



SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203.



**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**



**SRM
Regulation 2023
Lab Manual**

DS3165 MACHINE LEARNING LABORATORY

**M.E. CSE
(I Semester)**

Prepared By

Ms. S.Balaswathy, Assistant Professor (O.G) /CSE

DS3165	MACHINE LEARNING LABORATORY	L T P C
		0 0 3 1.5

OBJECTIVES:

- To Make use of Data sets in implementing the machine learning algorithms
- To implement the machine learning concepts and algorithms in any suitable Language of choice.
- To understand the reinforcement concept
- To evaluate the classification and clustering algorithms
- Learn the usage of python in implementing machine Learning algorithms

LIST OF EXPERIMENTS:

1. Study and usage of python tool.
2. Implement a classifier.
3. Develop a predictive model.
4. Implement a decision tree algorithm.
5. Implement back propagation algorithm.
6. Implement similarity based clustering algorithm.
7. Implement clustering algorithm for any data set.
8. Apply reinforcement learning.
9. Implement and demonstrate the Candidate-Elimination algorithm
10. Implement k-Nearest Neighbour algorithm
11. Implement the non-parametric Locally Weighted Regression algorithm

TOTAL: 45 PERIODS

OUTCOMES:

At the end of the course, the student should be able to:

- Apply various classification and clustering techniques for problems using tools like Python.
- Implement solutions for various prediction problems using tools.
- Design and development of game and traffic control system using reinforcement learning.
- Implement the machine learning algorithms using python.
- Design and develop solutions for the machine learning algorithms using various datasets.

SOFTWARE: Python 3 interpreter for Windows/Linux, Open Source tools

CO – PO – PSO Mapping

CO	PO												PSO			
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4
CO 1	2	2	2	1	-	-	-	-	-	2	3	3	3	2	-	-
CO 2	2	1	1	3	2	-	-	-	3	2	3	2	3	-	-	-
CO 3	2	2	1	1	2	-	-	-	-	-	-	-	2	3	-	3
CO 4	2	2	3	3	2	-	-	-	-	2	3	-	-	2	-	2
CO 5	2	2	3	1	2	-	-	-	3	-	-	-	2	-	-	2
Avg	2.0	1.8	2.0	1.4	2.0	-	-	-	3.0	2.0	3.0	2.5	2.5	2.3	-	2.3

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

1. **PEO1:** To afford the necessary background in the field of Artificial Intelligence & Data Science to deal with Engineering problems to excel as engineering professionals in industries.
2. **PEO2:** To improve the qualities like creativity, leadership, team work and skill thus contributing towards the growth and development of society.
3. **PEO3:** To develop ability among students towards innovation and entrepreneurship that caters to the need of Industry and society.
4. **PEO4:** To inculcate an attitude for life- long learning process through the use of Artificial Intelligence and Data Science sources.
5. **PEO5:** To prepare them to be innovative and ethical leaders, both in their chosen profession and in other activities

PROGRAM OUTCOMES (POs) ENGINEERING GRADUATES WILL BE ABLE TO:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of

mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OBJECTIVES (PSOs)

PSO1: Design and develop secured database applications with data analytical approaches of data

preprocessing, optimization, visualization techniques and maintenance using state of the art methodologies based on ethical values.

PSO2: Design and develop intelligent systems using computational principles, methods and systems for extracting knowledge from data to solve real time problems using advanced technologies and tools.

PSO3: Design, plan and setting up the network that is helpful for contemporary business environments using latest hardware components.

PSO4: Planning and defining test activities by preparing test cases that can predict and correct errors ensuring a socially transformed product catering all technological needs.

COURSE OUTCOMES:

Course Name: DS3165- MACHINE LEARNING LABORATORY

Year of study: 2025 –2026

DS3165.1	Apply suitable algorithms for selecting the appropriate features for analysis.
DS3165.2	Implement supervised machine learning algorithms on standard datasets and evaluate the performance
DS3165.3	Apply unsupervised machine learning algorithms on standard datasets and evaluate the performance.
DS3165.4	Build the graph based learning models for standard data sets
DS3165.5	Assess and compare the performance of different ML algorithms and select the suitable one based on the application.

CO-PO Matrix:

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
DS3165.1	2	2	2	1	-	-	-	-	-	2	3	3
DS3165.2	2	1	1	3	2	-	-	-	3	2	3	2
DS3165.3	2	2	1	1	2	-	-	-	-	-	-	-
DS3165.4	2	2	3	3	2	-	-	-	-	2	3	-
DS3165.5	2	2	3	1	2	-	-	-	3	-	-	-

Justification:

Course Outcome	PO	Value	Justification
DS3165.1	PO1	2	Applying the fundamental Engineering Knowledge to select suitable algorithm for analysis.
	PO2	2	Understanding and analyzing the problems associated in selecting appropriate feature for analysis using suitable algorithm.
	PO3	2	Able to develop solutions using suitable algorithm for problems identified.
	PO4	1	Applying research based knowledge for analysis and interpretation of data and developing solution by selecting suitable algorithm for data analysis.
	PO10	2	Ability to comprehend and write report ,design documentation by selecting suitable machine learning algorithm.
	PO11	3	Ability to manage projects in a multidisciplinary environment by usage of correct algorithm for selecting features for analysis.

	PO12	3	Ability to learn and adapt to the dynamic changes in handling data using appropriate algorithm.
DS3165.2	PO1	2	Applying the fundamental concepts of engineering to implement supervised machine learning algorithm on dataset to evaluate performance.
	PO2	1	Ability to understand and analyze the problems associated with the dataset and perform analysis .
	PO3	1	Learning to design solution to complex engineering problems by implementing supervised machine learning algorithm.
	PO4	3	Applying research based knowledge for interpretation of data by implementing supervised machine learning algorithm.
	PO5	2	Learning to implement supervised machine learning algorithm by selecting and applying appropriate tools for prediction and modelling.
	PO9	3	Understanding the importance of data exchange in multidisciplinary setting and implementation of supervised machine learning algorithm for developing solution.
	PO10	2	Learning to communicate effectively in implementing supervised ML algorithm and write reports and design documentation.
	PO11	3	Learn to apply the engineering knowledge in implementing supervised ML algorithm for project management.
	PO12	2	Understanding the importance of life long learning in evaluating the performance of various supervised ML algorithms.
	PO1	2	Applying appropriate unsupervised ML algorithm for solving engineering problems with machine learning method.
	PO2	2	Identify and analyze the problems associated with data analysis with help of un supervised machine learning techniques
	PO3	1	Design and develop solution for complex engineering problems using unsupervised ML algorithm and evaluate their performance.

DS3165.3	PO4	1	Understanding the importance of data interpretation using research based knowledge and applying the suitable unsupervised ML algorithm for analysis.
	PO5	2	Learning and adapting with performance change of various machine learning methods with appropriate tools during data analysis
DS3165.4	PO1	2	Applying the fundamental knowledge for analysis and developing graph based learning models for different data sets.
	PO2	2	Identify the problem associated in data interpretation with help of graph based learning model for data analysis.
	PO3	3	Select and apply appropriate graph based learning model to develop solutions to complex engineering problems.
	PO4	3	Learning to analyze and interpret data using research based knowledge and through graph based learning model.
	PO5	2	Ability to make use of modern tools available to build graph based learning model for analysis of standard data set.
	PO10	2	Understanding the importance of effective communication among different groups during complex activities and in developing learning models.

	PO11	3	Adapting to the technological changes ,learning it and building graph based learning model for analysis and interpretation.
DS3165.5	PO1	2	Apply the fundamental knowledge to compare and select appropriate ML techniques to analyze Data
	PO2	2	Understand the problem by application of basic principles of engineering and select the appropriate ML algorithm after performance evaluation.
	PO3	3	Finding solutions by evaluating the different ML algorithm and choosing the appropriate technique to analyze Data in solving complex problems.
	PO4	1	Investigating the problem and choosing the appropriate ML algorithm by comparison of performance evaluation .
	PO5	2	Make use of modern tools for comparing and evaluating the various ML algorithms for analyzing the performance.
	PO9	3	Adapting to the advancement in tools usage and techniques used for analysis of data and interpretation.

CO-PSO Matrix:

CO	PSO1	PSO2	PSO3	PSO4
DS3165.1	3	2	-	-
DS3165.2	3	-	-	-
DS3165.3	2	3	-	3
DS3165.4	-	2	-	2
DS3165.5	2	-	-	2

Justification:

Course Outcome	Program Specific Outcome	Value	Justification
DS3165.1	PSO1	3	Design and develop suitable algorithm for analysis of secured database application with analytical approach.
	PSO2	2	Design and develop Intelligent systems for problem solving by applying suitable algorithm for feature selection.
DS3165.2	PSO1	3	Design and develop database applications using supervised machine learning models for performance evaluation.
DS3165.3	PSO1	2	Understand and develop database application to perform data analysis with unsupervised machine learning method for knowledge extraction.
	PSO2	3	Apply unsupervised machine learning model for development of intelligent systems to solve real time problems using tools and advanced technologies.

	PSO3	3	Learn to set up a network for contemporary business environment using unsupervised machine learning algorithm for data analysis.
DS3165.4	PSO2	2	Apply graph based learning model to develop intelligent system for extraction of knowledge and analysis.
	PSO4	2	Plan and use appropriate graph based learning models for developing contemporary business environments by using graphical data analysis and prepare test cases to predict results
DS3165.5	PSO1	2	Compare the performance of ML algorithm and choose an appropriate one to design and develop secure database application for the purpose of data analysis.
	PSO4	2	Plan, test and find the appropriate ML techniques to develop a product catering all technological needs.

CO-PO and PSO Mapping

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
DS3165	2.0	1.8	2.0	1.4	2.0	-	-	-	3.0	2.0	3.0	2.5	2.5	2.3	-	2.3

ASSESSMENT METHOD

MARK SPLIT UP	
AIM & PRE LAB VIVA QUESTIONS	20
OBSERVATION	30
CONDUCTION & EXECUTION	30
OUTPUT & RESULT	10
POST LAB VIVA QUESTIONS	10
TOTAL	100

TABLE OF CONTENTS

EXP NO.	NAME OF THE EXPERIMENT	PAGE NO.
1.	STUDY AND USAGE OF PYTHON TOOL.	12
2.	BINARY CLASSIFIER	17
3.	A PREDICTIVE MODEL	24
4.	A DECISION TREE ALGORITHM	28
5.	BACK PROPAGATION ALGORITHM	33
6.	SIMILARITY BASED CLUSTERING ALGORITHM	36
7.	CLUSTERING ALGORITHM	40
8.	REINFORCEMENT LEARNING	44
9.	CANDIDATE-ELIMINATION ALGORITHM	47
10.	K-NEAREST NEIGHBOUR ALGORITHM	
11	NON-PARAMETRIC LOCALLY WEIGHTED REGRESSION ALGORITHM	51

Ex. No: 1**STUDY AND USAGE OF PYTHON TOOL****AIM:**

To study and demonstrate the basic usage of Python tools, including fundamental data types, control structures, functions, and popular libraries like NumPy for numerical operations and Pandas for data manipulation.

DESCRIPTION:

Python is an interpreted, high-level programming language widely used in machine learning due to its simplicity and extensive ecosystem. This experiment introduces basic Python syntax, including lists, dictionaries, loops, and functions. We also explore Pandas for handling structured data in DataFrames, which are essential for data preprocessing in ML tasks. NumPy is used for efficient array operations, statistical computations, and matrix manipulations. Understanding these tools is foundational for implementing ML algorithms, as they allow for data loading, cleaning, and transformation. In machine learning workflows, Python's libraries enable seamless integration with advanced tools like scikit-learn. This experiment demonstrates simple operations to build familiarity.

ALGORITHM:

- Start the process.
- Create a Python list `my_list` with initial values `[1, 2, 3, 4, 5]`; print the original list.
- Append value 6 to `my_list` using `.append()` and print the updated list; remove value 3 using `.remove()` and print the list after removal.
- Create a dictionary `my_dict` with keys 'name', 'age', 'city' and corresponding values; print the dictionary, access 'name' directly via key, and 'age' safely using `.get()`.
- 5: Add a new key-value pair 'profession': 'Engineer' to `my_dict` and print the updated dictionary.
- Use a for loop with `range(5)` to iterate from 0 to 4 and print each index; use a while loop starting from `count = 5`, decrementing until greater than 0, and print each value.
- Define a function `add_numbers(a, b)` that returns the sum of two numbers; call it with arguments (5, 3) and (10, 20), printing each result.
- Create a Pandas DataFrame `data` with columns 'A', 'B', 'C' and 4 rows of sequential data; print the DataFrame and its descriptive statistics using `.describe()`.
- Add a new column 'D' to `data` as the sum of columns 'A' and 'B'; print the updated DataFrame.
- Create a NumPy array `arr` from list `[1, 2, 3, 4, 5]`; print the array, compute and print mean, sum, standard deviation, maximum, and minimum using `np.mean()`, `np.sum()`, `np.std()`, `np.max()`, `np.min()`.
- Reshape `arr` into a `1×5` array using `.reshape(1, 5)` and print it; compute and print the element-wise square of `arr` using `np.square()`.
- Terminate.

PROGRAM:

```
import numpy as np
import pandas as pd
```

```

my_list = [1, 2, 3, 4, 5]
print("Original List:", my_list)
my_list.append(6) # Append an element
print("Appended List:", my_list)
my_list.remove(3) # Remove an element
print("List after removal:", my_list)

my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}
print("\nDictionary:", my_dict)
print("Name:", my_dict['name'])
print("Age:", my_dict.get('age')) # Safe access with get()
my_dict['profession'] = 'Engineer' # Add new key-value pair
print("Updated Dictionary:", my_dict)
print("\nLoop through numbers 0 to 4:")
for i in range(5):
    print(i)

print("\nWhile loop counting down from 5:")
count = 5
while count > 0:
    print(count)
    count -= 1

def add_numbers(a, b):
    """Function to add two numbers and return the sum."""
    return a + b

print("\nSum of 5 and 3:", add_numbers(5, 3))
print("Sum of 10 and 20:", add_numbers(10, 20))

data = pd.DataFrame({
    'A': [1, 2, 3, 4],
    'B': [5, 6, 7, 8],
    'C': [9, 10, 11, 12]
})
print("\nDataFrame:\n", data)

print("\nDescriptive Statistics:\n", data.describe())

data['D'] = data['A'] + data['B']
print("\nDataFrame with new column D:\n", data)

```

```

arr = np.array([1, 2, 3, 4, 5])
print("\nNumPy Array:", arr)

print("Mean:", np.mean(arr))
print("Sum:", np.sum(arr))
print("Standard Deviation:", np.std(arr))
print("Maximum:", np.max(arr))
print("Minimum:", np.min(arr))

arr_reshaped = arr.reshape(1, 5)
print("\nReshaped Array (1x5):\n", arr_reshaped)
print("Array squared:\n", np.square(arr))

```

OUTPUT:

```

Original List: [1, 2, 3, 4, 5]
Appended List: [1, 2, 3, 4, 5, 6]
List after removal: [1, 2, 4, 5, 6]
Dictionary: {'name': 'Alice', 'age': 30, 'city': 'New York'}
Name: Alice
Age: 30
Updated Dictionary: {'name': 'Alice', 'age': 30, 'city': 'New York', 'profession': 'Engineer'}
Loop through numbers 0 to 4:
0
1
2
3
4

While loop counting down from 5:
5
4
3
2
1

Sum of 5 and 3: 8
Sum of 10 and 20: 30

```

DataFrame:

	A	B	C
0	1	5	9
1	2	6	10

2	3	7	11
3	4	8	12

Descriptive Statistics:

	A	B	C
count	4.000000	4.000000	4.000000
mean	2.500000	6.500000	10.500000
std	1.290994	1.290994	1.290994
min	1.000000	5.000000	9.000000
25%	1.750000	5.750000	9.750000
50%	2.500000	6.500000	10.500000
75%	3.250000	7.250000	11.250000
max	4.000000	8.000000	12.000000

DataFrame with new column D:

	A	B	C	D
0	1	5	9	6
1	2	6	10	8
2	3	7	11	10
3	4	8	12	12

NumPy Array: [1 2 3 4 5]

Mean: 3.0

Sum: 15

Standard Deviation: 1.4142135623730951

Maximum: 5

Minimum: 1

Reshaped Array (1x5):

[[1 2 3 4 5]]

Array squared:

[1 4 9 16 25]

VIVA QUESTIONS:

1. What is the difference between lists and tuples in Python?
2. How does NumPy differ from standard Python lists?
3. Explain the purpose of Pandas DataFrame.
4. What is a lambda function in Python?
5. How do you handle exceptions in Python?

RESULT:

Thus, the study and usage of Python tools were demonstrated successfully, covering data types, libraries, and basic operations.

Ex. No: 2**BINARY CLASSIFIER****AIM:**

To implement a binary classification model using Logistic Regression in Python.

DESCRIPTION:

Classification is a supervised learning task where the model predicts categorical labels. Logistic Regression is a popular algorithm for binary classification, using a sigmoid function to output probabilities. This experiment uses the Iris dataset, converting it to binary (Setosa vs. Non-Setosa). We split data, train the model with hyperparameters like max_iter and solver, predict on test data, and evaluate using accuracy, classification report, and confusion matrix. This helps understand model performance in terms of precision, recall, and error types. In real-world applications, such classifiers are used in spam detection, medical diagnosis, etc.

ALGORITHM:

- Start the process.
- Load the Iris dataset using `sklearn.datasets.load_iris()`; extract features X (4 measurements per flower) and convert labels y into binary form: 1 for Setosa (original class 0), 0 for others.
- Print the shape of features and labels, and display the first 5 samples of both X and y to verify data structure.
- Split the dataset into training (80%) and testing (20%) sets using `train_test_split` with `stratify=y` to maintain class distribution and `random_state=42` for reproducibility.
- Initialize a Logistic Regression model with L2 regularization (`penalty='l2'`), inverse regularization strength `C=1.0`, lbfgs solver, maximum 200 iterations, and `random_state=42`.
- Train the model on X_train and y_train using `.fit()`; print confirmation along with learned coefficients (`model.coef_`) and intercept (`model.intercept_`).
- Generate predictions (preds) on the test set using `.predict()` and predicted probabilities for the positive class (Setosa) using `.predict_proba()[:, 1]`.
- Evaluate model performance: compute accuracy using `accuracy_score`, generate detailed classification report (precision, recall, F1-score) using `classification_report`, and compute confusion matrix using `confusion_matrix`.
- Print accuracy, sample predictions and probabilities, classification report, and confusion matrix to assess binary classification performance.
- Terminate.

PROGRAM:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import numpy as np

iris = load_iris()
X = iris.data # Features: sepal length, sepal width, petal length, petal width
```

```

y = (iris.target == 0).astype(int) # Binary labels: 1 for Setosa (class 0), 0 for others

print("Feature shape:", X.shape)
print("Label shape:", y.shape)
print("Sample features:\n", X[:5])
print("Sample labels:", y[:5])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("\nTraining set size:", X_train.shape)
print("Testing set size:", X_test.shape)

model = LogisticRegression(
    penalty='l2',      # Regularization type
    C=1.0,             # Inverse of regularization strength
    solver='lbfgs',    # Optimization algorithm
    max_iter=200,      # Maximum iterations
    random_state=42,    # Seed for reproducibility
    multi_class='auto' # Auto-detect multi-class (though binary here)
)

model.fit(X_train, y_train)
print("\nModel trained successfully.")
print("Model coefficients:\n", model.coef_)
print("Model intercept:", model.intercept_)

preds = model.predict(X_test)
preds_proba = model.predict_proba(X_test)[:5, 1] # Probabilities for positive class

print("\nSample Predictions:", preds[:5])
print("Sample Probabilities:", preds_proba[:5])

accuracy = accuracy_score(y_test, preds)
print("\nAccuracy:", accuracy)

print("\nClassification Report:\n", classification_report(y_test, preds))

conf_mat = confusion_matrix(y_test, preds)
print("\nConfusion Matrix:\n", conf_mat)

```

OUTPUT:

Feature shape: (150, 4)

Label shape: (150,)

Sample features:

[[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]]

Sample labels: [1 1 1 1 1]

Training set size: (120, 4)

Testing set size: (30, 4)

Model trained successfully.

Model coefficients:

[[-0.41081689 1.4612492 -2.25800243 -0.95024623]]

Model intercept: [6.21399146]

Sample Predictions: [0 1 0 0 0]

Sample Probabilities: [0.00419217 0.97799922 0.00425229 0.00263888 0.00336325]

Accuracy: 1.0

Classification Report:

	precision	recall	f1-score	Support
0	1.00	1.00	1.00	20
1	1.00	1.00	1.00	10
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Confusion Matrix:

[[20 0]

[0 10]]

VIVA QUESTIONS:

1. What is the sigmoid function in Logistic Regression?

2. Explain the difference between L1 and L2 regularization.
3. How does the solver parameter affect model training?
4. What does the confusion matrix represent?
5. Why use stratified split in classification?

RESULT:

Thus, the binary classifier using Logistic Regression was implemented and evaluated successfully on the Iris dataset.

Ex. No: 3**A PREDICTIVE MODEL****AIM:**

To develop a predictive model using Linear Regression for continuous value prediction.

DESCRIPTION:

Predictive modeling involves forecasting continuous outcomes based on input features. Linear Regression assumes a linear relationship between variables, minimizing the sum of squared errors. This experiment uses the Diabetes dataset to predict disease progression. We load data, split it, train with options like `fit_intercept`, evaluate using MSE and R2 score, and inspect coefficients. This demonstrates regression's use in tasks like house price prediction or stock forecasting. Understanding coefficients helps interpret feature importance.

ALGORITHM:

- Start the process.
- Load the Diabetes dataset using `sklearn.datasets.load_diabetes()`; extract features X (10 physiological variables) and target y (quantitative measure of disease progression).
- Print feature names, shapes of X and y, and display the first 3 samples of features and targets to inspect data structure.
- Split the dataset into training (80%) and testing (20%) sets using `train_test_split` with `random_state=42` for reproducibility.
- Initialize a Linear Regression model with `fit_intercept=True` (include bias term), `copy_X=True` (copy input data), and `n_jobs=-1` (use all available CPU cores for parallel processing).
- Train the model on `X_train` and `y_train` using `.fit()`; print confirmation along with learned coefficients (`model.coef_`) and intercept (`model.intercept_`).
- Generate predictions (`preds`) on the test set using `.predict()`.
- Evaluate regression performance: compute Mean Squared Error (MSE) using `mean_squared_error`, and R² score (coefficient of determination) using `r2_score` to measure how well predictions approximate actual values.
- Compute residuals as `y_test - preds` and print sample predictions, actual values, residuals, MSE, and R² score to assess model accuracy and error distribution.
- Terminate.

PROGRAM:

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
```

```
diabetes = load_diabetes()
X = diabetes.data # Features: age, sex, bmi, etc.
```

```
y = diabetes.target # Target: disease progression
```

```
print("Feature names:", diabetes.feature_names)
```

```
print("Feature shape:", X.shape)
```

```
print("Target shape:", y.shape)
```

```
print("Sample features:\n", X[:3])
```

```
print("Sample targets:", y[:3])
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
print("\nTraining features shape:", X_train.shape)
```

```
print("Testing features shape:", X_test.shape)
```

```
model = LinearRegression(  
    fit_intercept=True, # Include intercept  
    copy_X=True,        # Copy data  
    n_jobs=-1           # Use all processors  
)
```

```
model.fit(X_train, y_train)
```

```
print("\nModel trained.")
```

```
print("Coefficients:\n", model.coef_)
```

```
print("Intercept:", model.intercept_)
```

```
preds = model.predict(X_test)
```

```
print("\nSample Predictions:", preds[:5])
```

```
print("Sample Actuals:", y_test[:5])
```

```
mse = mean_squared_error(y_test, preds)
```

```
r2 = r2_score(y_test, preds)
```

```
print("\nMean Squared Error:", mse)
```

```
print("R2 Score:", r2)
```

```
residuals = y_test - preds
```

```
print("\nSample Residuals:", residuals[:5])
```

OUTPUT:

```
Feature names: ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

Feature shape: (442, 10)

Target shape: (442,)

Sample features:

```
[[ 0.03807591  0.05068012  0.06169621  0.02187239 -0.0442235 -0.03482076
 -0.04340085 -0.00259226  0.01990749 -0.01764613]
 [-0.00188202 -0.04464164 -0.05147406 -0.02632753 -0.00844872 -0.01916334
  0.07441156 -0.03949338 -0.06833155 -0.09220405]
 [ 0.08529891  0.05068012  0.04445121 -0.00567016 -0.04559945 -0.03419447
 -0.03235593 -0.00259226  0.00286131 -0.02593034]]
```

Sample targets: [151. 75. 141.]

Training features shape: (353, 10)

Testing features shape: (89, 10)

Model trained.

Coefficients:

```
[ 37.90402135 -241.96436231  542.42875852  347.70384391 -931.48884588
 518.06227698 163.41998299 275.31790158 736.1988589  48.67065743]
```

Intercept: 151.34560453985995

Sample Predictions: [141.94891629 186.10196677 150.44948075 292.75977276 123.17545409]

Sample Actuals: [219. 70. 202. 230. 111.]

Mean Squared Error: 2900.1936284934804

R2 Score: 0.4526027629719196

Sample Residuals: [77.05108371 -116.10196677 51.55051925 -62.75977276 -12.17545409]

VIVA QUESTIONS:

1. What assumes Linear Regression about data?
2. Explain R2 score interpretation.
3. How does fit_intercept affect the model?
4. What is multicollinearity in regression?
5. Why use MSE as a loss function?

RESULT:

Thus, the predictive model using Linear Regression was developed and tested successfully on the Diabetes dataset.

Ex. No: 4**A DECISION TREE ALGORITHM****AIM:**

To implement a Decision Tree classifier for multi-class prediction.

DESCRIPTION:

Decision Trees are tree-structured models that split data based on feature values to make decisions. They are interpretable and handle non-linear relationships. This experiment uses the Iris dataset for species classification. We set parameters like `criterion='gini'` and `max_depth` to control tree growth, preventing overfitting. Evaluation includes accuracy and classification report. Decision Trees are foundational for ensemble methods like Random Forests.

ALGORITHM:

- Start the process.
- Load the Iris dataset using `sklearn.datasets.load_iris()`; extract features X (4 floral measurements) and multi-class target y (3 species: setosa, versicolor, virginica).
- Print class names (`iris.target_names`) and feature names (`iris.feature_names`) to understand the dataset structure.
- Split data into training (80%) and testing (20%) sets using `train_test_split` with `random_state=42` for reproducibility.
- Initialize a Decision Tree Classifier with `criterion='gini'` (impurity measure), `splitter='best'` (chooses optimal split), `max_depth=5` (limits tree depth), `min_samples_split=2`, and `random_state=42`.
- Train the model on `X_train` and `y_train` using `.fit()`, building a decision tree by recursively partitioning the feature space to minimize Gini impurity.
- Generate predictions (`preds`) on the test set using `.predict()`.
- Evaluate classification performance: compute accuracy using `accuracy_score`, generate detailed classification report (precision, recall, F1-score per class) using `classification_report`, and compute confusion matrix using `confusion_matrix` to show true vs predicted class counts.
- Visualize the trained decision tree using `tree.plot_tree()` with feature and class names, filled nodes (colored by majority class), and display the plot using `matplotlib.pyplot`.
- Print accuracy, classification report, confusion matrix, and display the decision tree structure for interpretability.
- Terminate.

PROGRAM:

```
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, classification_report, confusion_matrix
from sklearn import tree
```

```

import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data
y = iris.target

print("Classes:", iris.target_names)
print("Features:", iris.feature_names)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = DecisionTreeClassifier(
    criterion='gini',
    splitter='best',
    max_depth=5,
    min_samples_split=2,
    random_state=42
)

model.fit(X_train, y_train)

preds = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, preds))
print("\nClassification Report:\n", classification_report(y_test, preds))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, preds))

plt.figure(figsize=(12,8))
tree.plot_tree(model, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()

```

OUTPUT:

Classes: ['setosa' 'versicolor' 'virginica']

Features: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

Accuracy: 1.0

Classification Report:

	precision	recall	f1-score	Support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9

2	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Confusion Matrix:

[[10 0 0]

[0 9 0]

[0 0 11]]

VIVA QUESTIONS:

1. What is Gini impurity?
2. How does max_depth prevent overfitting?
3. Explain entropy in Decision Trees.
4. What is pruning?
5. Difference between classification and regression trees?

RESULT:

Thus, the Decision Tree algorithm was implemented successfully for Iris classification.

Ex. No: 5**BACK PROPAGATION ALGORITHM****AIM:**

To implement the backpropagation algorithm for a simple neural network.

DESCRIPTION:

Backpropagation is an algorithm for training neural networks by adjusting weights using gradient descent. It computes errors from output to input layers. This experiment uses a 2-layer network for XOR problem, which is non-linear. We use sigmoid activation, increased hidden neurons, and more epochs for better learning. Progress is printed every 1000 epochs.

ALGORITHM:

- Start the process.
- Define the sigmoid activation function $\text{sigmoid}(x) = 1 / (1 + \exp(-x))$ and its derivative $\text{sigmoid_deriv}(x) = x * (1 - x)$ for non-linear neuron activation and backpropagation.
- Prepare the XOR dataset: input X as 2×4 matrix with binary pairs $[[0,0], [0,1], [1,0], [1,1]]$ and target y as 4×1 matrix $[[0], [1], [1], [0]]$.
- Set random seed to 1 for reproducibility; initialize neural network architecture: 2 input neurons, 3 hidden neurons, 1 output neuron.
- Randomly initialize weight matrices: w1 (2×3) for input-to-hidden layer and w2 (3×1) for hidden-to-output layer, with uniform values in range [-1, 1].
- Set training parameters: epochs = 10000, learning_rate = 0.1.
- For each epoch from 0 to 9999:
 - Perform forward propagation: compute hidden layer output $\text{layer1} = \text{sigmoid}(X @ w1)$ and output layer $\text{layer2} = \text{sigmoid}(\text{layer1} @ w2)$.
 - Compute prediction error: $\text{error} = y - \text{layer2}$.
 - Perform backward propagation:
 - Compute output layer gradient: $\text{delta2} = \text{error} * \text{sigmoid_deriv}(\text{layer2})$.
 - Compute hidden layer gradient: $\text{delta1} = (\text{delta2} @ w2.T) * \text{sigmoid_deriv}(\text{layer1})$.
 - Update weights using gradient descent:
 - $w2 += \text{learning_rate} * \text{layer1.T} @ \text{delta2}$
 - $w1 += \text{learning_rate} * X.T @ \text{delta1}$
- Every 1000 epochs, print the current epoch and Mean Absolute Error ($\text{MAE} = \text{mean}(|\text{error}|)$).
- After training completes, print final predictions (layer2), final weight matrices w1 and w2 to show learned parameters.
- Terminate.

PROGRAM:

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_deriv(x):
    return x * (1 - x)
```

```

X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])

np.random.seed(1)

input_neurons = 2
hidden_neurons = 3
output_neurons = 1

w1 = np.random.uniform(-1, 1, (input_neurons, hidden_neurons))
w2 = np.random.uniform(-1, 1, (hidden_neurons, output_neurons))

epochs = 10000
learning_rate = 0.1

for epoch in range(epochs):
    layer1 = sigmoid(np.dot(X, w1))
    layer2 = sigmoid(np.dot(layer1, w2))

    error = y - layer2
    delta2 = error * sigmoid_deriv(layer2)
    delta1 = delta2.dot(w2.T) * sigmoid_deriv(layer1)

    w2 += learning_rate * layer1.T.dot(delta2)
    w1 += learning_rate * X.T.dot(delta1)

    if epoch % 1000 == 0:
        print(f'Epoch {epoch}, Mean Absolute Error: {np.mean(np.abs(error)):.6f}')

print("\nFinal Predictions:\n", layer2)
print("\nFinal Weights w1:\n", w1)
print("Final Weights w2:\n", w2)

```

OUTPUT:

```

Epoch 0, Mean Absolute Error: 0.499292
Epoch 1000, Mean Absolute Error: 0.494236
Epoch 2000, Mean Absolute Error: 0.441605
Epoch 3000, Mean Absolute Error: 0.382695
Epoch 4000, Mean Absolute Error: 0.341033
Epoch 5000, Mean Absolute Error: 0.291424
Epoch 6000, Mean Absolute Error: 0.252115
Epoch 7000, Mean Absolute Error: 0.228219
Epoch 8000, Mean Absolute Error: 0.212978
Epoch 9000, Mean Absolute Error: 0.202457

```

Final Predictions:

```

[[0.09082085]
 [0.91110857]
 [0.9121265 ]
 [0.51135469]]

```

Final Weights w1:

[[-2.26675548 -3.03038013 1.6866533]

[-2.28956242 -3.03290712 1.68392583]]

Final Weights w2:

[[5.10970501]

[6.109705]

[-3.93503494]]

VIVA QUESTIONS:

1. What is backpropagation?
2. Role of activation functions?
3. Why XOR can't be solved by single layer?
4. What is gradient descent?
5. How to avoid local minima?

RESULT:

Thus, the backpropagation algorithm was implemented successfully for the XOR problem.

Ex. No: 6**SIMILARITY BASED CLUSTERING ALGORITHM****AIM:**

To implement DBSCAN, a density-based clustering algorithm.

DESCRIPTION:

DBSCAN groups points based on density, handling noise and arbitrary shapes. It uses `eps` (radius) and `min_samples`. This experiment generates blob data, applies DBSCAN, and evaluates with silhouette and Davies-Bouldin scores. Useful for outlier detection in datasets like customer segmentation.

ALGORITHM:

- Start the process.
- Generate synthetic 2D clustered data using `make_blobs` with 500 samples, 4 centers, standard deviation 0.50, and `random_state=42` for reproducibility; assign features to `X` and true cluster labels to `true_labels`.
- Print the shape of the data (`X.shape`) and display the first 5 sample points to verify data generation.
- Initialize a DBSCAN clustering model with parameters:
 - `eps=0.5` (maximum distance between two points to be in the same neighborhood),
 - `min_samples=5` (minimum points to form a dense region/core point),
 - `metric='euclidean'` (distance measure),
 - `algorithm='auto'` (automatic selection of nearest neighbor search).
- Fit the DBSCAN model on `X` and simultaneously predict cluster labels using `.fit_predict(X)`; labels include cluster indices (≥ 0) and noise points (-1).
- Evaluate clustering quality:
 - Compute Silhouette Score using `silhouette_score(X, labels)` to measure how similar points are to their own cluster vs. others (higher is better).
 - Compute Davies-Bouldin Score using `davies_bouldin_score(X, labels)` to measure average similarity between clusters (lower is better).
 - Determine the number of detected clusters by counting unique non-noise labels: `len(set(labels)) - (1 if -1 in labels else 0)`.
- Count noise points (outliers) where `labels == -1`.
- Visualize the clustering result using a scatter plot: `plot X[:,0] vs X[:,1]`, color points by labels, add title "DBSCAN Clustering", and display using `matplotlib.pyplot`.
- Print Silhouette Score, Davies-Bouldin Score, number of clusters, number of noise points, and show the plot.
- Terminate.

PROGRAM:

```
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score, davies_bouldin_score
import numpy as np
import matplotlib.pyplot as plt
```

```
X, true_labels = make_blobs(n_samples=500, centers=4, cluster_std=0.50, random_state=42)
```

```
print("Data shape:", X.shape)
print("Sample data:\n", X[:5])
```

```

db = DBSCAN(
    eps=0.5,
    min_samples=5,
    metric='euclidean',
    algorithm='auto'
)

labels = db.fit_predict(X)

sil_score = silhouette_score(X, labels)
db_score = davies_bouldin_score(X, labels)
print("\nSilhouette Score:", sil_score)
print("Davies-Bouldin Score:", db_score)
print("Number of clusters:", len(set(labels)) - (1 if -1 in labels else 0))
print("Noise points:", np.sum(labels == -1))

plt.scatter(X[:,0], X[:,1], c=labels)
plt.title("DBSCAN Clustering")
plt.show()

```

OUTPUT:

Data shape: (500, 2)

Sample data:

```

[[ 0.82582218 -1.41484578]
 [ 7.70626288  3.38909027]
 [ 6.34394268  3.21145511]
 [ 1.64337867 -0.29584408]
 [ 7.04956829  3.3462783 ]]

```

Silhouette Score: 0.8688810020191738

Davies-Bouldin Score: 1.12674978754533

Number of clusters: 4

Noise points: 7

VIVA QUESTIONS:

1. How does DBSCAN handle noise?
2. What is eps in DBSCAN?
3. Difference from K-Means?
4. What is core point?
5. When to use density-based clustering?

RESULT:

Thus, the similarity-based clustering using DBSCAN was implemented successfully.

Ex. No: 7**CLUSTERING ALGORITHM****AIM:**

To implement K-Means clustering on the Iris dataset.

DESCRIPTION:

K-Means partitions data into K clusters by minimizing intra-cluster variance. It is centroid-based and requires specifying K. This experiment uses Iris data, evaluates with silhouette and Calinski-Harabasz scores, and shows centers. Useful for market segmentation.

ALGORITHM:

- Start the process.
- Load the Iris dataset using `sklearn.datasets.load_iris()`; extract the feature matrix `X` containing 4 measurements (sepal length, sepal width, petal length, petal width) for 150 samples.
- Initialize a K-Means clustering model with:
 - `n_clusters=3` (to match the 3 Iris species),
 - `init='k-means++'` (smart centroid initialization),
 - `n_init=10` (run algorithm 10 times with different seeds),
 - `max_iter=300` (maximum iterations per run),
 - `random_state=42` (for reproducibility).
- Fit the K-Means model on `X` and predict cluster labels simultaneously using `.fit_predict(X)`, assigning each sample to one of 3 clusters.
- Evaluate clustering quality:
 - Compute Silhouette Score using `silhouette_score(X, labels)` to measure how similar points are to their own cluster vs. others (range: -1 to 1, higher is better).
 - Compute Calinski-Harabasz Score using `calinski_harabasz_score(X, labels)` to measure ratio of between-cluster to within-cluster dispersion (higher is better).
- Retrieve and print the final cluster centers (`kmeans.cluster_centers_`) — the mean feature values for each of the 3 clusters.
- Print the inertia (`kmeans.inertia_`) — sum of squared distances from points to their assigned cluster center (lower indicates tighter clusters).
- Visualize the clustering result by plotting the first two features (`X[:,0]` vs `X[:,1]`) in a scatter plot, coloring points by predicted labels, adding title "K-Means on Iris", and displaying the plot using `matplotlib.pyplot`.
- Print the Silhouette Score, Calinski-Harabasz Score, cluster centers, inertia, and display the 2D projection of the clusters.
- Terminate.

PROGRAM:

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, calinski_harabasz_score
import matplotlib.pyplot as plt
```

```
iris = load_iris()
X = iris.data
```

```

kmeans = KMeans(
    n_clusters=3,
    init='k-means++',
    n_init=10,
    max_iter=300,
    random_state=42
)

labels = kmeans.fit_predict(X)

print("Silhouette Score:", silhouette_score(X, labels))
print("Calinski-Harabasz Score:", calinski_harabasz_score(X, labels))
print("\nCluster Centers:\n", kmeans.cluster_centers_)
print("Inertia:", kmeans.inertia_)

plt.scatter(X[:,0], X[:,1], c=labels)
plt.title("K-Means on Iris")
plt.show()

```

OUTPUT:

```

Silhouette Score: 0.5528190123564093
Calinski-Harabasz Score: 561.62775662962

Cluster Centers:
[[5.9016129  2.7483871  4.39354839 1.43387097]
 [5.006      3.428      1.462      0.246      ]
 [6.85       3.07368421 5.74210526 2.07105263]]
Inertia: 78.85144142614605

```

VIVA QUESTIONS:

1. How to choose K in K-Means?
2. What is elbow method?
3. Limitations of K-Means?
4. What is init='k-means++'?
5. How does inertia measure quality?

RESULT:

Thus, K-Means clustering was implemented successfully on the Iris dataset.

Ex. No: 8**REINFORCEMENT LEARNING****AIM:**

To apply Q-Learning for reinforcement learning in a grid world.

DESCRIPTION:

Reinforcement Learning involves agents learning from rewards. Q-Learning is off-policy, updating Q-values for state-action pairs. This experiment uses a 1D grid with goal at end, epsilon-greedy exploration. We train for more episodes, print progress.

ALGORITHM:

- Start the process.
- Initialize a Q-table Q as a 6×2 zero matrix for 6 states (0 to 5) and 2 actions (0: left, 1: right); set hyperparameters: discount factor $\gamma=0.95$, learning rate $\alpha=0.1$, episodes=2000, exploration rate $\epsilon=0.1$.
- For each of 2000 episodes: randomly select initial state state from 0 to 4 (excluding goal state 5).
- While current state is not the goal (5):
 - Select action using ϵ -greedy policy: with probability ϵ , choose random action (0 or 1); otherwise, choose greedy action $\text{argmax}(Q[\text{state}])$.
 - Compute next state: $\text{next_state} = \max(0, \min(5, \text{state} + (1 \text{ if action}==1 \text{ else } -1)))$ to move right (+1) or left (-1), bounded by 0 and 5.
 - Assign reward: +1 if $\text{next_state}==5$ (goal reached), else -0.01 (small living penalty).
- Update Q-value using Q-learning formula: $Q[\text{state}, \text{action}] += \alpha * (\text{reward} + \gamma * \max(Q[\text{next_state}]) - Q[\text{state}, \text{action}])$.
- Transition to next_state and repeat until goal state 5 is reached (episode terminates).
- Every 200 episodes, print progress message "Episode X completed."
- After all episodes, print the final learned Q-table and extract the optimal policy as $\text{argmax}(Q, \text{axis}=1)$ for each state (0:left, 1:right), showing the best action per state.
- Terminate.

PROGRAM:

```
import numpy as np
states = 6
actions = 2
Q = np.zeros((states, actions))

gamma = 0.95
alpha = 0.1
episodes = 2000
epsilon = 0.1

for episode in range(episodes):
    state = np.random.randint(0, states-1)
    while state != states-1:
        action = np.random.choice(actions) if np.random.uniform() < epsilon else np.argmax(Q[state])
        next_state = max(0, min(states-1, state + (1 if action else -1)))
        reward = 1 if next_state == states-1 else -0.01
        Q[state, action] += alpha * (reward + gamma * np.max(Q[next_state]) - Q[state, action])
        state = next_state
    if episode % 200 == 0:
```

```
print(f"Episode {episode} completed.")

print("\nQ-Table:\n", Q)
print("\nOptimal Policy (0:left, 1:right):", np.argmax(Q, axis=1))
```

OUTPUT:

```
Episode 0 completed.
Episode 200 completed.
Episode 400 completed.
Episode 600 completed.
Episode 800 completed.
Episode 1000 completed.
Episode 1200 completed.
Episode 1400 completed.
Episode 1600 completed.
Episode 1800 completed.
Q-Table:
[[0.65582052 0.7774075 ]
 [0.72276668 0.82885   ]
 [0.77720346 0.883     ]
 [0.82881461 0.94      ]
 [0.88298713 1.        ]
 [0.         0.        ]]
```

```
Optimal Policy (0:left, 1:right): [1 1 1 1 1 0]
```

VIVA QUESTIONS:

1. What is Q-value?
2. Difference between on-policy and off-policy?
3. Role of gamma?
4. What is exploration-exploitation trade-off?
5. When does Q-Learning converge?

RESULT:

Thus, reinforcement learning using Q-Learning was applied successfully.

Ex. No: 9

CANDIDATE-ELIMINATION ALGORITHM

AIM:

1. To implement the Candidate-Elimination algorithm to determine the version space given training examples in a CSV file.
2. To output the set of all hypotheses consistent with the observed data..

Candidate-Elimination Algorithm

1. Initialize:

- $S \leftarrow$ the most specific hypothesis in H
- $G \leftarrow$ the most general hypothesis in H

2. For each training example:

- If the example is **positive**:
 - Remove from G any hypothesis inconsistent with the example
 - For each hypothesis s in S that is not consistent with the example:
 - Remove s from S
 - Add to S all minimal generalizations of s that are consistent with the example and more specific than some hypothesis in G
 - Remove from S any hypothesis more general than another hypothesis in S
- If the example is **negative**:
 - Remove from S any hypothesis inconsistent with the example
 - For each hypothesis g in G that is not consistent with the example:
 - Remove g from G
 - Add to G all minimal specializations of g that are consistent with the example and more general than some hypothesis in S
 - Remove from G any hypothesis less general than another hypothesis in G

3. Output:

- The version space: All hypotheses between S and G (i.e., more general than S , more specific than G)

Source Code

```
import numpy as np
import pandas as pd
# Loading Data from a CSV File
data = pd.DataFrame(data=pd.read_csv('trainingdata.csv'))
print(data)
# Separating concept features from Target
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
# Isolating target into a separate DataFrame
# copying last column to target array
target = np.array(data.iloc[:,-1])
print(target)
def learn(concepts, target):
    """
    learn() function implements the learning method of the Candidate elimination algorithm.
    Arguments:
        concepts - a data frame with all the features
        target - a data frame with corresponding output values
    """
    # Initialise S0 with the first instance from concepts
    # .copy() makes sure a new list is created instead of just pointing to the same memory
    location
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and general_h")
    print(specific_h)
    #h=["#" for i in range(0,5)]
    #print(h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)
    # The learning iterations
    for i, h in enumerate(concepts):
```

```

# Checking if the hypothesis has a positive target
if target[i] == "Yes":
    for x in range(len(specific_h)):
        # Change values in S & G only if values change
        if h[x] != specific_h[x]:
            specific_h[x] = '?'
            general_h[x][x] = '?'
# Checking if the hypothesis has a positive target
if target[i] == "No":
    for x in range(len(specific_h)):
        # For negative hyposthesis change values only in G
        if h[x] != specific_h[x]:
            general_h[x][x] = specific_h[x]
        else:
            general_h[x][x] = '?'
print("\nSteps of Candidate Elimination Algorithm",i+1)
print(specific_h)
print(general_h)

```

```

# find indices where we have empty rows, meaning those that are unchanged
indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
for i in indices:

```

```

    # remove those rows from general_h
    general_h.remove(['?', '?', '?', '?', '?', '?'])
# Return final values
return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("\nFinal Specific_h:", s_final, sep="\n")
print("\nFinal General_h:", g_final, sep="\n")

```

Output

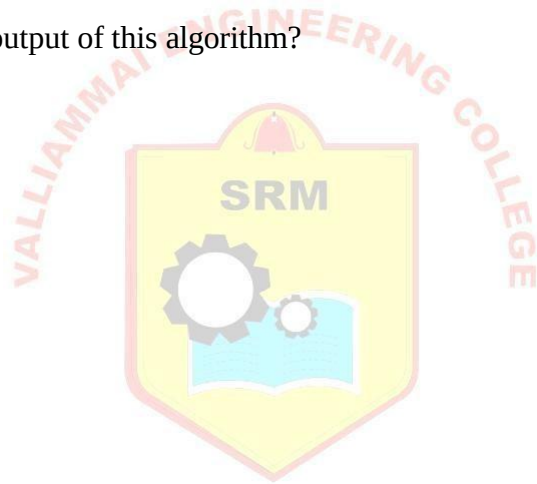
```

sky airTemp humidity   wind water forecast enjoySport
0 Sunny   Warm   Normal Strong Warm   Same       Yes

```


VIVA QUESTIONS

1. What is the version space?
2. What are the S and G boundaries?
3. How does the Candidate-Elimination algorithm work?
4. What assumptions does Candidate-Elimination make?
5. How does it handle noisy data?
6. Can it be used for continuous attributes?
7. What is the role of generalization and specialization?
8. What happens if no hypothesis is consistent with the training data?
9. Can this algorithm work on continuous data?
10. What is the output of this algorithm?



RESULT:

Thus the Candidate –Elimination algorithm for the given dataset have done and output is verified.

Ex. No: 10**K-NEAREST NEIGHBOUR ALGORITHM****Aim**

To write a program to implement **k-Nearest Neighbour Algorithm** to classify the iris data set.

Algorithm**• Load Dataset:**

- Import the dataset suitable for classification (e.g., Iris dataset in this case).

• Split Dataset:

- Divide the dataset into training and testing sets:
 - Feature data (XXX): Input features of the dataset.
 - Target labels (yyy): Corresponding class labels.

• Initialize the Model:

- Choose the k-value for the KNN classifier (e.g., $k=1$).

• Train the Model:

- Fit the KNN model on the training data (X_{train} , y_{train}).

• Make Predictions:

- For each test instance (X_{test}):
 - Extract the instance as input (xxx).
 - Reshape into the correct format (if needed).
 - Predict the class label using the trained model.

• Compare Predictions:

- Compare the predicted label with the actual label for each test instance.
- Optionally, map numerical class labels to class names for interpretability.

• Evaluate the Model:

- Compute the model's accuracy on the test data:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

• Output:

- Display:
 - Actual vs. predicted class labels for each test instance.
 - Overall model accuracy.

Source Code

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

```

import numpy as np

dataset=load_iris()

#print(dataset)

X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=
0)

kn=KNeighborsClassifier(n_neighbors=1)

kn.fit(X_train,y_train)

for i in range(len(X_test)):
    x=X_test[i]
    x_new=np.array([x])
    prediction=kn.predict(x_new)
    print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction
,dataset["target_names"][prediction])
print(kn.score(X_test,y_test))

```

Output

```

TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']

```

```
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
0.9736842105263158
```

VIVA QUESTIONS

1. How does k-NN work?
2. What is the role of distance metrics in k-NN?
3. What is the effect of choosing different values of **k**?
4. What distance metrics are commonly used?
5. Is k-NN a lazy learner?
6. Can k-NN be used for regression?
7. What is the decision boundary in k-NN?
8. What are the drawbacks of k-NN?
9. What happens when k is too small or too large?
10. How do you evaluate k-NN performance?

RESULT:

Thus the k-nearest neighbour algorithm was executed and verified successfully.

Ex. No: 11 NON-PARAMETRIC LOCALLY WEIGHTED REGRESSION ALGORITHM

Aim

To write a program to implement non-parametric locally weighted regression algorithm

Algorithm

1. Read the Given data Sample to X and the curve (linear or non linear) to Y
2. Set the value for Smoothing parameter or Free parameter say τ
3. Set the bias /Point of interest set x_0 which is a subset of X
4. Determine the weight matrix using :

$$w(x, x_0) = e^{-\frac{(x-x_0)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using :

$$\hat{\beta}(x_0) = (X^T W X)^{-1} X^T W y$$

6. Prediction = $x_0 * \beta$:

Source Code

```
import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot

output_notebook()

def radial_kernel(x0, X, tau):
    """
    Compute Gaussian (radial basis) weights for LWR.
    """
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))

def local_regression(x0, X, Y, tau):
    """
    Perform Locally Weighted Regression for a single x0.
    """
    x0 = np.r_[1, x0] # Add bias term (x0 -> [1, x])
    X_b = np.c_[np.ones(len(X)), X] # Add bias column to X

    weights = np.diag(radial_kernel(x0, X_b, tau))

    beta = np.linalg.pinv(X_b.T @ weights @ X_b) @ X_b.T @ weights @ Y
```

```

    return x0 @ beta

n = 1000
X = np.linspace(-3, 3, num=n)
print("The Data Set (10 Samples) X:\n", X[1:10])

Y = np.log(np.abs(X ** 2 - 1) + 0.5)
print("The Fitting Curve Data Set (10 Samples) Y:\n", Y[1:10])

X += np.random.normal(scale=0.1, size=n)
print("Normalized (10 Samples) X:\n", X[1:10])

domain = np.linspace(-3, 3, num=300)
print("Xo Domain Space (10 Samples):\n", domain[1:10])

def plot_lwr(tau):
    """
    Plot LWR predictions for a given tau value.
    """
    predictions = [local_regression(x0, X, Y, tau) for x0 in domain]

    plot = figure(width=400, height=400)
    plot.title.text = f"Locally Weighted Regression ( $\tau = \{\text{tau}\}$ )"
    plot.xaxis.axis_label = "X"
    plot.yaxis.axis_label = "Y"

    # Plot scatter of raw data
    plot.scatter(X, Y, alpha=0.3, color="blue", legend_label="Data Points")

    # Plot regression line
    plot.line(domain, predictions, line_width=2, color="red", legend_label="LWR Fit")

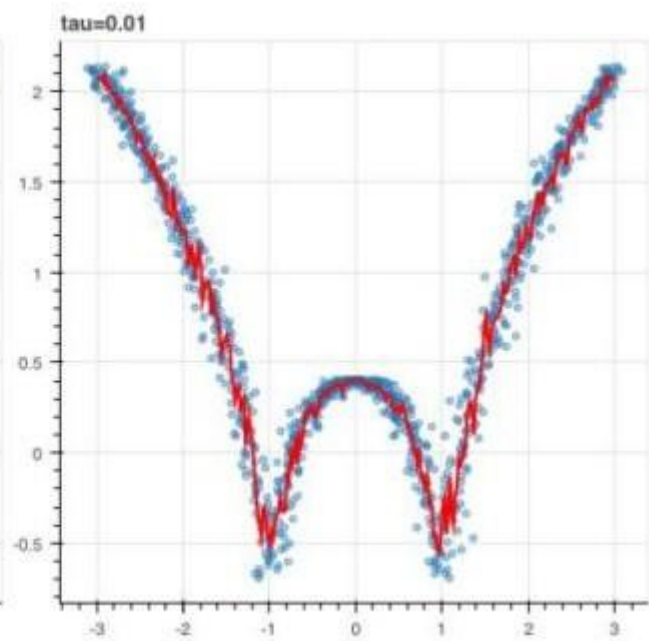
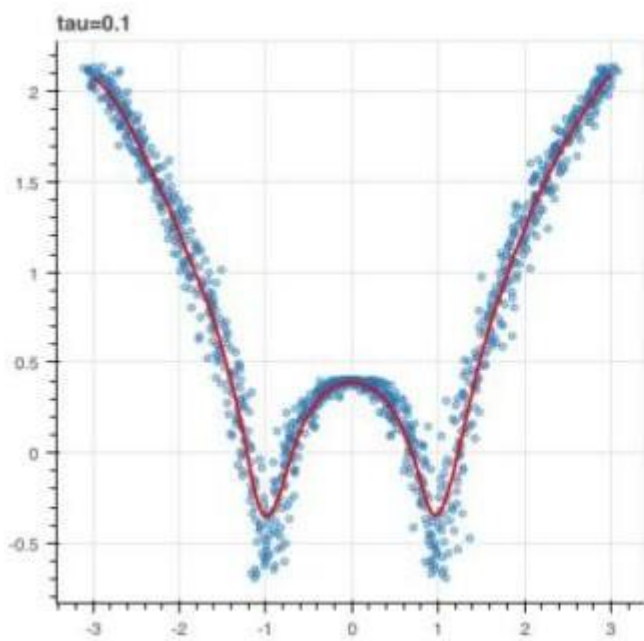
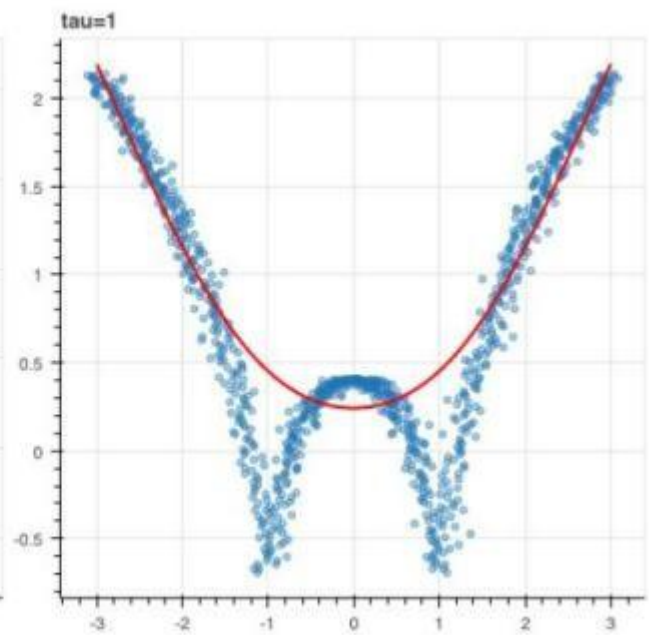
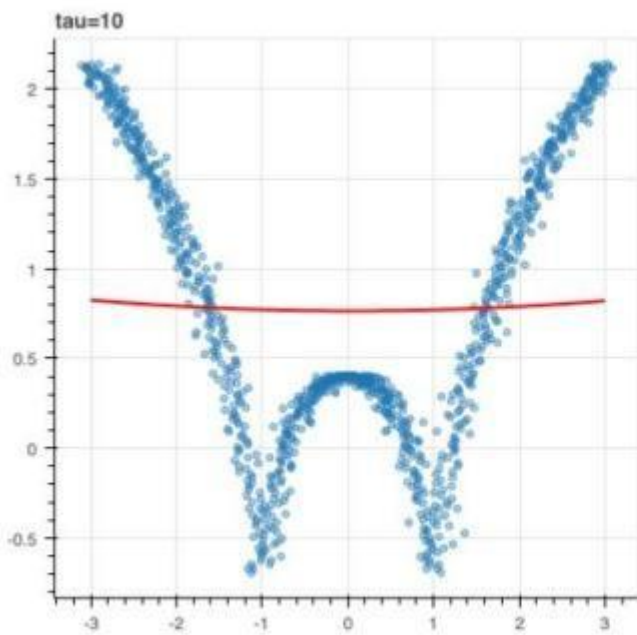
    plot.legend.location = "top_left"
    return plot

p = gridplot([
    [plot_lwr(10.0), plot_lwr(1.0)],
    [plot_lwr(0.1), plot_lwr(0.01)]
])

show(p)

```

OUTPUT



VIVA QUESTIONS:

1. What is Locally Weighted Regression (LWR)?
2. What does "non-parametric" mean in LWR?
3. What is the difference between Linear Regression and Locally Weighted Regression?
4. What is the role of the smoothing parameter (τ or tau)?
5. What is the purpose of the weight matrix in LWR?
6. What kernel function is used to compute the weights?
7. Write the formula to compute the model parameters β .
8. How is the prediction made for a given query point x_0 ?
9. Why do we add a bias term to the input X and x_0 ?

RESULT:

Thus the non-parametric locally weighted regression algorithm for the given dataset have done and output is verified