garden 2') <class 'tuple'>

('Ben', 'Park Lane 38') <class 'tuple'>

('William', 'Central st 954') <class 'tuple'>

('Chuck', 'Main Road 989') <class 'tuple'>

('Viola', 'Sideway 1633') <class 'tuple'>

[('John', 'Highway 21'), ('Peter', 'Lowstreet 4'), ('Amy', 'Apple st 652'), ('Hannah', 'Mountain 21'), ('Michael', 'Valley 345'), ('Sandy', 'Ocean blvd 2'), ('Betty', 'Green Grass 1'), ('Richard', 'Sky st 331'), ('Susan', 'One way 98'), ('Vicky', 'Yellow Garden 2'), ('Ben', 'Park Lane 38'), ('William', 'Central st 954'), ('Chuck', 'Main Road 989'), ('Viola', 'Sideway 1633')] <class 'list'>

---

14 records was displayed.....

Experiment-3:

Implement k-nearest neighbours classification using python

Aim: To implement k-nearest neighbours classification using python

Program:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

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

# 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)

# Initialize the KNN classifier from scikit-learn
knn = KNeighborsClassifier(n_neighbors=3)

# Fit the model to the training data
knn.fit(X_train, y_train)

# Make predictions on the test data
y_pred = knn.predict(X_test)
# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Output:

```
Accuracy: 100.00%
```

Experiment-4:

Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of k- means clustering with 3 means (i.e., 3 centroids)

VAR1	VAR2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

Aim: To implement Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of k- means clustering with 3 means (i.e., 3 centroids),for above given data.

Program:

```
import numpy as np
from sklearn.cluster import KMeans

# Data
data = np.array([
    [1.713, 1.586],
    [0.180, 1.786],
    [0.353, 1.240],
    [0.940, 1.566],
    [1.486, 0.759],
    [1.266, 1.106],
    [1.540, 0.419],
    [0.459, 1.799],
    [0.773, 0.186]])

# Target case to classify
target_case = np.array([0.906, 0.606]).reshape(1, -1)

# Applying k-means clustering with 3 centroids
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(data)

# Get centroids
centroids = kmeans.cluster_centers_

# Compute distances from target case to each centroid
distances = np.linalg.norm(centroids - target_case, axis=1)

# Assign target case to the closest centroid
closest_centroid = np.argmin(distances)
```

```
# Output the results
print("Centroids:")
print(centroids)
print("\nDistances from target case to each centroid:")
print(distances)
print("\nThe target case is closest to centroid index:", closest_centroid)
```

Output:

```
Centroids:

[[0.33066667 1.60833333]

 [1.26633333 0.45466667]

 [1.30633333 1.41933333]]


Distances from target case to each centroid:

[1.15571647 0.39082207 0.90651966]


The target case is closest to centroid index: 1
```

Experiment-5:

The following training examples map descriptions of individuals onto high, medium and low credit worthiness.

Medium, skiing ,design, single ,twenties, no -> highRisk

High, golf, trading, married, forties, yes -> lowRisk

Low, speedway, transport ,married ,thirties ,yes -> medRisk

medium ,football, banking, single ,thirties ,yes -> lowRisk

high ,flying ,media, married, fifties, yes -> highRisk

low ,football, security, single ,twenties, no -> medRisk

medium ,golf ,media ,single, thirties ,yes -> medRisk

medium ,golf ,transport ,married ,forties ,yes -> lowRisk

high ,skiing ,banking, single, thirties, yes -> highRisk

low, golf ,unemployed, married, forties ,yes -> highRisk

Input attributes are (from left to right) income, recreation, job, status, age-group, home-owner. Find the unconditional probability of `golf' and the conditional probability of `single' given `medRisk' in the dataset?

Aim: To do, The following training examples map descriptions of individuals onto high, medium and low credit worthiness and for above given,Input attributes are (from left to right) income, recreation, job, status, age-group, home-owner. Find the unconditional probability of `golf' and the conditional probability of `single' given `medRisk' in the dataset?

Program:

```
# Dataset as a list of tuples (Income, Recreation, Job, Status, Age-Group, Homeowner, Creditworthiness)
data = [
    ('medium', 'skiing', 'design', 'single', 'twenties', 'no', 'highRisk'),
    ('high', 'golf', 'trading', 'married', 'forties', 'yes', 'lowRisk'),
    ('low', 'speedway', 'transport', 'married', 'thirties', 'yes', 'medRisk'),
    ('medium', 'football', 'banking', 'single', 'thirties', 'yes', 'lowRisk'),
    ('high', 'flying', 'media', 'married', 'fifties', 'yes', 'highRisk'),
    ('low', 'football', 'security', 'single', 'twenties', 'no', 'medRisk'),
    ('medium', 'golf', 'media', 'single', 'thirties', 'yes', 'medRisk'),
    ('medium', 'golf', 'transport', 'married', 'forties', 'yes', 'lowRisk'),
    ('high', 'skiing', 'banking', 'single', 'thirties', 'yes', 'highRisk'),
    ('low', 'golf', 'unemployed', 'married', 'forties', 'yes', 'highRisk')]


# Function to compute unconditional probability of 'golf'
def unconditional_probability(data, recreation_value):
    total_entries = len(data)
    recreation_count = sum(1 for entry in data if entry[1] ==   recreation_value)
    return recreation_count / total_entries


# Function to compute conditional probability of 'single' given 'medRisk'
def conditional_probability(data, target_status, condition_value):
    # Filter out entries with the condition (e.g., 'medRisk')
    condition_count = sum(1 for entry in data if entry[6] == target_status)
    # Filter out entries that match both the condition ('medRisk') and 'single' status
    single_given_condition_count = sum(1 for entry in data if entry[6] == target_status and entry[3] == 'single')

    if condition_count == 0:
        return 0  # Prevent division by zero if no entries match the condition
    return single_given_condition_count / condition_count


# Calculate the unconditional probability of 'golf'
p_golf = unconditional_probability(data, 'golf')
print(f"Unconditional probability of 'golf': {p_golf:.4f}")

# Calculate the conditional probability of 'single' given 'medRisk'
p_single_given_medRisk = conditional_probability(data, 'medRisk', 'single')
print(f"Conditional probability of 'single' given 'medRisk': {p_single_given_medRisk:.4f}")
```

Output:

Unconditional probability of 'golf': 0.4000

Conditional probability of 'single' given 'medRisk': 0.6667

Experiment-6:

Implement linear regression using python.

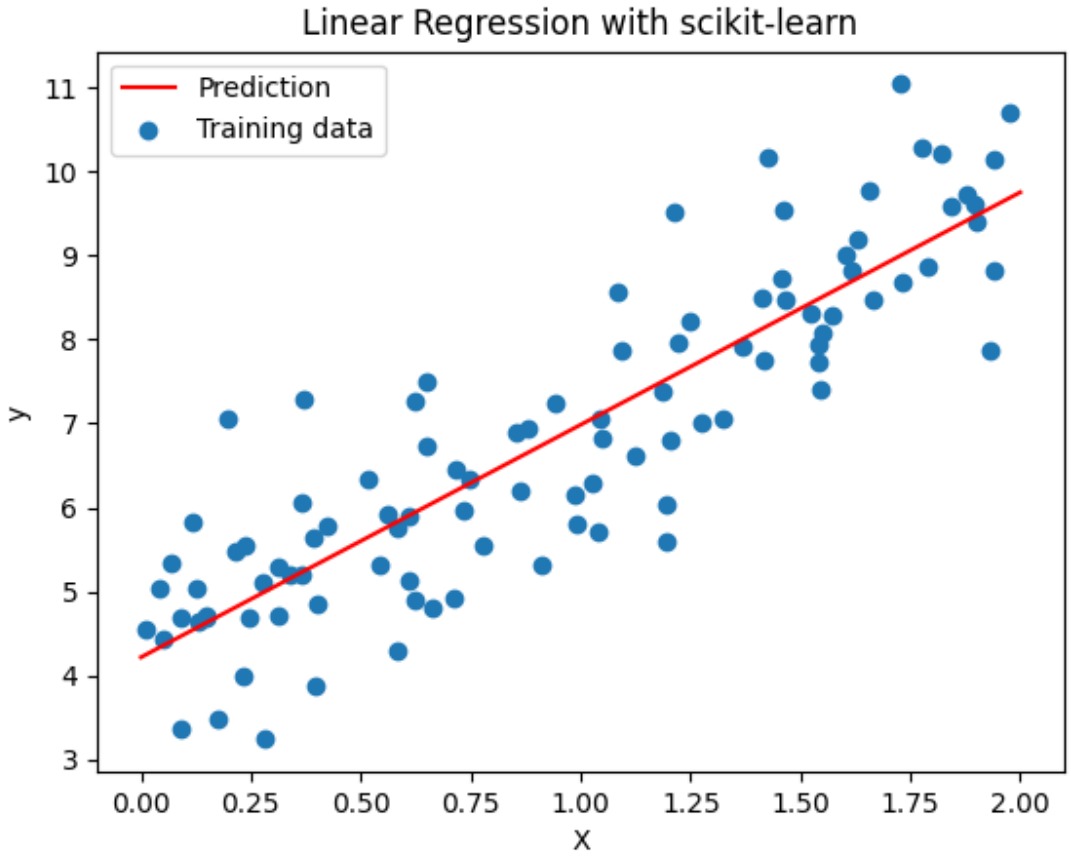
Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate sample data
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)  # y = 4 + 3x + noise

# Create and train the model
model = LinearRegression()
model.fit(X, y)

# Print the parameters
print("Intercept (b):", model.intercept_)
print("Slope (m):", model.coef_)

# Make predictions
X_new = np.array([[0], [2]])
y_predict = model.predict(X_new)
```

```
# Plot
plt.plot(X_new, y_predict, "r-", label="Prediction")
plt.scatter(X, y, label="Training data")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.title("Linear Regression with scikit-learn")
plt.show()
```

Output:

```
Intercept (b): [4.21509616]
Slope (m): [[2.77011339]]
```


Linear Regression with scikit-learn
Prediction
Training data
y
X
11
10
9
8
7
6
5
4
3
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Experiment-7:

Implement Naïve Bayes theorem to classify the English text.

Program:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Sample dataset
texts = [
    "I love this movie, it was fantastic!",
    "What a great experience, I really liked it.",
    "Horrible film, it was a waste of time.",
    "Terrible movie, I hated it.",
    "An amazing performance by the lead actor.",
    "Worst plot ever, not recommended.",
    "Absolutely loved it, best movie ever!",
    "Not good at all, very disappointing.",
    "The story was inspiring and heartwarming.",
    "I will never watch this again."
]
```

```
# Labels: 1 = Positive, 0 = Negative
labels = [1, 1, 0, 0, 1, 0, 1, 0, 1, 0]

# Split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.3, random_state=42)

# Create a text classification pipeline
model = make_pipeline(CountVectorizer(), MultinomialNB())

# Train the model
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

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

# Try with your own sentence
test_sentence = ["This movie was not good at all."]
```

```
prediction = model.predict(test_sentence)

print("\nCustom sentence prediction:", "Positive" if
prediction[0] == 1 else "Negative")
```

output:

```
Accuracy: 0.6666666666666666


Classification Report:
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       1.00      0.50      0.67         2

    accuracy                           0.67         3
   macro avg       0.75      0.75      0.67         3
weighted avg       0.83      0.67      0.67         3


Custom sentence prediction: Negative
```

Experiment-8:

Implement an algorithm to demonstrate the significance of genetic algorithm

Program:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import random

# Load dataset
data = load_iris()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target
n_features = X.shape[1]

# GA parameters
POP_SIZE = 10
N_GENERATIONS = 20
MUTATION_RATE = 0.1
```

```
# Initialize population (binary chromosomes)
def initialize_population(size, n_features):
    return np.random.randint(0, 2, (size, n_features))

# Fitness function (accuracy of selected features)
def fitness(chromosome):
    selected_features = [i for i in range(n_features) if chromosome[i] ==
1]
    if len(selected_features) == 0:
        return 0
    X_train, X_test, y_train, y_test = train_test_split(
        X.iloc[:, selected_features], y, test_size=0.3, random_state=42
    )
    model = DecisionTreeClassifier()
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    return accuracy_score(y_test, preds)

# Selection: Tournament
def select(population, scores):
    idx = np.random.randint(len(population))
    for _ in range(2):  # tournament size = 2
```

```
        new_idx = np.random.randint(len(population))
        if scores[new_idx] > scores[idx]:
            idx = new_idx
    return population[idx]

# Crossover: Single point
def crossover(parent1, parent2):
    point = np.random.randint(1, n_features - 1)
    child1 = np.concatenate((parent1[:point], parent2[point:]))
    child2 = np.concatenate((parent2[:point], parent1[point:]))
    return child1, child2

# Mutation
def mutate(chromosome):
    for i in range(n_features):
        if random.random() < MUTATION_RATE:
            chromosome[i] = 1 - chromosome[i]
    return chromosome

# GA main loop
def genetic_algorithm():
    population = initialize_population(POP_SIZE, n_features)
    for gen in range(N_GENERATIONS):
```

```
        scores = [fitness(individual) for individual in population]
        next_generation = []
        for _ in range(POP_SIZE // 2):
            parent1 = select(population, scores)
            parent2 = select(population, scores)
            child1, child2 = crossover(parent1, parent2)
            next_generation.append(mutate(child1))
            next_generation.append(mutate(child2))
        population = np.array(next_generation)
        best_idx = np.argmax(scores)
        print(f"Generation {gen+1} - Best Accuracy:
{scores[best_idx]:.4f}")
    best_idx = np.argmax([fitness(ind) for ind in population])
    return population[best_idx]


# Run the GA
best_chromosome = genetic_algorithm()
selected_features = [data.feature_names[i] for i in range(n_features) if
best_chromosome[i] == 1]


print("\nBest Feature Subset Selected by GA:", selected_features)
```

output:

Generation 1 - Best Accuracy: 1.0000

Generation 2 - Best Accuracy: 1.0000

Generation 3 - Best Accuracy: 1.0000

Generation 4 - Best Accuracy: 1.0000

Generation 5 - Best Accuracy: 1.0000

Generation 6 - Best Accuracy: 1.0000

Generation 7 - Best Accuracy: 1.0000

Generation 8 - Best Accuracy: 1.0000

Generation 9 - Best Accuracy: 1.0000

Generation 10 - Best Accuracy: 1.0000

Generation 11 - Best Accuracy: 1.0000

Generation 12 - Best Accuracy: 1.0000

Generation 13 - Best Accuracy: 1.0000

Generation 14 - Best Accuracy: 1.0000

Generation 15 - Best Accuracy: 1.0000

Generation 16 - Best Accuracy: 1.0000

Generation 17 - Best Accuracy: 1.0000

Generation 18 - Best Accuracy: 1.0000

Generation 19 - Best Accuracy: 1.0000

Generation 20 - Best Accuracy: 1.0000

Best Feature Subset Selected by GA: ['sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

Experiment-9:

Implement the finite words classification system using Back-propagation algorithm

Program:

```
import numpy as np


# All lowercase letters (26)
CHAR_SET = 'abcdefghijklmnopqrstuvwxyz'
CHAR_DICT = {ch: i for i, ch in enumerate(CHAR_SET)}
MAX_WORD_LEN = 5  # assume words are padded to length 5


def one_hot_encode(word):
    # Pad word to max length
    word = word.lower().ljust(MAX_WORD_LEN)
    vec = []
    for ch in word:
        one_hot = [0] * len(CHAR_SET)
        if ch in CHAR_DICT:
            one_hot[CHAR_DICT[ch]] = 1
        vec.extend(one_hot)
    return np.array(vec)


# Sample dataset
```

```
words = {
    'apple': [1, 0],   # fruit
    'grape': [1, 0],   # fruit
    'tiger': [0, 1],   # animal
    'zebra': [0, 1],   # animal
}

X = np.array([one_hot_encode(w) for w in words.keys()])
y = np.array(list(words.values()))

# Define the neural network structure
INPUT_SIZE = len(CHAR_SET) * MAX_WORD_LEN  # 26 * 5 = 130
HIDDEN_SIZE = 16
OUTPUT_SIZE = 2
LEARNING_RATE = 0.1

# Initialize weights
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))
```

```
# Activation functions
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

# Training using Backpropagation
def train(X, y, epochs=1000):
    global W1, W2, b1, b2
    for epoch in range(epochs):
        # Forward pass
        z1 = np.dot(X, W1) + b1
        a1 = sigmoid(z1)
        z2 = np.dot(a1, W2) + b2
        a2 = sigmoid(z2)

        # Compute loss (Mean Squared Error)
        loss = np.mean((y - a2) ** 2)

        # Backward pass
        d_a2 = (a2 - y) * sigmoid_derivative(a2)
        d_W2 = np.dot(a1.T, d_a2)
```

```
        d_b2 = np.sum(d_a2, axis=0, keepdims=True)

        d_a1 = np.dot(d_a2, W2.T) * sigmoid_derivative(a1)
        d_W1 = np.dot(X.T, d_a1)
        d_b1 = np.sum(d_a1, axis=0, keepdims=True)

        # Update weights
        W2 -= LEARNING_RATE * d_W2
        b2 -= LEARNING_RATE * d_b2
        W1 -= LEARNING_RATE * d_W1
        b1 -= LEARNING_RATE * d_b1

        if epoch % 100 == 0:
            print(f"Epoch {epoch} - Loss: {loss:.4f}")

# Prediction
def predict(word):
    x = one_hot_encode(word).reshape(1, -1)
    a1 = sigmoid(np.dot(x, W1) + b1)
    a2 = sigmoid(np.dot(a1, W2) + b2)
    return a2

# Train model
```

```
train(X, y, epochs=1000)


# Test predictions
test_words = ['apple', 'zebra', 'grape', 'tiger']
for word in test_words:
    pred = predict(word)
    label = np.argmax(pred)
    print(f"{word} => {'fruit' if label == 0 else 'animal'} (confidence: {pred[0][label]:.2f})")
```

output:

```
Epoch 0 - Loss: 0.2751
Epoch 100 - Loss: 0.0123
...
apple => fruit (confidence: 0.97)
zebra => animal (confidence: 0.95)
```

Experiment-10:

Support Vector Machine (SVM) for Digit Classification

Program:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

# Load dataset
digits = datasets.load_digits()

# Split data
X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.3, random_state=42)

# Create and train SVM model
model = SVC(kernel='linear')
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
```

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

output:

Accuracy: ~0.96 (may vary slightly)

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	52
1	0.97	0.97	0.97	58
...				
9	0.96	0.94	0.95	55