KNS INSTITUTE OF TECHNOLOGY

HEGDE NAGAR, TIRUMENAHALLI, KOGILU ROAD, BENGALURU – 64

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

MACHINE LEARNING LABORATORY

[21AIL66]

Academic Year 2023 - 2024

Prepared by:
Ms. Ayesha Javeriya
Assistant Professor, AI/ML
KNSIT

Reviewed by:
Dr. Ajaz Ali Khan
Head of the Department, ECE/AIML
KNSIT, Bengaluru

Name of the Student: _____

University Serial Number: _____

Semester: _____ **Batch:** _____

Program Outcomes

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
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.
3. **Design/development of solutions**: Design solutions for complex engineering problems anddesign 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.
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.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modernengineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assesssocietal, health, safety, legal and cultural issues and the consequent responsibilities relevant tothe professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and needfor sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leaderin diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and writeeffective reports and design documentation, make effective presentations, and give and receiveclear instructions.
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.

Course outcomes

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

Syllabus

MACHINE LEARNING LABORATORY			
Course Code	21AIL66	CIE Marks	50
Teaching Hours/Week(L:T:P:S)	0:0:2:0	SEE Marks	50
Total Hours of Pedagogy	24	Total Marks	100
Credits	1	Exam Hours	03

Course Learning Objectives:

CLO 2. To learn and understand the Importance Machine learning Algorithms
CLO 3. Compare and contrast the learning techniques like ANN approach, Bayesian learning and reinforcement learning.
CLO 4. Able to solve and analyse the problems on ANN, Instance based learning and Reinforcement learning techniques.
CLO 5. To impart the knowledge of clustering and classification Algorithms for predictions and evaluating Hypothesis.

	Prerequisite
	• Students should be familiarized about Python installation and setting Python environment • Usage and installation of Anaconda should be introduced https://www.anaconda.com/products/individual • Should have the knowledge about Probability theory,Statistics theory and linear Algebra. • Should have the knowledge of numpy,pandas,scikit-learn and scipy library packages.
Sl. No.	**PART A – List of problems for which student should develop program and execute in the Laboratory**
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. **Text Book 1: Ch2**
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. **Text Book 1: Ch2** Reference: https://www.youtube.com/watch?v=tfpAm4kxGQI
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. **Text Book 1: Ch 3**
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. **Text Book 1: Ch 4**

5	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. **Text Book 1: Ch6**
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. **Text Book 1: Ch 6**
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. **Text Book 1: Ch 8**
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. **Text Book 1: Ch 8**
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. **Text Book 1: Ch8**
10	Aim: Implement and demonstrate classification algorithm using Support vector machine Algorithm. Program: Implement and demonstrate the working of SVM algorithm for classification. **Text Book 2: Ch6**
Pedagogy	For the above experiments the following pedagogy can be considered. Problem based learning, Active learning, MOOC, Chalk & Talk
PART B	
	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.

Course Outcomes: At the end of the course the student will be able to:

CO 1. Understand the Importance of different classification and clustering algorithms.
CO 2. Demonstrate the working of various algorithms with respect to training and test data sets.
CO 3. Illustrate and analyze the principles of Instance based and Reinforcement learning techniques. CO 4. Elicit the importance and Applications of Supervised and unsupervised machine learning.
CO 5. Compare and contrast the Bayes theorem principles and Q learning approach.

Assessment Details (both 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)
    - *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).
- The duration of SEE is 03 hours
- Rubrics suggested in Annexure-II of Regulation book

Text 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)

Suggested Web Links / E Resource

1. https://www.kaggle.com/general/95287
2. https://web.stanford.edu/~hastie/Papers/ESLII.pdf

Introduction

Machine learning

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome.

Machine learning tasks

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:

Supervised learning: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback:

Semi-supervised learning: the computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.

Active learning: the computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labeling.

Reinforcement learning: training data (in form of rewards and punishments) is given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent.

Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

<table>
<tr><th>Supervised learning</th><th>Un Supervised learning</th><th>Instance based learning</th></tr>
<tr><td>Find-s algorithm</td><td>EM algorithm</td><td rowspan="8">Locally weighted Regression algorithm</td></tr>
<tr><td>Candidate elimination algorithm</td><td rowspan="7">K means algorithm</td></tr>
<tr><td>Decision tree algorithm</td></tr>
<tr><td>Back propagation Algorithm</td></tr>
<tr><td>Naïve Bayes Algorithm</td></tr>
<tr><td>K nearest neighbour algorithm(lazy learningalgorithm)</td></tr>
</table>

Machine learning applications

In classification, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised manner. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam". In regression, also a supervised problem, the outputs are continuous rather than discrete.

In clustering, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task. Density estimation finds the distribution of inputs in some space. Dimensionality reduction simplifies inputs by mapping them into a lower- dimensional space. Topic modeling is a related problem, where a program is givena list of human language documents and is tasked with finding out which documents cover similar topics.

Machine learning Approaches

Decision tree learning: Decision tree learning uses a decision tree as a predictive model, which maps observations about an item to conclusions about the item's target value. Association rule learning Association rule learning is a method for discovering interesting relations between variables in large databases.

Artificial neural networks

An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is vaguely inspired by biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.

Deep learning

Falling hardware prices and the development of GPUs for personal use in the last few years have contributed to the development of the concept of deep learning which consists of multiple hidden layers in an artificial neural network. This approach tries to model the way the human brain processes light and sound into vision and hearing. Some successful applications of deep learning are computer vision and speech recognition.

Inductive logic programming

Inductive logic programming (ILP) is an approach to rule learning using logic programming as a uniform representation for input examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program that entails all positive and no negative examples. Inductive programming is a related field that considers any kind of programming languages for representing hypotheses (and not only logic programming), such as functional programs.

Support vector machines

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

Clustering

Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to some pre designated criterion or criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some similarity metric and evaluated for example by internal compactness (similarity between members of the same cluster) and separation between different clusters. Other methods are based on estimated density and graph connectivity. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis.

Bayesian networks

A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Reinforcement learning

Reinforcement learning is concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

Similarity and metric learning

In this problem, the learning machine is given pairs of examples that are considered similar and pairs of less similar objects. It then needs to learn a similarity function (or a distance metric functions that can predict if objects are similar. It is sometimes used in Recommendation systems.

Lab Programs

Program 1

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 *tennis.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('tennis.csv', 'r') as f:
    reader = csv.reader(f)
    your_list = list(reader)

h = [['0', '0', '0', '0', '0', '0']]

for i in your_list:
    print(i)
    if i[-1] == "True":
        j = 0
        for x in i:
            if x != "True":
                if x != h[0][j] and h[0][j] == '0':
                    h[0][j] = x
                elif x != h[0][j] and h[0][j] != '0':
                    h[0][j] = '?'
                else:
                    pass
            j = j + 1
print("Most specific hypothesis is")
print(h)
```

Output:

Maximally Specific set

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

Program 2

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

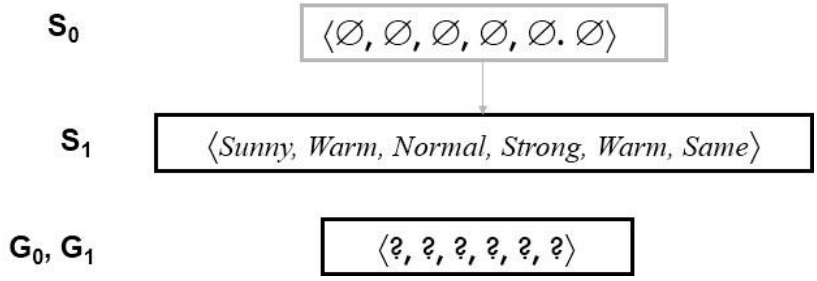
G0 ⟨?, ?, ?, ?, ?, ?⟩

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

S0 ⟨∅, ∅, ∅, ∅, ∅, ∅⟩

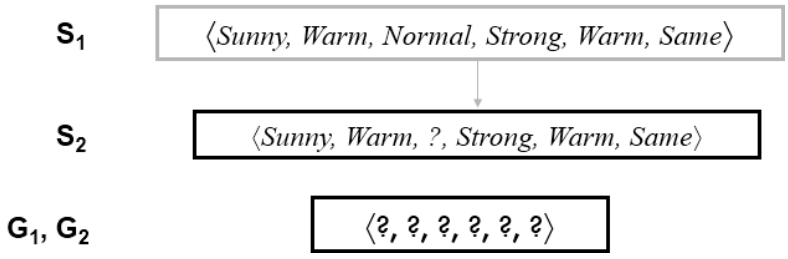
- When the first training example is presented, the CANDIDATE-ELIMINTION algorithm checks the S boundary and finds that it is overly specific and it fails to cover the positive example.
- The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example
- 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 ⟩ +



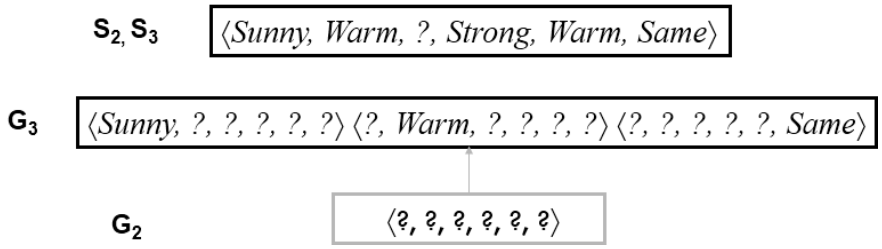
- When the second training example is observed, it has a similar effect of generalizing S further to S_2, leaving G again unchanged i.e., $G_2 = G_1 = G_0$

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 ⟩ –

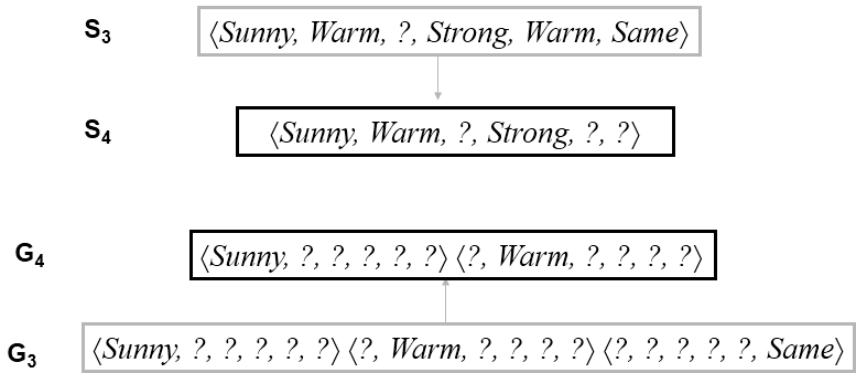


Given that there are six attributes that could be specified to specialize G_2, why are there only three new hypotheses in G_3?

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.

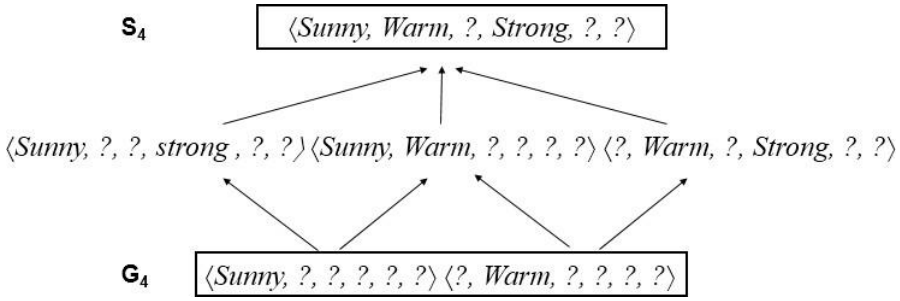
For training example d, ⟨Sunny, Warm, High, Strong, Cool Change ⟩ +



- 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 *tennist.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:

```
class Holder:
    factors={} #Initialize an empty dictionary
    attributes = () #declaration of dictionaries parameters with an arbitrary length

    '''
    Constructor of class Holder holding two parameters,
    self refers to the instance of the class
    '''
    def_init_(self,attr): #
        self.attributes = attr
        for i in attr:
            self.factors[i]=[]

    def add_values(self,factor,values):
        self.factors[factor]=values

class CandidateElimination:
    Positive={} #Initialize positive empty dictionary
    Negative={} #Initialize negative empty dictionary
```

```
    def init (self,data,fact):
        self.num_factors = len(data[0][0])
        self.factors = fact.factors
        self.attr = fact.attributes
        self.dataset = data

    def run_algorithm(self):
        '''
        Initialize the specific and general boundaries, and loop the dataset against the
algorithm
        '''
        G = self.initializeG()
        S = self.initializeS()

        '''
        Programmatically populate list in the iterating variable trial_set
        '''
        count=0
        for trial_set in self.dataset:
            if self.is_positive(trial_set): #if trial set/example consists of positive examples
                G = self.remove_inconsistent_G(G,trial_set[0]) #remove inconsitent data from
the general boundary
                    S_new = S[:] #initialize the dictionary with no key-value
                pairprint (S_new)
                for s in S:
                    if not self.consistent(s,trial_set[0]):
                        S_new.remove(s)
                        generalization = self.generalize_inconsistent_S(s,trial_set[0])
                        generalization = self.get_general(generalization,G)
                        if generalization:
                            S_new.append(generalization)
                    S = S_new[:]
                    S = self.remove_more_general(S)
                print(S)

            else:#if it is negative

                S = self.remove_inconsistent_S(S,trial_set[0]) #remove inconsitent data from
the specific boundary
                G_new = G[:] #initialize the dictionary with no key-value pair (dataset can
take any value)
                print (G_new)
                for g in G:
                    if self.consistent(g,trial_set[0]):
                        G_new.remove(g)
                        specializations = self.specialize_inconsistent_G(g,trial_set[0])
                        specializationss = self.get_specific(specializations,S)
                        if specializations != []:
                            G_new += specializationss
                    G = G_new[:]
                    G = self.remove_more_specific(G)
                print(G)
```

```
        print (S)
        print (G)

    def initializeS(self):
        ''' Initialize the specific boundary '''
        S = tuple(['-' for factor in range(self.num_factors)]) #6 constraints in the vector
        return [S]

    def initializeG(self):
        ''' Initialize the general boundary '''
        G = tuple(['?' for factor in range(self.num_factors)]) # 6 constraints in the vector
        return [G]

    def is_positive(self,trial_set):
        ''' Check if a given training trial_set is positive '''
        if trial_set[1] == 'Y':
                return True
        elif trial_set[1] == 'N':
            return False
        else:
            raise TypeError("invalid target value")

    def match_factor(self,value1,value2):
        ''' Check for the factors values match,
            necessary while checking the consistency of
            training trial_set with the hypothesis '''
        if value1 == '?' or value2 == '?':
            return True
        elif value1 == value2 :
            return True
        return False

    def consistent(self,hypothesis,instance):
        ''' Check whether the instance is part of the hypothesis '''
        for i,factor in enumerate(hypothesis):
            if not self.match_factor(factor,instance[i]):
                return False
        return True

    def remove_inconsistent_G(self,hypotheses,instance):
        ''' For a positive trial_set, the hypotheses in G
            inconsistent with it should be removed '''
        G_new = hypotheses[:]

        for g in hypotheses:
            if not self.consistent(g,instance):
                G_new.remove(g)
        return G_new

    def remove_inconsistent_S(self,hypotheses,instance):
        ''' For a negative trial_set, the hypotheses in S
            inconsistent with it should be removed '''
```

```
        S_new = hypotheses[:]
        for s in hypotheses:
            if self.consistent(s,instance):
                S_new.remove(s)
        return S_new

    def remove_more_general(self,hypotheses):
        ''' After generalizing S for a positive trial_set, the hypothesis in S
         general than others in S should be removed '''
        S_new = hypotheses[:]
        for old in hypotheses:
                for new in S_new:
            if old!=new and self.more_general(new,old):
                S_new.remove[new]
        return S_new

    def remove_more_specific(self,hypotheses):
        ''' After specializing G for a negative trial_set, the hypothesis in G
        specific than others in G should be removed '''
        G_new = hypotheses[:]
         for old in hypotheses:
           for new in G_new:
            if old!=new and self.more_specific(new,old):
                G_new.remove[new]
        return G_new


    def generalize_inconsistent_S(self,hypothesis,instance):
        ''' When a inconsistent hypothesis for positive trial_set is seen in the specific
    boundary S,
            it should be generalized to be consistent with the trial_set ... we will get one
    hypothesis'''
        hypo = list(hypothesis) # convert tuple to list for mutability
        for i,factor in enumerate(hypo):
            if factor == '-':
                hypo[i] = instance[i]
            elif not self.match_factor(factor,instance[i]):
                hypo[i] = '?'
        generalization = tuple(hypo) # convert list back to tuple for immutability
        return generalization

    def specialize_inconsistent_G(self,hypothesis,instance):
        ''' When a inconsistent hypothesis for negative trial_set is seen in the general
    boundary G
            should be specialized to be consistent with the trial_set.. we will get a set of
    hypotheses '''
        specializations = []
        hypo = list(hypothesis) # convert tuple to list for mutability
        for i,factor in enumerate(hypo):
            if factor == '?':
                values = self.factors[self.attr[i]]
                for j in values:
                    if instance[i] != j:
                        hyp=hypo[:]
```

```
                    hyp[i]=j
                    hyp=tuple(hyp) # convert list back to tuple for immutability
                    specializations.append(hyp)
return specializations

        def get_general(self,generalization,G):
            ''' Checks if there is more general hypothesis in G
                for a generalization of inconsistent hypothesis in S
                in case of positive trial_set and returns valid generalization '''

            for g in G:
                if self.more_general(g,generalization):
                    return generalization
            return None

        def get_specific(self,specializations,S):
            ''' Checks if there is more specific hypothesis in S
                for each of hypothesis in specializations of an
                inconsistent hypothesis in G in case of negative trial_set
                and return the valid specializations'''
            valid_specializations = []
            for hypo in specializations:
                for s in S:
                    if self.more_specific(s,hypo) or s==self.initializeS()[0]:
                        valid_specializations.append(hypo)
            return valid_specializations


        def exists_general(self,hypothesis,G):
            '''Used to check if there exists a more general hypothesis in
                general boundary for version space'''

            for g in G:
                if self.more_general(g,hypothesis):
                    return True
            return False

        def exists_specific(self,hypothesis,S):
            '''Used to check if there exists a more specific hypothesis in
                general boundary for version space'''
            for s in S:
                if self.more_specific(s,hypothesis):
                    return True
            return False

        def more_general(self,hyp1,hyp2):
            ''' Check whether hyp1 is more general than hyp2 '''
            hyp = zip(hyp1,hyp2)
            for i,j in hyp:
                if i == '?':
                    continue
                        elif j == '?':
                    if i != '?':
                        return False
```

```
                elif i != j:
                    return False
                else:
                    continue
            return True

        def more_specific(self,hyp1,hyp2):
            ''' hyp1 more specific than hyp2 is
                equivalent to hyp2 being more general than hyp1 '''
            return self.more_general(hyp2,hyp1)

dataset=[(('sunny','warm','normal','strong','warm','same'),'Y'),(('sunny','warm','high','stron
g','warm','same'),'Y'),(('rainy','cold','high','strong','warm','change'),'N'),(('sunny','warm','hi
gh','strong','cool','change'),'Y')]
attributes =('Sky','Temp','Humidity','Wind','Water','Forecast')
f = Holder(attributes)
f.add_values('Sky',('sunny','rainy','cloudy')) #sky can be sunny rainy or cloudy
f.add_values('Temp',('cold','warm')) #Temp can be sunny cold or warm
f.add_values('Humidity',('normal','high')) #Humidity can be normal or high
f.add_values('Wind',('weak','strong')) #wind can be weak or strong
f.add_values('Water',('warm','cold')) #water can be warm or cold
f.add_values('Forecast',('same','change')) #Forecast can be same or change
a = CandidateElimination(dataset,f) #pass the dataset to the algorithm class and call the
run algoritm method
a.run_algorithm()
```

Output:

```
[('sunny', 'warm', 'normal', 'strong', 'warm', 'same')]
[('sunny', 'warm', 'normal', 'strong', 'warm','same')]
[('sunny', 'warm', '?', 'strong', 'warm', 'same')]
[('?', '?', '?', '?', '?', '?')]
[('sunny', '?', '?', '?', '?', '?'), ('?', 'warm', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'same')]
[('sunny', 'warm', '?', 'strong', 'warm', 'same')]
[('sunny', 'warm', '?', 'strong', '?', '?')]
[('sunny', 'warm', '?', 'strong', '?', '?')]
[('sunny', '?', '?', '?', '?', '?'), ('?', 'warm', '?', '?', '?', '?')]
```

Program 3

3	**Aim**	To construct the Desicision 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.

ID3 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 valueof 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* v_i, be the subset of Examples that have value v_i for *A*
    - If *Examples* v_i , 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* v_i, Targe_tattribute, Attributes – {A}))
- End
- Return Root

* The best attribute is the one with highest information gain

ENTROPY:

Entropy measures the impurity of a collection of examples.

$$Entropy\ (S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

Where, $p+$ is the proportion of positive examples in S

$p-$ is the proportion of negative examples in S.

INFORMATION GAIN:

- ***Information gain,*** is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

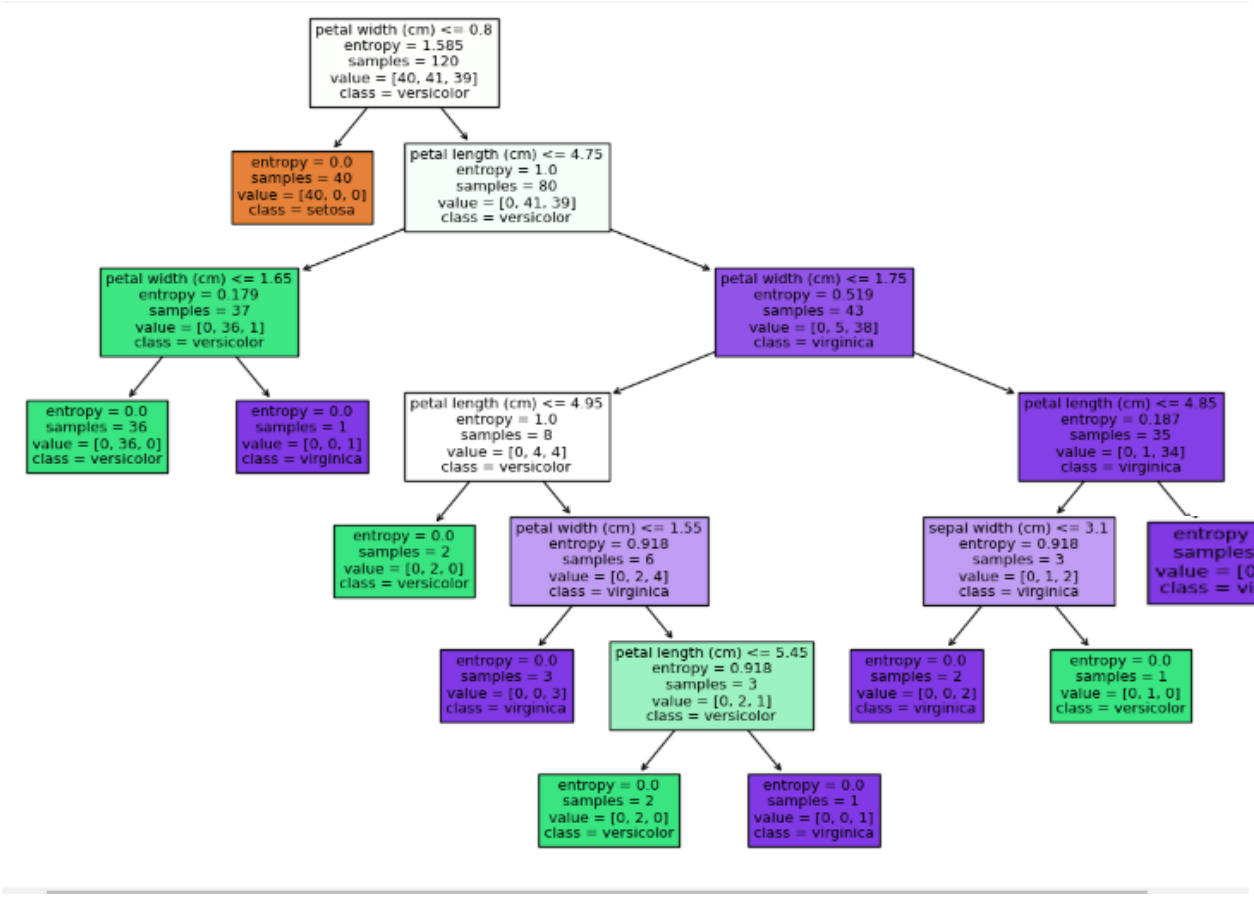
Training Dataset:

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:

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier, plot_tree

import matplotlib.pyplot as plt


# Load the Iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# 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)


# Train the decision tree classifier using ID3 algorithm

clf = DecisionTreeClassifier(criterion='entropy')  # ID3 algorithm uses entropy as criterion

clf.fit(X_train, y_train)


# Plot the decision tree

plt.figure(figsize=(15, 10))

plot_tree(clf, filled=True, feature_names=iris.feature_names, class_names=iris.target_names)

plt.show()


# Predict the class of a new sample

new_sample = [[5.1, 3.5, 1.4, 0.2]]  # Example new sample

predicted_class = clf.predict(new_sample)

print("Predicted class for new sample:", iris.target_names[predicted_class[0]])
```

Output


petal width (cm) <= 0.8
entropy = 1.585
samples = 120
value = [40, 41, 39]
class = versicolor
entropy = 0.0
samples = 40
value = [40, 0, 0]
class = setosa
petal length (cm) <= 4.75
entropy = 1.0
samples = 80
value = [0, 41, 39]
class = versicolor
petal width (cm) <= 1.65
entropy = 0.179
samples = 37
value = [0, 36, 1]
class = versicolor
petal width (cm) <= 1.75
entropy = 0.519
samples = 43
value = [0, 5, 38]
class = virginica
entropy = 0.0
samples = 36
value = [0, 36, 0]
class = versicolor
entropy = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
petal length (cm) <= 4.95
entropy = 1.0
samples = 8
value = [0, 4, 4]
class = versicolor
petal length (cm) <= 4.85
entropy = 0.187
samples = 35
value = [0, 1, 34]
class = virginica
entropy = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
petal width (cm) <= 1.55
entropy = 0.918
samples = 6
value = [0, 2, 4]
class = virginica
sepal width (cm) <= 3.1
entropy = 0.918
samples = 3
value = [0, 1, 2]
class = virginica
entropy = 0.0
samples = 32
value = [0, 0, 32]
class = virginica
entropy = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
petal length (cm) <= 5.45
entropy = 0.918
samples = 3
value = [0, 2, 1]
class = versicolor
entropy = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
entropy = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
entropy = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
entropy = 0.0
samples = 1
value = [0, 0, 1]
class = virginica

Program: 4

4. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Program:

```
import numpy as np


class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # Initialize weights and biases
        self.weights_input_hidden = np.random.randn(self.input_size, self.hidden_size)
        self.biases_input_hidden = np.zeros((1, self.hidden_size))

        self.weights_hidden_output = np.random.randn(self.hidden_size, self.output_size)
        self.biases_hidden_output = np.zeros((1, self.output_size))

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        return x * (1 - x)

    def forward_propagation(self, X):
        self.hidden_input = np.dot(X, self.weights_input_hidden) + self.biases_input_hidden
        self.hidden_output = self.sigmoid(self.hidden_input)

        self.output = np.dot(self.hidden_output, self.weights_hidden_output) + self.biases_hidden_output
        self.predicted_output = self.sigmoid(self.output)
```

```
        return self.predicted_output

    def backward_propagation(self, X, y, learning_rate):
        # Compute gradients
        error = y - self.predicted_output
        output_delta = error * self.sigmoid_derivative(self.predicted_output)

        error_hidden = output_delta.dot(self.weights_hidden_output.T)
        hidden_delta = error_hidden * self.sigmoid_derivative(self.hidden_output)

        # Update weights and biases
        self.weights_hidden_output += self.hidden_output.T.dot(output_delta) * learning_rate
        self.biases_hidden_output += np.sum(output_delta, axis=0, keepdims=True) * learning_rate

        self.weights_input_hidden += X.T.dot(hidden_delta) * learning_rate
        self.biases_input_hidden += np.sum(hidden_delta, axis=0, keepdims=True) * learning_rate

    def train(self, X, y, epochs, learning_rate):
        for epoch in range(epochs):
            output = self.forward_propagation(X)
            self.backward_propagation(X, y, learning_rate)
            loss = np.mean(np.square(y - output))
            if epoch % 100 == 0:
                print(f'Epoch {epoch}, Loss: {loss:.4f}')

    def predict(self, X):
        return self.forward_propagation(X)

# Sample data
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])
```

```
# Initialize and train the neural network
input_size = 2
hidden_size = 4
output_size = 1
nn = NeuralNetwork(input_size, hidden_size, output_size)
nn.train(X, y, epochs=1000, learning_rate=0.1)

# Predictions
print("Predictions:")
print(nn.predict(X))
```

Output:

```
Epoch 0, Loss: 0.2794
Epoch 100, Loss: 0.2521
Epoch 200, Loss: 0.2513
Epoch 300, Loss: 0.2507
Epoch 400, Loss: 0.2503
Epoch 500, Loss: 0.2500
Epoch 600, Loss: 0.2496
Epoch 700, Loss: 0.2492
Epoch 800, Loss: 0.2488
Epoch 900, Loss: 0.2482
Predictions:
[[0.49629468]
 [0.53202543]
 [0.4804809 ]
 [0.5049312 ]]
```

Program: 5

5. Write a program to 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.

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

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

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

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

(Ignoring P(D) since it is a constant)

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

Representation of Gaussian Naive Bayes

We calculate the probabilities for input values for each class using a frequency. With real- valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.

Gaussian Naive Bayes model from data

The probability density function for the normal distribution is defined by two parameters (mean and standard deviation) and calculating the mean and standard deviation values of each input variable (x) for each class value.

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

$$\sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2\right]^{0.5}$$ Standard deviation

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

Program:

```
import csv
import random
import math

def loadCsv(filename):
 lines = csv.reader(open(filename, "r"));
 dataset = list(lines)
 for i in range(len(dataset)):
     #converting strings into numbers for processing
        dataset[i] = [float(x) for x in dataset[i]]

 return dataset

def splitDataset(dataset, splitRatio):
   #67% training size
 trainSize = int(len(dataset) * splitRatio);
 trainSet = []
 copy = list(dataset);
 while len(trainSet) < trainSize:
#generate indices for the dataset list randomly to pick ele for training data
        index = random.randrange(len(copy));
        trainSet.append(copy.pop(index))
 return [trainSet, copy]
def separateByClass(dataset):
 separated = {}
#creates a dictionary of classes 1 and 0 where the values are the instacnes belonging to
each class
 for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
                separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
 return separated
```

```
def mean(numbers):
  return sum(numbers)/float(len(numbers))

def stdev(numbers):
  avg = mean(numbers)
  variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
  return math.sqrt(variance)
def summarize(dataset):
  summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
  del summaries[-1]
  return summaries

def summarizeByClass(dataset):
  separated = separateByClass(dataset);
  summaries = {}
  for classValue, instances in separated.items():
#summaries is a dic of tuples(mean,std) for each class value
        summaries[classValue] = summarize(instances)
  return summaries

def calculateProbability(x, mean, stdev):
  exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
  return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateClassProbabilities(summaries, inputVector):
  probabilities = {}
  for classValue, classSummaries in summaries.items():#class and attribute information
as mean and sd
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
              mean, stdev = classSummaries[i] #take mean and sd of every attribute
for class 0 and 1 seperaely
              x = inputVector[i] #testvector's first attribute
              probabilities[classValue] *= calculateProbability(x, mean, stdev);#use
normal dist
  return probabilities

def predict(summaries, inputVector):
  probabilities = calculateClassProbabilities(summaries, inputVector)
  bestLabel, bestProb = None, -1
  for classValue, probability in probabilities.items():#assigns that class which has he
highest prob
        if bestLabel is None or probability > bestProb:
              bestProb = probability
              bestLabel = classValue
  return bestLabel

def getPredictions(summaries, testSet):
  predictions = []
  for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
  return predictions
```

```
def getAccuracy(testSet, predictions):
  correct = 0
  for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
              correct += 1
  return (correct/float(len(testSet))) * 100.0

def main():
  filename = '5data.csv'
  splitRatio = 0.67
  dataset = loadCsv(filename);

  trainingSet, testSet = splitDataset(dataset, splitRatio)
  print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset),
len(trainingSet), len(testSet)))
  # prepare model
  summaries = summarizeByClass(trainingSet);
  # test model
  predictions = getPredictions(summaries, testSet)
  accuracy = getAccuracy(testSet, predictions)
  print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()
```

Output:

```
confusion matrix is as
follows [[17 0 0]
 [ 0 17 0]
 [ 0 0 11]]
Accuracy metrics
      precision recall f1-score support
         0      1.00     1.00     1.00    17
         1      1.00     1.00     1.00    17
         2      1.00     1.00     1.00    11
         Avg  1.00      1.00     1.00    45
```

Program: 6

6. Write a program to construct a Bayesian network considering medical data. Usethis model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

Program:

```
From pomegranate import*
Asia=DiscreteDistribution({ „True“:0.5, „False“:0.5 })
Tuberculosis=ConditionalProbabilityTable(
[[ „True“, „True“, 0.2],
[„True“, „False“, 0.8],
[ „False“, „True“, 0.01],
[ „False“, „False“, 0.98]], [asia])

Smoking = DiscreteDistribution({ „True“:0.5, „False“:0.5 })
Lung = ConditionalProbabilityTable(
[[ „True“, „True“, 0.75],
[„True“, „False“,0.25].
[ „False“, „True“, 0.02],
[ „False“, „False“, 0.98]], [ smoking])

Bronchitis = ConditionalProbabilityTable(
[[ „True“, „True“, 0.92],
[„True“, „False“,0.08].
[ „False“, „True“,0.03],
[ „False“, „False“, 0.98]], [ smoking])

Tuberculosis_or_cancer = ConditionalProbabilityTable(
 [[ „True“, „True“, „True“, 1.0],
[„True“, „True“, „False“, 0.0],
[„True“, „False“, „True“, 1.0],
[„True“, „False“, „False“, 0.0],
[„False“, „True“, „True“, 1.0],
[„False“, „True“, „False“, 0.0],
[„False“, „False“ „True“, 1.0],
[„False“, „False“, „False“, 0.0]], [tuberculosis, lung])

Xray = ConditionalProbabilityTable(

[[ „True“, „True“, 0.885],
[„True“, „False“, 0.115],
[ „False“, „True“, 0.04],


[ „False“, „False“, 0.96]], [tuberculosis_or_cancer])
dyspnea = ConditionalProbabilityTable(
[[ „True“, „True“, „True“, 0.96],
[„True“, „True“, „False“, 0.04],
[„True“, „False“, „True“, 0.89],
[„True“, „False“, „False“, 0.11],
[„False“, „True“, „True“, 0.96],
```

```
[„False“, „True“, „False“, 0.04],
[„False“, „False“ „True“, 0.89],
[„False“, „False“, „False“, 0.11 ]], [tuberculosis_or_cancer, bronchitis])
s0 = State(asia, name=”asia”)
s1 = State(tuberculosis, name=” tuberculosis”)
s2 = State(smoking, name=” smoker”)

network = BayesianNetwork(“asia”)
network.add_nodes(s0,s1,s2)
network.add_edge(s0,s1)
network.add_edge(s1.s2)
network.bake()
print(network.predict_probal({„tuberculosis“: „True“}))
```

Program: 7

7. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same dataset 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.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=100, centers =
4,Cluster_std=0.60,random_state=0)
X = X[:, ::-1]
#flip axes for better plotting
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture (n_components = 4).fit(X)
lables = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap="viridis");
probs = gmm.predict_proba(X)
print(probs[:5].round(3))
size = 50 * probs.max(1) ** 2 # square emphasizes differences
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap="viridis", s=size);


from matplotlib.patches import Ellipse
def draw_ellipse(position, covariance, ax=None, **kwargs);
        """Draw an ellipse with a given position and covariance"""
Ax = ax or plt.gca()
# Convert covariance to principal axes
if covariance.shape ==(2,2):

  U, s, Vt = np.linalg.svd(covariance)
  Angle = np.degrees(np.arctan2(U[1, 0], U[0,0]))
  Width, height = 2 * np.sqrt(s)
else:
      angle = 0
      width, height = 2 * np.sqrt(covariance)
#Draw the Ellipse
for nsig in range(1,4):
     ax.add_patch(Ellipse(position, nsig * width, nsig *height,
                          angle, **kwargs))

 def plot_gmm(gmm, X, label=True, ax=None):
   ax = ax or plt.gca()
   labels = gmm.fit(X).predict(X)
   if label:
      ax.scatter(X[:, 0], x[:, 1], c=labels, s=40, cmap="viridis", zorder=2)
 else:
      ax.scatter(X[:, 0], x[:, 1], s=40, zorder=2)
```

```
    ax.axis(„equal“)

    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
      draw_ellipse(pos, covar, alpha=w * w_factor)

    gmm = GaussianMixture(n_components=4, random_state=42)
    plot_gmm(gmm, X)
    gmm = GaussianMixture(n_components=4, covariance_type=“full“,
                random_state=42)
    plot_gmm(gmm, X)
```

Output:

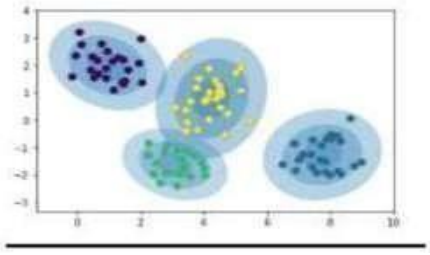
[[1, 0, 0, 0]

[0, 0, 1, 0]

[1, 0, 0, 0]

[1, 0, 0, 0]

[1, 0, 0, 0]]



Program: 8

8. Write a program to implement *k*-Nearest Neighbour algorithm to classify the irisdata set. Print both correct and wrong predictions. Java/Python ML library classescan be used for this problem.

Program:

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score


# Load Iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# Split 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)


# Initialize k-NN classifier

knn = KNeighborsClassifier(n_neighbors=3)


# Train the classifier

knn.fit(X_train, y_train)


# Predict the labels for test set

y_pred = knn.predict(X_test)


# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
```

```
# Print correct and wrong predictions

print("\nCorrect predictions:")

for i in range(len(y_test)):

    if y_pred[i] == y_test[i]:

        print("Predicted:", y_pred[i], "Actual:", y_test[i])


print("\nWrong predictions:")

for i in range(len(y_test)):

    if y_pred[i] != y_test[i]:

        print("Predicted:", y_pred[i], "Actual:", y_test[i])
```

Output:

```
Accuracy: 1.0

Correct predictions:
Predicted: 1 Actual: 1
Predicted: 0 Actual: 0
Predicted: 2 Actual: 2
Predicted: 1 Actual: 1
Predicted: 1 Actual: 1
Predicted: 0 Actual: 0
Predicted: 1 Actual: 1
Predicted: 2 Actual: 2
Predicted: 1 Actual: 1
Predicted: 1 Actual: 1
Predicted: 2 Actual: 2
Predicted: 0 Actual: 0
Predicted: 0 Actual: 0
Predicted: 0 Actual: 0
Predicted: 0 Actual: 0
Predicted: 1 Actual: 1
Predicted: 2 Actual: 2
Predicted: 1 Actual: 1
Predicted: 1 Actual: 1
Predicted: 2 Actual: 2
Predicted: 0 Actual: 0
Predicted: 2 Actual: 2
Predicted: 0 Actual: 0
Predicted: 2 Actual: 2
Predicted: 2 Actual: 2

Predicted: 2 Actual: 2
Predicted: 1 Actual: 1
Predicted: 1 Actual: 1
Predicted: 0 Actual: 0
Predicted: 0 Actual: 0

Wrong predictions:
```

Program: 9

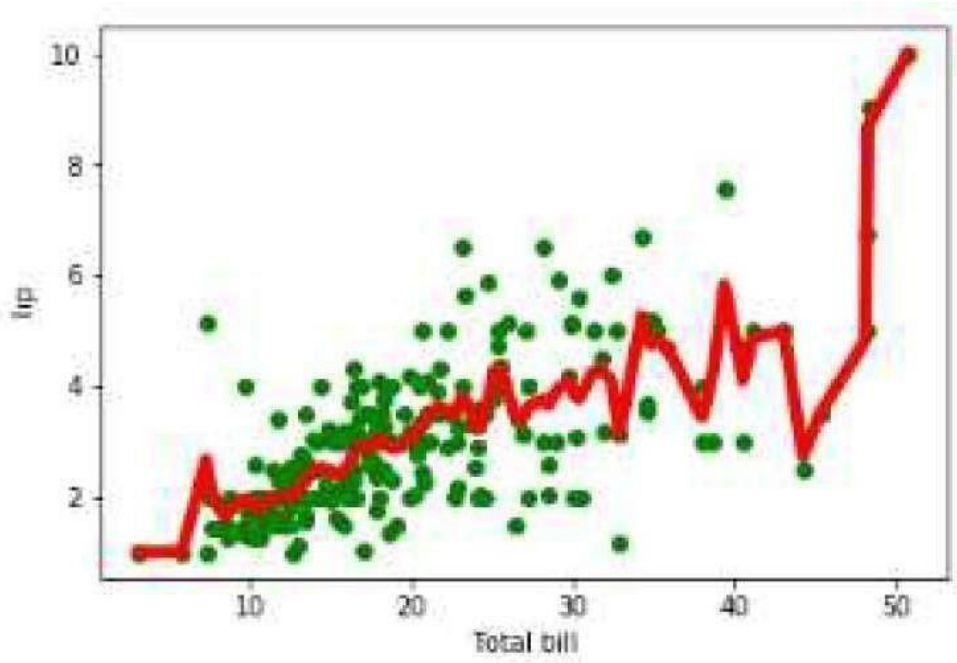
9. Implement the non-parametric Locally Weighted Regression algorithm in orderto fit data points. Select appropriate data set for your experiment and draw graphs.

Program:

```
from numpy import *
import operator
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1 import
numpy.linalg as np
from scipy.stats.stats import pearsonr

def kernel(point,xmat, k):
    m,n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W=(X.T*(wei*X)).I*(X.T*(wei*ymat.T))return
    W

def localWeightRegression(xmat,ymat,k):
    m,n = np1.shape(xmat)
    ypred = np1.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

# load data points
data = pd.read_csv('data10.csv')
bill = np1.array(data.total_bill) tip
= np1.array(data.tip)

#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip)
m= np1.shape(mbill)[1]
one = np1.mat(np1.ones(m)) X=
np1.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,2)
```

```
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
```

Output:



Program: 10

10. Implement and demonstrate the working of SVM algorithm for classification.

Program:

```
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the iris dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # Taking only the first two features for simplicity
y = iris.target

# Splitting 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)

# Feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Training the SVM model
svm_classifier = SVC(kernel='linear', random_state=42)
svm_classifier.fit(X_train, y_train)

# Predicting the test set results
```

```
y_pred = svm_classifier.predict(X_test)

# Calculating the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Visualizing the decision boundary
def plot_decision_boundary(classifier, X, y):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                         np.arange(y_min, y_max, 0.1))
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.4)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolors='k')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('SVM Decision Boundary')
    plt.show()

# Plotting decision boundary
plot_decision_boundary(svm_classifier, X_train, y_train)
```

Output: