# SIDDHARTH INSTITUTE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS)

(Approved by AICTE, New Delhi& Affiliated to JNTUA, Ananthapuramu) (Accredited by NBA for Civil, EEE, Mech., ECE & CSE

Accredited by NAAC with ‘A+’ Grade)

Puttur -517583, Tirupati District, A.P. (India)



**Department of Computer Science and Engineering**

# (CSE with Specialization in Artificial Intelligence & Machine Learning)

**(**20CS0908**) ADVANCED MACHINE LEARNING LAB**

**III B.Tech - II Semester**

Lab Observation Book

Academic Year:

Name

:

Roll. Number : Year & Branch:

Specialization :



# SIDDHARTH INSTITUTE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS)

(Approved by AICTE, New Delhi& Affiliated to JNTUA, Ananthapuramu) (Accredited by NBA for Civil, EEE, Mech., ECE & CSE

Accredited by NAAC with ‘A+’ Grade)

Puttur -517583, Tirupati District, A.P. (India)

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## INSTITUTE VISION

To emerge as one of the premier institutions through excellence in education and research, producing globally competent and ethically strong professionals and entrepreneurs

## INSTITUTE MISSION

|  |  |
| --- | --- |
| **M1:** | Imparting high-quality technical and management education through the state-of-  the- art resources. |
| **M2:** | Creating an eco-system to conduct independent and collaborative research for the  betterment of the society |
| **M3:** | Promoting entrepreneurial skills and inculcating ethics for the socio-economic  development of the nation. |

**DEPARTMENT VISION**

To impart quality education and research in Computer Science and Engineering for producing technically competent and ethically strong IT professionals with contemporary knowledge

**DEPARTMENT MISSION**

|  |  |
| --- | --- |
| **M1:** | Achieving academic excellence in computer science through effective pedagogy,  modern curriculum and state-of-art computing facilities. |
| **M2:** | Encouraging innovative research in Computer Science and Engineering by  collaborating with Industry and Premier Institutions to serve the nation. |
| **M3:** | Empowering the students by inculcating professional behavior, strong ethical  values and leadership abilities |



# SIDDHARTH INSTITUTE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS)

(Approved by AICTE, New Delhi& Affiliated to JNTUA, Ananthapuramu) (Accredited by NBA for Civil, EEE, Mech., ECE & CSE

Accredited by NAAC with ‘A+’ Grade)

Puttur -517583, Tirupati District, A.P. (India)

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Program Outcomes**

**PO1: Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

**PO2: 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.

**PO3: 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.

**PO4: 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.

**PO5: Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

**PO6: 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.

**PO7: 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. **PO8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**PO9: Individual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**PO10: 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.

**PO11: 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.

**PO12: 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.



# SIDDHARTH INSTITUTE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS)

(Approved by AICTE, New Delhi& Affiliated to JNTUA, Ananthapuramu) (Accredited by NBA for Civil, EEE, Mech., ECE & CSE

Accredited by NAAC with ‘A+’ Grade)

Puttur -517583, Tirupati District, A.P. (India)

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

|  |  |
| --- | --- |
| **PEO1:** | To provide software solutions for arising problems in diverse areas with strong  knowledge in innovative technologies of computer science. |
| **PEO2:** | To serve in IT industry as professionals and entrepreneurs or in pursuit of higher  education and research. |
| **PEO3:** | To attain professional etiquette, soft skills, leadership, ethical values meld with a  commitment for lifelong learning. |

**PROGRAM SPECIFIC OUTCOMES (PSOs)**

|  |  |
| --- | --- |
| **PSO1:** | **Analysis & Design:** Ability to design, develop and deploy customized applications in all applicable domains using various algorithms and programming  languages. |
| **PSO2:** | **Computational Logic:** Ability to visualize and configure computational need in  terms of hardware and software to provide solutions for various complex applications. |
| **PSO3:** | **Software Development:** Ability to apply standard procedures, tools and strategies  for software development. |

## COURSE OBJECTIVES

The Objectives of this Course:

*1. Develop an appreciation for what is involved in learning from data.*

*2. Demonstrate a wide variety of learning algorithms.*

*3. Understand the important of supervised learning and their applications.*

*4. Understand unsupervised learning like clustering and EM algorithms.*

*5. Demonstrate how to apply a variety of learning algorithms to various fields.*

## COURSE OUTCOMES

On successful completion of the course students will be able to

*1. To articulate a machine learning problem*

*2. Domain Knowledge for Productive use of Machine Learning and Diversity of Data.*

*3. Apply Supervised, unsupervised algorithms for real time applications*

*4. Analyze on Statistics in learning techniques and Logistic Regression*

*5. Develop various models using Support Vector Machines and Perceptron Algorithm*

*6. Select an appropriate pattern analysis tool for analyzing data in each feature space.*

## LIST OF EXPERIMENTS

* 1. Implement k-nearest neighbors classification using python.
  2. Extract the data from database using python
  3. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file
  4. Implement an algorithm to demonstrate the significance of genetic algorithm
  5. Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of k-means clustering with 3 means (i.e., 3 centroids) VAR1 VAR2 CLASS

1.713 1.586 0

0.180 1.786 1

0.353 1.240 1

0.940 1.566 0

1.486 0.759 1

1.266 1.106 0

1.540 0.419 1

0.459 1.799 1

0.773 0.186 1

* 1. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
  2. Write a program to implement Principle Component Analysis for Dimensionality Reduction.
  3. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
  4. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.
  5. Write a program to demonstrate the working of the decision tree-based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Date** | **Name of the Experiment** | **Page No.** | **Signature of Faculty** |
| 1 |  | **K-Nearest Neighbors Classification using python** |  |  |
| 2 |  | **Extract the data from database using python** |  |  |
| 3 |  | **Implement and demonstrate the FIND-S algorithm** |  |  |
| 4 |  | **Demonstrate the significance of genetic algorithm** |  |  |
| 5 |  | **Classifications for nine combinations of VAR1 and VAR2 predict** |  |  |
| 6 |  | **Apply EM algorithm to cluster a set of data stored in a .CSV file** |  |  |
| 7 |  | **Implement Principle Component Analysis for Dimensionality Reduction** |  |  |
| 8 |  | **K-Nearest Neighbor algorithm to classify the iris data set** |  |  |
| 9 |  | **Non-parametric Locally Weighted Regression algorithm in order to fit data** |  |  |
| 10 |  | **Demonstrate the working of the decision tree-based ID3 algorithm** |  |  |

|  |  |  |
| --- | --- | --- |
| **Ex.No. 1** | **K-Nearest Neighbors Classification using python** | **Date:** |

## AIM:

To Implement k-nearest neighbors classification using python.

**Description:**

We load the famous Iris dataset from scikit-learn using the load\_iris() function. We then split the dataset into training and testing sets using the train\_test\_split() function. Next, we create a K Neighbors Classifier object with k set to 3, and fit the model to the training data using the fit() method. We make predictions on the testing data using the predict() method, and calculate the accuracy of the model using the accuracy\_score() function from scikit-learn. Note that the Iris dataset is a well-known dataset for multi-class classification, where the goal is to classify iris flowers into one of three species based on four features (sepal length, sepal width, petal length, petal width).

## Implementation:

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_iris

# Load the iris dataset

iris = load\_iris()

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.2, random\_state=42)

# Create a KNeighborsClassifier object with k=3

k = 3

knn = KNeighborsClassifier(n\_neighbors=k)

# Fit the model to the training data

knn.fit(X\_train, y\_train)

# Make predictions on the testing data

y\_pred = knn.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

**OUTPUT:**

## Result:

.

|  |  |  |
| --- | --- | --- |
| **Ex.No. 2** | **Extract the data from database using python** | **Date:** |

**AIM:**

To extract the data from database using python.

## Description:

## Connect to the SQLite database using the connect() function and create a cursor object to interact with the database. We then execute a query using the execute() method and fetch all the rows using the fetchall() method. Finally, we loop through the fetched rows and print them. Don't forget to close the database connection using the close() method after you're done with it. Note that you need to replace 'example.db' with the actual name of your database file, and 'my\_table' with the actual name of the table from which you want to fetch data. Also, you may need to modify the query according to your specific requirements. Other database libraries such as MySQLdb, psycopg2, or pymongo may have slightly different syntax and usage, so you would need to refer to their respective documentation.

**Source Code:**

import sqlite3

# Connect to the database

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

# Execute a query to retrieve data

cursor.execute('SELECT \* FROM example\_table')

rows = cursor.fetchall() # Fetch all rows returned by the query

# Process the retrieved data

for row in rows:

print(row) # You can access the columns in the row using row[0], row[1], etc.

# Close the database connection

conn.close()

**Output**:

## Result:

|  |  |  |
| --- | --- | --- |
| **Ex.No. 3** | **Implement and demonstrate the FIND-S algorithm** | **Date:** |

**AIM:**

To Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file

**Description:**

The **read\_training\_data()** function reads the training data from the CSV file using the **csv** module in Python. The **find\_s\_algorithm()** function implements the FIND-S algorithm, which finds the most specific hypothesis based on the training data. The **demonstrate\_find\_s\_algorithm()** function demonstrates the FIND-S algorithm by printing the training data and the most specific hypothesis. You can replace **'training\_data.csv'** with the actual name of your CSV file. The assumption is that the last column in the CSV file represents the class label, and 'Yes' denotes positive examples. The algorithm iterates through the training data and updates the specific hypothesis based on positive examples. The most specific hypothesis is printed at the end. Note that this implementation assumes binary attributes and does not handle missing values or continuous attributes. You may need to modify the code accordingly based on your specific requirements.

**Source Code:**

import csv

# Read the training data from CSV file

def read\_training\_data(file\_name):

data = []

with open(file\_name, 'r') as file:

csv\_reader = csv.reader(file)

for row in csv\_reader:

data.append(row)

return data

# Implement FIND-S algorithm

def find\_s\_algorithm(training\_data):

specific\_hypothesis = ['0'] \* (len(training\_data[0]) - 1)

for row in training\_data:

if row[-1] == 'Yes': # positive example

for i in range(len(row) - 1):

if specific\_hypothesis[i] == '0':

specific\_hypothesis[i] = row[i]

elif specific\_hypothesis[i] != row[i]:

specific\_hypothesis[i] = '?'

return specific\_hypothesis

# Demonstrate FIND-S algorithm

def demonstrate\_find\_s\_algorithm(training\_data):

print("Training Data:")

for row in training\_data:

print(row)

print("\nMost Specific Hypothesis:")

specific\_hypothesis = find\_s\_algorithm(training\_data)

print(specific\_hypothesis)

# Driver code

if \_\_name\_\_ == "\_\_main\_\_":

# Replace 'training\_data.csv' with the actual name of your CSV file

training\_data = read\_training\_data('training\_data.csv')

demonstrate\_find\_s\_algorithm(training\_data)

**OUTPUT:**

.

## Result:

|  |  |  |
| --- | --- | --- |
| **Ex.No. 4** | **Demonstrate the significance of genetic algorithm** | **Date:** |

**AIM:**

To Implement an algorithm to demonstrate the significance of genetic algorithm

## Description:

## Genetic algorithm that maximizes the sum of a sequence of integers. The algorithm consists of generating an initial population of random individuals, evaluating their fitness using a fitness function, selecting the top individuals as parents for crossover, performing crossover to create offspring, mutating some offspring, and updating the population with offspring. The process is repeated for a number of generations. The algorithm demonstrates the significance of genetic algorithms in optimization and search problems, as it can efficiently explore the search space and converge towards optimal or near-optimal solutions. The significance of genetic algorithms lies in their ability to effectively handle complex optimization problems, search large solution spaces, and find good solutions in a reasonable amount of time.

## Source Code:

import random

# Define a fitness function

def fitness\_function(individual):

# Define the fitness function based on the problem

# In this example, we use a simple fitness function that maximizes the sum of values

return sum(individual)

# Define the genetic algorithm

def genetic\_algorithm(population, fitness\_func, generations, mutation\_rate):

for gen in range(generations):

# Calculate fitness for each individual in the population

fitness\_scores = [fitness\_func(individual) for individual in population]

# Select top individuals as parents for crossover

parents = [population[i] for i in range(len(population)) if random.random() < fitness\_scores[i]/sum(fitness\_scores)]

# Perform crossover to create offspring

offspring = []

while len(offspring) < len(population):

parent1 = random.choice(parents)

parent2 = random.choice(parents)

crossover\_point = random.randint(1, len(parent1)-1)

child1 = parent1[:crossover\_point] + parent2[crossover\_point:]

child2 = parent2[:crossover\_point] + parent1[crossover\_point:]

offspring.append(child1)

offspring.append(child2)

# Perform mutation

for i in range(len(offspring)):

if random.random() < mutation\_rate:

mutation\_point = random.randint(0, len(offspring[i])-1)

mutation\_value = random.randint(0, 9)

offspring[i][mutation\_point] = mutation\_value

# Update the population with offspring

population = offspring

# Return the final population

return population

# Demonstrate the significance of genetic algorithm

def demonstrate\_genetic\_algorithm():

# Define the problem, e.g., finding a sequence of integers that maximizes their sum

problem\_size = 10

population\_size = 100

generations = 100

mutation\_rate = 0.1

# Generate an initial population of random individuals

population = [[random.randint(0, 9) for \_ in range(problem\_size)] for \_ in range(population\_size)]

# Run the genetic algorithm

final\_population = genetic\_algorithm(population, fitness\_function, generations, mutation\_rate)

# Find the best individual in the final population

best\_individual = max(final\_population, key=fitness\_function)

print("Initial Population:")

print(population)

print("\nFinal Population:")

print(final\_population)

print("\nBest Individual:")

print(best\_individual)

print("\nFitness Score of Best Individual:")

print(fitness\_function(best\_individual))

# Driver code

if \_\_name\_\_ == "\_\_main\_\_":

demonstrate\_genetic\_algorithm()

**Output:**

## Result:

|  |  |  |
| --- | --- | --- |
| **Ex.No. 5** | **Classifications for nine combinations of VAR1 and VAR2 predict** | **Date:** |

**AIM:**

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

## Description:

## Represents the classification for each combination of VAR1 and VAR2. The KMeans class from scikit-learn is used to perform k-means clustering with 3 clusters. The fit() method is used to fit the data to the k-means model, and the predict() method is used to predict the cluster assignment for the new case with VAR1=0.906 and VAR2=0.606. Finally, based on the predicted cluster, we determine the corresponding classification for the new case.

**Source Code:**

import numpy as np

from sklearn.cluster import KMeans

# Given data specifying classifications for nine combinations of VAR1 and VAR2

data = np.array([[1.713, 1.586, 0],

[0.180, 1.786, 1],

[0.353, 1.240, 1],

[0.940, 1.566, 0],

[1.486, 0.759, 1],

[1.266, 1.106, 0],

[1.540, 0.419, 1],

[0.459, 1.799, 1],

[0.773, 0.186, 1]])

# Preprocessing data - not required in this case as data is already given

# Perform k-means clustering with 3 means

kmeans = KMeans(n\_clusters=3, random\_state=0)

kmeans.fit(data[:, :2]) # Use only the first two columns (VAR1 and VAR2) for clustering

# Predict for a new case with VAR1=0.906 and VAR2=0.606

new\_case = np.array([[0.906, 0.606]])

predicted\_cluster = kmeans.predict(new\_case)

# Determine the classification based on the predicted cluster

if predicted\_cluster == 0:

classification = "Class 1"

elif predicted\_cluster == 1:

classification = "Class 2"

else:

classification = "Class 3"

print("Predicted classification for VAR1=0.906 and VAR2=0.606: ", classification)

**Output:**

## Result:

|  |  |  |
| --- | --- | --- |
| **Ex.No. 6** | **Apply EM algorithm to cluster a set of data stored in a .CSV file** | **Date:** |

**AIM:**

To Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API.

## Description:

## First load the data from a .CSV file and extract the features from the dataset. We then standardize the features using Standard Scaler from scikit-learn to ensure that they are on the same scale. Next, we apply the EM algorithm and the k-Means algorithm to cluster the data into 3 clusters. We print the clustering labels obtained from both algorithms and calculate the silhouette score, which is a measure of the quality of clustering. Higher silhouette score indicates better clustering quality.By comparing the results of the EM algorithm and the k-Means algorithm, we can evaluate the quality of clustering. The silhouette score is a measure of how similar the data points are within the same cluster compared to other clusters. A higher silhouette score indicates that the data points within the same cluster are more similar to each other, and the clusters are well-separated. From the comparison, we can comment on the quality of clustering achieved by each algorithm. Higher silhouette score indicates better clustering quality, while a lower score indicates poorer clustering quality. Additionally, visualizing the clusters using plots or other methods can provide further insights into the quality of clustering. It's important to note that the results may vary depending on the dataset and the specific parameters used in each algorithm, so it's recommended to experiment with different settings and evaluate the results based on the specific requirements of the problem at hand.

## Source Code:

import pandas as pd

import numpy as np

from sklearn.mixture import GaussianMixture

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score

# Load the data from .CSV file

data = pd.read\_csv('D:\ML\credit\_data.csv')

# Extract features from the data

X = data.iloc[:, :-1].values

# Standardize the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# EM Algorithm

em = GaussianMixture(n\_components=3) # specify number of clusters

em.fit(X\_scaled)

em\_labels = em.predict(X\_scaled)

# K-Means Algorithm

kmeans = KMeans(n\_clusters=3) # specify number of clusters

kmeans.fit(X\_scaled)

kmeans\_labels = kmeans.labels\_

# Compare the results of EM and K-Means algorithms

print("EM Algorithm Clustering Labels:")

print(em\_labels)

print("K-Means Algorithm Clustering Labels:")

print(kmeans\_labels)

# Calculate silhouette score to evaluate the quality of clustering

em\_silhouette\_score = silhouette\_score(X\_scaled, em\_labels)

kmeans\_silhouette\_score = silhouette\_score(X\_scaled, kmeans\_labels)

print("Silhouette Score - EM Algorithm: ", em\_silhouette\_score)

print("Silhouette Score - K-Means Algorithm: ", kmeans\_silhouette\_score)

## Output

## 

## Result:

|  |  |  |
| --- | --- | --- |
| **Ex.No. 7** | **Implement Principle Component Analysis for Dimensionality Reduction** | **Date:** |

**AIM**

To implement Principle Component Analysis for Dimensionality Reduction.

## Description:

## We first load the data from a .CSV file and extract the features from the dataset. We then standardize the features using StandardScaler from scikit-learn to ensure that they are on the same scale. Next, we apply PCA for dimensionality reduction by creating an instance of PCA with the desired number of components (dimensions) to reduce to, and then calling the fit\_transform() method on the standardized data. The reduced dimensionality data is stored in the X\_pca variable, which can be used for further analysis or visualization.Note that the number of components to reduce to in PCA can be adjusted based on the specific requirements of the problem at hand. Choosing the right number of components is important as it determines the amount of information retained after dimensionality reduction. Additionally, visualizing the reduced dimensionality data using plots or other methods can provide insights into the structure of the data in the reduced space. It's recommended to experiment with different settings and evaluate the results based on the specific requirements of the problem.

## Source Code:

import pandas as pd

import numpy as np

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

# Load the data from a .CSV file

data = pd.read\_csv('D:\ML\credit\_data.csv')

# Extract features from the data

X = data.iloc[:, :-1].values

# Standardize the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Perform PCA for dimensionality reduction

pca = PCA(n\_components=2) # specify the number of components (dimensions) to reduce to

X\_pca = pca.fit\_transform(X\_scaled)

# Print the reduced dimensionality data

print("Reduced Dimensionality Data (First 5 rows):")

print(X\_pca[:5, :])

## Output

## Result:

|  |  |  |
| --- | --- | --- |
| **Ex.No. 8** | **K-Nearest Neighbor algorithm to classify the iris data set** | **Date:** |

**AIM:**

Implements k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

## Description:

## CSV file and extract the features (sepal length, sepal width, petal length, petal width) and labels (species) from the dataset. We then split the dataset into a train set and a test set using the train\_test\_split() function from scikit-learn. Next, we initialize a k-NN classifier with the desired number of neighbors to consider (in this example, 5). We then train the classifier on the train set using the fit() method. After that, we predict the labels of the test set using the predict() method and evaluate the accuracy of the classifier using the accuracy\_score() function. Finally, we print both correct and wrong predictions, as well as the confusion matrix to further evaluate the performance of the classifier. Note that the k-NN algorithm is a simple and intuitive method for classification, but it may not always be the best choice depending on the specific problem and dataset. It's recommended to experiment with different values of k (number of neighbors) and other hyperparameters, as well as try other classification algorithms to find the best approach for a given problem.

## Source Code:

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Load the Iris dataset

data = pd.read\_csv('iris.csv')

# Extract features and labels from the dataset

X = data.iloc[:, :-1].values

y = data.iloc[:, -1].values

# Split the dataset into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize k-NN classifier

knn = KNeighborsClassifier(n\_neighbors=5) # specify the number of neighbors to consider

# Train the classifier

knn.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = knn.predict(X\_test)

# Evaluate the accuracy of the classifier

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy: ", accuracy)

# Print correct and wrong predictions

correct\_predictions = 0

wrong\_predictions = 0

for i in range(len(y\_test)):

if y\_test[i] == y\_pred[i]:

print("Correct prediction: Expected:", y\_test[i], " Predicted:", y\_pred[i])

correct\_predictions += 1

else:

print("Wrong prediction: Expected:", y\_test[i], " Predicted:", y\_pred[i])

wrong\_predictions += 1

print("Total Correct Predictions: ", correct\_predictions)

print("Total Wrong Predictions: ", wrong\_predictions)

# Print confusion matrix

confusion = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", confusion)

## OUTPUT

## Result:

|  |  |  |
| --- | --- | --- |
| **Ex.No. 9** | **Non-parametric Locally Weighted Regression algorithm in order to fit data** | **Date:** |

## AIM:

## To Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

## Description:

## Generate synthetic data using the numpy library, which consists of X values and corresponding y values. Then, we implement the LWR algorithm as a function locally\_ weighted\_regression() that takes in the training data (X and y), the test point (x) for which we want to make a prediction, and the bandwidth parameter (tau) which controls the width of the weight function. The algorithm calculates weights for each training data point based on their distance to the test point using a Gaussian kernel, and then fits a linear regression model to the weighted data points using the normal equation. Finally, we use the trained model to make predictions for a range of X values and plot the results Note that the LWR algorithm is a non-parametric method for fitting data points that can capture non-linear patterns in the data. It is sensitive to the bandwidth parameter (tau), which controls the trade-off between bias and variance. Smaller values of tau result in higher bias but lower variance, while larger values of tau result in lower bias but higher variance. It's important to choose an appropriate value of tau based on the specific dataset and problem at hand. Additionally, LWR can be computationally expensive for large datasets, as it requires calculating weights for each training data point for every test point.

## Source Code:

import numpy as np

import matplotlib.pyplot as plt

# Generate synthetic data

np.random.seed(0)

X = np.sort(5 \* np.random.rand(80, 1), axis=0)

y = np.sin(X).ravel()

y += 0.5 \* (np.random.rand(80) - 0.5)

# LWR algorithm

def locally\_weighted\_regression(X, y, x, tau):

m, n = X.shape

X\_ = np.hstack((np.ones((m, 1)), X)) # add bias term

x\_ = np.array([[1, x]])

# Calculate weights

weights = np.exp(-np.sum((X\_ - x\_)\*\*2, axis=1) / (2 \* tau\*\*2))

W = np.diag(weights)

# Calculate theta using normal equation

theta = np.linalg.inv(X\_.T.dot(W).dot(X\_)).dot(X\_.T).dot(W).dot(y)

# Predict y\_hat

y\_hat = x\_.dot(theta)

return y\_hat

# Fit data using LWR

tau = 0.3 # bandwidth parameter

X\_test = np.linspace(0, 5, 100)

y\_test = []

for x in X\_test:

y\_hat = locally\_weighted\_regression(X, y, x, tau)

y\_test.append(y\_hat)

# Plot the results

plt.scatter(X, y, color='blue', label='Data points')

plt.plot(X\_test, y\_test, color='red', label='LWR')

plt.xlabel('X')

plt.ylabel('y')

plt.title('Locally Weighted Regression')

plt.legend()

plt.show()

**OUTPUT:**

## Result:

|  |  |  |
| --- | --- | --- |
| **Ex.No. 10** | **Demonstrate the working of the decision tree-based ID3 algorithm** | **Date:** |

**AIM:**

To demonstrate the working of the decision tree-based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

## Source Code:

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris\_df = pd.read\_csv('iris.csv')

# Split the dataset into features (X) and labels (y)

X = iris\_df.iloc[:, :-1]

y = iris\_df.iloc[:, -1]

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define the ID3 algorithm

def id3(X, y):

# Base case: all samples have the same label

if y.nunique() == 1:

return y.iloc[0]

# Base case: no more attributes to split on

if X.shape[1] == 0:

return y.value\_counts().idxmax()

# Calculate the information gain for each attribute

information\_gains = []

for attribute in X.columns:

entropy\_parent = entropy(y)

unique\_values = X[attribute].unique()

entropy\_children = 0

for value in unique\_values:

y\_subset = y[X[attribute] == value]

entropy\_children += len(y\_subset) / len(y) \* entropy(y\_subset)

information\_gain = entropy\_parent - entropy\_children

information\_gains.append(information\_gain)

# Select the attribute with the highest information gain

best\_attribute = X.columns[np.argmax(information\_gains)]

# Create a new decision node with the selected attribute

node = {'attribute': best\_attribute, 'children': {}}

# Recursively split on the selected attribute

unique\_values = X[best\_attribute].unique()

for value in unique\_values:

X\_subset = X[X[best\_attribute] == value].drop(best\_attribute, axis=1)

y\_subset = y[X[best\_attribute] == value]

node['children'][value] = id3(X\_subset, y\_subset)

return node

# Function to calculate entropy

def entropy(y):

p = y.value\_counts(normalize=True)

entropy = -np.sum(p \* np.log2(p))

return entropy

# Function to predict the class label of a sample

def predict(node, x):

if 'children' not in node:

return node

attribute = node['attribute']

value = x[attribute]

if value not in node['children']:

return None

child = node['children'][value]

return predict(child, x)

# Build the decision tree using ID3

decision\_tree = id3(X\_train, y\_train)

# Predict the class labels of the test samples

y\_pred = X\_test.apply(lambda x: predict(decision\_tree, x), axis=1)

# Evaluate the accuracy of the decision tree

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy of decision tree:", accuracy)

# Print both correct and wrong predictions

for i in range(len(y\_pred)):

if y\_pred.iloc[i] == y\_test.iloc[i]:

print("Sample:", X\_test.iloc[i].values, "Predicted class:", y\_pred.iloc[i], "True class:", y\_test.iloc[i], "Correct")

else:

print("Sample:", X\_test.iloc[i].values, "Predicted class:", y\_pred.iloc[i], "True class:", y\_test.iloc[i], "Wrong")

## Output

**Result:**