INDEX

S. No	Name of the Experiment	Page	Date of Experiment	Date of Submission	Faculty Sign
1	Write a program to implement BFS and DFS Traversal.	1 – 7	17/07/25	17/07/25	17/7/25
2	Write a program to implement A* Search.	8-13	31/07/25	31/07/25	31/7
3	Write a program to implement Travelling Salesman Problem and Graph Coloring Problem	14 – 19	24/07/25	24/07/25	24/7/25
4	Write a program to implement Knowledge Representation	20 – 25	7/8/25	7/8/25	7/8
5	Write a program to implement Bayesian Network.	26 – 31	14/08/25	14/8/25	14/8.
6	Write a program to implement Hidden Markov Model.	32 – 37	21/08/25	21/8/25	21/8
7	Write a program to implement Regression algorithm	38 – 44	18/9/25	18/9/25	18/9
8	Write a program to implement decision tree based ID3 algorithm.	45 – 52	25/9/25	25/9/25	25/9
9	Write a program to implement K-Means Clustering algorithm.	53 – 58	23/10/25	23/10/25	23/10
10	Write a program to implement K-Nearest Neighbor algorithm (K-NN).	59 – 64	9/10/25	9/10/25	9/10.
11	Write a program to implement Back Propagation Algorithm.	65 – 70	23/10/25	23/10/25	23/10.
12	Write a program to implement Support Vector Machine.	71 – 76	16/10/25	16/10/25	16/10.

Experiment-1

Date: 17/09/25

AIM

Write a program to implement BFS and DFS Traversal.

BFS

THEORY

Breadth-First Search (BFS) is a graph traversal algorithm used in AI to explore all nodes at the present depth level before moving to the next level. It is particularly useful for finding the shortest path in unweighted graphs, as it ensures the first time a node is visited, it is through the shortest possible path. BFS is widely applied in AI for tasks like pathfinding, puzzle solving, and game state exploration. By using a queue, BFS processes nodes in a systematic, level-by-level manner.

ALGORITHM

1. Initialize an empty list **visited[]** and a queue with the starting node.
2. Mark the starting node as visited.
3. While the queue is not empty, dequeue a node from the queue and process it.
4. For each unvisited neighbor of the current node, mark it as visited and enqueue it.
5. Continue this process until the queue is empty.
6. The algorithm terminates when all nodes have been visited.

DFS

THEORY

Depth-First Search (DFS) is a graph traversal algorithm used in Artificial Intelligence to explor as deeply as possible along each branch before backtracking. It is commonly applied in task like puzzle solving, game tree exploration, and searching through large state spaces. DFS i particularly useful in scenarios where we need to explore all possibilities in a deep hierarchy o structure, such as in decision-making and problem-solving. While DFS can be memory efficient it doesn't always guarantee the shortest or optimal solution compared to BFS.

ALGORITHM

1. Initialize an empty list **visited[]** and a stack with the starting node.
2. Mark the starting node as visited.
3. While the stack is not empty, pop a node from the stack and process it.
4. For each unvisited neighbor of the current node, mark it as visited and push it onto th stack.
5. Continue this process until the stack is empty.
6. The algorithm terminates when all nodes have been visited.

SOURCE CODE

```
graph = { }
edge_set = set()
def add_node(node):
    if node in graph:
        print(f"{node}' already exists,
            please enter a different node")
    return False
    graph[node] = []
    return True
```

Experiment-2

Date: 31/07/25

AIM

Write a Program to Implement A* Search.

THEORY

A* (A-star) Search is a popular algorithm in Artificial Intelligence used for finding the shortest path from a starting node to a goal node, while considering both the cost to reach a node and the estimated cost to reach the goal. It combines the advantages of Dijkstra's algorithm (which guarantees the shortest path) and Greedy Best-First Search (which focuses on promising paths). A* uses a heuristic function to guide its search efficiently, making it widely used in pathfinding and navigation tasks. Its optimality and efficiency make it ideal for AI applications like robot navigation and game AI.

ALGORITHM

1. Initialize the open_list with the starting node and the closed_list as empty.
2. While the open_list is not empty, select the node with the lowest f(n) value.
3. If the selected node is the goal, stop and reconstruct the path.
4. Generate neighbors of the current node and evaluate them based on g(n) and h(n).
5. Add unvisited neighbors to the open_list and move the current node to the closed_list.
6. Repeat until the goal is found or the open_list is empty.

Experiment-3

Date: 24/4/25

AIM

Write a Program to Implement Travelling Salesman Problem and Graph Coloring Problem

THEORY

Travelling Saleman Problem

The Traveling Salesman Problem (TSP) seeks the shortest route that visits each city once ar returns to the start. It's an NP-hard problem, making it challenging for large numbers of citie Solutions involve methods like dynamic programming or heuristic algorithms. TSP ha applications in logistics, route optimization, and network design.

ALGORITHM

1. Consider city1as the starting and ending point.
2. Generate all (n-1)! Permutations of cities.
3. Calculate cost of every permutation and keep track of minimumcostpermutation.
4. Return the permutation with minimum cost.

THEORY

Graph Coloring Problem

The Graph Coloring Problem involves assigning colors to vertices in a graph such that no two adjacent vertices share the same color, using the fewest colors possible. It is widely used in scheduling, resource allocation, and network design. The problem is NP-hard, meaning it becomes

difficult to solve for large graphs, often requiring heuristic or approximation methods for practical solutions.

ALGORITHM

1. Color first vertex with first color.
2. Do following for remainingV-1 vertices.

Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

SOURCE CODE

```
from sys import maxsize
from itertools import permutations

v=4
def travellingSalesmanProblem(graph,s):
    vertex=[]
    for i in range(v):
        if i!=s:
            vertex.append(i)
            print(vertex)

    min_path= maximize.
    for i in next_permutation:
        print(i)
        current_pathweight=0

        K=s
```

Experiment-4

Date: 7/03/25

AIM

Write a Program to Implement Knowledge Representation

THEORY

Knowledge Representation

Knowledge Representation in artificial intelligence involves modeling information about t world in a form that a computer can understand and reason about. It enables machines to stor retrieve, and manipulate data to solve problems, make decisions, and understand natural languag Common methods include logic, semantic networks, frames, and ontologies. Effective knowled representation is crucial for tasks like expert systems, machine learning, and natural langua processing.

ALGORITHM

Let's start with a Harry Potter example. Consider the following sentences.

1. If it didn't rain, Harry visited Hagrid today.
2. Harry visited Hagrid or Dumbledore today, but not both.
3. Harry visited Dumbledore today.

Based on these three sentences, we can answer the question "did it rain today?", eve though none of the individual sentences tell us anything about whether it is raining today. Her is how we can go about it: looking at sentence 3, we know Harry visited Dumbledore. Lookin

at sentence 2, we know Harry visited either Dumbledore or Hagrid, and thus we can conclude 4. Harry did not visit Hagrid.

4. Harry did not visit hagrid

Now, looking at sentence 1, we understand that if it didn't rain, Harry would have visited Hagrid. However, knowing sentence 4, we know that this is not the case. Therefore, we can conclude 5. It rained today.

5. It rained today

SOURCE CODE

```
from sympy import symbols, Or, Not, Implies, Xor,
satisfiable
Rain = symbols("Rain")
Harry-visited_Hagrid = symbols("Harry_visited_Hagrid")
Harry-visited_Dumbledore = symbols("Harry_visited_Dumbledore")

Sentence_1 = Implies( Not(Rain), Harry_visited_Hagrid )
Sentence_2 = Xor (Harry visited_hagrid, Harry_visited_
                                        Dumbledore )

Sentence_3 = Harry_visited_Dumbledore
Knowledge_base = sentence_1 & sentence_2 & sentence_3
Solution = satisfiable (knowledge_base, all_models
                                        = True)

for model in solutions
    if model[Rain]:
        print ("It rained today")
    else:
        print ("there is no rain today")
```

Experiment-5

Date: 14/08/25

AIM

Write a Program to Implement Bayesian Network

THEORY

Bayesian Network

A Bayesian Network is a graphical model that represents probabilistic relationships amon variables using directed acyclic graphs (DAGs). Each node in the network represents a rando variable, and the edges represent conditional dependencies between them. Bayesian Network are used for reasoning under uncertainty, where they help in making predictions, diagnosin problems, and understanding complex systems by calculating the probabilities of differen outcomes based on prior knowledge and observed evidence.

ALGORITHM

1. Define the conditional probability tables (CPTs) for each event (e.g., burglary, earthquake alarm, David's call, and Sophia's call).
2. Accept input parameters (alarm, burglary, earthquake, David's call, Sophia's call).
3. Calculate the probability of the alarm going off based on the CPTs and input conditions.
4. Calculate the probability of David calling based on the alarm status and CPTs.
5. Calculate the probability of Sophia calling based on the alarm status and CPTs.
6. Compute the joint probability by multiplying the probabilities of burglary, earthquake, alarm, David's call, and Sophia's call.
7. Output the calculated joint probability.

Experiment-6

Date: 21/8/25

AIM

Write a Program to Implement Hidden Markov Model.

THEORY

Hidden Markov Model.

A Hidden Markov Model (HMM) is a statistical model used to describe systems that transit between a series of hidden states over time. E ach state produces an observable output, but actual state is not directly visible. Instead, the system is modeled as a Markov process, where future state depends only on the current state and not on the past states. HMMs are commo used in areas like speech recognition, bioinformatics, and time series prediction due to th ability to model sequential data with latent structures.

FORWARD ALGORITHM STEPS

1. **Initialization**

- Initialize the alpha matrix for the first observation.

2. **Recursion**

- For each time step, update the alpha values based on previous alpha values, transiti probabilities, and emission probabilities.

3. **Termination**

- Calculate the total probability by summing the alpha values at the last time step.

Experiment-7

Date: 18/9/25

AIM

Write a program to implement Regression algorithm.

THEORY

Regression Algorithm

Regression is a statistical technique used to model the relationship between a dependent variab and one or more independent variables. The goal of regression analysis is to predict the value the dependent variable based on the values of the independent variables. The most common ty of regression is linear regression, where a straight line is fitted to the data to minimize the differen between the predicted and actual values. Regression algorithms are widely used in various field including economics, finance, and machine learning, for tasks like predicting sales, estimati risk, and forecasting trends.

ALGORITHM

1. **Import Libraries**

- Import necessary libraries like numpy, pandas, and matplotlib.

2. **Load Dataset**

- Load the dataset from a CSV file.

3. **Prepare Data**

- Extract the independent variables (X) and dependent variable (y).

4. **Split Data**

- Split the dataset into training and test sets.

5. **Initialize the Model**

- Create an instance of the linear regression model.

6. **Train the Model**

- Fit the linear regression model using the training data.

7. **Make Predictions**

- Use the trained model to predict the outcomes for the test set.

8. **Compare Actual and Predicted Values**

- Create a DataFrame to compare the actual vs. predicted values.

9. **Visualize Training Set**

- Plot the training data points and the regression line.

10. **Visualize Test Set**

- Plot the test data points and the regression line.

11. **Display Dataset**

- Display the first few rows of the dataset.

SOURCE CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset = pd.read_csv
print(dataset)

x= dataset.iloc[:,:-1].values
y = dataset.iloc[:, 1].values
from sklearn.model_selection import train_test_split
```

Experiment-8

Date: 25/9/25

AIM

Write a program to implement decision tree based ID3 algorithm.

THEORY

Decision tree based ID3

The ID3 (Iterative Dichotomiser 3) algorithm is a decision tree-building method used for classification tasks. It constructs the tree by selecting the attribute that offers the highest information gain at each decision node, effectively reducing uncertainty. The process is repeated recursively for each branch until all data is perfectly classified or no further attributes remain. ID3 uses entropy to measure the impurity of a dataset and splits it in a way that maximizes the gain in information. It's simple, intuitive, and forms the basis for more advanced decision tree algorithms like C4.5 and CART.

ALGORITHM

ID3 Algorithm Steps

1. **Start with the entire dataset** as the root of the decision tree.
2. **Calculate the entropy** for the dataset. Entropy measures the uncertainty or impurity of the dataset.
3. **For each attribute**, calculate the information gain. Information gain is based on how well the attribute splits the dataset to reduce entropy.
4. **Select the attribute** with the highest information gain as the decision node.

5. **Split the dataset** into subsets based on the selected attribute.
6. **Repeat steps 2-5** recursively for each subset, treating each as a new dataset.
7. **Stop the recursion** when one of the following occurs:

- All instances in the subset belong to the same class.
- There are no more attributes to split on.

8. **Label the leaves** of the tree with the most frequent class of the instances in that subset.
9. **Return the decision tree** built with the selected attributes and their corresponding splits.

SOURCE CODE

```
import pandas as pd
df = pd.read_csv("play_Tennis - csv")
print(df)
import math
def entropy(probs):
    return sum(-prob * math.log(prob,2) for
        prob in probs)
from collections import Counter
cnt = Counter (x for x in a_list)
print(cnt)
num_instances = len(a_list)
probs = [x/num_instances ]
print (num_instances)
print(probs)
return entropy(probs)
total_energy = entropy_list(df['playTennis'])
print(total_energy)
```

Experiment-9

Date: 9/10/25

AIM

Write a program to implement K-Means Clustering algorithm.

THEORY

K-Means Clustering algorithm

K-Means Clustering is an unsupervised machine learning algorithm used to partition a dataset into K distinct clusters based on similarity. The algorithm begins by selecting K initial centroids (randomly or through some initialization technique). Then, it assigns each data point to the nearest centroid, forming K clusters. Afterward, it recalculates the centroids as the mean of all points in each cluster. This process of assigning points and updating centroids repeats until convergence, where the centroids no longer change. K-Means is widely used for data segmentation, pattern recognition, and anomaly detection due to its simplicity and efficiency.

ALGORITHM

1. Initialize K Centroids

Randomly select K data points as initial centroids or use a more advanced initialization method (e.g., K-Means++).

2. Assign Points to Nearest Centroid:

For each data point, assign it to the nearest centroid based on a distance metric (typically Euclidean distance).

3. Update Centroids

Recalculate the centroids by finding the mean of all data points assigned to each centroid.

4. Repeat

Repeat steps 2 and 3 until the centroids no longer change significantly (convergence) or a maximum number of iterations is reached.

5. Output

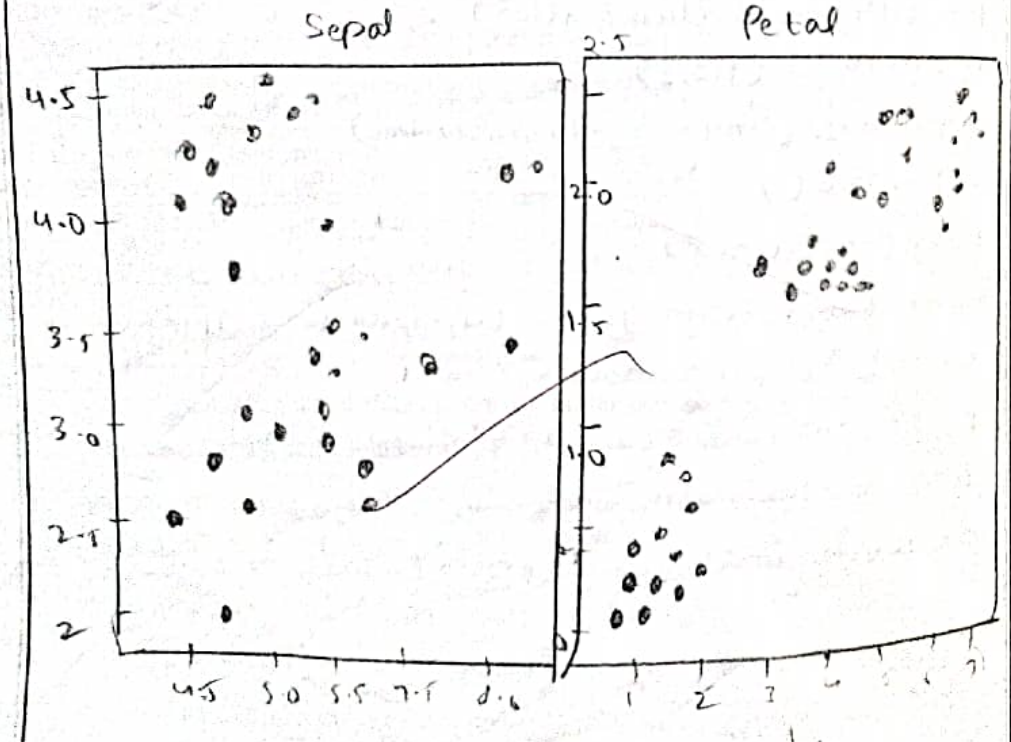
Once the algorithm has converged, output the final clusters and centroids.

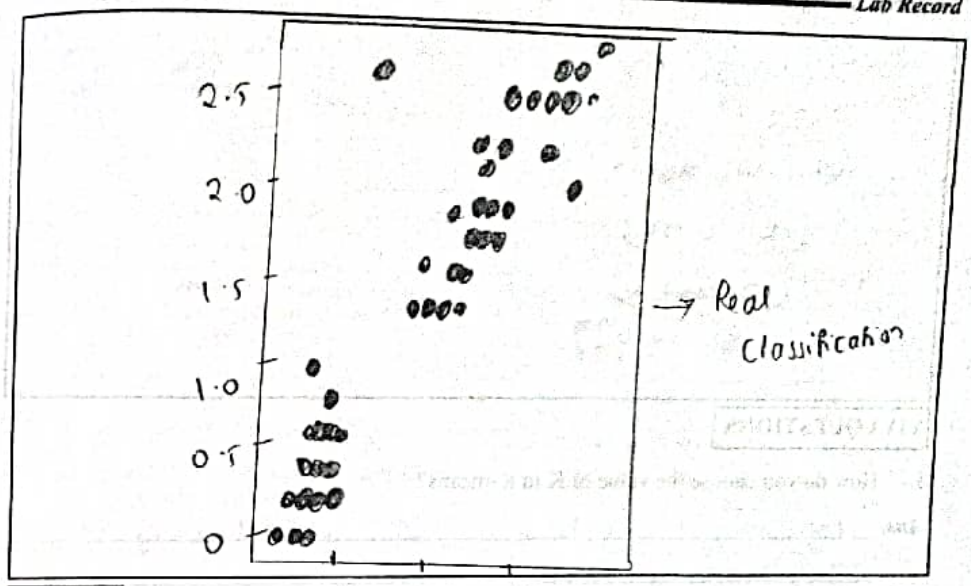
SOURCE CODE

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import kMeans
import sklearn.metrics as sm
import pandas as pd

iris = datasets.load_iris()
x = pd.DataFrame(iris.data)
x.columns = ['sepal_width', 'sepal_length']
print(x)

y = pd.DataFrame(iris.target)
y.columns = ['target']
print(y)
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(1,2,1)
```

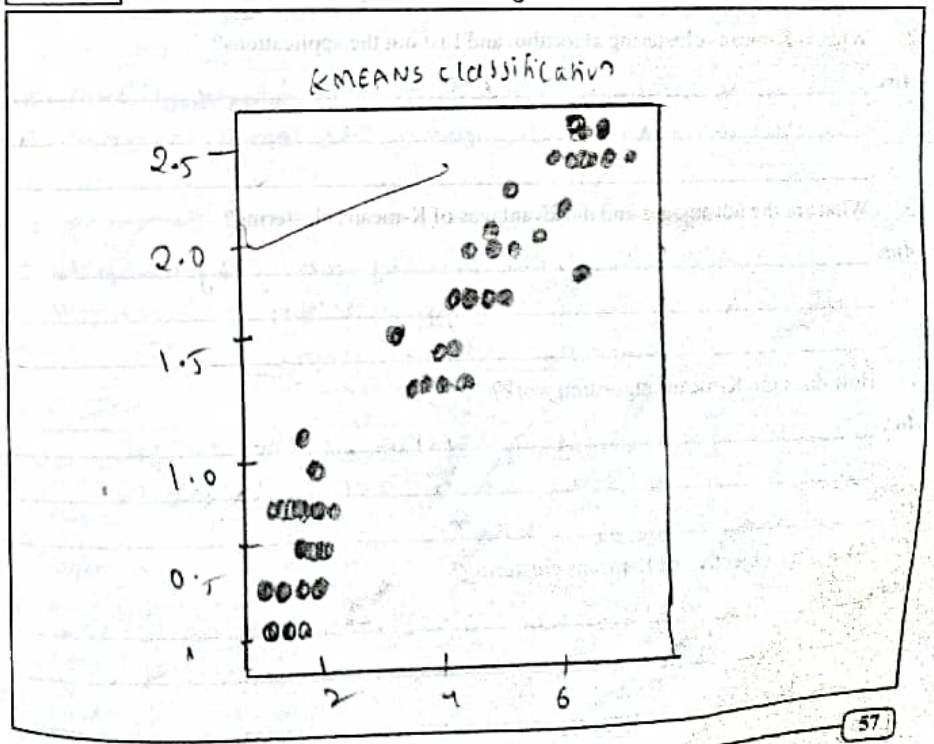
	Sepal_Length	Sepal_Width	Petal_Width
0	5.1	3.5	0.2
1	4.9	3.0	0.2
2	4.7	3.2	0.2
3	5.0	3.6	0.2
⋮	⋮	⋮	⋮
148	6.2	3.4	2.3
149	5.9	3.0	1.8

[150 rows x 4 columns]





OUTPUT



Experiment-10

Date: 23/10/25

AIM

Write a program to implement K-Nearest Neighbor algorithm (K-NN).

THEORY

K-Nearest Neighbor algorithm (K-NN)

The **K-Nearest Neighbor (K-NN)** algorithm is a simple, non-parametric, and lazy supervised learning algorithm used for classification and regression tasks. It works by finding the K nearest data points to a given test point in the feature space, based on a distance metric (commonly Euclidean distance). For classification, the test point is assigned the most common class among its K nearest neighbors. For regression, the prediction is typically the average of the values of the K nearest neighbors. The K-NN algorithm does not require any training phase, making it computationally efficient for small datasets but potentially slow for large datasets due to the need to calculate distances for each query. It is sensitive to the choice of K and the scale of features.

ALGORITHM

K-Nearest Neighbor (K-NN) Algorithm for Iris Dataset

1. **Import Libraries**

- Import necessary libraries such as pandas, sklearn, and specific functions for dataset loading, splitting, and K-NN classification.

2. **Load Dataset**

- Load the Iris dataset using load_iris() from sklearn.datasets.

3. **Visualize the Data**

- Convert the Iris dataset into a pandas DataFrame for better visualization and display the feature names and first few rows of the dataset.

4. **Define Features and Target**

- Extract the feature columns (X) and target labels (y) from the dataset.

5. **Split Data**

- Split the dataset into training and testing sets using train_test_split() with 67% for training and 33% for testing.

6. **Initialize the K-NN Classifier**

- Initialize the K-NN classifier with n_neighbors=3.

7. **Train the Model**

- Fit the K-NN model using the training data (X_train, y_train).

8. **Make Predictions**

- Use the trained model to predict the labels for the test data (X_test).

9. **Evaluate the Model (Test Data)**

- Generate the confusion matrix for the predicted labels (y_pred) on the test data.
- Calculate and print the accuracy score for the test data.

10. **Evaluate the Model (Training Data)**

- Predict the labels for the training data (X_train).
- Generate the confusion matrix for the predicted labels on the training data.

11. **Output the Results**

- Display the confusion matrix and accuracy score for both the test and training sets.

SOURCE CODE

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
import pandas as pd
import numpy as np
```

OUTPUT

Confusion matrix

```
[[7 0 0]
 [0 5 0]
 [0 0 3]]
```

Accuracy Metrics

	precision	recall	f1-score	support
0	1.00	1.00	1.00	7
1	1.00	1.00	1.00	5
2	1.00	1.00	1.00	3
accuracy			1.00	15
macro avg	1.00	1.00	1.00	15
weighted avg	1.00	1.00	1.00	15

Experiment-11

Date: 23/10/25

AIM

Write a program to implement Back Propagation Algorithm.

THEORY

Back Propagation Algorithm

The **Backpropagation algorithm** is a supervised learning technique used for training artificial neural networks. It works by calculating the gradient of the loss function with respect to the network's weights using the chain rule, and then propagating this gradient backward through the network to update the weights. The algorithm consists of two main phases: the forward pass, where the input is passed through the network to get the output, and the backward pass, where the error is calculated and propagated back to adjust the weights. This process is repeated iteratively to minimize the error and improve the model's performance. Backpropagation is crucial in training deep learning models and is often used with optimization techniques like gradient descent.

ALGORITHM

Backpropagation Algorithm Steps

1. **Initialize the weights**

- Initialize the weights of the neural network randomly.

2. **Forward Pass**

- Input the training data into the network.

- Compute the output of each neuron layer by passing the input through the network using the current weights.

3. **Calculate the Error**

- Compute the error by comparing the network's output with the actual target value using a loss function (e.g., Mean Squared Error).

4. **Backward Pass (Backpropagate the error)**

- Calculate the gradient of the error with respect to each weight by applying the chain rule.
- Start from the output layer and propagate the error backward through the network to the input layer.

5. **Update the Weights**

- Adjust the weights using an optimization algorithm (e.g., Gradient Descent) to minimize the error. Update the weights by subtracting the gradient multiplied by a learning rate.

6. **Repeat**

- Repeat the process for all training samples in the dataset and for multiple iterations (epochs) until the error is minimized or converges to an acceptable level.

7. **Output the Trained Model**

- Once the training process is complete, the network's weights are optimized and the model is ready for prediction on new data.

SOURCE CODE

```
import numpy as np
x = np.array([[2,9],[1,5],[3,6]])
y = np.array([[92],[86],[89]])
y = y/100
```

OUTPUT

Input:

[[2, 9]
 [1 5]
 [3 5]]

Actual output:

[[0.92]
 [0.86]
 [0.89]]

Predicted output:

[[0.89416634]
 [0.88012057]
 [0.89553052]].

Experiment-12

Date: 16/10/25

AIM

Write a program to implement Support Vector Machine

THEORY

Support Vector Machine

The **Support Vector Machine (SVM)** is a supervised machine learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that best separates the data into different classes in a high-dimensional space. The goal is to maximize the margin, or the distance between the closest data points of each class, called support vectors, to the hyperplane. In cases where the data is not linearly separable, SVM uses a technique called the **kernel trick** to map the data into a higher-dimensional space where a hyperplane can be found. SVM is particularly effective in high-dimensional spaces and is commonly used for classification tasks like image recognition, text classification, and bioinformatics.

ALGORITHM

Support Vector Machine (SVM) Algorithm Steps

1. **Prepare the Data**

- Collect and preprocess the dataset. Ensure it is properly labeled for supervised learning tasks.

2. **Choose the Kernel Function**

- Decide the type of kernel to use (e.g., linear, polynomial, radial basis function (RBF)) based on the data's complexity and separability.

3. **Define the Objective**

- Maximize the margin between the two classes by finding the optimal hyperplane that separates the data points.

4. **Solve the Optimization Problem**

- Formulate an optimization problem to maximize the margin while ensuring that each data point is correctly classified.
- Use a convex optimization technique to solve for the weights (coefficients) of the hyperplane.

5. **Identify Support Vectors**

- Find the data points that are closest to the hyperplane, known as support vectors. These points determine the position of the hyperplane.

6. **Training the Model**

- Use the training data to compute the optimal hyperplane by adjusting the weights and biases based on the kernel function and margin maximization.

7. **Make Predictions**

- Use the trained SVM model to classify new data points by determining on which side of the hyperplane they lie.

8. **Evaluate the Model**

- Assess the performance of the model using metrics such as accuracy, precision, recall, or F1-score based on the test data.

SOURCE CODE

```
from sklearn import datasets
import pandas as pd
import numpy as np
iris = datasets.load_iris()
iris = datasets.load_iris()
```

Accuracy Kenig

	Precision	recall	f1-score	support
0	1.00	1.00	1.00	10
2	1.00	1.00	1.00	5
accuracy				
macro avg	1.00	1.00	1.00	15
weighted avg	1.00	1.00	1.00	15

OUTPUT

	Sepal_length	Sepal_width	Petal_length	Petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	5.0	3.6	1.4	0.2
...	...	...	...	...
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

[150 rows x 4 columns]

	Targets
0	0
1	0
2	0
3	0
...	
147	2
148	2
149	2

[150 rows x 1 columns]

confusion matrix

[[10 0]
 [0 5]]