K.S. SCHOOL OF ENGINEERING & MANAGEMENT, BANGALORE - 109

"ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY"

Subject code: 18CSL76

Prepared by:-

Mr. Santhosh Kumar J
Associate Professor
Dept. of CSE, KSSEM

Mr. Ashoka S
Assistant Professor
Dept. of CSE, KSSEM

Mrs. Shalini K V
Assistant Professor
Dept. of CSE, KSSEM

Department of Computer Science & Engineering

K.S SCHOOL OF ENGINEERING & MANAGEMENT, BANGALORE - 109

K.S. SCHOOL OF ENGINEERING AND MANAGEMENT
15, Mallasandra, off Kanakapura Road**,** Bengaluru-**560 109**
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course: Artificial Intelligence and Machine Learning Laboratory			
Type: **Core**	**Course Code:18CSL76**	**Academic Year: 2021-2022**	
No. of Hours per week			
Theory (Lecture Class)	Practical/Field Work/Allied Activities	Total/Week	Total teaching hours
0	3	3	36 Hours
Marks			
Internal Assessment	Examination	Total	Credits
40	60	100	2

Aim/Objective of the Course:

1. Implement and evaluate AI and ML algorithms in and Python programming language.
2. Make use of Data sets in implementing the machine learning algorithms.
3. Implement the machine leaning concepts and algorithms in any suitable language of choice.

Course Learning Outcomes:

After completing the course, the students will be able to

CO1	**Implement** AI algorithms to solve searching problems	**Applying (K3)**
CO2	**Make use of** machine learning algorithms to analyze and interpret the results from the data set.	**Applying (K3)**
CO3	**Apply** machine leaning concepts and algorithms to implement the same using any suitable programming language.	**Applying (K3)**

Institution Vision & Mission

VISION: "To impart quality education in engineering and management to meet technological, business and societal needs through holistic education and research"

MISSION:

K.S. School of Engineering and Management shall,

- Establish state-of-art infrastructure to facilitate effective dissemination of technical and Managerial knowledge.
- Provide comprehensive educational experience through a combination of curricular and experiential learning, strengthened by industry-institute-interaction.
- Pursue socially relevant research and disseminate knowledge.
- Inculcate leadership skills and foster entrepreneurial spirit among students.

Department Vision & Mission

VISION: **"**To produce quality Computer Science professional, possessing excellent technical knowledge, skills, personality through education and research."

MISSION:

Department of Computer Science and Engineering shall,

- Provide good infrastructure and facilitate learning to become competent engineers who meet global challenges.
- Encourages industry institute interaction to give an edge to the students.
- Facilitates experimental learning through interdisciplinary projects.
- Strengthen soft skill to address global challenges.

K.S. SCHOOL OF ENGINEERING AND MANAGEMENT
 # 15, Mallasandra, off Kanakapura Road, Bengaluru-**560 109**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Artificial Intelligence and Machine Learning Laboratory (18CSL76)

INDEX

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY
(Effective from the academic year 2018 - 2019)

SEMESTER – VII

Course Code	**18CSL76**	**CIE Marks**	40
Number of Contact Hours/Week	0:0:2	**SEE Marks**	60
Total Number of Lab Contact Hours	36	**Exam Hours**	03

Credits – 2

Course Learning Objectives: This course (18CSL76) will enable students to:

- Implement and evaluate AI and ML algorithms in and Python programming language.

Descriptions (if any):

Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

Programs List:

1.	Implement A* Search algorithm.
2.	Implement AO* Search algorithm.
3.	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.
4.	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 newsample.
5.	Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.
6.	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.
7.	Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
8.	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.
9.	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

Laboratory Outcomes: The student should be able to:

- Implement and demonstrate AI and ML algorithms.
- Evaluate different algorithms.

Conduct of Practical Examination:

- Experiment distribution
  - For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
  - For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Courseed to change in accoradance with university regulations)*
  - q) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  - r) For laboratories having PART A and PART B
    - i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
    - ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

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:

1. **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:
2. **Semi-supervised learning:** the computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.
3. **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.
4. **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.
5. **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>Un Supervised learning</th></tr>
<tr><td>Find-s algorithm EM algorithm</td><td>Find-s algorithm EM algorithm</td><td rowspan="7">Locally weighted Regression algorithm</td></tr>
<tr><td>Candidate elimination algorithm</td><td rowspan="6">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 neighbor algorithm(lazy learning algorithm)</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 given a list of human language documents and is tasked with finding out which documents cover similar topics.

Machine learning Approaches

1. **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.

2. **Association rule learning**
   Association rule learning is a method for discovering interesting relations between variables in large databases.

3. **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.

4. **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.

5. 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.

6. 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.

7. 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.

8. 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. Efficient algorithms exist that perform inference and learning.

9. 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.

10. 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 function) that can predict if new objects are similar. It is sometimes used in Recommendation systems.

11. Genetic algorithms

A genetic algorithm (GA) is a search heuristic that mimics the process of natural selection, and uses methods such as mutation and crossover to generate new genotype in the hope of finding good solutions to a given problem. In machine learning, genetic algorithms found some uses in the 1980s and 1990s. Conversely, machine learning techniques have been used to improve the performance of genetic and evolutionary algorithms.

12. Rule-based machine learning

Rule-based machine learning is a general term for any machine learning method that identifies, learns, or evolves "rules" to store, manipulate or apply, knowledge. The defining characteristic of a rule- based machine learner is the identification and utilization of a set of relational rules that collectively represent the knowledge captured by the system. This is in contrast to other machine learners that commonly identify a singular model that can be universally applied to any instance in order to make a prediction. Rule-based machine learning approaches include learning classifier systems, association rule learning, and artificial immune systems.

13. Feature selection approach

Feature selection is the process of selecting an optimal subset of relevant features for use in model construction. It is assumed the data contains some features that are either redundant or irrelevant, and can thus be removed to reduce calculation cost without incurring much loss of information. Common optimality criteria include accuracy, similarity and information measures.

Artificial Intelligence:-

It is a branch of Computer Science that pursues creating the computers or machines as intelligent as human beings. It is the science and engineering of making intelligent machines, especially intelligent computer programs.

Definition: Artificial Intelligence is the study of how to make computers do things, which, at the moment, people do better.

AI Vocabulary:-

1. Intelligence - relates to tasks involving higher mental processes, e.g. creativity, solving problems, pattern recognition, classification, learning, induction, deduction, building analogies, optimization, language processing, knowledge and many more. Intelligence is the computational part of the ability to achieve goals.

2. Intelligent behaviour - is depicted by perceiving one's environment, acting in complex environments, learning and understanding from experience, reasoning to solve problems and discover hidden knowledge, applying knowledge successfully in new situations, thinking abstractly, using analogies, communicating with others and more.

3. Science based goals of AI pertain to developing concepts, mechanisms and understanding biological intelligent behaviour. The emphasis is on understanding intelligent behaviour.

4. Engineering based goals of AI relate to developing concepts, theory and practice of building intelligent machines. The emphasis is on system building.

5. AI Techniques depict how we represent, manipulate and reason with knowledge in order to solve problems. Knowledge is a collection of 'facts'. To manipulate these facts by a program, a suitable representation is required. A good representation facilitates problem solving.

6. Learning means that programs learn from what facts or behaviour can represent. Learning denotes changes in the systems that are adaptive in other words, it enables the system to do the same task(s) more efficiently next time.

7. Applications of AI refers to problem solving, search and control strategies, speech recognition, natural language understanding, computer vision, expert systems, etc.

Branches of AI:

A list of branches of AI is given below.

1. Logical AI — In general the facts of the specific situation in which it must act, and its goals are all represented by sentences of some mathematical logical language. The program decides what to do by inferring that certain actions are appropriate for achieving its goals.

2. Search — Artificial Intelligence programs often examine large numbers of possibilities – for example, moves in a chess game and inferences by a theorem proving program. Discoveries are frequently made about how to do this more efficiently in various domains.

3. Pattern Recognition — When a program makes observations of some kind, it is often planned to compare what it sees with a pattern. For example, a vision program may try to match a pattern of eyes and a nose in a scene in order to find a face. More complex patterns are like a natural language text, a chess position or in the history of some event. These more complex patterns require quite different methods than do the simple patterns that have been studied the most.

4. Representation — Usually languages of mathematical logic are used to represent the facts about the world.

5. Inference — Others can be inferred from some facts. Mathematical logical deduction is sufficient for some purposes, but new methods of non-monotonic inference have been added to the logic since the 1970s. The simplest kind of non-monotonic reasoning is default reasoning in which a conclusion is to be inferred by default. But the conclusion can be withdrawn if there is evidence to the divergent. For example, when we hear of a bird, we infer that it can fly, but this conclusion can be reversed when we hear that it is a penguin. It is the possibility that a conclusion may have to be withdrawn that constitutes the non-monotonic character of the reasoning. Normal logical reasoning is monotonic, in that the set of conclusions can be drawn from a set of premises, i.e. monotonic increasing function of the premises. Circumscription is another form of non-monotonic reasoning.

6. Common sense knowledge and Reasoning — This is the area in which AI is farthest from the human level, in spite of the fact that it has been an active research area since the 1950s. While there has been considerable progress in developing systems of non-monotonic reasoning and theories of action, yet more new ideas are needed.

7. Learning from experience — There are some rules expressed in logic for learning. Programs can only learn what facts or behaviour their formalisms can represent, and unfortunately learning systems are almost all based on very limited abilities to represent information.

8. Planning — Planning starts with general facts about the world (especially facts about the effects of actions), facts about the particular situation and a statement of a goal. From these, planning programs generate a strategy for achieving the goal. In the most common cases, the strategy is just a sequence of actions.

9. Epistemology — This is a study of the kinds of knowledge that are required for solving problems in the world.

10. Ontology — Ontology is the study of the kinds of things that exist. In AI the programs and sentences deal with various kinds of objects and we study what these kinds are and what their basic properties are. Ontology assumed importance from the 1990s.

11. Heuristics — A heuristic is a way of trying to discover something or an idea embedded in a program. The term is used variously in AI. Heuristic functions are used in some approaches to search or to measure how far a node in a search tree seems to be from a goal. Heuristic predicates that compare two nodes in a search tree to see if one is better than the other, i.e. constitutes an advance toward the goal, and may be more useful.

12. Genetic programming — Genetic programming is an automated method for creating a working computer program from a high-level problem statement of a problem. Genetic programming starts from a high-level statement of 'what needs to be done' and automatically creates a computer program to solve the problem.

Applications of AI:-

- Perception
  - Machine vision
  - Speech understanding
  - Touch (tactile or haptic) sensation

- Robotics
- Natural Language Processing
  - Natural Language Understanding
  - Speech Understanding
  - Language Generation
  - Machine Translation

- Planning
- Expert Systems
- Machine Learning
- Theorem Proving
- Symbolic Mathematics
- Game Playing

1. Implement A* Search algorithm.

```
#from collections import deque

class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis

    def get_neighbors(self, v):
        return self.adjac_lis[v]

    # This is heuristic function which is having equal values for all nodes
    def h(self, n):
        H = {'A': 1,'B': 1,'C': 1,'D': 1}
         return H[n]
    def a_star_algorithm(self, start, stop):
        open_lst = set([start])
        closed_lst = set([])

        poo = {}
        poo[start] = 0

        par = {}
        par[start] = start

        while len(open_lst) > 0:
            n = None

            # it will find a node with the lowest value of f() -
            for v in open_lst:
                if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):
                    n = v;

            if n == None:
                print('Path does not exist!')
                return None

            # if the current node is the stop
            # then we start again from start
            if n == stop:
                reconst_path = []

                while par[n] != n:
                    reconst_path.append(n)
                    n = par[n]
                 reconst_path.append(start)
```

```
            reconst_path.reverse()

        print('Path found: {}'.format(reconst_path))
            return reconst_path

        # for all the neighbors of the current node do
        for (m, weight) in self.get_neighbors(n):

          # if the current node is not presentin both open_lst and closed_lst
            # add it to open_lst and note n as it's par
            if m not in open_lst and m not in closed_lst:
                open_lst.add(m)
                par[m] = n
                poo[m] = poo[n] + weight
            else:
                if poo[m] > poo[n] + weight:
                    poo[m] = poo[n] + weight
                    par[m] = n

                    if m in closed_lst:
                        closed_lst.remove(m)
                        open_lst.add(m)

        open_lst.remove(n)
        closed_lst.add(n)

    print('Path does not exist!')
    return None

adjac_lis = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjac_lis)
graph1.a_star_algorithm('A', 'D')
```

OUTPUT

```
Path found: ['A', 'B', 'D']

['A', 'B', 'D']
```

2. Implement AO* Search algorithm.

```
#from collections import deque

class Graph:
    def __init__(self, adjac_lis):
        print('adjac_lis values are ==== ' , adjac_lis)
        self.adjac_lis = adjac_lis

    def get_neighbors(self, v):
        print(" v values are --------", v)
        return self.adjac_lis[v]

    # This is heuristic function which is having equal values for all nodes
    def h(self, n):
        H = {'A': 1,'B': 1,'C': 1,'D': 1}
        #print(H)
        return H[n]

    def a_star_algorithm(self, start, stop):
        open_lst = set([start])
        closed_lst = set([])

        print("open list initially )))))))", open_lst)
        print("closed list initially ))))))))))", closed_lst)

        poo = {}
        poo[start] = 0

        par = {}
        par[start] = start

        print("par value &&&&&&&&&&", par)

        while len(open_lst) > 0:
            n = None

            # it will find a node with the lowest value of f() -
            for v in open_lst:
                if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):
                    print("n value ", n)
                    print("v value ", v)
                    n = v;

            if n == None:
                print('Path does not exist!')
```

```
        return None
    # if the current node is the stop
    # then we start again from start
    if n == stop:
        reconst_path = []
        print("reconst_path", reconst_path)

        while par[n] != n:
            reconst_path.append(n)
            n = par[n]
            print("reconst_path within while", reconst_path)
        reconst_path.append(start)

        reconst_path.reverse()

        print('Path found: {}'.format(reconst_path))
        print("reconst_path", reconst_path)
        return reconst_path

    # for all the neighbors of the current node do
    for (m, weight) in self.get_neighbors(n):
      # if the current node is not presentin both open_lst and closed_lst
        # add it to open_lst and note n as it's par
        if m not in open_lst and m not in closed_lst:
            print("open list  within while and if ",open_lst)
            print("closed  list  within while and if ",closed_lst)
            open_lst.add(m)
            par[m] = n
            poo[m] = poo[n] + weight

            print("open list  within while and if after ... ",open_lst)
            print("closed list  within while and if after ... ",closed_lst)
            print("poo values within if ", poo)

        else:
            if poo[m] > poo[n] + weight:
                poo[m] = poo[n] + weight
                par[m] = n

                print("poo values within else ", poo)

                if m in closed_lst:
                    closed_lst.remove(m)
                    open_lst.add(m)
                    print("closed list within while and if ",closed_lst)

    print("open==========----  list within while and if ",open_lst)
```

```
            print("closed ========== list within while and if ",closed_lst)

            open_lst.remove(n)
            closed_lst.add(n)

        print('Path does not exist!')
        return None

adjac_lis = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjac_lis)
# print("graph1 is ", graph1)
graph1.a_star_algorithm('A', 'D')

adjac_lis = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjac_lis)
# print("graph1 is ", graph1)
graph1.a_star_algorithm('A', 'D')
```

OUTPUT

```
adjac_lis values are ====  {'A': [('B', 1), ('C', 3), ('D', 7)], 'B': [('D', 5)], 'C': [('D', 12)]}
open list initially ))))))) {'A'}
closed list initially )))))))))))) set()
par value &&&&&&&&&& {'A': 'A'}
n value  None
v value  A
 v values are -------- A
open list  within while and if  {'A'}
closed  list  within while and if  set()
open list  within while and if after ...  {'B', 'A'}
closed list  within while and if after ...  set()
poo values within if  {'A': 0, 'B': 1}
open list  within while and if  {'B', 'A'}
closed  list  within while and if  set()
open list  within while and if after ...  {'B', 'A', 'C'}
closed list  within while and if after ...  set()
poo values within if  {'A': 0, 'B': 1, 'C': 3}
open list  within while and if  {'B', 'A', 'C'}
```

```
closed  list  within while and if  set()
open list  within while and if after ...  {'D', 'B', 'A', 'C'}
closed list  within while and if after ...  set()
poo values within if  {'A': 0, 'B': 1, 'C': 3, 'D': 7}
open==========----  list within while and if  {'D', 'B', 'A', 'C'}
closed ========== list within while and if  set()
n value  None
v value  D
n value  D
v value  B
 v values are -------- B
poo values within else  {'A': 0, 'B': 1, 'C': 3, 'D': 6}
open==========----  list within while and if  {'D', 'B', 'C'}
closed ========== list within while and if  {'A'}
n value  None
v value  D
n value  D
v value  C
 v values are -------- C
open==========----  list within while and if  {'D', 'C'}
closed ========== list within while and if  {'B', 'A'}
n value  None
v value  D
reconst_path []
reconst_path within while ['D']
reconst_path within while ['D', 'B']
Path found: ['A', 'B', 'D']
reconst_path ['A', 'B', 'D']

Out[1]:  ['A', 'B', 'D']
```

3. 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.

```
import numpy as np
import pandas as pd

# Loading Data from a CSV File
data = pd.DataFrame(data = pd.read_csv("C:/Users/mbacc/Documents/dataset/Training.csv"))
data

# Separating concept features from Target
concepts = np.array(data.iloc[:,0:-1])
concepts
target = np.array(data.iloc[:,-1])
target
def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]

    # The learning iterations
    for i, h in enumerate(concepts):

        # Checking if the hypothesis has a positive target
        if target[i] == "Yes":
            for x in range(len(specific_h)):

                # Change values in S & G only if values change
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'

        # Checking if the hypothesis has a positive target
        if target[i] == "No":
            for x in range(len(specific_h)):
                print(f"specific={specific_h[x]}")
                # For negative hyposthesis change values only  in G
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                    #print(f"general{x}={general_h[x][x]}")
                else:
                    general_h[x][x] = '?'
    # find indices where we have empty rows, meaning those that are unchanged
    indices = [i for i,val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
```

```
    for i in indices:
        # remove those rows from general_h
        general_h.remove(['?', '?', '?', '?', '?', '?'])

    # Return final values
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
s_final
g_final
```

Dataset

Training.csv

Sky	Airtemp	Humidity	Wind	Water	Forecast	WaterSport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Cloudy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

OUTPUT

```
specific=Sunny
specific=Warm
specific=?
specific=Strong
specific=Warm
specific=Same

Out[7]:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

4. 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 newsample.

```
# import libraries
import numpy as np
import pandas as pd

#load dataset
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
data.data
data.feature_names
data.target
data.target_names

#create dataframe
df = pd.DataFrame(np.c_[data.data, data.target], columns=[list(data.feature_names)+['target']])
df.head()
df.tail()
row1=df.iloc[3]
row1
df.shape

#Split Data
X = df.iloc[:, 0:-1]
y = df.iloc[:, -1]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2020)
print('Shape of X_train = ', X_train.shape)
print('Shape of y_train = ', y_train.shape)
print('Shape of X_test = ', X_test.shape)
print('Shape of y_test = ', y_test.shape)

#Train Decision Tree Classification Model
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='gini')
classifier.fit(X_train, y_train)
classifier.score(X_test, y_test)
classifier_entropy = DecisionTreeClassifier(criterion='entropy')
classifier_entropy.fit(X_train, y_train)
```

```
classifier_entropy.score(X_test, y_test)
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_sc = sc.transform(X_train)
X_test_sc = sc.transform(X_test)
classifier_sc = DecisionTreeClassifier(criterion='gini')
classifier_sc.fit(X_train_sc, y_train)
classifier_sc.score(X_test_sc, y_test)

#Predict Cancer
patient1 = [17.99,
 0.38,
 122.8,
 1001.0,
 0.1184,
 0.2776,
 0.3001,
 0.1471,
 0.2419,
 0.07871,
 1.095,
 0.9053,
 8.589,
 153.4,
 0.006399,
 0.04904,
 0.05373,
 0.01587,
 0.03003,
 0.006193,
 25.38,
 17.33,
 184.6,
 2019.0,
 0.1622,
 0.6656,
 0.7119,
 0.2654,
 0.4601,
 0.1189]
```

```
patient1 = np.array([patient1])
patient1
classifier.predict(patient1)
data.target_names
pred = classifier.predict(patient1)

if pred[0] == 0:
  print('Patient has Cancer (malignant tumor)')
else:
  print('Patient has no Cancer (malignant benign)')
```

OUTPUT

```
Patient has Cancer (malignant tumor)
```

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

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)     # X = (hours sleeping, hours studying)
y = np.array(([92], [86], [89]), dtype=float)           # y = score on test

# scale units
X = X/np.amax(X, axis=0)       # maximum of X array
y = y/100                      # max test score is 100

print(X)
print(y)

class Neural_Network(object):
   def __init__(self):
                        # Parameters
      self.inputSize = 2
      self.outputSize = 1
      self.hiddenSize = 3
                         # Weights
      self.W1 = np.random.randn(self.inputSize, self.hiddenSize)
                                                  # (3x2) weight matrix from input to hidden layer
      self.W2 = np.random.randn(self.hiddenSize, self.outputSize)
                                                  # (3x1) weight matrix from hidden to output layer
   def forward(self, X):
                                          # forward propagation through our network
      self.z = np.dot(X, self.W1)         # dot product of X (input) and first set of 3x2 weights
      self.z2 = self.sigmoid(self.z)      # activation function
      self.z3 = np.dot(self.z2, self.W2)  # dot product of hidden layer (z2) and second set of 3x1 weights
      o = self.sigmoid(self.z3)           # final activation function
      return o

   def sigmoid(self, s):
      return 1/(1+np.exp(-s))             # activation function

   def sigmoidPrime(self, s):
      return s * (1 - s)                  # derivative of sigmoid
    def backward(self, X, y, o):
                                      # backward propgate through the network
      self.o_error = y - o            # error in output

      self.o_delta = self.o_error*self.sigmoidPrime(o)          # applying derivative of sigmoid to

      self.z2_error = self.o_delta.dot(self.W2.T)    # z2 error: how much our hidden layer weights
contributed to output error
```

```
        self.z2_delta = self.z2_error*self.sigmoidPrime(self.z2) # applying derivative of sigmoid to z2 error

        self.W1 += X.T.dot(self.z2_delta)       # adjusting first set (input --> hidden) weights
        self.W2 += self.z2.T.dot(self.o_delta)  # adjusting second set (hidden --> output) weights

    def train (self, X, y):
        o = self.forward(X)
        self.backward(X, y, o)

NN = Neural_Network()
for i in range(1000):                    # trains the NN 1,000 times
    print ("\nInput: \n" + str(X))
    print ("\nActual Output: \n" + str(y))
    print ("\nPredicted Output: \n" + str(NN.forward(X)))
    print ("\nLoss: \n" + str(np.mean(np.square(y - NN.forward(X))))    # mean sum squared loss)
    NN.train(X, y)
```

OUTPUT

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]

Predicted Output:
[[0.52902915]
 [0.52826157]
 [0.53631347]]

Loss:
0.12933425267706222

Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]
```

Predicted Output:
[[0.5855257]
 [0.58103597]
 [0.60227742]]

Loss:
0.09082609014210542

Input:
[[0.66666667 1.   ]
 [0.33333333 0.55555556]
 [1.   0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]

Predicted Output:
[[0.89499911]
 [0.86577542]
 [0.91033262]]

Loss:
0.0003572717610598643

Input:
[[0.66666667 1.   ]
 [0.33333333 0.55555556]
 [1.   0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]

Predicted Output:
[[0.89500363]
 [0.86577342]
 [0.91032839]]

Loss:
0.0003571313973137464

6. 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.

```
# import necessary libarities
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# load data from CSV
data = pd.read_csv('C:/Users/mbacc/Documents/dataset/tennisdata.csv')
print("THe first 5 values of data is :\n",data.head())

# obtain Train data and Train output
X = data.iloc[:,:-1]
print("\nThe First 5 values of train data is\n",X.head())
y = data.iloc[:,-1]
print("\nThe first 5 values of Train output is\n",y.head())

# Convert then in numbers
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

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

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

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

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

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)

classifier = GaussianNB()
classifier.fit(X_train,y_train)

from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

Dataset

tennisdata.csv

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	FALSE	No
Sunny	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Rainy	Mild	High	FALSE	Yes
Rainy	Cool	Normal	FALSE	Yes
Rainy	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Sunny	Mild	High	FALSE	No
Sunny	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	FALSE	Yes
Sunny	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Rainy	Mild	High	TRUE	No

OUTPUT

The first 5 values of data is :

	Outlook	Temperature	Humidity	Windy	PlayTennis
0	Sunny	Hot	High	False	No
1	Sunny	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Rainy	Mild	High	False	Yes
4	Rainy	Cool	Normal	False	Yes

The First 5 values of train data is

	Outlook	Temperature	Humidity	Windy
0	Sunny	Hot	High	False
1	Sunny	Hot	High	True
2	Overcast	Hot	High	False
3	Rainy	Mild	High	False
4	Rainy	Cool	Normal	False

The first 5 values of Train output is
0 No
1 No
2 Yes
3 Yes
4 Yes
Name: PlayTennis, dtype: object

Now the Train data is :

	Outlook	Temperature	Humidity	Windy
0	2	1	0	0
1	2	1	0	1
2	0	1	0	0
3	1	2	0	0
4	1	0	1	0

Now the Train output is
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Accuracy is: 1.0

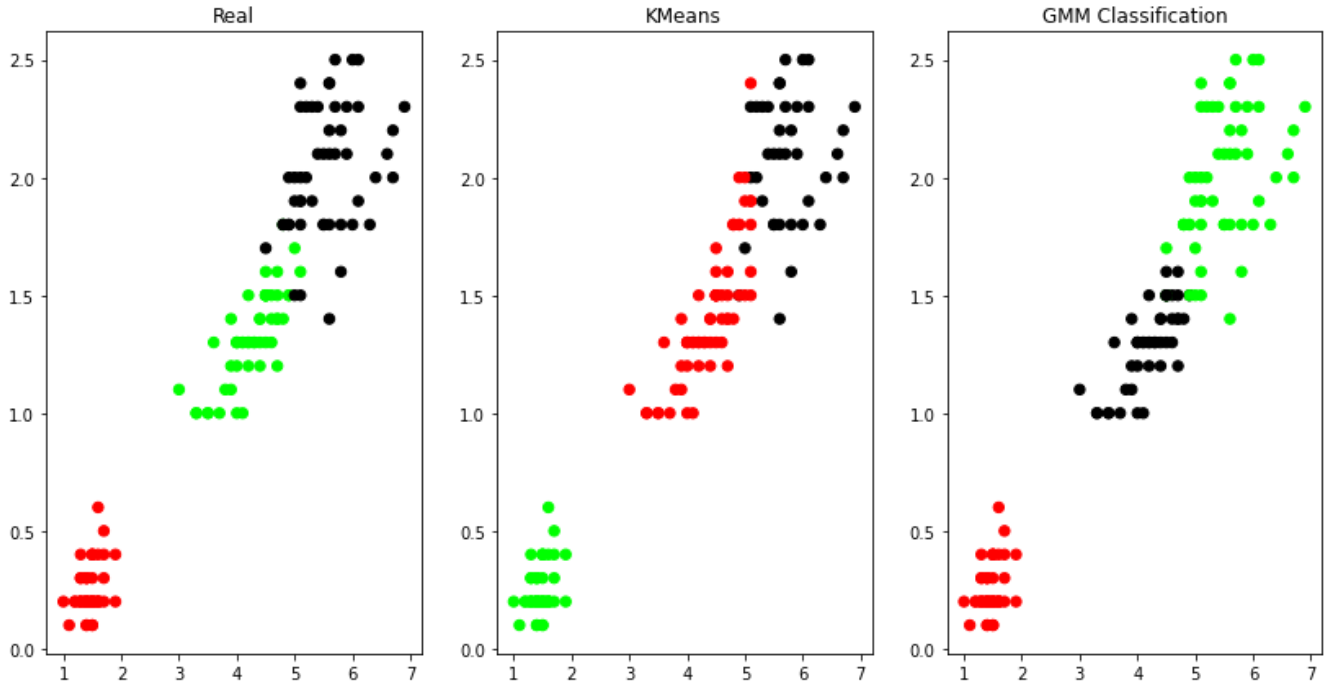
7. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using *k*-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset=load_iris()
# print(dataset)

X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']


y=pd.DataFrame(dataset.target)
y.columns=['Targets']
# print(X)

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

# K-PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')

# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
```

```
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)

y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

OUTPUT

Text(0.5, 1.0, 'GMM Classification')



8. 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.

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np

dataset=load_iris()
#print(dataset)
X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=0)

kn=KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train,y_train)

for i in range(len(X_test)):

  x=X_test[i]
  x_new=np.array([x])
  prediction=kn.predict(x_new)

print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction,
dataset["target_names"][prediction])
print(kn.score(X_test,y_test))
```

OUTPUT

```
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
0.9736842105263158
```

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

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

def kernel(point,xmat, k):
    m,n =np.shape(xmat)
    weights = np.mat(np.eye((m)))
    for j in range(m):
        diff = point - X[j]
    weights[j,j] = np.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 = np.shape(xmat)
    ypred = np.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('C:/Users/mbacc/Documents/dataset/tips.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

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

#set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig=plt.figure()
ax=fig.add_subplot(1,1,1)
```

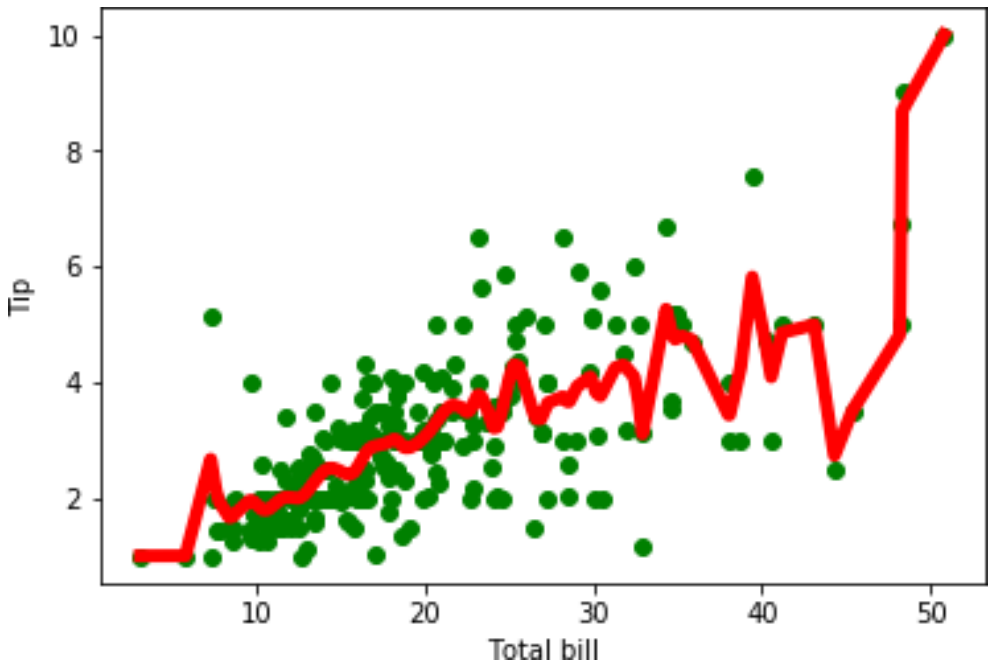
```
ax.scatter(bill,tip,color='green')
ax.plot(xsort[:,1],ypred[SortIndex],color='red',linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

Dataset

Add Tips.csv (256 rows)

OUTPUT

(244, 2)



VIVA Questions

1. What is machine learning?
2. Define supervised learning?
3. Define unsupervised learning?
4. Define semi supervised learning?
5. Define reinforcement learning?
6. What do you mean by hypotheses?
7. What is classification?
8. What is clustering?
9. Define precision, accuracy and recall?
10. Define entropy?
11. Define regression?
12. How Knn is different from k-means clustering?
13. What is concept learning?
14. Define specific boundary and general boundary?
15. Define target function?
16. Define decision tree?
17. What is ANN?
18. Explain gradient descent approximation?
19. State Bayes theorem?
20. Define Bayesian belief networks?
21. Differentiate hard and soft clustering?
22. Define variance?
23. What is inductive machine learning?
24. Why K nearest neighbor algorithm is lazy learning algorithm?
25. Why naïve Bayes is naïve?
26. Mention classification algorithms?
27. Define pruning?
28. Differentiate Clustering and classification?
29. Mention clustering algorithms?
30. Define Bias?
31. What is A* Search Algorithm?
32. Why A* Search Algorithm?
33. What are the applications of AI?
34. Why is A* Search Algorithm Preferred?
35. How is A* calculated?
36. What is a Heuristic Function?
37. Is A* better than Dijkstra?
38. Why is A* optimal?

39. Give a real-time example that does not use A* Search?
40. How overestimation is handled in the A* algorithm?
41. What are the advantages and disadvantages of A*?
42. How does A* Search work?
43. What is AO* Algorithm?
44. List the advantages and disadvantages of AO*.
45. Which is best A* or AO* Search?
46. Differentiate between A* and AO* Search.
47. What is the applications of AO* Search?
48. Which is better A* or AO* Search?
49. What is problem and problem space?
50. What is Search?
51. Name few Heuristic Search Techniques.
52. What are the features of Hill Climbing?
53. What is Constraint Satisfaction?
54. What is AI Technique and its characteristics?

-------- *********---------