SRI KRISHNA COLLEGE OF TECHNOLOGY
(An Autonomous Institution)
Affiliated to Anna University | Approved by AICTE
Accredited by NAAC with 'A' Grade
KOVAIPUDUR, COIMBATORE – 641 042.
DEPARTMENT OF CSE (AI & ML)

22AM502 – MACHINE LEARNING TECHNIQUES LABORATORY

(REGULATION 2022)

LAB MANUAL

Laboratory In-charge

Ms. A. Sugitha

Assistant Professor

PREFACE

Machine Learning (ML) is an essential subfield of Artificial Intelligence (AI) that focuses on the development of algorithms and statistical models enabling computers to learn from and make decisions based on data. As the world generates ever-increasing amounts of data, the ability to analyze and derive meaningful insights from it becomes crucial. Machine Learning techniques are at the forefront of this data-driven revolution, powering advancements in various domains such as healthcare, finance, transportation, and more.

he primary objective of this laboratory manual is to provide hands-on experience with fundamental Machine Learning algorithms and techniques. Through a series of practical exercises, students will learn to implement, train, and evaluate various ML models using Python. This manual aims to bridge the gap between theoretical knowledge and practical application, preparing students for real-world data science challenges.

This laboratory manual is designed to equip students with practical skills and a deeper understanding of key Machine Learning techniques. By the end of these experiments, students will be capable of implementing and evaluating various ML models, preparing them for more advanced studies and professional work in the field of data science and machine learning.

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

SRI KRISHNA COLLEGE OF TECHNOLOGY

COIMBATORE – 641 042

Prepared by	**Verified by**	**Approved by**
Ms. A. Sugitha	Dr. Suma Sira Jacob	Dr. T. Senthilnathan
Assistant Professor	Associate Professor	Dean- SoC
CSE	PC/ CSE (AI & ML, CYS, IoT)	

PROFILE OF THE INSTITUTION

Nestled at the foothills of the Western Ghats, located in a sprawling 52-acre campus in Kovaipudur, Coimbatore, Sri Krishna College of Technology (SKCT) is a vibrant institute of higher education established in 1985 promoted by Sri Krishna Institutions. An extraordinary freedom of opportunity—to explore, to collaborate and to challenge oneself is the hallmark of the Institute. Being an autonomous institute, affiliated to Anna University, Chennai, and approved by AICTE, New Delhi, SKCT lays strong emphasis on collaborative research and stands apart from other institutes by its participatory work culture, student care Programmes and high industry interaction.

In a span of 38 years, it has emerged as one of the premier engineering colleges for learning, discovery and innovation due to the dynamic leadership of the Chairperson and Managing Trustee Smt. S. Malarvizhi. Being an acclaimed educationalist, she continues to contribute profusely for the glory and happiness of advancing generations. The college is accredited with A Grade by NAAC and eligible undergraduate programs are accredited by the National Board of Accreditation (NBA), New Delhi. The college offers 11 undergraduate Programmes, 6 Postgraduate Programmes and 5 Doctorial Programmes in Engineering, Technology, and Management Studies.

VISION:

Sri Krishna College of Technology aspires to be recognized as one of the pioneers in imparting world class technical education through technology enabled innovative teaching learning processes with a focus on research activities to cater, to the societal needs.

MISSION:

To be recognized as centre of excellence in science, engineering and technology through effective teaching and learning processes by providing a conducive learning environment.

To foster research and development with creative and entrepreneurial skills by means of innovative applications of technology. Accomplish expectations of the society and industry by nurturing the students to be competent professionals with integrity.

COURSES OFFERED

UNDER GRADUATE PROGRAMMES (Four Years B.E / B.Tech)

- B.E - Civil Engineering
- B.E - Computer Science and Engineering
- B.E - Computer Science and Engineering (Cyber Security)
- B.E - Computer Science and Engineering (Internet of Things)
- B.E - Computer Science and Engineering (Artificial Intelligence and Machine Learning)
- B.E - Electronics and Communication Engineering
- B.E - Electrical and Electronics Engineering
- B.E - Instrumentation and Control Engineering
- B.E - Mechanical Engineering
- B.Tech - Artificial Intelligence and Data Science
- B.Tech - Information Technology

POST GRADUATE PROGRAMMES (Two Years)

- Master of Business Administration
- M.E - Applied Electronics
- M.E – Computer Science Engineering
- M.E –Engineering Design
- M.E - Power System Engineering
- M.E - Structural Engineering

DOCTORAL PROGRAMMES (Ph.D.)

- Civil Engineering
- Computer Science and Engineering
- Electronics and Communication Engineering
- Electrical and Electronics Engineering
- Mechanical Engineering

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING (AI & ML)

The Department of Computer Science and Engineering (Artificial Intelligence and Machine Learning) is established in the Year 2022 with an Intake of 60. The Department aims to produce students with a solid foundation in the concepts of AI & ML as well as advanced areas like Deep Learning, Natural Language Processing, Computer Vision, etc. Throughout the Programme, students will compete their skills in various events like Hackathons, coding contests, others technical & automations competitions. The objective is to meet the pressing demands of the nation in the areas of Artificial Intelligence and Machine Learning.

VISION:

The department of CSE fosters a conducive ambience to meet the global standards by equipping the students with modern techniques in the area of Computer Science and relevant research to address the societal needs.

MISSION:

- To provide positive working environment that would help the students perform to their highest abilities in various fields of computer science.
- To enable students and faculty with the best of technologies and knowledge emerging in the domain of Computer Science and Engineering.
- To establish nationally and internationally recognized research centers and expose the students to broad research experience.

PROGRAM EDUCTIONAL OBJECTIVES:

PEO1: Graduates will be able to demonstrate technical skills and proficiency in modern Intelligent computing practices.

PEO2: Graduates will be able to navigate scientific and societal modernization through technological innovation and entrepreneurship in the Computational Intelligence. PEO3: Graduates will be able to equip productively in the field of Artificial Intelligence Engineering through strong technical communication and Entrepreneurship skills.

PROGRAM OUTCOMES:

PO 1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO 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.

PO 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.

PO 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.

PO 5: 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.

PO 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.

PO 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.

PO 8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 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.

PO 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.

PO 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 OUTCOMES

PSO1: Apply principles and concepts of Artificial Intelligence and Machine Learning models to build Expert systems.

PSO2: Develop computational skills and techniques to solve real-time problems in the field of Deep Learning, Natural Language Processing and Intelligent Automation.

22AM502	MACHINE LEARNING TECHNIQUES LABORATORY	0/0/3
Nature of the Course: Practical (N/A)		
Pre-requisite(s):Nil		
Course Objectives:		
1	Implement various machine learning techniques and can be able to demonstrate using Python	
2	Implement and develop a machine learning algorithms based application in Python programming language.	
Course Outcomes:		
CO1	Implement the Classification and regression algorithms in supervised learning	AP
CO2	Develop Supervised Machine Learning models in machine learning	AP
CO3	Implement the unsupervised learning algorithms using Python	AP
CO4	Construct evolutionary and graph models of machine learning using Python	AP
CO5	Build Artificial Neural Network models for various datasets.	AP
CO6	Develop a real time application using machine learning techniques	AP

Lab Components:			
S.No.	**List of Experiments**	**CO Mapping**	**RBT**
1	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.	CO1	AP
2	Implement and demonstrate the Support vector machine classification on an appropriate given set of training data samples.	CO1	AP
3	Implement Regression algorithms using Python.	CO2	AP
4	Implement k-nearest neighbor classification using Python.	CO2	AP
5	Implement Gaussian Mixture Models using Python.	CO3	AP
6	Implement Dimensionality reduction using Python.	CO3	AP
7	Implement K-Means clustering using Python.	CO3	AP
8	Implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.	CO4	AP
9	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.	CO5	AP
10	Develop a mini project for a real time application.	CO6	AP
Text Book:			

1	EthemAlpaydin, -Introduction to Machine Learning 3e (Adaptive Computation and Machine Learning Series), Third Edition, MIT Press, 2014
2	Jason Bell- Machine learning - Hands on for Developers and Technical Professionals, First Edition, Wiley, 2014
Reference Book:	
1	Andreas C Muller & Sarah Guido, Introduction to Machine Learning with Python: A Guide for Data Scientists, O'Reilly, 2016
Web Reference:	
1	https://www.javatpoint.com/machine-learning
2	https://www.w3schools.com/python/python_ml_getting_started.asp
3	https://www.tutorialspoint.com/machine_learning/index.htm

TABLE OF CONTENT

RUBRIC ASSESSMENT FOR : 22CSE05 – BIG DATA ANALYTICS

Items	Excellent	Good	Satisfactory	Needs Improvement
OBJECTIVE & ALGORITHM (20 MARKS)				
	Objective and Algorithm are highly / maximally efficient and effective, demonstrating strong understanding	Objective and Algorithm are efficient andeffective, demonstrating moderate understanding	Objective and Algorithm are somewhat efficient and effective, demonstrating	Objective and Algorithm Inefficient and/or ineffective, demonstrating limited
PROGRAM WITH SYNTAX AND STRUCTURE (30 Marks)	27-30	21-26	15-20	0-14
	The program design uses appropriate Syntax and Structures. The program overall design is	The program design generally usesa ppropriate structures. Program elements	Not all of the selected structures are appropriate. Some of the program elements	Few of the selectedstructures
COMPILATION AND DEBUGGING (30 MARKS)	27-30	21-26	15-20	0-14
	Program compiles and contains no evidence of misunderstanding or misinterpreting the syntax of the language. Program produces correct answers or appropriate results for all inputs tested.	Program compiles andis free from major syntactic misunderstandings, but may contain non-standard usage or superfluous elements.Program producescorrect answers orappropriate results formost inputs.	Program compiles, but contains errors that signal misunderstanding of syntax. Program approaches orrect answers or appropriate results for most inputs, but can contain miscalculationsin some cases.	Program does not compile orcontains typographical errorsleading to undefined names. Program does not producecorrect answers or appropriate resultsfor most inputs.
DOCUMENT ATION (10 MARKS)	9-10	7-8	5-6	0-4
	Clearly and effectively documented including descriptions of all class variables.Specific purpose noted for each function, control structure, input requirements, and output results.	Clearly documented including descriptions of all class variables. Specific purpose is noted for each function and control structure.	Basic documentation has been completed including descriptions of all class variables. Purpose is noted for each function.	Very limited or no documentation included. Documentation does not help the reader understand the code.
VIVA (10 MARKS)	9-10	7-8	5-6	0-4
	Masterfully defends by providing clear and insightful answers to questions	Competently defends by providing very helpful answers	Answers questions, but often with little insight	Very less answers /Does not answer

EXPT.NO.1	**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.**

AIM:

To understand and implement the ID3 algorithm for building a decision tree and use it to classify new samples.

INTRODUCTION:

The ID3 (Iterative Dichotomiser 3) algorithm is used to generate a decision tree based on a dataset. The decision tree is built by selecting the attribute that provides the highest information gain at each node of the tree. The ID3 algorithm uses entropy to measure the impurity or disorder in the dataset and chooses attributes that reduce entropy the most.

DATASET:

For this lab, we'll use a simple dataset known as the "Play Tennis" dataset. The dataset has features related to weather conditions and whether a game of tennis is played or not. Here's a sample dataset:

Outlook	Temperature	Humidity	Windy	Play Tennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes

Steps:

1. **Calculate Entropy**:
   - Compute the entropy of the target attribute ("Play Tennis") in the dataset.
2. **Calculate Information Gain**:

- For each attribute, calculate the entropy of the dataset after splitting by that attribute and compute the information gain.

3. **Build the Decision Tree**:
   - Choose the attribute with the highest information gain as the decision node.
   - Recursively apply steps 1-3 for each branch until all examples are classified.
4. **Classify New Sample**:
   - Use the constructed decision tree to classify a new sample.

Sample Code:

```
import pandas as pd
import numpy as np
# Define the dataset
data = pd.DataFrame({
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'High'],
    'Windy': [False, True, False, False, False, True, True, False, False, True, True, False, True],
    'Play Tennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes']
})
# Define a function to compute entropy
def entropy(column):
    counts = column.value_counts()
    probabilities = counts / len(column)
    return -np.sum(probabilities * np.log2(probabilities))
# Define a function to compute information gain
def information_gain(data, feature, target):
    total_entropy = entropy(data[target])
    feature_entropy = 0
    for value in data[feature].unique():
        subset = data[data[feature] == value]
        feature_entropy += (len(subset) / len(data)) * entropy(subset[target])
    return total_entropy - feature_entropy
# Determine the best feature to split on
def best_feature(data, features, target):
    gains = {feature: information_gain(data, feature, target) for feature in features}
    return max(gains, key=gains.get)
```

```
# Build the decision tree
def build_tree(data, features, target):
    classes = data[target].unique()
    if len(classes) == 1:
        return classes[0]
    if not features:
        return data[target].mode()[0]
    best = best_feature(data, features, target)
    tree = {best: {}}
    for value in data[best].unique():
        subset = data[data[best] == value]
        new_features = [f for f in features if f != best]
        tree[best][value] = build_tree(subset, new_features, target)
    return tree
# Build the decision tree
features = ['Outlook', 'Temperature', 'Humidity', 'Windy']
target = 'Play Tennis'
decision_tree = build_tree(data, features, target)
print("Decision Tree:")
print(decision_tree)
# Classify a new sample
def classify(tree, sample):
    if not isinstance(tree, dict):
        return tree
    feature = list(tree.keys())[0]
    feature_value = sample[feature]
    subtree = tree[feature].get(feature_value)
    if subtree is None:
        return None
    return classify(subtree, sample)
# New sample
new_sample = {'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'High', 'Windy': False}
classification = classify(decision_tree, new_sample)
print(f"Classification for new sample: {classification}")
```

RESULTS:

Thus, the decision tree is built based on the ID3 algorithm and displayed and also the new sample is classified according to the decision tree.

Possible viva questions:

Pre-viva questions:

1. What is the ID3 algorithm used for?
2. Explain the concept of information gain. Why is it important in decision tree construction?
3. How do you compute entropy for a dataset?
4. Discuss the Python libraries used in your implementation (e.g., pandas, numpy).
5. How did you handle categorical and numerical data in the "Play Tennis" dataset?

Post-viva questions:

1. Can the ID3 algorithm handle missing values in the dataset? If not, how could you modify it to handle missing data?
2. How would you evaluate the performance of the decision tree built using ID3?
3. What are some strategies for pruning decision trees to avoid overfitting?
4. What are some advanced techniques or modifications that could improve the ID3 algorithm or decision tree construction in general?
5. How could you adapt the ID3 algorithm to handle regression tasks instead of classification?

EXPT.NO.2	Implement and demonstrate the Support vector machine classification on an appropriate given set of training data samples.

AIM:

To understand and implement the Support Vector Machine (SVM) algorithm for classification, and to use it to classify new samples based on a given dataset.

INTRODUCTION:

Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression tasks. The main idea behind SVM is to find the optimal hyperplane that best separates the classes in the feature space. For binary classification, the hyperplane is the decision boundary that maximizes the margin between two classes. In cases where classes are not linearly separable, SVM uses kernel functions to map the data into a higher-dimensional space where a linear separation is possible.

DATASET:

For this lab, we'll use the "Iris" dataset, a classic dataset in machine learning, which contains measurements of iris flowers from three different species. The dataset is provided in the `sklearn` library and includes the following features:

- Sepal Length
- Sepal Width
- Petal Length
- Petal Width

We'll use a subset of this dataset for binary classification, focusing on two species for simplicity.

Steps:

1. **Load and Prepare Data**:
   - Load the dataset and select a subset for binary classification.
   - Split the data into training and testing sets.
2. **Train the SVM Model**:
   - Implement and train the SVM model using the training data.
   - Choose an appropriate kernel function (e.g., linear or radial basis function).
3. **Evaluate the Model**:
   - Test the model on the testing set.
   - Compute performance metrics such as accuracy, precision, recall, and F1-score.
4. **Classify New Samples**:
   - Use the trained model to classify new samples.

Step 1: Load and Prepare Data

We'll use the Iris dataset for this lab. First, we need to load the data and prepare it for binary classification.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
# Select only two classes for binary classification
X = X[y != 2]
y = y[y != 2]
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Step 2: Train the SVM Model

```
from sklearn.svm import SVC
# Create the SVM model with RBF kernel
model = SVC(kernel='rbf', gamma='scale', C=1.0)
# Train the model
model.fit(X_train, y_train)
```

Step 3: Evaluate the Model

After training, evaluate the model's performance on the test data.

```
from sklearn.metrics import accuracy_score, classification_report
# Predict on the test set
y_pred = model.predict(X_test)
# Calculate accuracy and other metrics
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=iris.target_names[:2])
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)
```

Step 4: Classify New Samples

Classify new samples using the trained model. Let's assume we have a new sample with the following features:

- Sepal Length: 5.1
- Sepal Width: 3.5
- Petal Length: 1.4
- Petal Width: 0.2

 # New sample

new_sample = np.array([[5.1, 3.5, 1.4, 0.2]])

 # Predict the class of the new sample

new_prediction = model.predict(new_sample)

print(f"New Sample Classification: {iris.target_names[new_prediction][0]}")

Example Code:

import numpy as np

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, classification_report

 # Load the Iris dataset

iris = load_iris()

X = iris.data

y = iris.target

 # Select only two classes for binary classification

X = X[y != 2]

y = y[y != 2]

 # Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

 # Create the SVM model with RBF kernel

model = SVC(kernel='rbf', gamma='scale', C=1.0)

 # Train the model

model.fit(X_train, y_train)

 # Predict on the test set

y_pred = model.predict(X_test)

 # Calculate accuracy and other metrics

accuracy = accuracy_score(y_test, y_pred)

report = classification_report(y_test, y_pred, target_names=iris.target_names[:2])

```
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)
# New sample
new_sample = np.array([[5.1, 3.5, 1.4, 0.2]])
# Predict the class of the new sample
new_prediction = model.predict(new_sample)
print(f"New Sample Classification: {iris.target_names[new_prediction][0]}")
```

RESULTS:

Thus, the model's accuracy and detailed classification metrics are obtained and the new sample is classified according to the trained SVM model.

Possible Viva Questions:

Pre-viva questions:

1. What is the main objective of the SVM algorithm?
2. Walk through the steps involved in training an SVM model.
3. How do you handle data preprocessing before applying SVM?
4. How does SVM handle multi-class classification tasks?
5. Discuss the computational complexity of SVM. When might SVM become computationally expensive?

Post-viva questions:

1. What strategies can be used to choose an appropriate kernel function?
2. How does SVM perform with datasets that have a large number of features or instances?
3. Can SVM handle datasets with missing values? If not, how could you preprocess such data?
4. Discuss a scenario where SVM might not perform well. What could be potential reasons for this?
5. How can you interpret the decision boundary generated by an SVM model?

EXPT.NO.3	Implement Regression algorithms using Python.

AIM:

To understand and implement various regression algorithms, including Linear Regression, Polynomial Regression, and Ridge Regression, and evaluate their performance using a synthetic dataset.

INTRODUCTION:

Regression algorithms are used to model the relationship between a dependent variable and one or more independent variables. The goal is to predict the dependent variable based on the independent variables. In this lab, we will implement three types of regression algorithms:

1. **Linear Regression**: Models the relationship between the dependent and independent variables using a straight line.
2. **Polynomial Regression**: Extends linear regression by fitting a polynomial function to the data.
3. **Ridge Regression**: A type of linear regression that includes a regularization term to prevent overfitting.

DATASET:

We will use a synthetic dataset generated using numpy. The dataset will consist of one feature and one target variable.

Steps:

1. **Generate Synthetic Data**:
   - Create a synthetic dataset for regression tasks.
   - Visualize the dataset.
2. **Implement and Train Regression Models**:
   - Implement and train Linear Regression.
   - Implement and train Polynomial Regression.
   - Implement and train Ridge Regression.
3. **Evaluate and Compare Models**:
   - Evaluate the performance of each model using metrics such as Mean Squared Error (MSE) and R^2 score.
   - Visualize the results and compare the models.

Step 1: Generate Synthetic Data

Generate a synthetic dataset with one feature and a target variable that follows a quadratic relationship.

```
import numpy as np
import matplotlib.pyplot as plt
# Generate synthetic data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + 1.5 * X**2 + np.random.randn(100, 1)
# Visualize the dataset
plt.scatter(X, y, color='blue', label='Data points')
plt.xlabel('Feature')
plt.ylabel('Target')
plt.title('Synthetic Dataset')
plt.legend()
plt.show()
```

Step 2: Implement and Train Regression Models

Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Create and train the model
lin_reg = LinearRegression()
lin_reg.fit(X, y)
# Predict
y_pred_lin = lin_reg.predict(X)
# Evaluate
mse_lin = mean_squared_error(y, y_pred_lin)
r2_lin = r2_score(y, y_pred_lin)
print(f"Linear Regression MSE: {mse_lin:.2f}")
print(f"Linear Regression R² Score: {r2_lin:.2f}")
```

Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
# Create a polynomial feature transformer
poly_features = PolynomialFeatures(degree=2, include_bias=False)
```

```
X_poly = poly_features.fit_transform(X)
# Create and train the model
poly_reg = LinearRegression()
poly_reg.fit(X_poly, y)
# Predict
y_pred_poly = poly_reg.predict(X_poly)
# Evaluate
mse_poly = mean_squared_error(y, y_pred_poly)
r2_poly = r2_score(y, y_pred_poly)
print(f"Polynomial Regression MSE: {mse_poly:.2f}")
print(f"Polynomial Regression R² Score: {r2_poly:.2f}")
# Visualize
plt.scatter(X, y, color='blue', label='Data points')
X_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
X_range_poly = poly_features.transform(X_range)
y_range_poly = poly_reg.predict(X_range_poly)
plt.plot(X_range, y_range_poly, color='red', label='Polynomial Fit')
plt.xlabel('Feature')
plt.ylabel('Target')
plt.title('Polynomial Regression')
plt.legend()
plt.show()
```

Ridge Regression

```
from sklearn.linear_model import Ridge

# Create and train the model
ridge_reg = Ridge(alpha=1.0)
ridge_reg.fit(X, y)
# Predict
y_pred_ridge = ridge_reg.predict(X)
# Evaluate
mse_ridge = mean_squared_error(y, y_pred_ridge)
r2_ridge = r2_score(y, y_pred_ridge)
print(f"Ridge Regression MSE: {mse_ridge:.2f}")
print(f"Ridge Regression R² Score: {r2_ridge:.2f}")
```

Step 3: Evaluate and Compare Models

 # Comparison of models

```
print(f"Linear Regression MSE: {mse_lin:.2f}")
print(f"Polynomial Regression MSE: {mse_poly:.2f}")
print(f"Ridge Regression MSE: {mse_ridge:.2f}")
print(f"Linear Regression R² Score: {r2_lin:.2f}")
print(f"Polynomial Regression R² Score: {r2_poly:.2f}")
print(f"Ridge Regression R² Score: {r2_ridge:.2f}")
```

Results:

Thus, the Mean Squared Error (MSE) and R^2 scores for each regression model are computed and visualizations for Polynomial Regression are displayed.

Possible Viva Questions:

Pre-viva questions:

1. What is the key difference between Polynomial Regression and Linear Regression?
2. Why is regularization used in Ridge Regression? How does it help prevent overfitting?
3. Walk through the steps involved in generating synthetic data for regression tasks.
4. How do you assess the quality of a regression model using metrics like Mean Squared Error (MSE) and R^2 score?
5. Discuss the impact of polynomial degree in Polynomial Regression. How do higher degrees affect the model?

Post-viva questions:

1. How do you interpret the values of MSE and R^2 score in regression evaluation?
2. Compare and contrast the performance of Linear Regression, Polynomial Regression, and Ridge Regression on a dataset. Under what conditions might each be preferred?
3. Discuss the limitations of using R^2 score as an evaluation metric for regression models.
4. How do you handle multicollinearity in regression analysis? Why is it important?
5. Explain the concept of bias-variance tradeoff in the context of regression models.

EXPT.NO.4	Implement k-nearest neighbor classification using Python.

AIM:

To understand and implement the K-Nearest Neighbor (KNN) algorithm for classification, and to evaluate its performance using a given dataset.

INTRODUCTION:

The K-Nearest Neighbor (KNN) algorithm is a simple, yet effective, supervised learning algorithm used for classification and regression. For classification tasks, KNN assigns a class to a sample based on the majority class among its K-nearest neighbors in the feature space. The choice of K (number of neighbors) and distance metric significantly affect the performance of the model.

DATASET:

For this lab, we'll use the Iris dataset, a classic dataset in machine learning. The Iris dataset includes measurements of iris flowers from three different species, but for simplicity, we'll use a subset for binary classification.

Steps:

1. **Load and Prepare Data**:
   - Load the dataset and prepare it for binary classification.
   - Split the data into training and testing sets.
2. **Implement and Train KNN Model**:
   - Implement the KNN algorithm using the KNeighborsClassifier from the sklearn library.
   - Train the model using the training data.
3. **Evaluate the Model**:
   - Test the model on the testing set.
   - Compute performance metrics such as accuracy, precision, recall, and F1-score.
4. **Classify New Samples**:
   - Use the trained KNN model to classify new samples.

Step 1: Load and Prepare Data

Load the Iris dataset, select a subset for binary classification, and split the data into training and testing sets.

The iris dataset conatins the following features

```
---> sepal length (cm)
---> sepal width (cm)
---> petal length (cm)
---> petal width (cm)

The Sample data in iris dataset format is  [5.4 3.4 1.7 0.2]

Where     5.4 ---> sepal length (cm)
                  3.4 ---> sepal width (cm)
                  1.7 ---> petal length (cm)
                  0.2 ---> petal width (cm)
```

Sample Code:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import random
# Loading data
data_iris = load_iris()
# To get list of target names
label_target = data_iris.target_names
print()
print("Sample Data from Iris Dataset")
print("*"*30)
# to display the sample data from the iris dataset
for i in range(10):
        rn = random.randint(0,120)
        print(data_iris.data[rn],"===>",label_target[data_iris.target[rn]])
# Create feature and target arrays
X = data_iris.data
y = data_iris.target
# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
                        X, y, test_size = 0.3, random_state=1)
print("The Training dataset length: ",len(X_train))
```

```
print("The Testing dataset length: ",len(X_test))
try:
    nn = int(input("Enter number of neighbors :"))
    knn = KNeighborsClassifier(nn)
    knn.fit(X_train, y_train)
    # to display the score
    print("The Score is :",knn.score(X_test, y_test))
    # To get test data from the user
    test_data = input("Enter Test Data :").split(",")
    for i in range(len(test_data)):
        test_data[i] = float(test_data[i])
    print()
    v = knn.predict([test_data])
    print("Predicted output is :",label_target[v])
except:
    print("Please supply valid input.....")
```

Output:

```
D:\Machine Learning\Lab>python Week3.py

Sample Data from Iris Dataset
*****************************
[6.8 2.8 4.8 1.4] ===> versicolor
[5.8 2.7 3.9 1.2] ===> versicolor
[6.6 2.9 4.6 1.3] ===> versicolor
[5.1 3.8 1.6 0.2] ===> setosa
[4.9 2.5 4.5 1.7] ===> virginica
[7.3 2.9 6.3 1.8] ===> virginica
[4.9 3.1 1.5 0.1] ===> setosa
[4.5 2.3 1.3 0.3] ===> setosa
[7.6 3.  6.6 2.1] ===> virginica
[6.6 3.  4.4 1.4] ===> versicolor
The Training dataset length:  105
The Testing dataset length:  45
Enter number of neighbors :10
The Score is : 0.9777777777777777
Enter Test Data :6.2,2.6,3.4,0.6

Predicted output is : ['versicolor']
```

RESULTS:

Thus, the accuracy and detailed classification metrics for the KNN model are displayed and the new sample is classified according to the trained KNN model.

Possible Questions:

Pre-viva questions:

1. Discuss the importance of choosing an appropriate value for K in KNN.
2. How does KNN handle categorical versus numerical data in its classification process?
3. Describe the steps involved in implementing the KNN algorithm for classification using Python and sklearn.
4. How do you evaluate the performance of a KNN model?
5. Explain the process of splitting data into training and testing sets. Why is this important?

Post-viva questions:

1. How does KNN perform with datasets that have a large number of features or instances?
2. Can KNN be used for regression tasks? If so, how is it adapted?
3. Discuss scenarios where KNN might not perform well. What are potential reasons for its poor performance?
4. How can you handle imbalanced datasets when using KNN for classification?
5. What strategies can be employed to optimize the performance of KNN?

EXPT.NO.5	Implement Gaussian Mixture Models using Python.

AIM:

To understand and implement Gaussian Mixture Models (GMM) for clustering, and to evaluate its performance using a synthetic dataset.

INTRODUCTION:

Gaussian Mixture Models (GMM) are a probabilistic model used for clustering and density estimation. GMM assumes that the data is generated from a mixture of several Gaussian distributions with unknown parameters. The algorithm tries to estimate these parameters to best fit the data.

DATASET:

for this lab, we will use a synthetic dataset generated with a mixture of gaussian distributions to illustrate the gmm clustering.

Steps:

1. **Generate Synthetic Data**:
   - Create a synthetic dataset with multiple Gaussian clusters.
   - Visualize the dataset.
2. **Implement and Train GMM Model**:
   - Implement the GMM using the GaussianMixture class from the sklearn library.
   - Train the model on the synthetic dataset.
3. **Evaluate the Model**:
   - Visualize the clustered data and the GMM components.
   - Assess the performance using clustering metrics (if true labels are available).
4. **Classify New Samples**:
   - Use the trained GMM model to classify new samples.

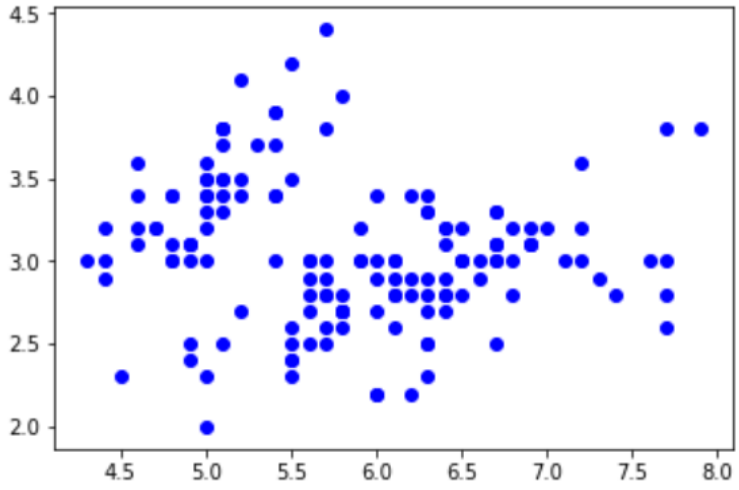
Sample Code:

In this example, iris Dataset is taken. In Python, there is a Gaussian mixture class to implement GMM. Load the iris dataset from the datasets package. To keep things simple, take the only first two columns (i.e sepal length and sepal width respectively). Now plot the dataset.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas import DataFrame
from sklearn import datasets
```

```
from sklearn.mixture import GaussianMixture
# load the iris dataset
iris = datasets.load_iris()
# select first two columns
X = iris.data[:, :2]
# turn it into a dataframe
d = pd.DataFrame(X)
# plot the data
plt.scatter(d[0], d[1])
plt.show()
```

Output:

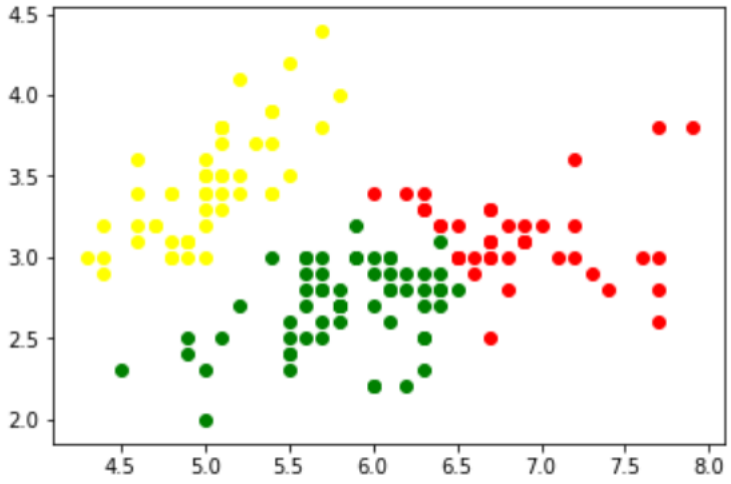


Now fit the data as a mixture of 3 Gaussians. Then do the clustering, i.e assign a label to each observation. Also, find the number of iterations needed for the log-likelihood function to converge and the converged log-likelihood value.

```
gmm = GaussianMixture(n_components = 3)
# Fit the GMM model for the dataset
# which expresses the dataset as a
# mixture of 3 Gaussian Distribution
gmm.fit(d)

# Assign a label to each sample
labels = gmm.predict(d)
```

```
d['labels']= labels

d0 = d[d['labels']== 0]

d1 = d[d['labels']== 1]

d2 = d[d['labels']== 2]

# plot three clusters in same plot

plt.scatter(d0[0], d0[1], c ='r')

plt.scatter(d1[0], d1[1], c ='yellow')

plt.scatter(d2[0], d2[1], c ='g')

plt.show()
```

Output:



RESULTS:

Thus, the GMM model clusters the synthetic data, and the results are visualized and the GMM components (mean of each Gaussian) are visualized on the dataset.

Possible Viva Questions:

Pre-viva questions:

1. Describe the steps involved in implementing GMM using Python and sklearn.
2. How does GMM estimate the parameters (mean, covariance, and mixture coefficients) from the data?
3. What are the advantages and disadvantages of using GMM for clustering compared to other methods?

4. Discuss the impact of initialization and convergence criteria on the performance of GMM.
5. Explain the Expectation-Maximization (EM) algorithm and its role in fitting GMM.

Post-viva questions:

1. Can GMM be used for anomaly detection? If so, how?
2. Explain strategies for selecting the optimal number of clusters in GMM.
3. How can GMM be applied to datasets where the number of clusters is unknown?
4. Describe a real-world application where GMM has been successfully applied. What were the key challenges and how were they addressed?
5. How can GMM be adapted for online learning scenarios where new data points arrive continuously?

EXPT.NO.6	Implement Dimensionality reduction using Python.

AIM:

To understand and implement dimensionality reduction techniques, specifically Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE), and evaluate their effectiveness using a dataset.

INTRODUCTION:

Dimensionality reduction techniques are used to reduce the number of features in a dataset while retaining as much information as possible. PCA and t-SNE are popular methods for achieving this:

- **PCA**: Reduces dimensionality by projecting data onto the directions of maximum variance.
- **t-SNE**: Preserves the local structure of the data, useful for visualizing high-dimensional data in 2D or 3D.

Steps to Apply PCA in Python for Dimensionality Reduction

We will understand the step by step approach of applying Principal Component Analysis in Python with an example. In this example, we will use the iris dataset, which is already present in the sklearn library of Python.

Step-1: Import necessary libraries

All the necessary libraries required to load the dataset, pre-process it and then apply PCA on it are mentioned below:

```
# Import necessary libraries
from sklearn import datasets # to retrieve the iris Dataset
import pandas as pd # to load the dataframe
from sklearn.preprocessing import StandardScaler # to standardize the features
from sklearn.decomposition import PCA # to apply PCA
import seaborn as sns # to plot the heat maps
```

Step-2: Load the dataset

After importing all the necessary libraries, we need to load the dataset. Now, the iris dataset is already present in sklearn. First, we will load it and then convert it into a pandas data frame for ease of use.

```
#Load the Dataset
iris = datasets.load_iris()
#convert the dataset into a pandas data frame
```

df = pd.DataFrame(iris['data'], columns = iris['feature_names'])

#display the head (first 5 rows) of the dataset

df.head()

Output:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
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	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Step-3: Standardize the features

Before applying PCA or any other Machine Learning technique it is always considered good practice to standardize the data. For this, Standard Scalar is the most commonly used scalar. Standard Scalar is already present in sklearn. So, now we will standardize the feature set using Standard Scalar and store the scaled feature set as a pandas data frame.

#Standardize the features

#Create an object of StandardScaler which is present in sklearn.preprocessing

scalar = StandardScaler()

scaled_data = pd.DataFrame(scalar.fit_transform(df)) #scaling the data

scaled_data

Output:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444
4	-1.021849	1.249201	-1.340227	-1.315444

Step-4: Applying Principal Component Analysis

We will apply PCA on the scaled dataset. For this Python offers yet another in-built class called PCA which is present in sklearn.decomposition, which we have already imported in step-1. We need to create an object of PCA and while doing so we also need to initialize n_components – which is the number of principal components we want in our final dataset. Here, we have taken n_components = 3, which means our final feature set will have 3 columns. We fit our scaled data to the PCA object which gives us our reduced dataset.

#Applying PCA

#Taking no. of Principal Components as 3

```
pca = PCA(n_components = 3)
pca.fit(scaled_data)
data_pca = pca.transform(scaled_data)
data_pca = pd.DataFrame(data_pca,columns=['PC1','PC2','PC3'])
data_pca.head()
```

Output:

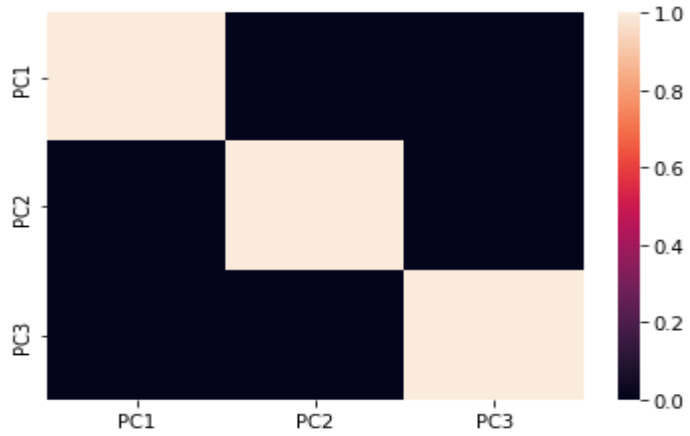
	PC1	PC2	PC3
0	-2.264703	0.480027	-0.127706
1	-2.080961	-0.674134	-0.234609
2	-2.364229	-0.341908	0.044201
3	-2.299384	-0.597395	0.091290
4	-2.389842	0.646835	0.015738

Step-5: Checking Co-relation between features after PCA

#Checking Co-relation between features after PCA

```
sns.heatmap(data_pca.corr())
```

Output:



Heatmap clearly depicts that there is no correlation between various obtained principal components (PC1, PC2, and PC3). Thus, we have moved from higher dimensional feature space to a lower-dimensional feature space while ensuring that there is no correlation between the so obtained PCs is minimum. Hence, we have accomplished the objectives of PCA.

RESULTS:

Thus, the Visualizations of the original data, PCA results, and t-SNE results are displayed and PCA and t-SNE both reduce the dataset to 2 dimensions for comparison.

Possible Viva Questions:

Pre-Viva questions:

1. How does PCA achieve dimensionality reduction while preserving information?
2. Describe the key assumptions made by PCA and t-SNE about the data.
3. Discuss scenarios where PCA might be more suitable compared to t-SNE, and vice versa.
4. Describe the step-by-step process of applying PCA on a dataset using Python.
5. What are the advantages and disadvantages of standardizing data before applying PCA?

Post-viva questions:

1. Discuss the interpretability of the transformed data after applying PCA and t-SNE.
2. What are some common metrics used to evaluate the quality of dimensionality reduction techniques?
3. Explain strategies for visualizing high-dimensional data using PCA and t-SNE.
4. Can PCA be used for feature selection? If so, how would you determine which features to select?
5. Discuss the robustness of t-SNE to different parameter settings (e.g., perplexity).

EXPT.NO.7	Implement K-Means clustering using Python.

AIM:

To understand and implement the K-Means clustering algorithm, and to evaluate its performance using a dataset.

INTRODUCTION:

The K-Means clustering algorithm is an unsupervised learning method used to partition a dataset into K distinct clusters. The algorithm iterates to minimize the variance within each cluster, assigning each data point to the nearest cluster center. The number of clusters (K) must be specified in advance.

Scenario:

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

Step 1:

To run this program you need to install the sklearn Module

Step 2:

Open Command propmt and then execute the following command to install sklearn Module

---> pip install scikit-learn

In this program, we are going to use the following data

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

Step 3: And, we need apply k-means clustering with 3 means (i.e., 3 centroids)

Step 4:

Finally, you need to predict the class for the VAR1=0.906 and VAR2=0.606

Sample Code:

```
from sklearn.cluster import KMeans
```

```
import numpy as np
X = np.array([[1.713,1.586], [0.180,1.786], [0.353,1.240],
[0.940,1.566], [1.486,0.759], [1.266,1.106],[1.540,0.419],[0.459,1.799],[0.773,0.186]])
y=np.array([0,1,1,0,1,0,1,1,1])
kmeans = KMeans(n_clusters=3, random_state=0).fit(X,y)
print("The input data is ")
print("VAR1 \t VAR2 \t CLASS")
i=0
for val in X:
        print(val[0],"\t",val[1],"\t",y[i])
        i+=1
print("="*20)
# To get test data from the user
print("The Test data to predict ")
test_data = []
VAR1 = float(input("Enter Value for VAR1 :"))
VAR2 = float(input("Enter Value for VAR2 :"))
test_data.append(VAR1)
test_data.append(VAR2)
print("="*20)
print("The predicted Class is : ",kmeans.predict([test_data]))
```

Output:

```
D:\Machine Learning\Lab>python Week4.py
The input data is
VAR1     VAR2    CLASS
1.713    1.586   0
0.18     1.786   1
0.353    1.24    1
0.94     1.566   0
1.486    0.759   1
1.266    1.106   0
1.54     0.419   1
0.459    1.799   1
0.773    0.186   1
====================
The Test data to predict
Enter Value for VAR1 :0.906
Enter Value for VAR2 :0.606
====================
The predicted Class is :  [0]
```

RESULTS:

Thus, the Visualizations of the original data, clustered data, and cluster centers are displayed at the Output terminal.

Possible Viva Questions:

Pre-viva questions:

1. What are the assumptions made by K-Means regarding the data and clusters?
2. Explain how K-Means minimizes variance within clusters and how it determines cluster centroids.
3. Discuss scenarios where K-Means might perform poorly and potential strategies to mitigate these issues.
4. Describe the role of the number of clusters (K) in the K-Means algorithm.
5. How does the initialization of cluster centroids impact the final clustering results?

Post-viva questions:

1. How does the choice of initialization method for centroids affect the convergence and final clustering outcome in K-Means?
2. Provide examples of real-world applications where K-Means clustering has been successfully applied.
3. How would you determine the optimal number of clusters (K) for a new dataset using K-Means?
4. Discuss the robustness of K-Means to noise and its sensitivity to different scaling methods of data.
5. How could you incorporate domain knowledge or constraints into the K-Means clustering process?

EXPT.NO.8	**Implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.**

AIM:

To understand and implement the Naïve Bayes classifier for classification tasks using a sample dataset stored in a CSV file. We will compute the accuracy of the classifier on given test datasets.

INTRODUCTION:

The Naïve Bayes classifier is a probabilistic model based on Bayes' theorem, assuming independence among features. It is widely used for classification tasks due to its simplicity and effectiveness, especially in text classification and spam detection.

DATASET:

For this lab, you will use a sample training dataset stored in a CSV file. The dataset should include features and a target class label.

Steps:

1. **Load and Prepare Data**:
   - Load the dataset from a CSV file.
   - Split the dataset into training and testing sets.
2. **Implement and Train Naïve Bayes Model**:
   - Implement the Naïve Bayes classifier using the GaussianNB class from the sklearn library.
   - Train the model on the training data.
3. **Evaluate the Model**:
   - Predict on the test set and compute the accuracy of the classifier.
   - Display the confusion matrix and classification report.

Source Code:

```
# import necessary libarities
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
# load data from CSV
data = pd.read_csv('tennisdata.csv')
print("THe first 5 values of data is :\n",data.head())
```

Output:

The first 5 values of data is :

Outlook Temperature Humidity Windy PlayTennis

0 Sunny Hot High False No

1 Sunny Hot High True No

2 Overcast Hot High False Yes

3 Rainy Mild High False Yes

4 Rainy Cool Normal False Yes

 # obtain Train data and Train output

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

print("\nThe First 5 values of train data is\n",X.head())

Output:

The First 5 values of train data is

Outlook Temperature Humidity Windy

0 Sunny Hot High False

1 Sunny Hot High True

2 Overcast Hot High False

3 Rainy Mild High False

4 Rainy Cool Normal False

y = data.iloc[:,-1]

print("\nThe first 5 values of Train output is\n",y.head())

Output:

The first 5 values of Train output is

0 No

1 No

2 Yes

3 Yes

4 Yes

Name: PlayTennis, dtype: object

 # Convert then in numbers

le_outlook = LabelEncoder()

X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()

X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()

X.Humidity = le_Humidity.fit_transform(X.Humidity)

```
le_Windy = LabelEncoder()
X.Windy = le_Windy.fit_transform(X.Windy)
print("\nNow the Train data is :\n",X.head())
```

Output:

```
Now the Train data is :
Outlook  Temperature  Humidity  Windy
0        2           1         0      0
1        2           1         0      1
2        0           1         0      0
3        1           2         0      0
4        1           0         1      0
```

```
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
```

Output:

```
Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)
classifier = GaussianNB()
classifier.fit(X_train,y_train)
from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

Output:

```
Accuracy is: 0.6666666666666666
```

RESULTS:

Thus, the accuracy of the Naïve Bayes classifier on the test data is printed.

Possible Viva Questions:

Pre-viva Question:

1. How does the classifier handle numerical and categorical features differently? Provide examples.
2. Compare Naïve Bayes with other popular classifiers like Logistic Regression and Decision Trees in terms of assumptions and performance.

3. Walk through the steps involved in implementing Naïve Bayes using the GaussianNB class from sklearn.
4. Explain the role of LabelEncoder in preprocessing categorical target labels and features for Naïve Bayes.
5. Discuss strategies for handling missing data and outliers in the dataset before applying Naïve Bayes.

Post-viva questions:

1. Discuss techniques for improving the performance of Naïve Bayes on datasets with high-dimensional features or imbalanced classes.
2. How does feature selection impact the performance of Naïve Bayes? Describe methods for selecting relevant features.
3. Explain strategies for deploying and maintaining a Naïve Bayes model in a production environment.
4. Application and Interpretation
5. How would you handle scenarios where features in the dataset violate the independence assumption of Naïve Bayes?
6. Describe how you would interpret probabilities assigned by Naïve Bayes to different classes.

EXPT.NO.9	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

AIM:

To understand and implement an Artificial Neural Network (ANN) using the Backpropagation algorithm, and to evaluate its performance on a given dataset.

INTRODUCTION:

An Artificial Neural Network (ANN) is a computational model inspired by the human brain, consisting of interconnected nodes (neurons). The Backpropagation algorithm is a supervised learning technique used to train ANNs by minimizing the error between predicted and actual outputs.

Training Examples:

Example	Sleep	Study	Expected % in Exams
1	2	9	92
2	1	5	86
3	3	6	89

Normalize the input

Example	Sleep	Study	Expected % in Exams
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

Source Code:

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 layers neurons
output_neurons = 1          #number of neurons at output layer
```

```
#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neur
ons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neuron
s))
bout=np.random.uniform(size=(1,output_neurons))


#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):

#Forward_Propogation
    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 wts contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad

# dotproduct of nextlayererror and currentlayerop
     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:
 [[0.89726759]
 [0.87196896]
 [0.9000671]]
```

RESULTS:

Thus, the accuracy of the ANN on the test data is printed and the model's performance can be further evaluated using additional metrics such as confusion matrix and classification report.

Possible Viva Questions:

Pre-viva questions:

1. Compare supervised learning, unsupervised learning, and reinforcement learning. In which category does backpropagation fall, and why?
2. Walk through the steps involved in implementing an ANN using a library like TensorFlow or Keras. What are the essential components required for setting up an ANN model?
3. What metrics are commonly used to evaluate the performance of an ANN on classification tasks? How are these metrics calculated?
4. Discuss the importance of a validation set in training an ANN. How does it help in monitoring and improving model performance?
5. Explain the purpose of a confusion matrix and a classification report in evaluating the results of a classification model like an ANN. How are these metrics interpreted?

Post-viva questions:

1. Describe techniques for visualizing and interpreting the learned features in an ANN, particularly in complex datasets.
2. What are some computational challenges associated with training large-scale ANNs? How can these challenges be addressed?
3. How would you handle scenarios where the dataset for training an ANN is highly imbalanced? What techniques can be used to improve model performance in such cases?
4. Discuss the ethical implications of deploying ANNs in decision-making processes, particularly in sensitive domains like healthcare or finance.
5. What are some emerging trends or advancements in ANN research and applications? How might these impact future developments in machine learning and artificial intelligence?