

**522CIP07 -ARTIFICAL INTELLIGENCE**

**AND MACHINE LEARNING LABORATORY**

(REGULATION - 2022)

**STAFF INCHARGE HOD**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Vision of the Institute**

To foster ACE as a centre for nurturing and developing world class Engineers and Managers who

convert global challenges into opportunities through value-based quality education.

**Mission of the Institute**

**M 1 :** To impart value-based quality education through effective teaching-learning processes

**M 2 :** To nurture creativity, excellence and critical thinking by applying global competency factors to contribute and excel in the rapidly growing technological world.

**M 3 :** To continuously develop and improve holistic and innovative personality for global Mobility.

**M 4 :** To make ACE a centre for excellence.

**Vision of the Department**

To empower young minds to become resilient professionals, instilled with ethical principles and equipped with cutting-edge technologies to meet the evolving demands of the world

**Mission of the Department**

**M 1 :** To empower individuals with a comprehensive understanding of computer engineering principles and its applications through effective teaching and learning practices.

**M 2 :** To cultivate excellence and critical thinking, while leveraging global competency, thus enabling significant contributions to societal challenges in the fast-paced technological landscape.

**M 3 :** To facilitate the students to work with modern tools and technologies to foster innovation, a zest for higher studies and to build leadership qualities by inculcating the spirit of ethical values.

**Program Educational Objectives (PEOs)**

**PEO1 :** The graduates will have sound knowledge in Mathematics, Science and Engineering concepts necessary to formulate, analyse, design and solve Engineering problems and to prepare them for higher learning, research and industry.

**PEO2 :** The graduates will possess innovative skills to assess and apply the rapid changes in technology and to engage in research leading to novel solutions for human, social and global competency.

**PEO3 :** The graduates will acquire knowledge and grab opportunities to work as teams in a multidisciplinary environment, communicate ideas effectively with diverse audiences demonstrate leadership qualities with ethical values and engage in lifelong learning.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Name of the Experiment** | **Page No** |
| 1. | Implementation of Uninformed search algorithm (BFS,DFS) | 4 |
| 2. | Implementation of Informed Search algorithm | 8 |
| 3. | Implement Candidate Elimination algorithm | 10 |
| 4. | Implement linear regression | 14 |
| 5. | Implement Back-propagation algorithm | 16 |
| 6. | Implement Support Vector Machine algorithm | 19 |
| 7. | Implement Decision Tree algorithm | 23 |
| 8. | Implement k-Nearest Neighbors algorithm | 25 |
| 9. | Implement K- Means Clustering algorithm | 27 |
| 10. | Project |  |

# EX.NO: 1 Implementation of Uninformed search algorithm (BFS,DFS)

**AIM**

To implement uninformed search algorithms such as BFS and DFS.

**Breadth First Search:**

# ALGORITHM

1. Initialize an empty list called 'visited to keep track of the nodes visited during the traversal.

2. Initialize an empty queue called 'queue' to keep track of the nodes to be traversed in the future.

3. Add the starting node to the 'visited' list and the 'queue',

4. While the 'queue' is not empty, do the following:

a. Dequeue the first node from the 'queue' and store it in a variable called current

b. Print 'current'.

c. For each of the neighbours of 'current' that have not been visited or do the following:

Mark the neighbour as visited and add it to the 'queue'.

5.When all the nodes reachable from the starting node hi the algorithm. been visited, terminate

**PROGRAM**

graph = '5': ['3','7'],

'3': ['2', '4'], ‘7’: ['8'],

‘2’:[],

'4': ['8'].

8:[];

visited = []

queue=[]

def bfs(visited. node):

visited.append(node)

queue.append(node)

while queue:

mqueue.pop(0)

print (m, end="") for neighbour in graph[m]:

if neighbour not in visited: visited.append(neighbour)

queue.append(neighbour)

print("Following is the Breadth-First Search")

bfs(visited, graph, '5')

**OUTPUT**

Following is the Breadth-First Search 537248

**Depth first Search:**

**ALGORITHM**

1: Initialize an empty set called 'visited' to keep track of the nodes visited during the traversal.

2: Define a DFS function that takes the current node, the graph, and the 'visited' set as input.

3: If the current node is not in the 'visited' set, do the following:

a. Print the current node.

b. Add the current node to the 'visited' set.

c. For each of the neighbors of the current node, call the DFS function recursively with the neighbor as the current node.

4: When all the nodes reachable from the starting node have been visited terminate the algorithm.

**PROGRAM**

graph = [

'5': ['3',7'], '3': ['2', '4'].

7': ['8'].

2:[]

'4': ['8'].

'8': []

1

the SHA ENGINEERING COLLEGE

visited = set()

def dfs(visited, graph, node):

if node not in visited:

print (node)

visited.add(node) for neighbour in grap

dfs(visited, graph, neighbour)

print("Following the Depth-First Search")

dfs(visited, graph, '5')

**OUTPUT**

Following is the Depth-First Search

5

3

2

4

8

7

**RESULT**

Thus to implement uninformed search algorithms such as BFS and DFS has been successfully executed.

**EX.NO: 2 IMPLEMENTATION OF INFORMED SEARCH ALGORITHM**

**AIM**

The aim of the A\* algorithm is to find the shortest path from a start node to a goal node in a graph by efficiently navigating through the nodes using both path cost and heuristic estimates.

# ALGORITHM

1. Initialize the open set with the start node and set g to 0 for the start node.

2. Select the node n with the lowest combined cost (actual distance g plus heuristic) from the open set.

3. Expand node n, updating neighbors’ costs and parents if a shorter path is found.

4. Move the node n from the open set to the closed set.

5. Reconstruct the path from the goal node to the start node once the goal is reached or declare that no path exists if the open set is empty.

**PROGRAM**

def aStarAlgo(start\_node, stop\_node):

open\_set = set([start\_node])

closed\_set = set()

g = {}

parents = {}

g[start\_node] = 0

parents[start\_node] = start\_node

while open\_set:

n = None

for v in open\_set:

if n is None or g[v] + heuristic(v) < g[n] + heuristic(n):

n = v

if n is None:

print('Path does not exist!')

return None

if n == stop\_node:

path = []

while parents[n] != n:

path.append(n)

n = parents[n]

path.append(start\_node)

path.reverse()

print('Path found: {}'.format(path))

return path

open\_set.remove(n)

closed\_set.add(n)

for m, weight in get\_neighbors(n):

if m not in open\_set and m not in closed\_set:

open\_set.add(m)

parents[m] = n

g[m] = g[n] + weight

else:

if g[m] > g[n] + weight:

g[m] = g[n] + weight

parents[m] = n

if m in closed\_set:

closed\_set.remove(m)

open\_set.add(m)

print('Path does not exist!')

return None

def get\_neighbors(v):

return Graph\_nodes.get(v, None)

def heuristic(n):

H\_dist = {

'A': 10, 'B': 8, 'C': 5, 'D': 7, 'E': 3,

'F': 6, 'G': 5, 'H': 3, 'I': 1, 'J': 0

}

return H\_dist[n]

Graph\_nodes = {

'A': [('B', 6), ('F', 3)],

'B': [('C', 3), ('D', 2)],

'C': [('D', 1), ('E', 5)],

'D': [('C', 1), ('E', 8)],

'E': [('I', 5), ('J', 5)],

'F': [('G', 1), ('H', 7)],

'G': [('I', 3)],

'H': [('I', 2)],

'I': [('E', 5), ('J', 3)],

}

aStarAlgo('A', 'J')

**OUTPUT**

Path found: ['A', 'F', 'G', 'I', 'J']

**RESULT**

Thus the aim of the A\* algorithm to find the shortest path from a start node to a goal node in a graph by efficiently navigating through the nodes using both path cost and heuristic estimates has been successfully executed.

**EX.NO: 3 IMPLEMENTATION CANDIDATE ELIMINATION ALGORITHM**

**AIM**

The aim of the code is to implement the Candidate Elimination Algorithm to learn hypotheses from training examples by refining the most specific and most general hypotheses based on positive and negative examples.

**ALGORITHM**

1. **Read Data**: Load training examples from a CSV file into a list, excluding the header.

2**. Initialize Hypotheses**: Set up the most specific hypothesis (exact match) and the most general hypothesis (all attributes are variable).

3. **Process Examples**:

- For positive examples ("yes"), update the specific hypothesis to match the example and adjust the general hypotheses.

- For negative examples ("no"), adjust general hypotheses to exclude the example.

4**. Update Hypotheses**: Print the specific and general hypotheses after processing each example.

5**. Final Hypotheses**: Collect and print the final specific and general hypotheses after processing all examples.

**PROGRAM**

import csv

file\_path = r"C:\ai\trainingexamples.csv"

try:

with open(file\_path, mode='r') as f:

csv\_file = csv.reader(f)

data = list(csv\_file)

specific = data[1][:-1]

general = [['?' for \_ in range(len(specific))] for \_ in range(len(specific))]

for i in data[1:]:

if i[-1].strip().lower() == "yes":

for j in range(len(specific)):

if i[j] != specific[j]:

specific[j] = "?"

general[j][j] = "?"

elif i[-1].strip().lower() == "no":

for j in range(len(specific)):

if i[j] != specific[j]:

general[j][j] = specific[j]

else:

general[j][j] = "?"

print(f"\nStep {data.index(i)} of Candidate Elimination Algorithm")

print("Specific Hypothesis:", specific)

print("General Hypothesis:", genera

gh = []

for i in general:

if any(j != '?' for j in i):

gh.append(i)

print("\nFinal Specific Hypothesis:\n", specific)

print("\nFinal General Hypothesis:\n", gh)

except OSError as e:

print(f"Error opening file: {e}")

**OUTPUT**

Step 1 of Candidate Elimination Algorithm

Specific Hypothesis: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

General Hypothesis: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 2 of Candidate Elimination Algorithm

Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

General Hypothesis: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 3 of Candidate Elimination Algorithm

Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

General Hypothesis: [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Step 4 of Candidate Elimination Algorithm

Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']

General Hypothesis: [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific hypothesis:

['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final General hypothesis:

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

**RESULT**

Thusto implement the Candidate Elimination Algorithm to learn hypotheses from training examples by refining the most specific and most general hypotheses based on positive and negative examples has been successfully executed.

**EX.NO: 4 IMPLEMENT LINEAR REGRESSION**

**AIM**

The aim of the code is to perform linear regression on a dataset to model the relationship between two variables and visualize this relationship with a scatter plot and a regression line.

**ALGORITHM**

1. **Import Libraries**: Import matplotlib for plotting and scipy.stats for performing linear regression.

2**. Define Data**: Specify the x and y data points representing the independent and dependent variables.

3. **Compute Regression**: Use stats.linregress to calculate the slope, intercept, and other statistics for the linear regression model.

4**. Apply Model:** Define a function to calculate the predicted y values based on the linear regression model and apply it to the x values.

5**. Plot Results**: Create a scatter plot of the original data points and plot the regression line, then display the plot.

**PROGRAM**

import matplotlib.pyplot as plt

from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]

y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std\_err = stats.linregress(x, y)

def myfunc(x):

return slope \* x + intercept

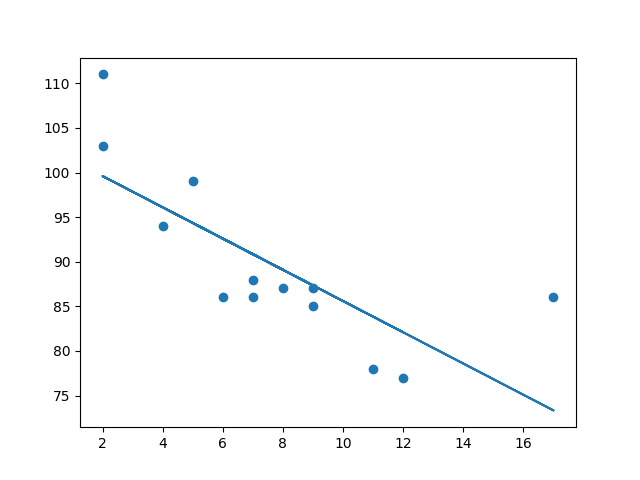
mymodel = list(map(myfunc, x))

plt.scatter(x, y)

plt.plot(x, mymodel)

plt.show()

**OUTPUT**



**RESULT**

Thusto perform linear regression on a dataset to model the relationship between two variables and visualize this relationship with a scatter plot and a regression line has been successfully executed.

**EX.NO:5 IMPLEMENT BACK-PROPOGATION ALGORITHM**

**AIM**

The aim of the code is to implement a simple neural network with one hidden layer to perform regression using forward propagation and backpropagation for training.

**ALGORITHM**

1. **Initialize Data:** Normalize input features X and target values y.

2.**Define Functions:** Implement the sigmoid activation function and its derivative for use in the neural network.

3. **Initialize Variables:** Set the number of epochs, learning rate, and initialize weights and biases for the input, hidden, and output layers.

4. **Train the Network:**

- **Forward Propagation:** Compute the output of the network by passing inputs through the hidden layer and output layer.

- **Backpropagation:** Calculate errors and gradients to update weights and biases using the learning rate.

5. **Output Results:** Print the input data, actual output values, and the predicted output after training.

**PROGRAM**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X / np.amax(X, axis=0) # Maximum of X array longitudinally

y = y / 100

# Sigmoid Function

def sigmoid(x):

return 1 / (1 + np.exp(-x))

# Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

return x \* (1 - x)

# Variable initialization

epoch = 5000 # Setting training iterations

lr = 0.1 # Setting learning rate

inputlayer\_neurons = 2 # Number of features in data set

hiddenlayer\_neurons = 3 # Number of hidden layer neurons

output\_neurons = 1 # Number of neurons at output layer

# Weight and bias initialization

wh = np.random.uniform(size=(inputlayer\_neurons, hiddenlayer\_neurons))

bh = np.random.uniform(size=(1, hiddenlayer\_neurons))

wout = np.random.uniform(size=(hiddenlayer\_neurons, output\_neurons))

bout = np.random.uniform(size=(1, output\_neurons))

# Training

for i in range(epoch):

# Forward Propagation

hinp1 = np.dot(X, wh)

hinp = hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1 = np.dot(hlayer\_act, wout)

outinp = outinp1 + bout

output = sigmoid(outinp)

# Backpropagation

EO = y - output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

# How much hidden layer weights contributed to error

hiddengrad = derivatives\_sigmoid(hlayer\_act)

d\_hiddenlayer = EH \* hiddengrad

# Update weights and biases

wout += hlayer\_act.T.dot(d\_output) \* lr

wh += X.T.dot(d\_hiddenlayer) \* lr

print("Input:\n" + str(X))

print("Actual Output:\n" + str(y))

print("Predicted Output:\n", output)

**OUTPUT**

**Input:**

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

**Actual Output:**

[[0.92]

[0.86]

[0.89]]

**Predicted Output**:

[[0.89597569]

[0.87889218]

[0.89500469]]

**RESULT**

Thusto implement a simple neural network with one hidden layer to perform regression using forward propagation and back propagation for training has been successfully executed.

**EX.NO:6 IMPLEMENT SUPPORT VECTOR MACHINE ALGORITHM**

**AIM**

The aim of the code is to train a Support Vector Machine (SVM) with a radial basis function (RBF) kernel on a subset of features from the breast cancer dataset and visualize the decision boundary along with the data points.

**ALGORITHM**

1. **Import Libraries**: Load necessary libraries for data handling, plotting, and machine learning.

2. **Load and Prepare Data:** Load the breast cancer dataset and select the first two features for simplicity.

3. **Train the Model:** Create and train an SVM model with an RBF kernel using the selected features and target labels.

4. **Plot Decision Boundary:** Use DecisionBoundaryDisplay to visualize the decision boundary of the trained model.

5. **Scatter Plot:** Plot the data points with their class labels to show how the decision boundary separates different classes.

**PROGRAM**

from sklearn.datasets import load\_breast\_cancer

import matplotlib.pyplot as plt

from sklearn.inspection import DecisionBoundaryDisplay

from sklearn.svm import SVC

# Load the datasets

cancer = load\_breast\_cancer()

X = cancer.data[:, :2]

y = cancer.target

# Build and train the model

svm = SVC(kernel=”rbf”, gamma=0.5, C=1.0)

svm.fit(X, y)

# Plot Decision Boundary

DecisionBoundaryDisplay.from\_estimator(

svm,

X,

response\_method=”predict”,

cmap=plt.cm.Spectral,

alpha=0.8,

xlabel=cancer.feature\_names[0],

ylabel=cancer.feature\_names[1]

)

# Scatter plot

plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolors=”k”)

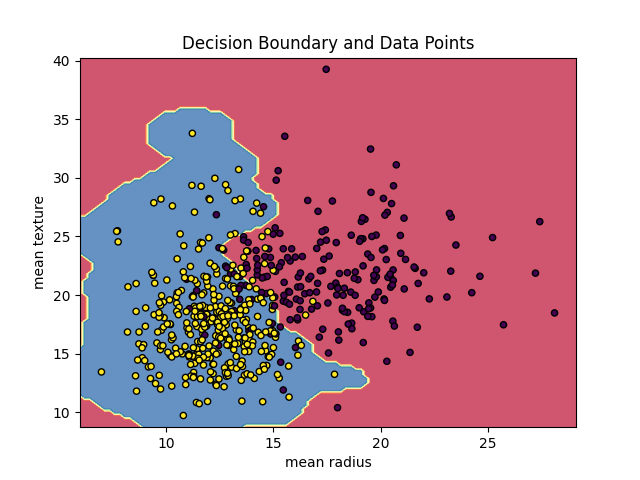
plt.title(‘Decision Boundary and Data Points’)

plt.xlabel(cancer.feature\_names[0])

plt.ylabel(cancer.feature\_names[1])

plt.show()

**OUTPUT**



**RESULT**

Thusto train a Support Vector Machine (SVM) with a radial basis function (RBF) kernel on a subset of features from the breast cancer dataset and visualize the decision boundary along with the data points has been successfully executed.

**EX.NO:7 IMPLEMENT DECISION TREE ALGORITHM**

**AIM**

The aim of the code is to train a Decision Tree Classifier on a dataset and visualize the trained decision tree structure.

**ALGORITHM**

1. **Import Libraries**: Load necessary libraries for data manipulation, machine learning, and plotting.

2. **Load and Prepare Data:** Read the dataset from a CSV file and map categorical variables to numerical values.

3. **Define Features and Target**: Specify the feature columns and target column for training the Decision Tree.

4. **Train the Model:** Create and fit a DecisionTreeClassifier using the prepared data.

5. **Visualize and Save Tree:** Plot the decision tree structure and save the plot to the standard output.

**PROGRAM**

import sys

import matplotlib

matplotlib.use(‘Agg’)

import pandas

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

df = pandas.read\_csv(“data.csv”)

d = {‘UK’: 0, ‘USA’: 1, ‘N’: 2}

df[‘Nationality’] = df[‘Nationality’].map(d)

d = {‘YES’: 1, ‘NO’: 0}

df[‘Go’] = df[‘Go’].map(d)

features = [‘Age’, ‘Experience’, ‘Rank’, ‘Nationality’]

X = df[features]

y = df[‘Go’]

dtree = DecisionTreeClassifier()

dtree = dtree.fit(X, y)

tree.plot\_tree(dtree, feature\_names=features)

plt.savefig(sys.stdout.buffer)

sys.stdout.flush()

**data.csv:**

Age,Experience,Rank,Nationality,Go

36,10,9,UK,NO

42,12,4,USA,NO

23,4,6,N,NO

52,4,4,USA,NO

43,21,8,USA,YES

44,14,5,UK,NO

66,3,7,N,YES

35,14,9,UK,YES

52,13,7,N,YES

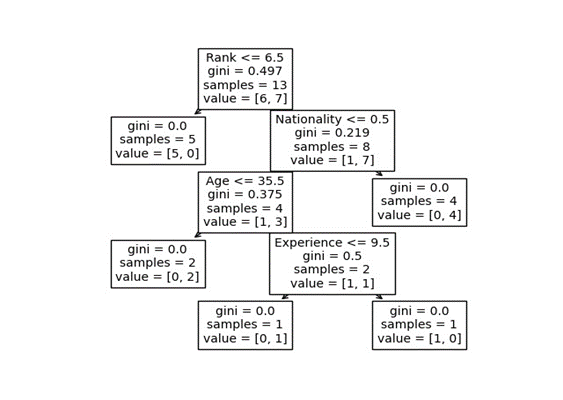
35,5,9,N,YES

24,3,5,USA,NO

18, 3,7,UK,YES

45, 9,9,UK,YES

**OUTPUT**

****

**RESULT**

Thusto train a Decision Tree Classifier on a dataset and visualize the trained decision tree structure has been successfully executed.

**EX.NO:8 IMPLEMENT K-NEAREST NEIGHBORS ALGORITHM**

**AIM**

The aim of the code is to visualize a dataset with class labels and use a k-Nearest Neighbors (k-NN) classifier to predict the class of a new data point, then display this prediction on a scatter plot.

**ALGORITHM**

1. **Import Libraries**: Import matplotlib for plotting the scatter plot.

2. **Define Data**: Specify the data points (x, y) and their associated class labels.

3. **Plot Existing Data:** Create a scatter plot of the existing data points with colors representing their class labels.

4. **Predict New Point:** Use a k-Nearest Neighbors (k-NN) classifier (assumed to be defined elsewhere) to predict the class of a new data point.

5. **Visualize Prediction:** Add the new data point to the scatter plot, display its predicted class, and show the updated plot.

**Note:** Ensure the knn classifier is defined and trained before calling knn.predict(new\_point)

**PROGRAM**

import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14, 8, 10, 12]

y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]

plt.scatter(x, y, c=classes)

new\_x = 8

new\_y = 21

new\_point = [(new\_x, new\_y)]

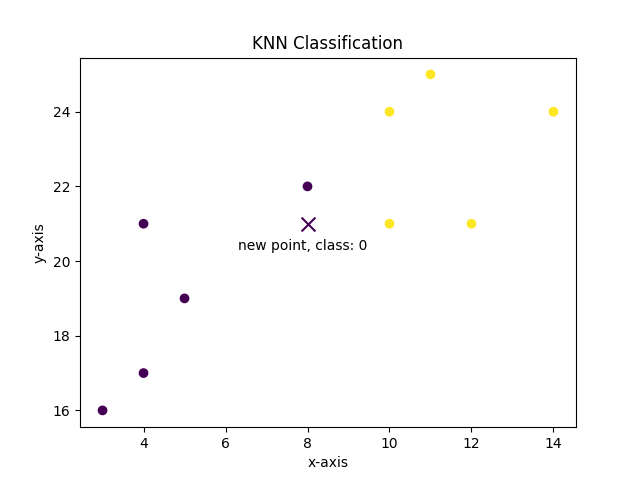
prediction = knn.predict(new\_point)

plt.scatter(x + [new\_x], y + [new\_y], c=classes + [prediction[0]])

plt.text(x=new\_x-1.7, y=new\_y-0.7, s=f"new point, class: {prediction[0]}")

plt.show()

**OUTPUT**

****

**RESULT**

Thusto visualize a dataset with class labels and use a k-Nearest Neighbors (k-NN) classifier to predict the class of a new data point, then display this prediction on a scatter plot has been successfully executed.

**EX.NO:9 IMPLEMENT K- MEANS CLUSTERING ALGORITHM**

**AIM**

The aim of the code is to determine the optimal number of clusters for K-Means clustering using the Elbow method, which helps identify the point where adding more clusters does not significantly improve the model.

**ALGORITHM**

1. **Import Libraries**: Import matplotlib for plotting and KMeans from sklearn for clustering.

2. **Prepare Data**: Combine x and y into a list of tuples representing data points.

3. **Compute Inertia**: For each number of clusters from 1 to 10, fit a K-Means model and record the inertia (sum of squared distances to nearest cluster center).

4. **Plot Inertia**: Create a plot of the number of clusters versus inertia to visualize the Elbow method.

5. **Interpret Results**: Identify the "elbow" point in the plot, where inertia starts to decrease more slowly, indicating the optimal number of clusters.

**PROGRAM**

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]

y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))

inertias = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i)

kmeans.fit(data)

inertias.append(kmeans.inertia\_)

plt.plot(range(1, 11), inertias, marker='o')

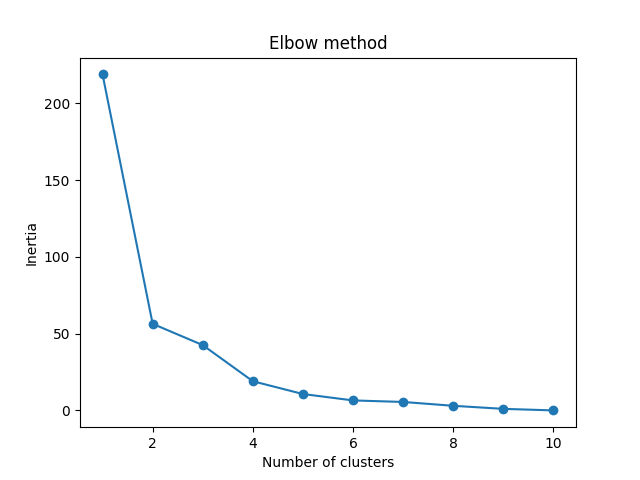
plt.title('Elbow method')

plt.xlabel('Number of clusters')

plt.ylabel('Inertia')

plt.show()

**OUTPUT**



**RESULT**

Thusto determine the optimal number of clusters for K-Means clustering using the Elbow method, which helps identify the point where adding more clusters does not significantly improve the model has been successfully executed.

**PROJECTS**

1. **PREDICTIVE TEXT GENERATOR**

* **Description:** This project involves creating a text generation model that predicts the next word or sequence of words in a sentence based on the context provided by previous words. The goal is to generate coherent and contextually appropriate text.
* **Explanation:** Students will use natural language processing (NLP) techniques to build the predictive text generator. They will start by collecting and pre-processing a large corpus of text data. Techniques such as n-grams, which predict the next word based on the previous n-1 words, and more advanced models like Long Short-Term Memory (LSTM) networks or Transformer models (e.g., GPT) will be employed. The project requires understanding of sequence modelling and training neural networks for text generation. Students will evaluate their models based on metrics such as perplexity and coherence of generated text.

**2. AI CHAT BOT**

* **Description:** This project focuses on developing a Chabot capable of understanding and responding to user queries in natural language. The Chabot should be able to engage in meaningful conversations and handle a range of topics or tasks.
* **Explanation:** Students will design and implement a Chabot using natural language understanding (NLU) and dialogue management techniques. They will start by defining the intents (user goals) and entities (key information) the Chabot should recognize. Tools such as Rasa, Dialog flow, or libraries like NLTK and spaCy can be used to process user inputs and generate responses. Students will build a conversation flow, handle user inputs, and integrate external APIs if needed. This project helps students understand conversational AI, user intent recognition, and response generation.

3. **VOICE-ACTIVATED SYSTEM**

* **Description:** This project involves creating a voice-activated system that can recognize spoken commands and perform corresponding actions or responses. The system should be able to interpret voice commands accurately and execute tasks based on them.
* **Explanation:** Students will use speech recognition technologies to develop the system. They will integrate APIs like Google Speech-to-Text or use libraries like CMU Sphinx to convert spoken language into text. The system will then process this text to determine the appropriate action, such as controlling smart devices or performing specific tasks. Students will work on handling different accents, background noise, and command variations. The project provides experience with speech processing, voice command recognition, and practical applications of voice-controlled systems.

4. **RECOMMENDATION SYSTEM**

* **Description:** Build a recommendation engine that suggests products or content to users based on their historical behavior and preferences. The system should provide personalized recommendations to enhance user experience.
* **Explanation:** Students will develop a recommendation system using techniques such as collaborative filtering (which relies on user-item interactions) and content-based filtering (which considers item attributes). They may also explore hybrid approaches that combine both techniques. The project involves data collection, feature extraction, model training, and evaluation. Tools and libraries like Surprise, TensorFlow, or Scikit-learn can be used for building recommendation algorithms. This project helps students understand personalization strategies and recommendation algorithms.

.

**5.**  **DISEASE PREDICTION USING MACHINE LEARNING**

* **Description:** This project involves creating a machine learning model that predicts the likelihood of a disease based on patient data, such as symptoms, medical history, and demographic information.
* **Explanation:** Students will use machine learning algorithms to analyze medical datasets and predict disease outcomes. The process includes data preprocessing (handling missing values, normalization), feature selection, and model training using algorithms like logistic regression, decision trees, or ensemble methods. Evaluation metrics such as accuracy, precision, recall, and F1 score will be used to assess model performance. This project provides insights into predictive modeling, healthcare data analysis, and the practical application of machine learning in medicine.

**6.**  **FAKE NEWS DETECTION USING MACHINE LEARNING**

* **Description:** Develop a machine learning model to classify news articles as real or fake based on their content. The goal is to create a system that can help identify and combat misinformation.
* **Explanation:** Students will collect and preprocess news data, including text normalization and feature extraction. They will use classification algorithms such as Naive Bayes, Support Vector Machines (SVM), or deep learning models to build the fake news detection system. The project involves training the model on labeled datasets, evaluating its performance, and fine-tuning it for better accuracy. This project emphasizes the importance of reliable information and equips students with skills in text classification and misinformation detection.

7. **SENTIMENT ANALYSIS WITH TWITTER/FACEBOOK DATA**

* **Description:** Analyze social media data to determine the sentiment expressed in tweets or Facebook posts. The system should classify sentiments into categories such as positive, negative, or neutral.
* **Explanation:** Students will collect tweets or Facebook posts using APIs and preprocess the data for sentiment analysis. Techniques such as tokenization, lemmatization, and feature extraction will be used. Students will implement sentiment analysis models using libraries like TextBlob, VADER, or deep learning models such as LSTM or BERT. They will evaluate the model's performance using metrics like accuracy, precision, and recall. This project demonstrates the application of sentiment analysis in understanding public opinion and social media trends.

**8. HANDWRITTEN DIGIT RECOGNITION USING NEURAL NETWORK**

* **Description:** Develop a neural network model to recognize and classify handwritten digits from images. The goal is to create a system that can accurately identify digits from the MNIST dataset.
* **Explanation:** Students will use Convolutional Neural Networks (CNNs), which are well-suited for image recognition tasks, to build the digit recognition model. The project involves preprocessing the MNIST dataset, designing the CNN architecture, training the model, and evaluating its performance using metrics such as accuracy and loss. Students will gain practical experience with image classification and deep learning techniques. This project provides insights into computer vision and neural network training.