CANARA ENGINEERING COLLEGE
Benjanapadavu, Mangaluru - 574219

LABORATORY MANUAL

MACHINE LEARNING LABORATORY
(21AIL66)

PREPARED BY

Mr. Deepak D

Assistant Professor

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

CANARA ENGINEERING COLLEGE

Sudhindra Nagar, Benjanapadavu,
Bantwal Taluk, D.K., Karnataka- 574 219

VISION OF THE INSTITUTE

To be an Engineering Institute of highest repute and produce world- class engineers catering to the needs of mankind.

MISSION OF THE INSTITUTE

- Provide the right environment to develop quality education for all, irrespective of caste, creed or religion to produce future leaders.
- Create opportunities for pursuit of knowledge and all-round development.
- Impart value education to students to build sense of integrity, honesty and ethics.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

VISION OF THE DEPARTMENT

To be identified as learning centre in the domain of Artificial Intelligent and Machine Learning education that delivers competent and professional engineers to meet the needs of industry, society and the nation.

MISSION OF THE DEPARTMENT

- To impart skill-based (Artificial Intelligent and Machine Learning) education through competent teaching-learning process.
- To establish a research and innovation ecosystem that provides solution for technological challenges of industry, society and the nation.
- To set-up industry-institute interface for overall development of students through practical internship and team work activities.
- To promote innovation and start-up culture among staff and students for addressing the challenges of needy.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

PROGRAM EDUCATIONAL OBJECTIVES

The graduates of BE-AIML program four years after graduation will

1. Design and develop learning-based intelligent systems in the field of artificial intelligent, machine learning, and allied engineering sectors.
2. Apply skills and knowledge of computer science to address relevant industry and societal problems or pursue higher education and research
3. Graduates will design and deploy software that meets the needs of individuals and the industries
4. Engage in lifelong learning, career advancement and adoption of changing professional and societal needs

PROGRAMME SPECIFIC OUTCOMES

1. Intelligent Systems: Select appropriate technologies to analyse, design, implement, and deployment of smart and intelligent systems
2. Contemporary Systems: Design, and development of efficient IT solutions for challenging issues through experiential learning

Laboratory Manual of Machine Learning

The Laboratory Manual of Machine Learning is an all-encompassing guide created for students who are studying and applying machine learning algorithms and techniques. This manual is a resource for conducting practical experiments and projects related to machine learning, providing students with hands-on experience in this fast-evolving field.

Purpose

The primary purpose of the Laboratory Manual of Machine Learning is to provide students with a structured framework for learning and implementing machine learning concepts and algorithms in real-world scenarios. Through a series of carefully designed experiments and projects, students can explore the principles, methodologies, and applications of machine learning, ultimately enhancing their understanding and proficiency in this cutting-edge technology.

Quality Assurance

The content of the Laboratory Manual of Machine Learning has been developed and reviewed by experienced faculty members and subject matter experts in the field of machine learning. Every effort has been made to ensure the accuracy, relevance, and reliability of the information presented in the manual, with a focus on providing students with a high-quality learning experience.

MACHINE LEARNING LABORATORY

Course Code	21AIL66	Semester	VI
Teaching Hours/ Week (L:T:P:S)	0:0:2:0	CIE MARKS	50
Total Hours of Pedagogy	24	SEE MARKS	50
Credits	01	Total Marks	100

COURSE DESCRIPTION

The Machine Learning Laboratory course provides students with hands-on experience in applying machine learning algorithms and techniques to real-world datasets. Through a series of practical exercises and projects, students will learn how to analyze data, build predictive models, and evaluate their performance. The course covers a wide range of topics in machine learning, including classification, regression, clustering

COURSE OBJECTIVES

This course will enable the students to:

1. To learn and understand the importance of machine learning algorithms
2. Illustrate the supervised and unsupervised learning techniques
3. Compare and contrast the learning techniques like ANN approach, Bayesian learning and reinforcement learning
4. Able to solve and analyze the problems on ANN, Instance based learning and Reinforcement learning techniques.
5. To impart the knowledge of clustering and classification Algorithms for predictions and evaluating Hypothesis.

COURSE OUTCOMES

After completion of the course, the students will be able to:

1. Demonstrate the working of various supervised, unsupervised, and reinforcement machine learning techniques in solving real-world problems.
2. Analyze the fundamental principles underlying concept learning algorithms, including the Find-S algorithm and the Candidate Elimination algorithm.
3. Apply various supervised learning techniques such as Decision trees, Artificial Neural Networks (ANN), Naive Bayes, K-Nearest Neighbors (KNN), and Bayesian belief networks
4. Apply unsupervised learning techniques such as K-means clustering and hierarchical clustering to make predictions and validate hypotheses in diverse datasets.
5. Apply the concepts of regression and classification algorithm techniques such as the non-parametric locally weighted regression algorithm and support vector machine.

SYLLABUS

SL. No.	CONTENT	HOURS
1.	Introduction to Machine Learning, Python installation and setting Python environment. Demonstration on installation and usage of Anaconda, explanation on numpy, pandas, scikit-learn and scipy library packages.	**2**
2.	Aim: Illustrate and demonstrate the working model and principle of Find-S algorithm. Program: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.	**2**
3.	Aim: Demonstrate the working model and principle of candidate elimination algorithm. Program: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.	**2**
4.	Aim: To construct the Decision tree using the training data sets under supervised learning concept. Program: 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.	**2**
5.	Aim: To understand the working principle of Artificial Neural network with feed forward and feed backward principle. Program: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.	**2**
6.	Aim: Demonstrate the text classifier using Naïve bayes classifier algorithm. Program: Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.	**2**
7.	Aim: Demonstrate and analyze the results sets obtained from Bayesian belief network Principle. Program: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Python ML library classes/API.	**2**
8.	Aim: Implement and demonstrate the working model of K-means clustering algorithm with Expectation Maximization Concept. Program: 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 Python ML library classes/API in the program.	**2**

SL. No.	CONTENT	HOURS
9.	Aim: Demonstrate and analyze the results of classification based on KNN Algorithm. Program: Write a program to implement 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.	**2**
10.	Aim: Understand and analyze the concept of Regression algorithm techniques. Program: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.	**2**
11.	Aim: Implement and demonstrate classification algorithm using Support vector machine Algorithm. Program: Implement and demonstrate the working of SVM algorithm for classification.	**2**
	PART B - Practical based learning	
	A problem statement for each batch is to be generated in consultation with the co-examiner and student should develop an algorithm, program and execute the program for the given problem with appropriate outputs.	**2**

TEXT, REFERENCE BOOKS

1. Tom M Mitchell, “Machine Lerning”,1st Edition, McGraw Hill Education, 2017.
2. Nello Cristianini, John Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, 2013
3. Allen B. Downey, “Think Python: How to Think Like a Computer Scientist”, 2nd Edition, Green Tea Press, 2015. (Available under CC-BY-NC license at http://greenteapress.com/thinkpython2/thinkpython2.pdf)

E-RESOURCES

1. https://deepakdvallur.weebly.com/ml-lab-21ail66.html
2. https://www.kaggle.com/general/95287
3. https://web.stanford.edu/~hastie/Papers/ESLII.pdf

COURSE PRE-REQUISITES

- Proficiency in programming is crucial, preferably in Python
- A strong understanding of mathematics is essential for grasping the underlying concepts of machine learning algorithms. Topics such as linear algebra, calculus, probability, and statistics.

ASSESSMENT DETAILS (CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each course. The student has to secure not less than 35% (18 Marks out of 50) in the semester-end examination (SEE).

Continuous Internal Evaluation (CIE):

- CIE marks for the practical course is 50 Marks.
- The split-up of CIE marks for record/ journal and test are in the ratio 60:40.
- Each experiment to be evaluated for conduction with observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments designed by the faculty who is handling the laboratory session and is made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled downed to 30 marks (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct 02 tests for 100 marks, the first test shall be conducted after the 8th week of the semester and the second test shall be conducted after the 14th week of the semester.
- In each test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability. Rubrics suggested in Annexure-II of Regulation book
- The average of 02 tests is scaled down to 20 marks (40% of the maximum marks). The Sum of scaled-down marks scored in the report write-up/journal and average marks of two tests is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

SEE marks for the practical course is 50 Marks.

- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the University
- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. OR based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the internal /external examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.
- General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

Note:

- *Students can pick one experiment from the questions lot of PART A with equal choice to all the students in a batch. For PART B examiners should frame a question for each batch, student should*
- *develop an algorithm, program, execute and demonstrate the results with appropriate output for the given problem.*
- *Weightage of marks for PART A is 80% and for PART B is 20%. General rubrics suggested to be followed for part A and part B.*
- *Change of experiment is allowed only once and Marks allotted to the procedure part to be made zero (Not allowed for Part B).*

COURSE CONTINUOUS AND COMPREHENSIVE ASSESSMENT PLAN

Sl. No.	Assessment Type	Maximum Marks	Minimum Marks	Assessment		
				Methods	Marks	Weightage (%)
1	CIE – CCAs	30	12	WRITEUP & CONDUCTION RECORD VIVA TOTAL	10 10 10 30	60
2	CIE - IAT	20	08	IAT (60:40) SCALEDOWN	100 20	40
	TOTAL CIE	**50**	**20**	**-**	**-**	**100**

CONTENT

1	Aim	Illustrate and demonstrate the working model and principle of Find-S algorithm
	Program	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples

CONCEPT - FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS

FIND-S Algorithm

1. Initialize *h* to the most specific hypothesis in *H*
2. For each positive training instance *x*

    For each attribute constraint a_i in *h*

    If the constraint a_i is satisfied by *x*

        Then do nothing

    Else replace a_i in *h* by the next more general constraint that is satisfied by *x*

3. Output hypothesis *h*

To illustrate this algorithm, assume the learner is given the sequence of training examples from the ***EnjoySport*** task

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

The first step of FIND-S is to initialize h to the most specific hypothesis in H

h - (Ø, Ø, Ø, Ø, Ø, Ø)

Consider the first training example

x1 = <Sunny, Warm, Normal, Strong, Warm, Same>, +

Observing the first training example, it is clear that hypothesis *h* is too specific. None of the "Ø" constraints in h are satisfied by this example, so each is replaced by the next *more general constraint* that fits the example

h1 = <Sunny, Warm, Normal, Strong, Warm, Same>

Consider the second training example

x2 = <Sunny, Warm, High, Strong, Warm, Same>, +

The second training example forces the algorithm to further generalize h, this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example

h2 = <Sunny, Warm, ?, Strong, Warm, Same>

Consider the third training example

x3 = <Rainy, Cold, High, Strong, Warm, Change>, -

Upon encountering the third training the algorithm makes no change to h. The FIND-S algorithm simply ignores every negative example.

h3 = < Sunny, Warm, ?, Strong, Warm, Same>

Consider the fourth training example

x4 = <Sunny Warm High Strong Cool Change>, +

The fourth example leads to a further generalization of h

h4 = < Sunny, Warm, ?, Strong, ?, ? >

The key property of the FIND-S algorithm

- It is incremental learning i.e., algorithm learns by processing one training example at a time, updating its hypothesis based on each example
- FIND-S is computationally efficient, especially for small to medium-sized datasets and can handle noisy data and incomplete training sets
- FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples
- FIND-S algorithm's final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H, and provided the training examples are correct.

Training Instances: (The below data is saved as *enjoysport.csv* file)

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Program:

```
import csv
with open('enjoysport.csv', 'r') as file:
   data = [row for row in csv.reader(file)]
   print("The total number of training instances are:",len(data)-1,'\n',data[1:])

num_attribute = len(data[0])-1

# Initial hypothesis
hypothesis = ['0']*num_attribute

for i in range(0, len(data)):
   if data[i][num_attribute] == 'yes':
      for j in range(0, num_attribute):
         if hypothesis[j] == '0' or hypothesis[j] == data[i][j]:
            hypothesis[j] = data[i][j]
         else:
            hypothesis[j] = '?'
   print("\n The hypothesis for the training instance {} is : \n".format(i),hypothesis)

print("\n The Maximally specific hypothesis for the training instances is ", hypothesis)
```

Output:

```
The total number of training instances are: 4
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']

The hypothesis for the training instance 0 is:
 ['0', '0', '0', '0', '0', '0']

The hypothesis for the training instance 1 is:
 ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

The hypothesis for the training instance 2 is:
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 3 is:
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 4 is:
 ['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instance is
['sunny', 'warm', '?', 'strong', '?', '?']
```

2	Aim	Demonstrate the working model and principle of candidate elimination algorithm
	Program	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples

CONCEPT - CANDIDATE-ELIMINATION LEARNING ALGORITHM

The CANDIDATE-ELIMINTION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

CANDIDATE-ELIMINATION Algorithm

Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      - h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S

- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      - h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

To illustrate this algorithm, assume the learner is given the sequence of training examples from the ***EnjoySport*** task

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

CANDIDATE-ELIMINTION algorithm begins by initializing the version space to the set of all hypotheses in H;

Initializing the G boundary set to contain the most general hypothesis in H

$$G0 \langle ?, ?, ?, ?, ?, ? \rangle$$

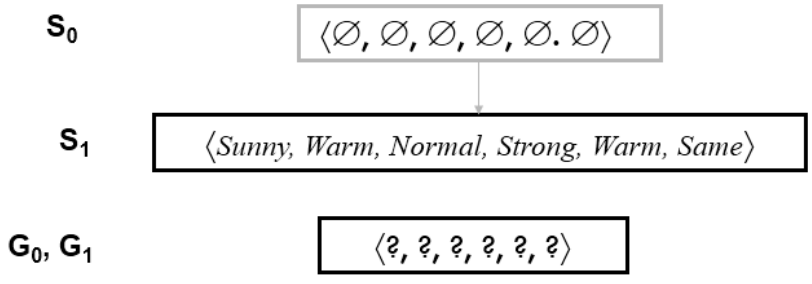
Initializing the S boundary set to contain the most specific (least general) hypothesis

$$S0 \langle \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing \rangle$$

- When the first training example is presented, the CANDIDATE-ELIMINTION algorithm checks the S boundary and finds that it is overly specific and it fails to cover the positive example.
- The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example
- No update of the G boundary is needed in response to this training example because G_o correctly covers this example

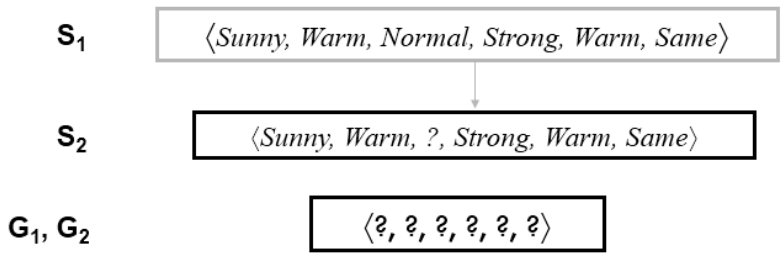
For training example d,

⟨Sunny, Warm, Normal, Strong, Warm, Same ⟩ +



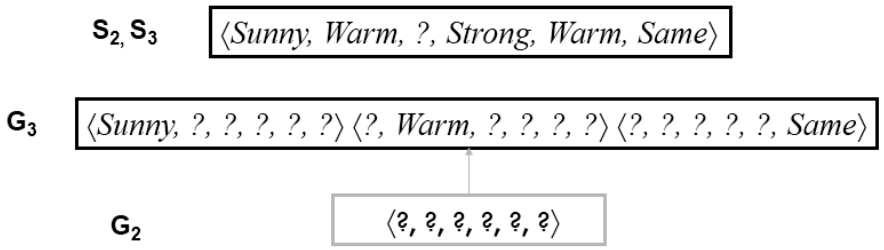
- When the second training example is observed, it has a similar effect of generalizing S further to S_2, leaving G again unchanged i.e., $G_2 = G_1 = G_0$

For training example d, ⟨Sunny, Warm, High, Strong, Warm, Same⟩ +



- Consider the third training example, this negative example reveals that the G boundary of the version space is overly general, that is, the hypothesis in G incorrectly predicts that this new example is a positive example.
- The hypothesis in the G boundary must therefore be specialized until it correctly classifies this new negative example

For training example d, ⟨Rainy, Cold, High, Strong, Warm, Change ⟩ –

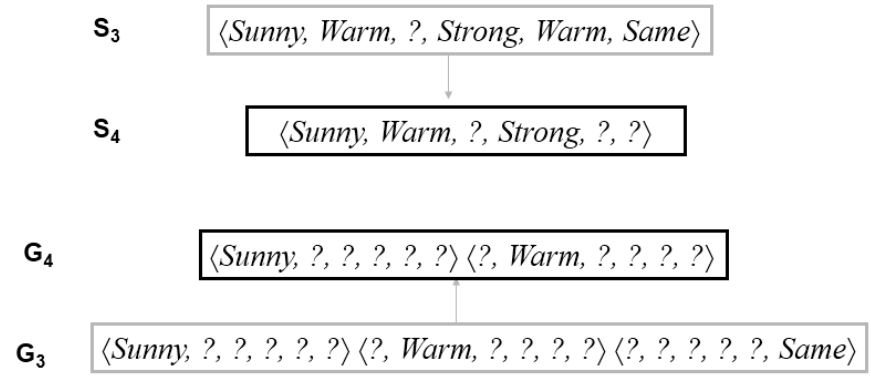


Given that there are six attributes that could be specified to specialize G_2, why are there only three new hypotheses in G_3?

For example, the hypothesis h = (?, ?, Normal, ?, ?, ?) is a minimal specialization of G_2 that correctly labels the new example as a negative example, but it is not included in G_3. The reason this hypothesis is excluded is that it is inconsistent with the previously encountered positive examples

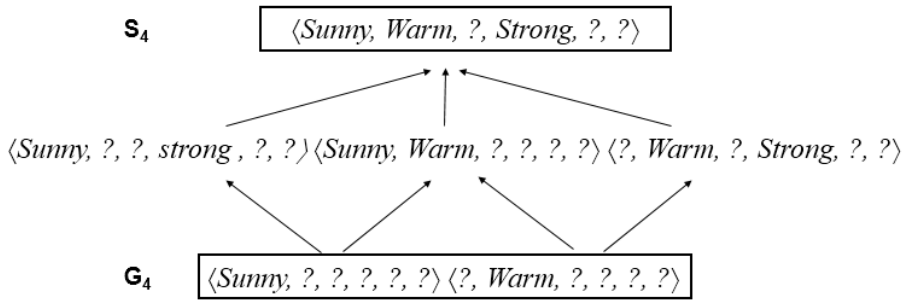
- Consider the fourth training example.

For training example d, ⟨Sunny, Warm, High, Strong, Cool Change ⟩ +



- This positive example further generalizes the S boundary of the version space. It also results in removing one member of the G boundary, because this member fails to cover the new positive example

After processing these four examples, the boundary sets S_4 and G_4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.



Training Instances: (The below data is saved as *enjoysport.csv* file)

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Program:

```
import pandas as pd

data = pd.read_csv('enjoysport.csv')
concepts = data.iloc[:, :-1].values
target = data.iloc[:, -1].values
n=len(concepts[0])-1
specific_h = ['0'] * n
general_h = ['?'] * n
print("The initialization of the specific and general hypothesis ")
print(" S0:",specific_h,"\n G0:",general_h)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for _ in range(len(specific_h))] for _ in range(len(specific_h))]
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            print(f"\n the {i+1} training instance is Positive \n",concepts[i])
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        else:
            print(f"\nThe {i+1} training instance is Negative \n",concepts[i])
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print(f"S{i+1}:\n", specific_h)
        print(f"G{i+1}:\n", general_h)

    general_h = [h for h in general_h if h != ['?' for _ in range(len(specific_h))]]
    return specific_h, general_h

s_final, g_final = learn(concepts, target)
print("\nThe Final Specific Hypothesis:")
print(s_final)
print("\nThe Final General Hypothesis:")
print(g_final)
```

Output:

```
The initialization of the specific and general hypothesis
S0: ['0', '0', '0', '0', '0']
G0: ['?', '?', '?', '?', '?']

The 1 training instance is Positive
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
S1:
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
G1:
[['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?']]

The 2 training instance is Positive
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
S2:
['sunny' 'warm' '?' 'strong' 'warm' 'same']
G2:
[['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?']]

The 3 training instance is Negative
['rainy' 'cold' 'high' 'strong' 'warm' 'change']
S3:
['sunny' 'warm' '?' 'strong' 'warm' 'same']
G3:
[['sunny', '?', '?', '?', '?', '?'],
['?', 'warm', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', 'same']]

The 4 training instance is Positive
['sunny' 'warm' 'high' 'strong' 'cool' 'change']
S4:
['sunny' 'warm' '?' 'strong' '?' '?']
```

```
G4:
[['sunny', '?', '?', '?', '?', '?'],
['?', 'warm', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?']]

The Final Specific Hypothesis:
['sunny' 'warm' '?' 'strong' '?' '?']

The Final General Hypothesis:
[['sunny', '?', '?', '?', '?', '?'],
['?', 'warm', '?', '?', '?', '?']]
```

3	Aim	To construct the Decision tree using the training data sets under supervised learning concept.
	Program	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.

CONCEPT - DECISION TREE

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.

THE BASIC DECISION TREE LEARNING ALGORITHM

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a Root node for the tree
- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = -
- If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples

- Otherwise Begin
  - A ← the attribute from Attributes that best* classifies Examples
  - The decision attribute for Root ← A
  - For each possible value, v_i, of A,
    - Add a new tree branch below *Root*, corresponding to the test A = v_i
    - Let *Examples* $_{vi}$, be the subset of Examples that have value v_i for *A*
    - If *Examples* $_{vi}$, is empty
      - Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples
      - Else below this new branch add the subtree
        ID3(*Examples* $_{vi}$, Targe_tattribute, Attributes – {A}))
- End
- Return Root

---

* The best attribute is the one with highest information gain

---

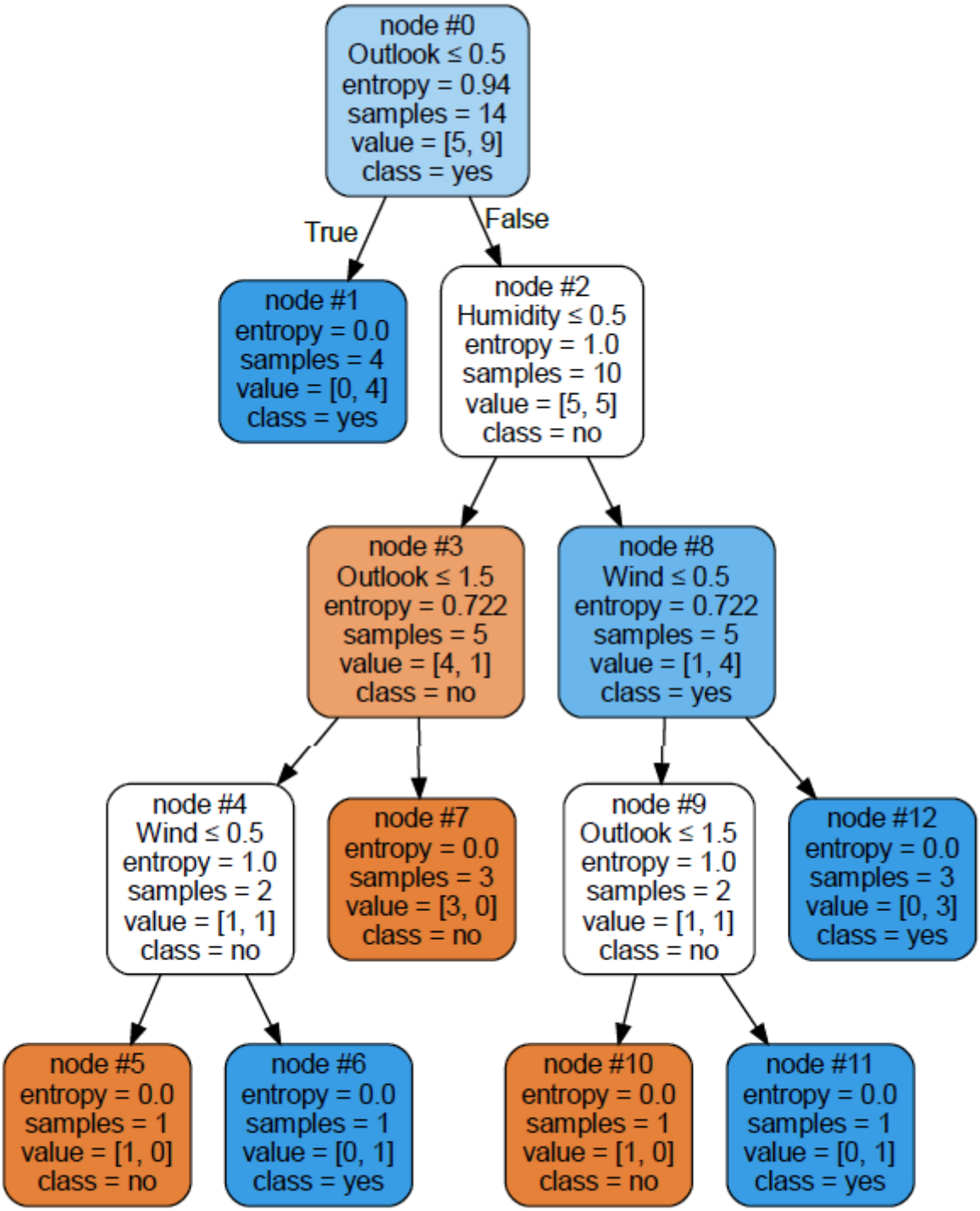
To illustrate this algorithm, consider the learning task represented by the training examples of below table. Here the target attribute ***PlayTennis***, which can have values yes or no for different days.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Program:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import export_graphviz
import graphviz

# Load the dataset
data = pd.read_csv('playtennis.csv')
df = pd.DataFrame(data)

# Convert categorical variables to numeric using LabelEncoder
label_encoder = LabelEncoder()
for column in df.columns:
    if df[column].dtype == 'object':
        df[column] = label_encoder.fit_transform(df[column])

# Separate features and target variable
X = df.drop('PlayTennis', axis=1)
y = df['PlayTennis']

# 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)

# Initialize the decision tree classifier
clf = DecisionTreeClassifier(criterion='entropy')

# Train the classifier
clf.fit(X, y)

# Visualize the decision tree
dot_data = export_graphviz(clf, out_file=None, feature_names=X.columns,
class_names=label_encoder.classes_, filled=True, rounded=True,
special_characters=True, node_ids=True)

graph = graphviz.Source(dot_data)
graph.view()
```

Output:


node #0
Outlook ≤ 0.5
entropy = 0.94
samples = 14
value = [5, 9]
class = yes
True
False
node #1
entropy = 0.0
samples = 4
value = [0, 4]
class = yes
node #2
Humidity ≤ 0.5
entropy = 1.0
samples = 10
value = [5, 5]
class = no
node #3
Outlook ≤ 1.5
entropy = 0.722
samples = 5
value = [4, 1]
class = no
node #8
Wind ≤ 0.5
entropy = 0.722
samples = 5
value = [1, 4]
class = yes
node #4
Wind ≤ 0.5
entropy = 1.0
samples = 2
value = [1, 1]
class = no
node #7
entropy = 0.0
samples = 3
value = [3, 0]
class = no
node #9
Outlook ≤ 1.5
entropy = 1.0
samples = 2
value = [1, 1]
class = no
node #12
entropy = 0.0
samples = 3
value = [0, 3]
class = yes
node #5
entropy = 0.0
samples = 1
value = [1, 0]
class = no
node #6
entropy = 0.0
samples = 1
value = [0, 1]
class = yes
node #10
entropy = 0.0
samples = 1
value = [1, 0]
class = no
node #11
entropy = 0.0
samples = 1
value = [0, 1]
class = yes

4	Aim	To understand the working principle of Artificial Neural network with feed forward and feed backward principle.
	Program	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

CONCEPT

An **Artificial Neural Network (ANN)** is a computational model inspired by the way biological neural networks in the human brain process information. It is used in machine learning and artificial intelligence to recognize patterns, make decisions, and solve complex problems.

Components of ANN:

Neurons (Nodes)

- Basic units of the network, similar to neurons in the brain.
- Each neuron receives input, processes it, and passes the output to the next layer.

Layers

An artificial neural network can be divided into three parts (layers), which are known as:

- Input layer: This layer is responsible for receiving information (data), signals, features, or measurements from the external environment. These inputs are usually normalized within the limit values produced by activation functions
- Hidden, intermediate, or invisible layers: These layers are composed of neurons which are responsible for extracting patterns associated with the process or system being analysed. These layers perform most of the internal processing from a network.
- Output layer: This layer is also composed of neurons, and thus is responsible for producing and presenting the final network outputs, which result from the processing performed by the neurons in the previous layers.

Weights and Biases

- Weights: Parameters that adjust the input signal's importance.
- Biases: Parameters that allow the activation function to shift to the left or right, improving model performance.

Activation Function:

- Determines whether a neuron should be activated or not by calculating a weighted sum and applying a threshold.
- Common activation functions include sigmoid, tanh, and ReLU (Rectified Linear Unit).

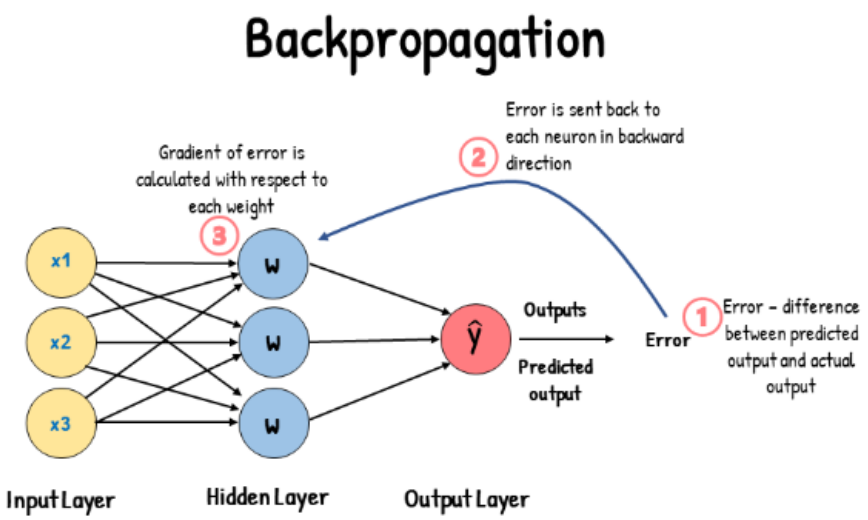


Image Source: https://www.analyticsvidhya.com/blog/2023/01/gradient-descent-vs-backpropagation-whats-the-difference/

Algorithm:

BACKPROPAGATION (*training_example, η, n_{in}, n_{out}, n_{hidden}*)

Each training example is a pair of the form $(\vec{x}, \vec{t})$, where $(\vec{x})$ is the vector of network input values, $(\vec{t})$ and is the vector of target network output values.
η is the learning rate (e.g., .05). n_i, is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.
The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
  - For each $(\vec{x}, \vec{t})$, in training examples, Do
    Propagate the input forward through the network:
    1. Input the instance $\vec{x}$, to the network and compute the output o_u of every unit u in the network.

    Propagate the errors backward through the network:
    2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

    3. For each hidden unit *h*, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

    4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

    Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

Instance:

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

Normalize the input

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

Program:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)     # two inputs [sleep, study]
y = np.array(([92], [86], [89]), dtype=float)           # one output [Expected % in Exams]
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
#weight of the link from input node to hidden node
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
# bias of the link from input node to hidden node
bh=np.random.uniform(size=(1,hiddenlayer_neurons))

#weight of the link from hidden node to output node
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
#bias of the link from hidden node to output node
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 weights 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]]
```

5	Aim	Demonstrate the classifier using Naïve bayes classifier algorithm
	Program	Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

CONCEPT

- Naive Bayes is a probabilistic classifier based on Bayes' theorem, which provides a way to calculate the probability of a class given the observed features (evidence).
- The "naive" part of Naive Bayes assumes that the features are independent given the class label, which simplifies the computation.

Bayes' Theorem is stated as:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

where,

- **P(h|D)** is the probability of hypothesis h given the data D. This is called the **posterior probability**.
- **P(D|h)** is the probability of data d given that the hypothesis h was true.
- **P(h)** is the probability of hypothesis h being true. This is called the **prior probability of h.**
- **P(D)** is the probability of the data. This is called the **prior probability of D**

After calculating the posterior probability for a number of different hypotheses h, and is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any such maximally probable hypothesis is called a ***maximum a posteriori (MAP) hypothesis***.

Bayes theorem to calculate the posterior probability of each candidate hypothesis is h_{MAP} is a MAP hypothesis provided

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D|h)P(h)$$

Gaussian Naive Bayes

- A Gaussian Naive Bayes algorithm is a special type of Naïve Bayes algorithm. It's specifically used when the features have continuous values. It's also assumed that all the features are following a Gaussian distribution i.e., normal distribution
- In GaussianNB, the likelihood P(X|C) is assumed to be Gaussian

Mean $$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Variance: $$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

Normal distribution $$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where:

- x_i is the value of feature i.
- μ_C is the mean of the feature i for class C.
- σ_C^2 is the variance of the feature i for class C.

Steps in GaussianNB:

1. Calculate Prior Probabilities P(C) for each class C from the training data.
2. Calculate Mean and Variance for each feature and each class
3. Use the Gaussian distribution to compute the likelihood P(x_i|C) for each feature value.
4. Calculate Posterior Probability: Combine the prior and likelihood to calculate the posterior probability P(C|X) for each class.
5. Predict Class: The class with the highest posterior probability is chosen as the predicted class.

Instance:

- The dataset used in this program is the ***Pima Indians Diabetes problem***.
- This dataset is comprised of 768 observations of medical details for Pima Indians patents. The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.
- The attributes are ***Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabeticPedigreeFunction, Age, Outcome***
- Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0)

Sample Instance:

Examples	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0	0.232	54	1

Program:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

# Load data
df = pd.read_csv("pima_indian.csv")

# Define feature columns and predicted class names
features = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
target = ['diabetes']

# Prepare features and target variable
X = df[features].values
y = df[target].values

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

print('Total number of Training Data:', y_train.shape)
print('Total number of Test Data:', y_test.shape)

# Train Naive Bayes (NB) classifier
clf = GaussianNB()
clf.fit(X_train, y_train)

# Predictions
predicted = clf.predict(X_test)
predict_test_data = clf.predict([[6, 148, 72, 35, 0, 33.6, 0.627, 50]])

# Model Evaluation
print('\nConfusion matrix:')
print(metrics.confusion_matrix(y_test, predicted))
print('\nAccuracy of the classifier:', metrics.accuracy_score(y_test, predicted))
print('\nPrecision:', metrics.precision_score(y_test, predicted))
print('\nRecall:', metrics.recall_score(y_test, predicted))
print("\nPredicted Value for individual Test Data:", predict_test_data)
```

Output:

```
Total number of Training Data: (614, 1)
Total number of Test Data: (154, 1)

Confusion matrix:
[[91 14]
 [22 27]]

Accuracy of the classifier: 0.7662337662337663

Precision: 0.6585365853658537

Recall: 0.5510204081632653

Predicted Value for individual Test Data: [1]
```

6	Aim	Demonstrate and Analyse the results sets obtained from Bayesian belief network Principle.
	Program	Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Python ML library classes/API.

CONCEPT –

A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable.

Bayesian network consists of two major parts: a directed acyclic graph and a set of conditional probability distributions

- The directed acyclic graph is a set of random variables represented by nodes.
- The conditional probability distribution of a node (random variable) is defined for every possible outcome of the preceding causal node(s).

For illustration, consider the following example. Suppose we attempt to turn on our computer, but the computer does not start (observation/evidence). We would like to know which of the possible causes of computer failure is more likely. In this simplified illustration, we assume only two possible causes of this misfortune: electricity failure and computer malfunction.
The corresponding directed acyclic graph is depicted in below figure.

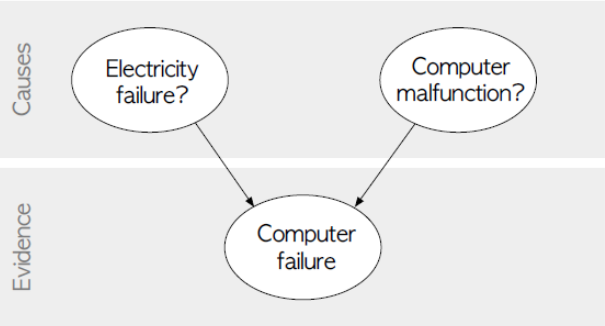


Fig: Directed acyclic graph representing two independent possible causes of a computer failure.

The goal is to calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e. P [Cause | Evidence].

Training Instances: (The below data is saved as *heart.csv* file)

Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Database	0	1	2	3	4	Total
Cleveland	165	55	36	35	13	303

Some instance from the dataset:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

Attribute Information

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
    - Value 1: typical angina
    - Value 2: atypical angina
    - Value 3: non-anginal pain
    - Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
    - Value 0: normal
    - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest

11. slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
13. Heartdisease: It is integer valued from 0 (no presence) to 4.
    0: No heart disease
    1: Mild heart disease
    2: Moderate heart disease
    3: Severe heart disease
    4: Very severe heart disease

Program:

```
import numpy as np
import pandas as pd
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianNetwork
from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv("heart.csv")
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())

model= BayesianNetwork([('age','heartdisease'), ('sex','heartdisease'),
('exang','heartdisease'), ('cp','heartdisease'), ('heartdisease','restecg'),
('heartdisease','chol')])

print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

Output:

```
Sample instances from the dataset are given below

   age  sex  cp  trestbps  chol  ...  oldpeak  slope  ca  thal  heartdisease
0   63    1   1       145   233  ...      2.3      3   0     6             0
1   67    1   4       160   286  ...      1.5      2   3     3             2
2   67    1   4       120   229  ...      2.6      2   2     7             1
3   37    1   3       130   250  ...      3.5      3   0     3             0
4   41    0   2       130   204  ...      1.4      1   0     3             0

[5 rows x 14 columns]

Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1016 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2361 |
+-----------------+---------------------+
| heartdisease(3) |              0.2017 |
+-----------------+---------------------+
| heartdisease(4) |              0.4605 |
+-----------------+---------------------+

 2. Probability of HeartDisease given evidence= cp
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3742 |
+-----------------+---------------------+
| heartdisease(1) |              0.2018 |
+-----------------+---------------------+
| heartdisease(2) |              0.1375 |
+-----------------+---------------------+
| heartdisease(3) |              0.1541 |
+-----------------+---------------------+
| heartdisease(4) |              0.1323 |
+-----------------+---------------------+
```

7	Aim	Implement and demonstrate the working model of K-means clustering algorithm with Expectation Maximization Concept
	Program	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 Python ML library classes/API in the program

CONCEPT - Expectation-Maximization (EM)

- Clustering is a type of unsupervised learning wherein data points are grouped into different sets based on their degree of similarity.
- The Expectation-Maximization (EM) algorithm is a popular method for finding maximum likelihood estimates of parameters in statistical models, particularly when the model depends on unobserved latent variables. In the context of clustering, the EM algorithm is often used with Gaussian Mixture Models (GMMs).
- The goal of clustering is to partition data points into clusters such that points within the same cluster are more similar to each other than to points in other clusters. GMMs assume that the data points are generated from a mixture of several Gaussian distributions, each representing a cluster.

Steps of the EM Algorithm

1. Initialization: Initialize the parameters of the Gaussian components such as means, covariances, and mixing coefficients. These can be initialized randomly or using some heuristic method.
2. **Expectation Step (E-Step):**

   - Calculate the responsibility that each Gaussian component takes for each data point. This is done using the current parameter estimates.
   - The responsibility r_{ij} is the probability that the j-th data point belongs to the i-th Gaussian component.

$$r_{ij} = \frac{\pi_i \mathcal{N}(x_j|\mu_i, \Sigma_i)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_j|\mu_k, \Sigma_k)}$$

   where π_i is the mixing coefficient, $\mathcal{N}(x_j|\mu_i, \Sigma_i)$ is the Gaussian probability density function, and K is the number of Gaussian components.

3. **Maximization Step (M-Step):**

   - Update the parameters of the Gaussian components using the responsibilities calculated in the E-Step.
   - The new parameters are calculated as follows:
     - **Mixing Coefficients:**

$$\pi_i = \frac{1}{N} \sum_{j=1}^{N} r_{ij}$$

     - **Means:**

$$\mu_i = \frac{\sum_{j=1}^{N} r_{ij} x_j}{\sum_{j=1}^{N} r_{ij}}$$

     - **Covariances:**

$$\Sigma_i = \frac{\sum_{j=1}^{N} r_{ij} (x_j - \mu_i)(x_j - \mu_i)^T}{\sum_{j=1}^{N} r_{ij}}$$

where N is the number of data points.

4. **Convergence Check:**

   - Check if the parameters have converged. This can be done by monitoring the change in the log-likelihood of the data given the model parameters. If the change is below a certain threshold, the algorithm has converged.

$$\log L = \sum_{j=1}^{N} \log \left(\sum_{i=1}^{K} \pi_i \mathcal{N}(x_j | \mu_i, \Sigma_i) \right)$$

5. **Repeat:**

   - Repeat the E-Step and M-Step until convergence.

K – Means Clustering

The algorithm will categorize the items into k groups of similarity. To calculate that similarity, use the Euclidean distance as measurement.

Algorithm

Input: Data points X = {x1, x2, ..., xn}, number of clusters K
Output: Cluster assignments for each data point

1. Initialize K cluster centroids randomly
2. Repeat until convergence or a maximum number of iterations:
    a. Assign each data point x_i to the nearest centroid
    b. Recalculate the centroids as the mean of the data points assigned to each cluster
3. Return the cluster assignments

silhouette_score

The **silhouette_score** is a metric used to evaluate the quality of clustering results. It measures how similar an object is to its own cluster compared to other clusters. The silhouette score ranges from -1 to 1, where:

- **1** indicates that the data points are well-clustered, meaning they are close to other points in the same cluster and far from points in other clusters.
- **0** indicates that the data points are on or very close to the decision boundary between two neighboring clusters.
- **-1** indicates that the data points may have been assigned to the wrong cluster, as they are closer to points in other clusters than to points in their own cluster.

Formula

$$s = \frac{b-a}{\max(a,b)}$$

The silhouette score for a single data point is calculated using the formula:

- a - is the mean distance between a data point and all other points in the same cluster.
- b - is the mean distance between a data point and all other points in the nearest cluster (i.e., the cluster that minimizes this mean distance).

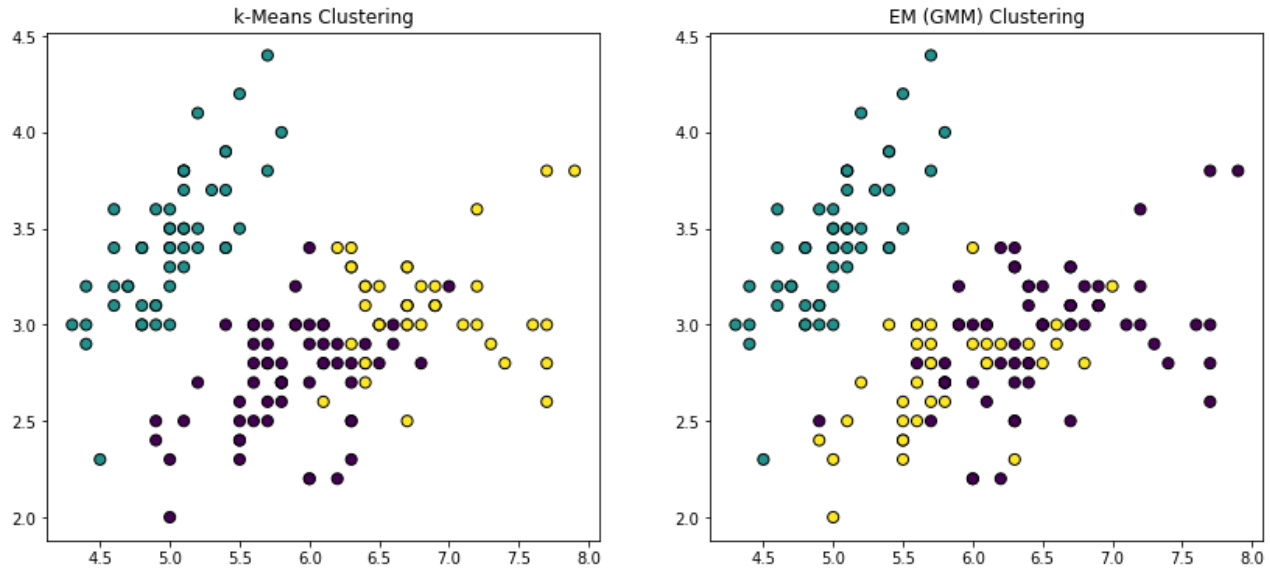
The overall silhouette score for a clustering solution is the mean silhouette score of all data points

Program:

```
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score

# Step 1: Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Step 2: Cluster the data using k-Means algorithm
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X)
kmeans_silhouette = silhouette_score(X, kmeans_labels)

# Step 3: Cluster the data using EM algorithm (Gaussian Mixture Model)
gmm = GaussianMixture(n_components=3, random_state=42)
gmm_labels = gmm.fit_predict(X)
gmm_silhouette = silhouette_score(X, gmm_labels)

# Step 4: Compare the clustering results and visualize them
print(f"Silhouette Score for k-Means: {kmeans_silhouette}")
print(f"Silhouette Score for EM (GMM): {gmm_silhouette}")

# Visualization of the clusters
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# k-Means Clustering
ax1.scatter(X[:, 0], X[:, 1], c=kmeans_labels, cmap='viridis', marker='o', edgecolor='k',
s=50)
ax1.set_title('k-Means Clustering')

# EM (GMM) Clustering
ax2.scatter(X[:, 0], X[:, 1], c=gmm_labels, cmap='viridis', marker='o', edgecolor='k',
s=50)
ax2.set_title('EM (GMM) Clustering')

plt.show()
```

```
# Step 5: Comment on the quality of clustering
if kmeans_silhouette > gmm_silhouette:
    print("k-Means clustering provides better quality clusters according to the silhouette score.")
else:
    print("EM (GMM) clustering provides better quality clusters according to the silhouette score.")
```

Output




```
Silhouette Score for k-Means: 0.551191604619592
Silhouette Score for EM (GMM): 0.5011761635067206

k-Means clustering provides better quality clusters according to the silhouette score.
```

8	Aim	Demonstrate and analyse the results of classification based on KNN Algorithm
	Program	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.

CONCEPT - k-Nearest Neighbour

- The most basic instance-based method is the K- Nearest Neighbor Learning. This algorithm assumes all instances correspond to points in the n-dimensional space R^n.
- The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.
- Let an arbitrary instance x be described by the feature vector

  $$((a_1(x), a_2(x), \ldots\ldots, a_n(x))$$

  Where, $a_r(x)$ denotes the value of the r^{th} attribute of instance x.

- Then the distance between two instances x_i and x_j is defined to be $d(x_i , x_j)$
  Where,

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

- In nearest-neighbor learning the target function may be either discrete-valued or real-valued.

Training algorithm:

- For each training example (x, f (x)), add the example to the list training examples

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from training examples that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

  - Where, $f(x_i)$ function to calculate the mean value of the k nearest training examples.

Training Instances:

- Iris Plants Dataset: Dataset contains 150 instances, 50 in each of three classes – Iris-setosa, Iris-versicolor, Iris-virginica
- There are four Attributes and values in centimetres- Sepal length, Sepal width, Petal length, Petal width

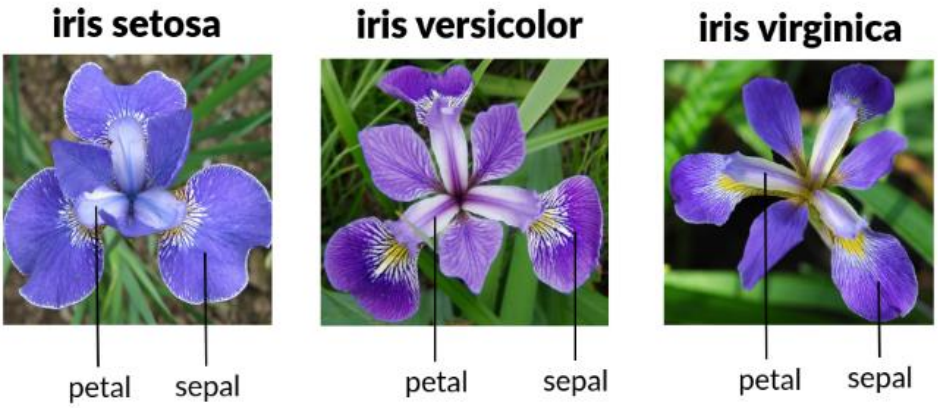
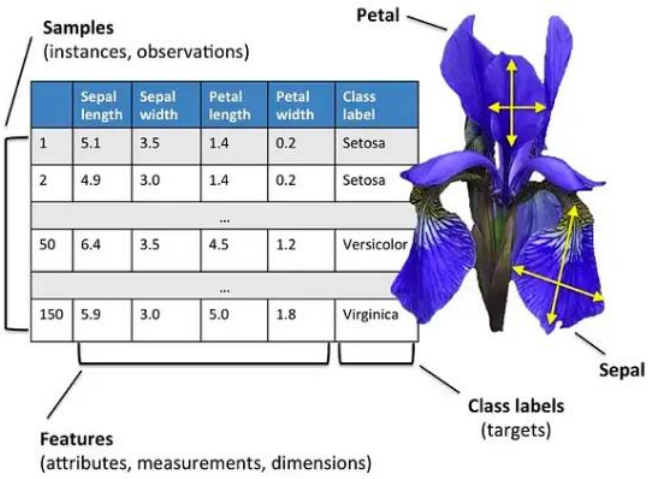


Image Source: https://www.analyticsvidhya.com/blog/2022/06/iris-flowers-classification-using-machine-learning/



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

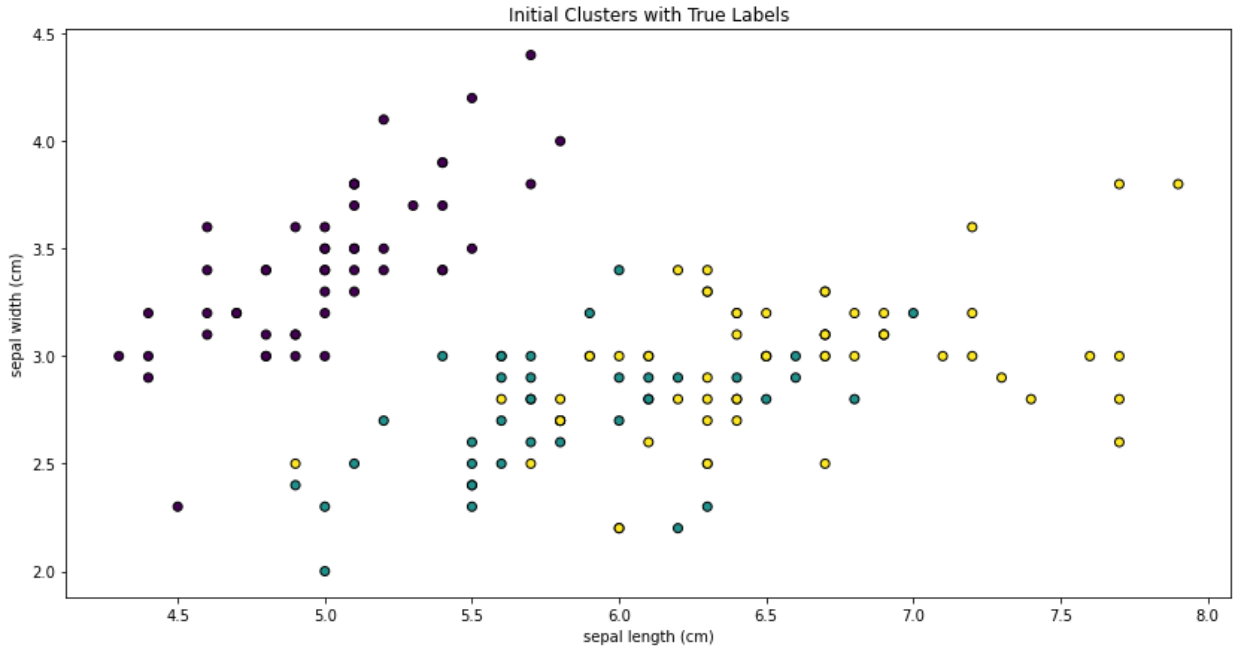
Image Source: https://eminebozkus.medium.com/exploring-the-iris-flower-dataset-4e000bcc266c

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Program:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = datasets.load_iris()
x = iris.data
y = iris.target

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)

# Plot the initial clusters
plt.figure(figsize=(14, 7))

# Plot the initial clusters (True labels)
plt.scatter(x[:, 0], x[:, 1], c=y, cmap='viridis', marker='o', edgecolor='k')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('Initial Clusters with True Labels')
plt.show()

#To Training the model and Nearest nighbors K=3
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(x_train, y_train)

#To make predictions on our test data
y_pred=classifier.predict(x_test)

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))

print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

Output:




```
Confusion Matrix
[[10  0  0]
 [ 0  8  1]
 [ 0  0 11]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

9	Aim	Understand and analyse the concept of Regression algorithm techniques.
	Program	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

CONCEPT - Locally Weighted Regression (LWR)

Introduction

- Locally Weighted Regression (LWR), also known as Locally Weighted Scatterplot Smoothing (LOWESS), is a non-parametric regression method that fits multiple regressions in local neighbourhoods. It is especially useful for non-linear datasets.
- Non-parametric regression is a type of regression analysis that does not assume a predetermined form for the relationship between the independent and dependent variables. Instead, it seeks to model the relationship directly from the data, allowing for greater flexibility in capturing the underlying patterns and structures.

Key Concepts of LWR

1. **Local Fitting**:
   - For each query point x_q, LWR fits a local regression model using data points that are close to x_q.
   - The "closeness" is determined by a weighting function, typically a kernel function.
2. **Weighting Function**:
   - A kernel function K assigns weights to data points based on their distance from the query point.
   - A common choice is the Gaussian kernel:

$$K(x, x_i) = \exp\left(-\frac{(x - x_i)^2}{2\tau^2}\right)$$

3. **Weighted Linear Regression**:
   - For a given query point x_q, LWR solves a weighted linear regression problem:

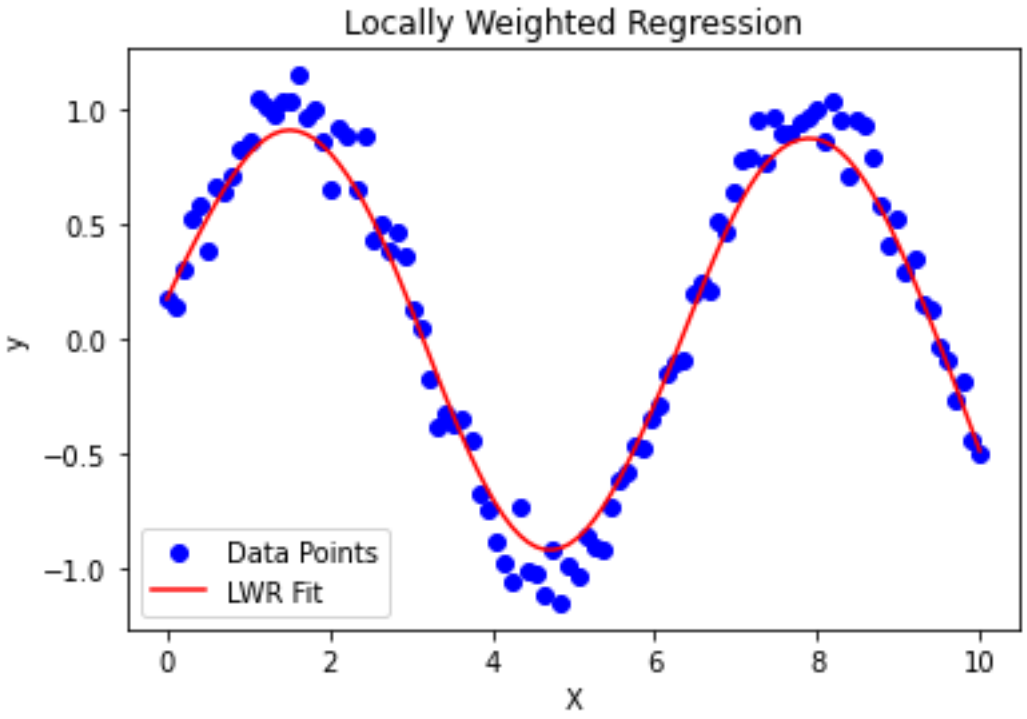
$$\theta = (X^T W X)^{-1} X^T W y$$

   where W is a diagonal matrix of weights calculated using the kernel function for each data point relative to x_q.
4. **Bandwidth Parameter (τ)**:
   - The bandwidth parameter τ determines the scale of the neighborhood. A smaller τ results in a more localized model, while a larger τ leads to a smoother, more global model.

Program:

```
import numpy as np
import matplotlib.pyplot as plt

def kernel(x, x_i, tau):
    return np.exp(-np.sum((x - x_i)**2) / (2 * tau**2))

def locally_weighted_regression(X, y, tau):
    m = X.shape[0]
    y_pred = np.zeros(m)

    for i in range(m):
        weights = np.array([kernel(X[i], X[j], tau) for j in range(m)])
        W = np.diag(weights)
        XTX = X.T @ W @ X
        XTX_inv = np.linalg.pinv(XTX)
        theta = XTX_inv @ X.T @ W @ y
        y_pred[i] = X[i] @ theta

    return y_pred

# Generate synthetic dataset
np.random.seed(0)
X = np.linspace(0, 10, 100)
y = np.sin(X) + np.random.normal(scale=0.1, size=X.shape)

# Add a column of ones to include the intercept term
X_bias = np.c_[np.ones(X.shape[0]), X]

# Perform Locally Weighted Regression
tau = 0.5  # Bandwidth parameter
y_pred = locally_weighted_regression(X_bias, y, tau)

# Plot the results
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X, y_pred, color='red', label='LWR Fit')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Locally Weighted Regression')
plt.legend()
plt.show()
```

Output:


Locally Weighted Regression
1.0
0.5
0.0
−0.5
−1.0
y
0
2
4
6
8
10
X
Data Points
LWR Fit

10	Aim	Implement and demonstrate classification algorithm using Support vector machine Algorithm.
	Program	Implement and demonstrate the working of SVM algorithm for classification.

CONCEPT - Support Vector Machine

- A Support Vector Machine (SVM) is a very powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression, and even outlier detection.
- SVMs are well suited for classification of complex but small- or medium-sized datasets.

Linear SVM Classification

Hyperplane

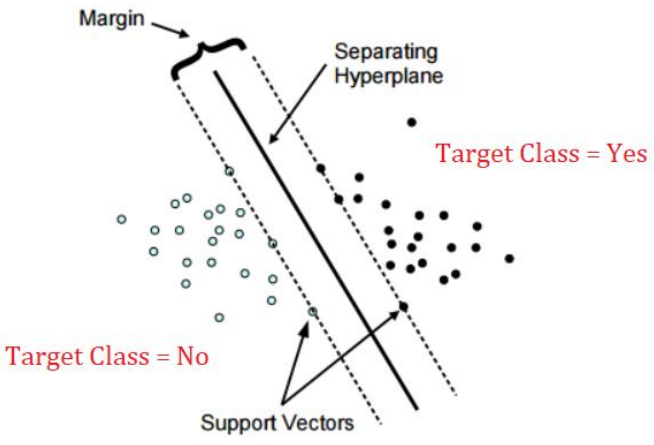
A hyperplane is a decision boundary which separates between given set of data points having different class labels. The SVM classifier separates data points using a hyperplane with the maximum amount of margin. This hyperplane is known as the maximum margin hyperplane and the linear classifier it defines is known as the maximum margin classifier.

Margin

A margin is a separation gap between the two lines on the closest data points. It is calculated as the perpendicular distance from the line to support vectors or closest data points. In SVMs, we try to maximize this separation gap so that we get maximum margin.

Support Vectors

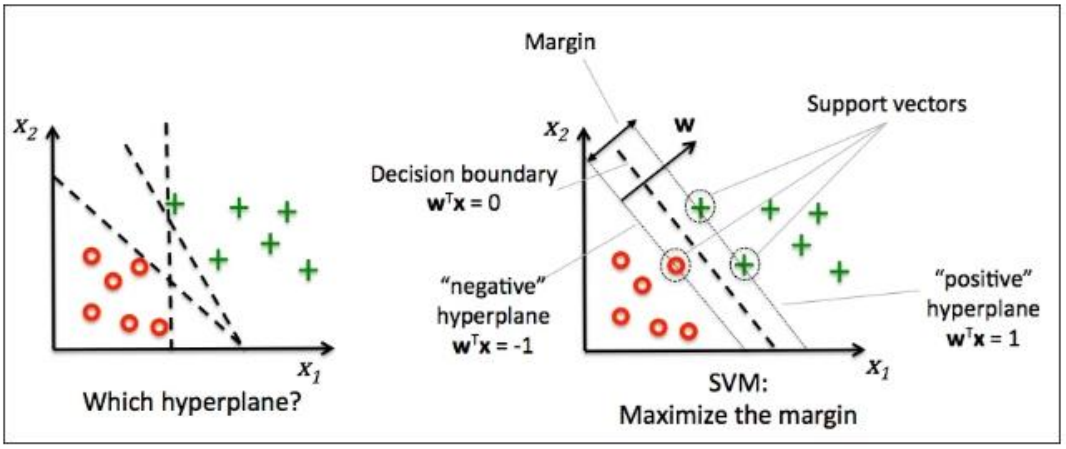
Support vectors are the sample data points, which are closest to the hyperplane. These data points will define the separating line or hyperplane better by calculating margins.



In SVMs, the main objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset.
SVM searches for the maximum margin hyperplane in the following 2 step process –

1. Generate hyperplanes which segregates the classes in the best possible way. There are many hyperplanes that might classify the data. We should look for the best hyperplane that represents the largest separation, or margin, between the two classes.
2. So, choose the hyperplane so that distance from it to the support vectors on each side is maximized. If such a hyperplane exists, it is known as the ***maximum margin hyperplane*** and the linear classifier it defines is known as a ***maximum margin classifier***.

The following diagram illustrates the concept of maximum margin and maximum margin hyperplane.



source: https://www.kaggle.com/code/prashant111/svm-classifier-tutorial#1.-Introduction-to-Support-Vector-Machines-

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.decomposition import PCA

# Step 1: Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Step 2: Split the data 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)

# Step 3: Train the SVM classifier
svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)
svm_classifier.fit(X_train, y_train)

# Step 4: Make predictions
y_pred = svm_classifier.predict(X_test)

# Step 5: Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
```

Output

```
Confusion Matrix:
[[10  0  0]
 [ 0  8  1]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30

Accuracy: 0.97
```

VIVA QUESTION

For a practical examination on the topics and programs mentioned, here are some possible viva questions along with brief explanations that might be useful for each topic:

1. Find-S Algorithm

What is the Find-S algorithm?
- The Find-S algorithm is a simple algorithm used to find the most specific hypothesis that fits all the positive examples in a given dataset.

What are the limitations of the Find-S algorithm?
- It only finds the most specific hypothesis and ignores negative examples, making it susceptible to noise and unable to handle inconsistencies in the training data.

How does the Find-S algorithm work step-by-step?
- It initializes the most specific hypothesis and then generalizes it only when necessary to cover positive examples.

2. Candidate Elimination Algorithm

What is the Candidate Elimination algorithm?
- The Candidate Elimination algorithm finds all hypotheses that are consistent with the training examples by maintaining a version space (a set of all plausible hypotheses).

How does the Candidate Elimination algorithm handle both positive and negative examples?
- It updates the version space by specializing the general hypotheses to exclude negative examples and generalizing the specific hypotheses to include positive examples.

What is the significance of the version space in the Candidate Elimination algorithm?
- The version space represents all hypotheses that are consistent with the observed training examples.

3. Decision Tree using ID3 Algorithm

What is the ID3 algorithm?
- The ID3 (Iterative Dichotomiser 3) algorithm is used to create a decision tree by using a top-down, greedy approach to select the attribute that maximizes information gain.

How does the ID3 algorithm determine the best attribute to split the data?
- It uses the concept of information gain, which is based on entropy, to determine the best attribute for splitting the data.

What is overfitting in the context of decision trees, and how can it be avoided?
- Overfitting occurs when the model captures noise in the training data. It can be avoided by pruning the tree or setting a limit on the tree's depth.

4. Artificial Neural Network with Backpropagation

What is a feedforward neural network?
- A feedforward neural network is a type of artificial neural network where the connections between the nodes do not form a cycle.

What is backpropagation, and why is it important?
- Backpropagation is an algorithm used for training neural networks by adjusting the weights based on the error rate obtained in the previous epoch (iteration). It is important for optimizing the neural network.

What are the main components of a neural network?
- Neurons (nodes), weights, biases, activation functions, and the architecture (layers and connections).

5. Naive Bayes Classifier

What is the Naive Bayes classifier?
- The Naive Bayes classifier is a probabilistic classifier based on Bayes' theorem, with the assumption of feature independence.

How does the Naive Bayes classifier handle continuous data?
- It can handle continuous data by assuming a Gaussian distribution or using other methods like binning.

What are some common applications of the Naive Bayes classifier?
- Spam detection, text classification, sentiment analysis.

6. Bayesian Belief Network

What is a Bayesian Belief Network (BBN)?
- A BBN is a graphical model that represents probabilistic relationships among variables.

How does a BBN help in medical diagnosis?
- It helps by modeling the probabilistic relationships between symptoms and diseases, allowing for inference and diagnosis based on observed symptoms.

What is conditional independence in the context of BBNs?
- Conditional independence is the property that a variable is independent of another variable given a third variable.

7. K-Means Clustering with Expectation Maximization

What is the K-means clustering algorithm?
- K-means is an unsupervised learning algorithm that partitions the data into K clusters by minimizing the variance within each cluster.

What is the Expectation Maximization (EM) algorithm?
- The EM algorithm is used for finding maximum likelihood estimates of parameters in probabilistic models, especially with incomplete data.

How do you compare the results of K-means and EM algorithms?
- By evaluating the clustering quality using metrics like the silhouette score or analyzing cluster cohesion and separation.

8. K-Nearest Neighbors (KNN) Algorithm

What is the KNN algorithm?
- KNN is a simple, instance-based learning algorithm that classifies a sample based on the majority class of its K nearest neighbors.

How is the value of K selected in the KNN algorithm?
- The value of K is typically selected using cross-validation to balance bias and variance.

What are the limitations of the KNN algorithm?
- It can be computationally expensive, especially with large datasets, and is sensitive to irrelevant features and the scale of the data.

9. Locally Weighted Regression

What is Locally Weighted Regression?
- Locally Weighted Regression is a non-parametric regression method that fits a model to a subset of data points near the query point.

How does Locally Weighted Regression handle the data?
- It assigns weights to the data points based on their distance from the query point and fits a regression model to the weighted data.

What are the advantages of Locally Weighted Regression?
- It is flexible and can model complex relationships without a predetermined form.

10. Support Vector Machine (SVM)

What is a Support Vector Machine (SVM)?
- SVM is a supervised learning algorithm used for classification and regression tasks that finds the optimal hyperplane separating different classes.

What are the kernel functions in SVM, and why are they important?
- Kernel functions (linear, polynomial, RBF) allow SVM to operate in higher-dimensional spaces, enabling it to handle non-linearly separable data.

How does SVM handle imbalanced datasets?
- By using techniques like class weighting, resampling, or adjusting the decision threshold.