Ahsanullah University of Science and Technology (AUST)
Department of Computer Science and Engineering

LABORATORY MANUAL

Course No.: CSE4108
Course Title: Formal Languages and Compilers Lab

For the students of 4th Year, 1st semester of
B.Sc. in Computer Science and Engineering program

TABLE OF CONTENTS

COURSE OBJECTIVES 1

PREFFERED TOOLS 1

TEXT/REFERENCE BOOK 1

ADMINISTRATIVE POLICY OF THE LABORATORY 1

LIST OF SESSIONS

SESSION 1:
Introduction to Declarative and Procedural ways of Representing Knowledge 2

SESSION 2:
Elements of Informed Search 5

SESSION 3:
Best First Search 9

SESSION 4:
Local Search and Optimization 11

SESSION 5:
Classification and Learning 13

SESSION 6:
Game Playing and Adversarial Search 15

SESSION 7:
Uncertainty and Probabilistic Reasoning 18

SUPPLIMENTARY MATERIALS FOR SESSIONS 19

MID TERM EXAMINATION 35

TERM FINAL EXAMINATION 35

COURSE OBJECTIVES

- To be able to use basic elements of procedural and declarative representation of a knowledgebase along with query processing environment
- To be able to implement simple heuristic functions and to use those for best-first search problems
- To be able to understand and implement local search and beam search as optimization strategies
- To gain insights of supervised and unsupervised learning through implementation of common classification and regression algorithms
- To gain insights of adversarial search through basic game playing algorithms
- To gain insights of acting under uncertainty using probabilistic reasoning

PREFFERED TOOL(S)

- Prolog and Python

TEXT/REFERENCE BOOK(S)

- Artificial Intelligence: A Modern Approach, S. J. Russell & P. Norvig, Pearson, 3rd Edition.
- The Art of Prolog, Leon Sterling & Ehud Shapiro, MIT Press, 2nd Edition.
- Learn Python the Hard Way, Zed Shaw, Addison-Wesley, 3rd Edition.

ADMINISTRATIVE POLICY OF THE LABORATORY

- Students must perform class assignments individually, without the help of others.
- Viva for lab exercises and assignments will be arranged as an important component of the assessment procedure.
- Plagiarism is strictly prohibited and will be dealt with strictly.

Session 1: Basics of Procedural and Declarative Knowledgebase

I. OBJECTIVES

- To be able to use basic elements of Python for procedural programming of knowledgebase.
- To be able to represent query processing environments declaring facts and rules in Prolog.

II. DEMONSTRATION OF USEFUL RESOURCES

Knowledgebase and Queries to a Knowledgebase

A simple knowledgebase (KB) from the Kinship Domain

Object relationships as a KB:

Hasib is a parent of *Rakib*. *Rakib* is a parent of *Sohel*. *Rakib* is a parent of *Rebeka*. *Rashid* is a parent of *Hasib*. If X is a parent of Y and Y is a parent of Z, then X is a grandparent of Z.

List of tuples and sample procedure to manipulate the KB **in Python**:

```
tupleList1=[('parent', 'Hasib', 'Rakib'),('parent', 'Rakib', 'Sohel'),
        ('parent', 'Rakib', 'Rebeka'),('parent', 'Rashid', 'Hasib')]

# Procedure to find the grandchildren of X

X=str(input("Grandparent:"))
print('Grandchildren:', end=' ')
i=0
while(i<=3):
        if ((tupleList1[i][0] == 'parent')&( tupleList1[i][1] == X)):
                for j in range(4):
                        if ((tupleList1[j][0] == 'parent') & ( tupleList1[i][2] ==
                                tupleList1[j][1])):
                          print(tupleList1[j][2], end=' ')
        i=i+1
```

Facts and Rules (KB) **in Prolog**:

```
parent('Hasib' , 'Rakib'). parent('Rakib' , 'Sohel'). parent('Rakib' , 'Rebeka').
parent('Rashid' , 'Hasib'). grandparent(X, Z) :- parent(X, Y), parent(Y, Z).

/* [Built-in KB is enhanced with the 4 facts and 1 rule; two 2-place predicates;  3 variables;
full stop
    (.) as the end marker of a clause/ sentence / statement;  :- as 'if';  comma (,) as logical
AND. ]*/

/* Procedure to find the grandchildren of X */

        findGc :- write(' Grandparent: '), read(X), write('Grandchildren: '),
                grandparent(X, Gc), write(Gc), tab(5), fail.
        findGc.
```

How *can we* ***modify*** *the codes to find the grandparents of somebody?*
Note *that we need to make more changes in Python than in Prolog.*
Moreover*, we can pose diverse queries to Prolog code and get interpretable answers.*

III. LAB EXERCISE

1) Explore thoroughly the supplementary material provided for this session at the end of the Manual.
2) Run and analyze the codes demonstrated in this session.
3) Modify the Python and Prolog codes demonstrated above to find the grandparents of somebody.
4) Enrich the KB demonstrated above with 'brother', 'sister', 'uncle' and 'aunt' rules in Python and Prolog.

Session 2: Elements of Informed Search

I. OBJECTIVES

- To be able to implement simple heuristic functions in Prolog and in Python.
- To be able to use heuristic functions for simple search problems.

II. DEMONSTRATION OF USEFUL RESOURCES

Heuristic functions for informed search

A. Heuristic functions for general graph search problems
   - A common practice for general graph search problems is to take 'straight line distance' between two nodes, computed somehow, as a heuristic function value.
   - We consider this type of heuristics as 'given' for solving problems and will involve in upcoming sessions.

B. Heuristic functions for other types of problems

i) Consider the following instance of the 8-puzzle problem.

Goal state:

1	2	3
8		4
7	6	5

Current state:

8	1	2
3	6	4
	7	5

Prolog representation of the states may have the following form

gtp(1,1,1). gtp(2,1,2). gtp(3,1,3). gtp(4,2,3).gtp(5,3,3). gtp(6,3,2). gtp(7,3,1). gtp(8,2,1). gblnk(2,2).

tp(1,1,2). tp(2,1,3). tp(3,2,1). tp(4,2,3).tp(5,3,3). tp(6,2,2). tp(7,3,2). tp(8,1,1). blnk(3,1).

- We can think of a **heuristic function (h_1)** that determines the number of mismatching tiles.

Possible Prolog code may have the following form:

```
go:- calcH(1,0,H), write('Heuristics: '),write(H).
calcH(9,X,X):-!.    calcH(T,X,Y):- check(T,V), X1 is X+V, T1 is T+1, calcH(T1,X1,Y).
check(T,V):-tp(T,A,B), gtp(T,C,D), A=C, B=D, V is 0,!.    check(_,1):-!.
```

Possible Python representation and procedure may have the following form:

```
gtp=[(1,1,1), (2,1,2), (3,1,3), (4,2,3), (5,3,3), (6,3,2), (7,3,1), (8,2,1)]
gblnk = (2,1)
tp=[(1,1,2), (2,1,3), (3,2,1), (4,2,3), (5,3,3), (6,2,2), (7,3,2), (8,1,1)]
blnk = (3,1)
```

1	2	3
8		4
7	6	5

```
# Procedure to find the number of mismatches
i,h=0,0
while(i<=7):
    if ((gtp[i][1] != tp[i][1])|(gtp[i][2] != tp[i][2])):
            h=h+1
    i=i+1
print('Heuristics 1: ',h)
```

8	1	2
3	6	4
	7	5

- We can think of another **heuristic function (h_2)** where Manhattan distances of the tiles are calculated.
  Possible Prolog code may have the following form:

```
go:- calcH(1,[],L), sumList(L,V),write('Heuristics: '),write(V).
calcH(9,X,X):-!.    calcH(T,X,Y):- dist(T,D), append(X,[D],X1), T1 is T+1,
calcH(T1,X1,Y).
dist(T,V):-tp(T,A,B), gtp(T,C,D), V is abs(A-C) + abs(B-D).
sumList([],0):-!.    sumList(L,V):-L=[H|T], sumList(T,V1), V is V1+H.
```

ii) Consider the following instance of 8-queens problem and a **heuristic function (h_3)** that returns the number of attacking pairs of queens.

8							Q	
7				Q				
6	Q							
5			Q					
4					Q			
3						Q		
2								
1		Q						Q
	1	2	3	4	5	6	7	8

State I

- Complete-state formulation of problem; I: 61574381, 1st queen is at the 6th row, 2nd queen at the 1st row, ….
- Any placement of queens can be taken as an initial state, but no fixed goal state.
- h will mean number of pairs of queens that are in attacking position (face to face); h(I) = 5; We try to minimize h; Global minimum = 0;

h(I) = face to face in the row + face to face diagonally up + face to face diagonally down
= 1+1+3 = 5.

How to compute this function using Prolog and Python?

III. LAB EXERCISE

1. Explore thoroughly the supplementary material provided for this session at the end of the Manual.
2. Run and analyze the codes demonstrated in this session.
3. Define a recursive procedure in Python and in Prolog to find the sum of 1st n terms of an equal-interval series given the 1st term and the interval.
4. Define a recursive procedure in Python and in Prolog to find the length of a path between two vertices of a directed weighted graph.
5. Modify the Python and Prolog codes demonstrated above to find h_2 and h_3 discussed above.

Session 3: Best-First search in Graph representation of Problems

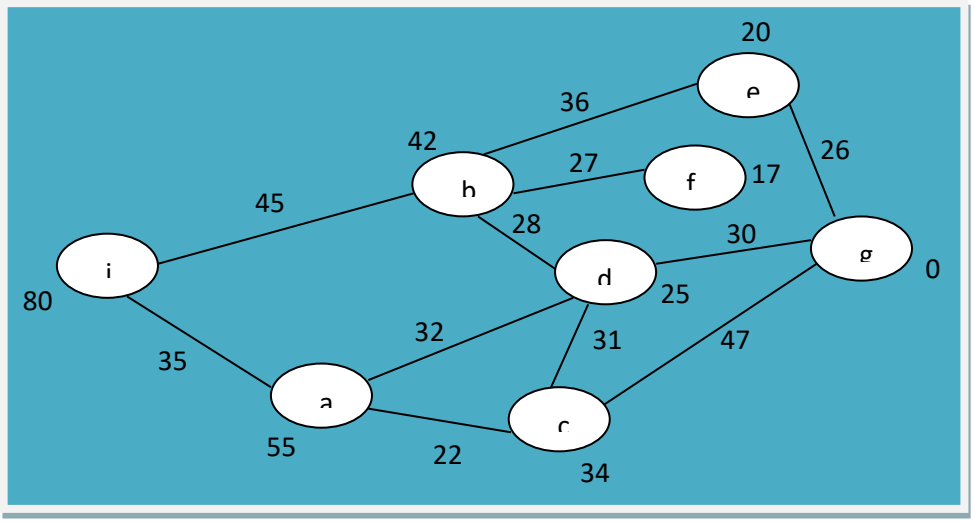
I. OBJECTIVES

- To be able to understand Greedy Best-First and A* search algorithms.
- To be able to implement Greedy Best-First and A* search algorithms in Prolog and in Python.

II. DEMONSTRATION OF USEFUL RESOURCES

C. Greedy Best-First Search

Consider a problem instance given in the following graph.



Node	Neighbor	Distance
i	a	35
i	b	45
a	c	20
…	…	…

Node	h(Node)
i	80
a	55
b	42
…	…

i - Initial state (source) g - Goal state (destination) h - heuristic function (straight line distance)

Basic idea and Major steps of the algorithm:

1) A node is selected for expansion based on an evaluation function, f(n), which is taken f(n) = h(n.
2) A Priority Queue (PQ), which contains nodes in ascending order of h-values, is maintained.
3) A Possible Path (PP) is maintained that contains nodes currently supposed to be in the solution.
4) A tree of visited nodes along with their children is also maintained which helps to update PQ and PP.
5) The process begins by placing the source node in the empty PQ, and initiating a tree by placing that node as its root.
6) The process terminates when the destination node is placed in the PQ, and consequently, selected for visit.
7) The 1st node from the PQ is selected repeatedly, and each time the tree, the PQ and the PP are updated:
   A. The node in the tree is marked visited and its neighbors from the graph are added to the tree as its children, while no repeated node is allowed in the tree;
   B. The node itself is deleted from the PQ, but its children are added to the PQ.
   C. The PP is straightened up to the root from the selected node.

Sample representation of tree, PQ and PP in Prolog:

```
t_node(i, nil). t_node(a, i).
t_node(b, i). t_node(d, b).
…
```

```
pq([node(b, 42), node(a, 55)]).
…
```

```
pp([i, b, e, g]).
```

D. **A* Search: Minimization of the total estimated solution cost**

Distinguishing features:

- Evaluation function,
  f(n) = g(n) + h(n), where
  g(n) = an actual path cost from initial node to node n,
  h(n) = estimated cost of the cheapest path from n to the goal.
- Generates all neighbors (repeatedly, if a path is there), and puts into PQ.
- Suboptimal solutions are avoided.

Sample representation of tree, PQ and PP in Prolog:

```
tr_node(i, 0, nil, 80).
tr_node(a, 1, 0, 90).
tr_node(b, 2, 0, 87).
tr_node(i, 3, 2, 170).
tr_node(d, 4, 2, 98).
tr_node(e, 5, 2, 101).
…
```

```
pq([node(g, 17, 10, 97),
node(d, 4, 2, 98),
node(e, 5, 2, 101),
node(g, 13, 9, 104),
      … ].
```

```
pp([i, a, d, g]).
```

III. LAB EXERCISE

1. Explore thoroughly the supplementary material provided for this session at the end of the Manual.
2. Run and analyze the codes demonstrated in this session.
3. Write a Python program that reads the file created as demonstrated into a dictionary taking ‘name’ as the key and a list consisting of ‘dept’ and ‘cgpa’ as the value for each line. Make changes in some ‘cgpa’ and then write back the whole file.
4. Implement in generic ways (as multi-modular and interactive systems) the Greedy Best-First and A* search algorithms in Prolog and in Python.

Session 4: Local Search and Optimization Strategy

I. OBJECTIVES

- To be able to understand hill-climbing local search and beam search strategies.
- To be able to implement simpler variants of hill-climbing and genetic algorithms in Prolog and in Python.

II. DEMONSTRATION OF USEFUL RESOURCES

E. Developing a multi-modular system for Hill-climbing Local Search

We take the 8-queens problem to demonstrate the working of the Hill-climbing search strategy. A state is represented as an eight-digit positive integer (with 1, 2, 3, …, 8 only). We generate all 56 successors of a current state and choose the one that appears best as per a heuristic function. The process is repeated until a state with a specified value is found. Here is the possible outcome of a typical implementation of the algorithm.

For the initial state 23456578, with threshold value 27, after 3 iterations a solution was found in the following form:

Iteration max: 20
Iteration max: 24
Iteration max: 25

Found! Id:45 s [7,3,4,6,1,5,2,8] Value:27

And the states were as follows:

state(1, c, [7, 3, 4, 6, 1, 5, 7, 8], 25).
state(2, s, [1, 3, 4, 6, 1, 5, 7, 8], 23).
state(3, s, [2, 3, 4, 6, 1, 5, 7, 8], 24).
…
state(44, s, [7, 3, 4, 6, 1, 5, 1, 8], 25).
state(45, s, [7, 3, 4, 6, 1, 5, 2, 8], 27).
state(46, s, [7, 3, 4, 6, 1, 5, 3, 8], 24).
…
state(55, s, [7, 3, 4, 6, 1, 5, 7, 5], 25).
state(56, s, [7, 3, 4, 6, 1, 5, 7, 6], 24).
state(57, s, [7, 3, 4, 6, 1, 5, 7, 7], 23).

The system gets stuck up frequently at local maxima if the threshold value is set at 28. To avoid the local maxima we consider the following three variants of the algorithm:

a) **Random restart hill climbing:** If stuck up at a local maximum, then begin with a new randomly **generated** state.
b) **Stochastic Hill-climbing:** Choose one at random from among the uphill moves.

c) Simulated annealing: Choose one at random from among the successors. Allow an uphill successor directly, but sometimes allow a downhill one, with a given probability. [Temperature change may be taken from 25.0 to 0.0 with an interval of 0.1, and the formula for probability as $e^{\Delta E/T}$, whereΔE means change in energy (downhill value) and T means temperature.]

F. Developing a multi-modular system for a typical genetic algorithm

We take a few states of the 8-queens problem as the initial population, set a threshold value for formation of parent generation, and set a target value of the fitness function. Crossover in parent population is allowed, and sometimes mutation in some new individual is also allowed.

Sample initial population may look as follows:

```
intl_sts(12345678).
intl_sts(87654321).
intl_sts(18273645).
intl_sts(45362718).
intl_sts(15263748).
intl_sts(84736251).
intl_sts(13572468).
intl_sts(24681357).
```

Formation of new population and evaluation of the individuals are carried out until an individual with the target fitness is found.

III. LAB EXERCISE

1. Explore thoroughly the supplementary material provided for this session at the end of the Manual.
2. Run and analyze the codes demonstrated in this session.
3. With the help of the supplementary materials and demonstrated codes implement the variants of hill-climbing and genetic algorithms discussed above in Prolog and Python.

Session 5: Classification and Learning

IV. OBJECTIVES

- To gain insights of supervised and unsupervised machine learning techniques;
- To be able to implement simple classification and regression algorithms using Python Libraries.

V. DEMONSTRATION OF USEFUL RESOURCES

Machine Learning is an application of artificial intelligence that provides systems the ability to improve from experience.

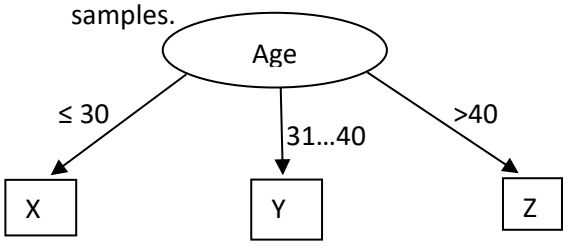
- ✓ Machine learning algorithms are often categorized as supervised or unsupervised.
- ✓ In supervised learning, the machine is 'trained' using data which are labeled, while unsupervised machine learning allows a model to work on its own to discover information.
- ✓ Regression and classification are two prominent approaches of learning.
- ✓ In regression the output variable takes a value from a continuous set of numbers, whereas in classification the output variable takes a class tag (label/category/discrete number).
- ✓ In regression analysis, curve fitting is a common process. There are many regression techniques such as linear regression and polynomial regression.
- ✓ There are different classification approaches such as Decision Tree, Naïve Bayes, Gradient Descent, K-Nearest Neighbor, Random Forrest, Support Vector Machine etc.
- ✓ Some classification approaches can be used for regression analysis as well, for example, Decision Tree regression and Support Vector regression.
- ✓ Clustering is a common unsupervised technique which is the process of grouping similar entities together. The goal is to find similarities in the data and group similar data.

1) Learning Decision Trees

Training Samples: [Described through attribute values along with the class they belong to, from Data Mining by Han & Kamber]

ID	**Age**	**Income**	**Student**	**Credit Rating**	*Decision/ Class/*
1	≤ 30	high	no	fair	negative
2	≤ 30	high	no	excellent	negative
3	31…40	high	no	fair	positive
4	> 40	medium	no	fair	positive
…	…	…	…	…	…

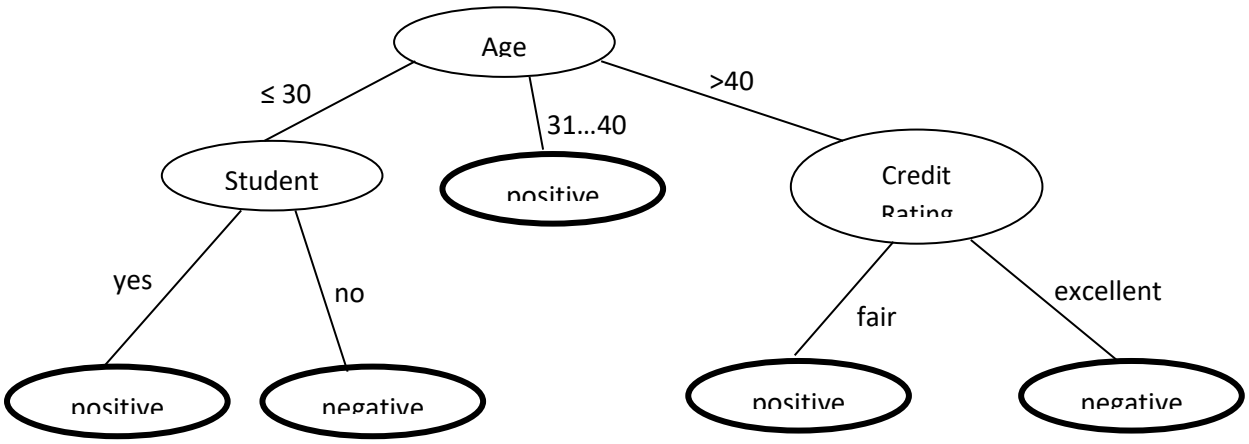
In each step a root node for a tree/subtree is generated based on best information gain from the samples.



X =

Age	**Income**	**Student**	**Credit**	*Decision/*
≤ 30	high	no	fair	negative
≤ 30	high	no	excellent	negative
≤ 30	medium	no	fair	negative
≤ 30	low	yes	fair	positive
≤ 30	medium	yes	excellent	positive

Finally, we get a tree like the one below from the given set of samples:



And it means that we have **learned the following 5 rules**.

a. If 'Age' = '≤ 30' and 'Student' = 'yes', then 'Class' = 'Buys a computer'.
b. If 'Age' = '≤ 30' and 'Student' = 'no', then 'Class' = 'Does not buy a computer'.
c. …..
d. …..
e. If 'Age' = '>40' and 'Credit Rating' = 'excellent', then 'Class' = 'Does not buy a computer'.

- These rules are used to find the class belonging of the samples in test set. For example, the tast case, X = (age = 22, income = 'medium', student = 'yes', credit_rating = 'fair') will belong to the class 'positive' ('Buys a computer').

2) Naïve Bayes Classifier

- ✓ We take the same data set and apply the following simplified forms of Bayes' theorem.
- ✓ For an unknown sample, X = (x_1, x_2, … , x_n), classifier should predict that X belongs to one of m classes, C_i with highest posterior probability

  $P(C_i \mid X) > P(C_j \mid X)$, $1 \leq j \leq m$ & $j \neq i$. [Maximun posterior probability]
- ✓ According to Bayes' theorem:

  $P(C_i \mid X) = (P(X \mid C_i) \times P(C_i)) / P(X)$

  As P(X) is constant for all classes, $P(X \mid C_i) \times P(C_i)$ needs to be maximized.
- ✓ $P(C_i) = S_i / S$, where S_i – no. of samples of class C_i, S – total no. of samples.
- ✓ And Discarding attribute dependence,

  $P(X \mid C_i) = \prod_{k=1:n} P(x_k \mid C_i)$.
- ✓ We take, C_1: 'Buys a computer' / 'positive' and C_2: 'Does not buy a computer' / 'negative'.
- ✓ The unknown sample we want to classify is

  X = (age = 22, income = 'medium', student = 'yes', credit_rating = 'fair')
- ✓ We now compute $P(X \mid C_i)$, for i = 1, 2 as follows:

  $P(\text{age} = \text{'<=30'} \mid C_1) = 2/9 = 0.222$

  ….
- ✓ We obtain,

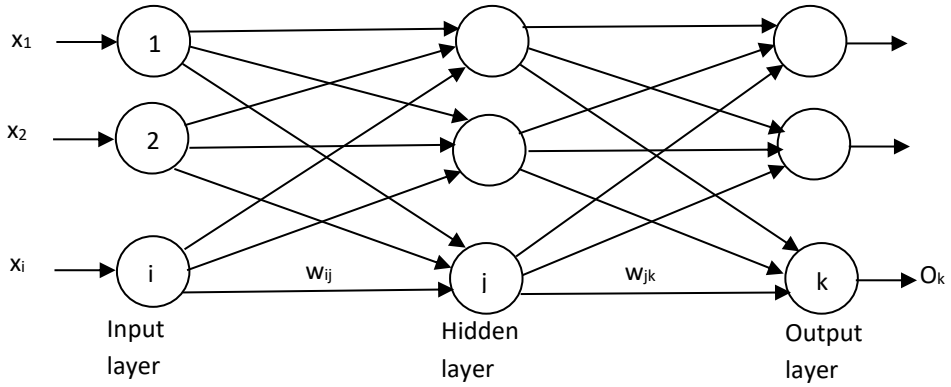
  $P(X \mid C_1)\, P(C_1) = 0.044 \times 0.643 =$ **0.028**

$P(X|\ C_2)\ P(C_2) = 0.019 \times 0.357 = 0.007$

- ✓ That is, prediction for sample X is 'positive' ('Buys a computer'), as before (with decision tree)

3) Neural Network Learning

- ✓ We consider the back-propagation algorithm using MLP(multilayer perceptron) concept
- ✓ A two-layer fully connected feed-forward Artificial Neural Network is shown below:



- ✓ ($x_1, x_2, \ldots, x_i$) – numerically scaled and normalized attribute values of a sample.
- ✓ Weighted output of one layer is passed on to the next.
- ✓ Training Samples are fed and network parameters like **weights are adjusted** based on feedback (the last layer output). Thus 'error' is back-propagated to adjust parameter, that is, to learn.

4) Linear Regression

- ✓ Data are modeled using a straight line.

  $Y = \alpha X + \beta$

  Y – random variable (response, dependent)

  X – random variable (predictor, independent)

  α, β - regression coefficients, that are to be learned

- ✓ To solve means to find estimated values of α and β that best describes the data.
- ✓ Methods of least squares can be used to find α and β minimizing error between the actual data and the estimate of the line.

  $\alpha = \sum_{i=1:s} (x_i - x')(y_i - y') / \sum_{i=1:s} (x_i - x')^2$, $\beta = y' - \alpha x'$,

  where x' - average of $x_1, x_2, \ldots, x_s$, y' - of $y_1, y_2, \ldots, y_s$, given sample data points $(x_1, y_1), (x_2, y_2), \ldots, (x_s, y_s)$.
- ✓ The line thus obtained can be used to predict an appropriate value of y, given an unknown x.

5) k-Nearest Neighbor Classifier

- ✓ Each sample represents a point in an n-dimensional 'pattern space' of samples.
- ✓ Closeness may be defined by Euclidian distance in the following way:

$$D(X, Y) = (\sum_{i=1:n} (x_i - y_i)^2)^{1/2} ,$$

where $X = (x_1, x_2, \dots , x_n)$ and $Y = (y_1, y_2, \dots , y_n)$ are two points in the pattern space.

- ✓ The unknown sample is assigned the most common class from among its k nearest neighbors.

6) k-Means Clustering

- ✓ Takes input parameter k and partitions the set of n objects into k clusters so that the intra-cluster similarity is high, while inter-cluster similarity is low.
- ✓ Similarity is measured with respect to the mean value of the objects in a cluster.
- ✓ Initially k objects are selected randomly as centers of clusters, and then others are assigned to the clusters based on the similarity (distance to a cluster center).
- ✓ Each time cluster center (mean of a cluster) is updated; Iterated until the criterion function converges.
- ✓ Typically, the squared error criterion is used:

$$E = \sum_{i=1:k} \sum_{p \in Ci} |p - m_i|^2$$

E – sum of the squared errors of all objects; minimized (until no change)
p – point in space representing a given object
m_i is the mean of cluster C_i

VI. LAB EXERCISE

1. Explore thoroughly the provided material along with the supplementary material at the end.
2. Run and analyze the demonstrated codes.
3. Implement Linear Regression and k-Nearest Neighbor Classifier without using Scikit-learn.

Session 06: Game Playing and Adversarial Search

I. OBJECTIVES

- To gain insights of adversarial search through basic game playing algorithms.
- To be able to implement search space improvement techniques for game playing.

II. DEMONSTRATION OF USEFUL RESOURCES

Game playing assumes multiple-agent environment, and thus offers ideal example for adversarial search. As the agents' goals are in conflict and they always plan against each other, the search space becomes complicated. Moreover, real games involve huge state spaces.

A. Finding Optimal Game Strategies using MINIMAX Algorithm

Two-player board game as a search problem:

✓ Players are usually named MAX & MIN. Anyone can start, and they make moves alternating one another.

✓ Search problem with 4 components: Initial state, Successor function, Terminal test, Utility function.

✓ Strategies of Players: MAX searches for the sequence of moves that leads to a terminal with maximum possible utility value, even if MIN plays in the best way; MIN searches for the opposite, that is, terminal with minimum possible utility.

✓ Major steps of the MINIMAX algorithm, from opener's point of view:

1. Generate the whole game tree.
2. Find the utility of the terminal nodes.
3. Determine the MINIMAX values of the non-terminal nodes, from lower nodes up to the root. If a level represents MAX's turn, then the highest values of the successors are taken, and in case of MIN's – lowest values.
4. Choose the best opening move.

✓ An imaginary game of small depth may be used for explanation. The game of Tic-Tac-Toe is suggested for implementation in Python or Prolog.

- A 3x3 grid is provided with the information of opener, and his/her symbol.
- All nodes up to the terminals are generated, and utilities (-1, 0, +1) are assigned to them.
- The MINIMAX values of non-terminals are computed up to the root, and the winning strategy is returned.

B) Improving the Performance of the MINIMAX search Strategy

Reduction of search space using alpha-beta pruning:

Cutting off in compliance with already calculated minimax values, that is, values of

- best choices for the maximizing player, α,
- best choices for the minimizing player, β.

Example:

Say, in absence of true minimax values we are given the following sequence of numbers from which we are supposed to assign a value to each newly generated 'terminal' state, that is, a state at the cutoff depth:

4, 3, -1, 4, 5, 2, 1, -1, -5, 3, 2, 1. [-5, +5]

We assume further:

- Branching factor = 2;
- Cutoff depth = 2 moves or 4 plies;
- MAX makes the opening move;
- Left to right expansion of the tree is followed.

We do keep in mind:

- It requires to try to prune in every occasion;
- To prune a branch, one must know at least the range of all other siblings, and it is not enough.

A pruned tree under the above conditions is a simple one, much reduced.

III. LAB EXERCISE

1. Implement the game of Tic-Tac-Toe as suggested in Python or Prolog.
2. Write a program in Prolog or Python to construct a pruned game tree using Alpha-Beta pruning. Take the sequence, [5, 3, 2, 4, 1, 3, 6, 2, 8, 7, 5, 1, 3, 4] of MINIMAX values for the nodes at the cutoff depth of 4 plies. Assume that branching factor is 2, MIN makes the first move, and nodes are generated from *right to left*.

Session 07: Uncertainty and Probabilistic Reasoning

I. OBJECTIVES

- To gain insights of acting under uncertainty using probabilistic reasoning.
- To be able to implement simple environments for making probabilistic inference.

II. DEMONSTRATION OF USEFUL RESOURCES

For decision making, rational agents are supposed to take help of probabilistic reasoning, beside utility theory for choosing from alternatives. Those tools are required for dealing with uncertainty due to partial knowledge of the environment, which is unavoidable.

B. Inference using Full Joint-Probability Distribution

✓ A full joint-probability distribution of random variables describing the whole of a domain can be used as a complete knowledgebase to answer any question involving the variables.

✓ For example, we can take a domain described using 3 Boolean random variables. The joint probabilities of the random variables, taken from a domain expert, may look as shown below.

	A		¬A	
	C	¬C	C	¬C
B	0.108	0.012	0.072	0.008
¬B	0.016	0.064	0.144	0.576

Observe that the sum of the entries is 1.

And we can compute probability of any compound proposition like $B \wedge \neg C$, $A \vee C$, $C \mid \neg B$, etc. from the given entries in the following way.

- $P(B \wedge \neg C) = P(A \wedge B \wedge \neg C) + P(\neg A \wedge B \wedge \neg C) = 0.012 + 0.008$
- $P(A \vee C) = P(A) + P(C)$
  $= P(A \wedge B \wedge C)+P(A \wedge B \wedge \neg C)+P(A \wedge \neg B \wedge C)+P(A \wedge \neg B \wedge \neg C)+$
  $P(\neg A \wedge B \wedge C)+P(\neg A \wedge \neg B \wedge C)$
- $P(C \mid \neg B) = P(C \wedge \neg B)/P(\neg B)$

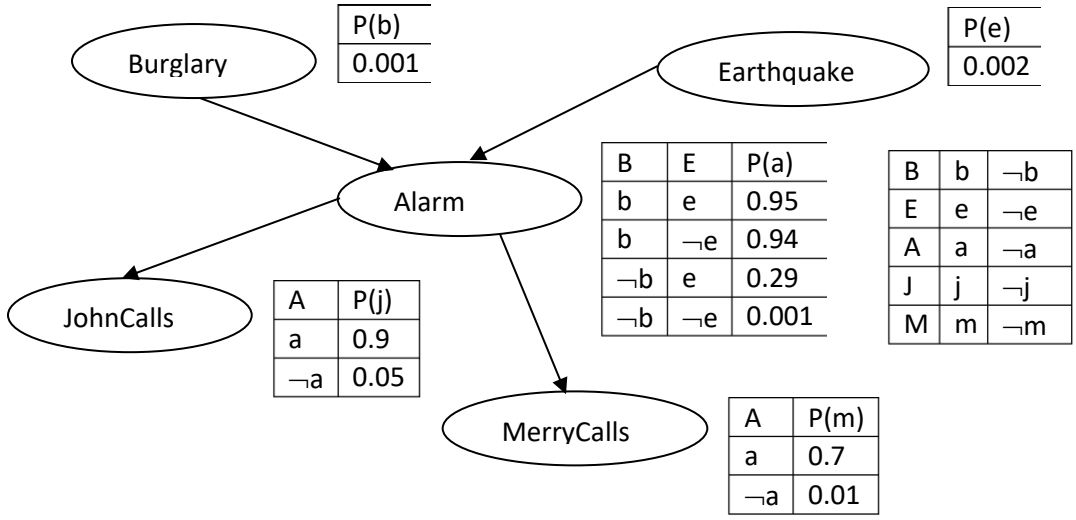
C) Probabilistic Reasoning using Bayesian Networks

A Bayesian Network is a data structure represented by a directed acyclic graph. A node represents a random variable and an arc represents a 'parent-child' relationship. A node X_i is assigned a conditional probability table that quantifies the effect of the parents on the node, that is, the distribution,

$P(X_i \mid Parents(X_i))$.

A Bayesian Network also provides a complete and useful description of the domain. It is as powerful as full joint-probability distribution, and at the same time, it is much easy to specify. It is thus appropriate for real world problems.

❖ **An example of Bayesian Networks with five Boolean random variables from the textbook**



Making inference using the joint probabilities in the network:

- P(a ∧¬j ∧ m ∧¬b ∧ e) = P(a | ¬b ∧ e) x P(¬j | a) x P(m | a) x P(¬b) x P(e)
  = 0.29 x 0.1 x 0.7 x 0.999 x 0.002
- P(b | a ∧ j ∧ m) = P(b ∧a ∧ j ∧ m) / P(a ∧ j ∧ m)
- P(b ∧a ∧ j ∧ m) = P(b ∧a ∧ j ∧ m∧ e) + P(b ∧a ∧ j ∧ m∧¬e)
- P(a ∧ j ∧ m) = P(a ∧ j ∧ m∧b∧e) +P(a ∧ j ∧ m∧b∧¬e) + P(a ∧ j ∧ m∧¬b∧e) +
  P(a ∧ j ∧ m∧¬b∧¬e)

III. LAB EXERCISE

1. Implement in Python or Prolog the environment for probabilistic inference using full joint-probability distribution as shown above.
2. Implement in Python or Prolog the environment for probabilistic inference using a Bayesian network as shown above.

Supplementary Material for Session 1

I. Queries to KB and finding an answer using Backward Chaining in Prolog:

Is Hasib a grandparent of Rebeka?

grandparent ('Hasib', 'Rebeka').

- parent ('Hasib', Y), parent(Y, 'Rebeka').

parent ('Hasib', 'Rakib'). [Y ← Rakib]

- parent ('Rakib', 'Rebeka').

Yes.

❖ Various types of queries are possible.

Who are parents of Rebeka? parent(X,'Rebeka').
Who are parents? parent(X, _).
Is Hasib a parent? parent('Hasib', _).
Is Hasib a parent of Rebeka? parent('Hasib', 'Rebeka').
Who have parents? parent(_, X).
Who are parents of whom? parent(X, Y).
Is there anybody who is a grandparent of somebody. grandparent(_, _).
Does Sohel have a grandparent? grandparent(_,'Sohel').
Who is a parent and, also, has a parent? parent(X, _),parent(_, X).
Who either is a parent or has a parent? parent(X, _);parent(_, X).

❖ Various rules may also be formulated for father, mother, brother, sister, aunt, uncle, etc. There may be more than one rule to define, for example, a grandfather.

[brother(X,Y):-parent(Z,X), parent(Z,Y), male(X), not(X=Y).]

❖ Nesting of the following type should be avoided.

greatGrandParent (X, Z) :- parent(X, Y), grandparent(Y, Z).
greatGreatGrandParet(X, Z) :- parent(X, Y), greatGrandParent(Y,Z).

II. Working with Structured Data and functions in Python:

- Lists, strings and tuples are ordered sequences of objects.
- Lists and tuples can contain any type of objects. Lists and tuples are like arrays.
- Lists are mutable, so they can be extended or reduced at will.
- Tuples, like strings, are immutable. Tuples are faster, and consume less memory.
- Strings contain only characters.
- A dictionary is an unordered collection of key-value pairs, which can be modified.

```
#List
l1=[0,2,1]
l1[1]
l1[1]=3
l1.append(5)
l2=[3,4,5]
l1.extend(l2)
print("Length:" ,len(l1))
#Tuple
L3=(2,4,1)
L3[1] # L3[1]= 5 not
allowed
```

```
#String
S="This is AUST"

#Dictionary
d = {"a":1, "b":2}
d["z"]=4    # d["b"] returns 2
for key in d:
print(key)
for value in d.values():
print(value)
for key, value in d.items():
print(key , ":", value)
```

```
#Python is Easy
#Observe the dialog in shell
>>> x=[1,2,3]
>>> y=(9,8)
>>> x
[1, 2, 3]
>>> y
(9, 8)
>>> x,y=y,x
>>> x
(9, 8)
>>> y
[1, 2, 3]
>>> for i in range(5):
      print(i)
>>> for i in range(1,10,2):
      print(i, end=' ')
```

```
#Python is Easy
#Observe the code of user defined function
def fssum():
        a=int(input("Start:"))
        d=int(input("Interval:"))
        n=int(input("n:"))
        i,s=1,0
        while(i<=n):
                s=s+a+d*(i-1)
                i=i+1
        print("Sum:",s)
        input("Press Enter to continue")
# Main
t=int(input("How many times?"))
for i in range(t):
        print("Iteration:",i+1)
        fssum()
```

Supplementary Material for Session 2

I. Recursion in Prolog:

```
parent('Hasib' , 'Rakib').
parent('Rakib' , 'Sohel').
parent('Rakib' , 'Rebeka').
parent('Rashid' , 'Hasib').
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
ancestor(X, Y) :- parent(X, Y), !.
findancestor:- write('Name:'), read(Y),
ancestor(X,Y), write(X), nl, fail.
findancestor.
```

i) Ancestor
   a. A parent is an ancestor.
   b. A parent of an ancestor is an ancestor.
   [X is an ancestor of Y, if X is a parent of an ancestor of Y.]

```
------------------------
ancestor(X, Y) :- parent(X, Y), !.
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```

ii) Factorial of 0 is 1.
If n is greater than 0, then factorial of n is the product of n and the factorial of n-1.

[Factorial of N is F, if N is greater than 0 and F is the product of N and factorial of N-1.]

```
-----------------------
factorial(0, 1):-!.
factorial(N, F) :- N>0, N1 is N-1, factorial(N1, F1), F is N*F1.
```

iii) 100+105+110+ … +(100+(n-1)x5)

Sum of the 1st one element is 100.
Sum of the 1st n elements is the sum of 1st n-1 elements and the nth element, which is 100+(n-1)x5.

```
--------------------
sum1(1, 100):-!.
```

sum1(N, S):-N1 is N-1, sum1(N1, S1), S is S1+100+(N-1)*5.

iv) Representing a weighted graph and finding the length of a path

```
neighbor(i,a,35). neighbor(i,b,45). neighbor(a,c,22).
neighbor(a,d,32). neighbor(b,d,28). neighbor(b,e,36).
neighbor(b,f,27). neighbor(c,d,31). neighbor(c,g,47).
neighbor(d,g,30). neighbor(e,g,26).

pathLength(X,Y,L):- neighbor(X,Y,L),!.
pathLength(X,Y,L):- neighbor(X,Z,L1), pathLength(Z,Y,L2), L is L1+L2.
```

II. Working with Python:

```
def ssum(N,I,F):
    if (N==0):
        return 0
    elif (N>=1):
 return ssum(N-1,I,F)+F+(N-1)*I
# Main
t=int(input('How many times?'))
for i in range(t):
    print('Iteration:',i+1)
    f=int(input('First element:'))
    d=int(input('Interval:'))
    n=int(input('n:'))
    print('Series sum:', ssum(n,d,f))
```

```
# Use of global variablesin Python
def f():
    global s
    print (s)
    s = "I love python!"
    print (s)

# Global Scope
s = "Python is great!"
f()
print (s)
```

III. h_3 in Prolog.

```
:-dynamic(hval/1).

/* Evaluates a 8-queens' state given as list of 8 digits */

evalState(L,V):- assert(hval(0)),hl(1,L), di_up(1,L),di_dn(1,L),hval(V),
 retractall(hval(_)).

hl(8,_):-!.  hl(I,L):- nthel(I,L,X), chk_incr(I,L,X), I1 is I+1, hl(I1,L).

chk_incr(8,_,_):-!.  chk_incr(I,L,X):- I1 is I+1, nthel(I1,L,Y),
 do_incr(X,Y),chk_incr(I1,L,X).
do_incr(X,Y):- X=Y, incr_hval.  do_incr(_,_).

incr_hval:-hval(V), V1 is V+1, retract(hval(_)), assert(hval(V1)).

di_up(8,_):-!.  di_up(I,L):- nthel(I,L,X), chkup_incr(I,L,X,0), I1 is I+1,
di_up(I1,L).
chkup_incr(8,_,_,_):-!.
chkup_incr(I,L,X,K):- I1 is I+1, nthel(I1,L,Y), K1 is K+1,
                doup_incr(X,Y,K1), chkup_incr(I1,L,X,K1).

doup_incr(X,Y,K1):- X1 is X+K1, Y=X1, incr_hval.  doup_incr(_,_,_).

di_dn(8,_):-!.  di_dn(I,L):- nthel(I,L,X), chkdn_incr(I,L,X,0), I1 is I+1,
 di_dn(I1,L).

chkdn_incr(8,_,_,_):-!.
chkdn_incr(I,L,X,K):- I1 is I+1, nthel(I1,L,Y),K1 is K+1,
                      dodn_incr(X,Y,K1), chkdn_incr(I1,L,X,K1).

dodn_incr(X,Y,K1):- X1 is X-K1, Y=X1, incr_hval.  dodn_incr(_,_,_).


% A procedure to find the nth element of a list

nthel(N,[_|T],El):- N1 is N-1, nthel(N1,T,El).
nthel(1,[H|_],H):-!.
```

Supplementary Material for Session 3

A. Major components of Prolog code for GBF and A* search:

```
% Including data files
:-use_module(inputGraph).
----------

% Declaration of dynamic data
:-dynamic(t_node/2).
:-dynamic(pq/1).
:-dynamic(pp/1).
----------
Changed: …:-dynamic(t_node/4).
Additional:      :-dynamic(t_n_indx/1).

% Search begins
search:-write('Enter start node:'),read(S),h_fn(S,HV),
        assert(t_node(S, 'nil')),assert(pq([node(S,HV)])),
        assert(pp([])),generate,find_path_length(L), display_result(L).
--------
Changed: … assert(t_node(S,0,nil,HV)),assert(pq([node(S,0,'nil',HV)])),assert(t_n_indx(1)), …

% Generating the solution
generate:-pq([H|_]),H=node(N,_),N=g, add_to_pp(g),!.
generate:-pq([H|_]),H=node(N,_),update_with(N), generate.
---------
Changed: … H=node(N,_,_,_) … H=node(N,I,_,_), update_with(N,I)

% Adding a node to possible path
add_to_pp(N):-pp(Lst), append(Lst,[N],Lst1), retract(pp(_)),
        assert(pp(Lst1)).
---------

% Updating data according to selected node.
update_with(N):-update_pq_tr(N), update_pp(N).
---------
Changed: … (N,I)…

% Updating Priority Queue and Tree
update_pq_tr(N):-pq(Lst), delete_1st_element(Lst,Lst1), retract(pq(_)),
        assert(pq(Lst1)), add_children(N).

delete_1st_element(Lst,Lst1):-Lst = [_|Lst1].

add_children(N):- neighbor(N,X,_), not(t_node(X,_)),insrt_to_pq(X),
        assert(t_node(X,N)),fail.
add_children(_).
---------
```

Changed: … (N,I)…

```
add_children(N,I):- neighbor(N,X,D), t_n_indx(I1), t_node(_,I,_,V),
        h_fn(N,V1), h_fn(X,V2),  FNV is V+D-V1+V2,
        insrt_to_pq(X,I1,I,FNV), assert(t_node(X,I1,I,FNV)),
        incr_indx, fail.
add_children(_,_).
incr_indx:-  t_n_indx(X), Y is X+1, retract(t_n_indx(X)),assert(t_n_indx(Y)).

% Inserting node to Priority Queue
insrt_to_pq(X):- pq(Lst), h_fn(X,V), insert12pq(node(X,V),Lst,Lst1),
        retract(pq(_)), assert(pq(Lst1)).

insert12pq(El,[], [El]):-!.
insert12pq(El, L1, L2):-L1=[H|_], El=node(_,V1), H=node(_,V2),
                not(V1 > V2), L2 = [El|L1], !.
insert12pq(El, L1, L2):-L1=[H|T], insert12pq(El, T, Lx), L2 = [H|Lx].
---------
```

Changed:

```
insrt_to_pq(X,I1,I,FNV):- pq(Lst), insert12pq(node(X,I1,I,FNV),Lst,Lst1),
        retract(pq(_)),   assert(pq(Lst1)).
```

… El=node(_,_,_,V1), H=node(_,_,_,V2)…

```
% Updating Possible Path
update_pp(N):- retract(pp(_)), assert(pp([])), renew_pp(N).
renew_pp(N):-t_node(N,nil), pp(X), append([N],X,X1),
        retract(pp(_)), assert(pp(X1)), !.
renew_pp(N):- pp(X), append([N],X,X1), retract(pp(_)), assert(pp(X1)),
        t_node(N,N1), renew_pp(N1).
---------
```

Changed:… (N,I)… t_node(N,I,nil,_) … t_node(N,I,I1,_),t_node(N1,I1,_,_), renew_pp(N1,I1).

```
% Finding 'shortest' path length
find_path_length(L):-pp(Lst),path_sum(Lst,L).
path_sum(Lst,0):- Lst=[g|_],!.
path_sum(Lst,L):-Lst=[N|T],T=[N1|_], neighbor(N,N1,D), path_sum(T,L1),L is L1+D.
---------

% Displaying 'shortest' path and its length
display_result(L):- pp(Lst), write('Solution:'), write(Lst),nl,
        write('Length:'), write(L).
---------
```

B. Utilities for Prolog code

```
%Arrange a menu of actions
start:- repeat,
        write('\n1. Clear database'),
        write('\n2. Execute GBFS'),
        write('\n3. Display database'),
        write('\n4. Save database'),
        write('\n5. Exit'),
        write('\n\nEnter your choice: '),
        read(N), N >0, N < 6,
        do(N), N=5,!.
```

```
do(1):-write('Done 1').
do(2):- write('Done 2').
do(3):- write('Done 3').
do(4):- write('Done 2').
do(5):- abort.
```

```
% List dynamic data
list_records:-listing(t_node), listing(pq), listing(pp).

% Save file with modified records in place of old ones.
save_records:-tell('gbfs_db.pl'), listing(t_node), listing(pq), listing(pp),told.
---------
Changed: …'astars_db.pl' …

%Clear the database
clr_db:-retractall(t_node(_,_)), retractall(pp(_)), retractall(pq(_)).
---------
Changed: …t_node(_,_,_,_)
Added:retractall(t_n_indx(_))
```

C. Utilities for Python code

 # Writing to and reading from a file in Python

```
f1=open(fn, "w")
print("\n")
for i in range(ln):
    name=str(input("Enter the name:"))
    dept=str(input("Enter the department:"))
    cgpa=str(input("Enter the cgpa:"))
    std=name+"\t"+dept+"\t"+cgpa
    print(std, end="\n", file=f1)
    print("\n")
f1.close
```

```
f1=open(fn, "r")
for l in f1:
    name, dept, cgpa =l.split("\t")
    print(name, dept, float(cgpa), end="\n")
f1.close
```

 # Including files

```
imports3Module1 as m1

m1.display_file_lines(fn,ln)

n=m1.num_of_lines(fn)
print("Number of lines in {} is {}.".format(fn,n))

m1.display_file(fn)
```

Supplementary Material for Session 4

I. Major Components of Prolog Code for Hill-Climbing Local Search and Genetic Algorithms:

% Code for generating the successors of a 8-queens' state given as a list of 8 digits

```
gnrt_sucsr(L):- assert(id(1)), assert(state(1, 'c', L, 50)),
                incr_id, mk_new(1, L), retract(id(_)), evaluate.

incr_id:-id(V), V1 is V+1, retract(id(_)), assert(id(V1)).

mk_new(9, _):-!.
mk_new(N, L):- nthel(N, L, X), del_el(X, [1,2,3,4,5,6,7,8], L1),
                cng_mk(N, L, L1), N1 is N+1, mk_new(N1, L).

cng_mk(_, _, []):-!.
cng_mk(N, L, L1):- L1=[H|T], rplc_nthel(N, H, L, L2), id(Id),    /* id/1 is a dynamic data */
                assert(state(Id, 's', L2, 50)), incr_id, cng_mk(N, L, T).
```

% Code for determination and display of the best state

```
checkall:- state(_, 'c', _, V1), threshold(V2), V1 >= V2, I is 1, dsply(I), !.
checkall:- best(I1,V1), threshold(V2), V1 >= V2, I is I1, dsply(I), !.
checkall:- state(_, 'c', _, V1), best(I, V2) ,V2>V1,state(I, _, L, _),
                retractall(state(_, _, _, _)),write_list(['\nIteration max: ', V2]),
                gnrt_sucsr(L), !.
checkall:- restrt, !.

best(I, Max):- state(_, 's', _, Val), assert(max_val(Val)),
                updt_max, max_val(Max), state(I, _, _, Max), retract(max_val(_)), !.

updt_max:- state(_, _, _, V2),  max_val(V1), V2>V1,
                retract(max_val(_)), assert(max_val(V2)), fail.
updt_max:-!.
```

% Performing Crossover

```
go_cross(X,Y,CP):- state(X,'p',L1,_), state(Y,'p',L2,_),CP1 is 8-CP,
        del_1st_n_el(L1,CP,L12),del_last_n_el(L1,CP1,L11),
        del_1st_n_el(L2,CP,L22),del_last_n_el(L2,CP1,L21),
        append(L11,L22,LO1),append(L21,L12,LO2), count_sts(_,N),
        N1 is N+1, N2 is N+2,
        assert(state(N1,'o',LO1,50)), assert(state(N2,'o',LO2,50)).
```

% Performing Mutation

```
do_mutn:- count_sts('o',N), N1 is random(N)+1,
        assert(id1(0)),get_offspr(N1,I,T,L,V), retract(id1(_)),
        N2 is random(8)+1, N3 is random(8)+1, rplc_nthel(N2,N3,L,L1),
        retract(state(I,T,L,V)), assert(state(I,T,L1,50)).

get_offspr(N1,I,'o',L,V):- state(I,'o',L,V),incr_id1, id1(N), N1=N,!.
```

II. Sample Codes for Object Oriented Programming with Python

1. Simple example of Class, Objects and Inheritance

```
class Animal:
    def __init__(self, m):
        self.movement = m
    def printAnimal(self):
        print("Movement: "+self.movement)

class Mammal(Animal):
    def __init__(mml, wb, m):
        Animal.__init__(mml, m)
        mml.warm_blooded= wb
    def printMammal(self):
        self.printAnimal()
        print("Warm blooded: "+self.warm_blooded)

class Cat(Mammal):
    def __init__(kt, c, nol, wb, m):
        Mammal.__init__(kt, wb, m)
        kt.color = c
        kt.no_of_legs= nol
    def printCat(kt):
        kt.printMammal()
        print("Color: "+kt.color+"\n"+
            "Number of Legs: "+str(kt.no_of_legs))

C1=Cat("White", 4, "Yes", "Yes")
print("\nDetailed Information of Cat:\n")
```

```
C1.printCat()

M1=Mammal("Yes", "Yes")
print("\nDetailed Information of Mammal:\n")
M1.printMammal()
```

2. Another simple example with Inheritance and Overloading

```
class Calculation:
    def calcVolume(self,arg1,arg2=None, arg3=None):
        if arg2 != None:
            return arg1*arg2*arg3
        else:
            return 4*3.14*arg1**3/3

class Sphere(Calculation):
    def __init__(sphr, r):
        sphr.baseRadius = r

    def displaySphere(self):
        print("Sphere volume: ", end='')
        print(self.calcVolume(self.baseRadius))

class Cube(Calculation):
    def __init__(cb, l, w, h):
        cb.length = l
        cb.width = w
        cb.height = h

    def displayCube(c):
        print("Cube volume: ", end='')
        print (c.calcVolume(c.length,
                    c.width,c.height))

S1=Sphere(float(input("\nSpere Radious:")))
print("\nSphere Volume Calculation")
S1.displaySphere()

C1=Cube(float(input("\nCube length:")),float(input("Cube width:")),
     float(input("Cube height:")))
print("\nCube Volume Calculation")
C1.displayCube()
```

III. Sample Code for a Rule based System in Prolog

% Rules:

```
hypothesis(Patient, flu):-
    symptom(Patient, headache), symptom(Patient, fever),
    symptom(Patient, runny_nose).
```

hypothesis(Patient, common_cold):-
symptom(Patient, sneezing),
symptom(Patient, runny_nose).

% Facts / Data:

symptom('Rahim', headache).
symptom('Karim', headache).
symptom('Hasib', headache).
symptom('Karim', fever).
symptom('Hasib', fever).
symptom('Hasib', sneezing).
symptom('Rahim', sneezing).
symptom('Karim', runny_nose).
symptom('Rahim', runny_nose).

Supplementary Material for Session 5

I. Some important Python libraries and packages for Machine Learning

In Python, there are many libraries and packages that make Machine Learning easier. Some of them are as below:

Libraries	Short description	Sample functions	command for installation
Numpy	Numerical Python; for working with arrays, also for linear algebra, fourier transform, and matrices; much faster than lists	numpy.sin(input_array), numpy.divide(arr1, arr2), numpy.convolve(arr1, arr2, mode)	pip install numpy
Scikit-learn	Tools for data analysis and data mining; depends on NumPy	train_test_split(X,Y,test_size=1/3), KNeighborsClassifier(n_neighbors=7). fit(X_train,Y_train)	pip install scikit-learn
Pandas	Provides data structures and operations for numerical tables and time series	pandas.read_csv(filename), datafile.head(n), datafile.tail(n)	pip install pandas
Matplotlib	For static, animated, and interactive visualizations	matplotlib.pyplot.scatter(X, Y, color='RED')	pip install matplotlib

The Python Package Index (PyPI) is a repository of software for the Python programming language. pip (acronym for "**Pip** Installs Packages") is the package management system used to install and manage software packages written in Python.

II. Reading and exploring a CSV file

```
# Import the necessary libraries
import matplotlib.pyplot as plot
import pandas
```

```
# Import the dataset
dataset = pandas.read_csv('salaryData.csv')

# Explore the dataset
print(dataset.shape) # number of rows and columns
print(dataset.head(5)) # display first five rows of the dataset

# Show the column heads
for col in dataset.columns:
    print(col)

# Capture specific attribute values, including values in the target
column
x = dataset['YearsExperience'].values
y = dataset['Salary'].values

# print(x.shape) # shape of x
# print(y.shape) # shape of y

X = x.reshape(len(x),1)
Y = y.reshape(len(y),1)
# print(X.shape) # shape of X
# print(Y.shape) # shape of Y

# Plot the data
plot.scatter(X, Y,  color='RED')
plot.show()
```

III. Example of Linear Regression using Python

```
import numpy
import matplotlib.pyplot as plot
import pandas
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataset=pandas.read_csv('salaryData.csv')

print(dataset.shape)
print(dataset.head(5))

x=dataset['YearsExperience'].values
y=dataset['Salary'].values

X = x.reshape(len(x),1)
Y = y.reshape(len(y),1)

xTrain,xTest,yTrain,yTest=train_test_split(X,Y,test_size=1/3)

# Creating a Linear Regression object and fitting it on our training set
linearRegressor=LinearRegression()
linearRegressor.fit(xTrain,yTrain)
```

```
# Predicting the test set results
yPrediction=linearRegressor.predict(xTest)

# Visualization
df=pandas.DataFrame({'Actual':yTest.flatten(),'Predicted':yPrediction.fla
tten()})
print(df)

df.plot(kind='bar')
plot.show()

plot.scatter(xTest,yTest,color='green')
plot.plot(xTest,yPrediction,color='red',linewidth=2)
plot.show()

# Displaying errors
print('Mean Absolute
Error:',metrics.mean_absolute_error(yTest,yPrediction))
print('Mean Squared
Error:',metrics.mean_squared_error(yTest,yPrediction))
print('Root Mean Squared Error:',
     numpy.sqrt(metrics.mean_squared_error(yTest,yPrediction)))
```

IV. Example of k-Nearest Neighbor classifier using Scikit Learn

```
from sklearn import datasets
from sklearn.metrics import confusion_matrix
froms klearn.model_selection import train_test_split

# Loading the iris dataset
iris=datasets.load_iris()


# X -> features, y -> label
X=iris.data
y=iris.target


print(X)
print(y)

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.67)

# Training a KNN classifier
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=7).fit(X_train,y_train)

# Accuracy on X_test
accuracy=knn.score(X_test,y_test)
print(accuracy)


# Displaying Predictions
knn_predictions=knn.predict(X_test)
```

```
df = pandas.DataFrame({'Actual': y_test.flatten(), 'Predicted':
knn_predictions.flatten()})
print(df)

print(knn_predictions)
print(y_test)

# Creating a confusion matrix
cm=confusion_matrix(y_test,knn_predictions)
print(cm)
```

V. Decision Tree

Example of Decision Tree Classifier

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from pandas import DataFrame
from sklearn import tree
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data
Y = iris.target

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3)

# Training a decision tree classifier
model = tree.DecisionTreeClassifier()
model.fit(X_train, Y_train)

# Testing
model_predictions = model.predict(X_test)
print('Testing Data Targets:', end='\n')
print(Y_test)
print('Predicted Data Targets:', end='\n')
print(model_predictions)


# Accuracy of prediction
accuracyScore = accuracy_score(Y_test, model_predictions)
print(accuracyScore)
```

Regression with Decision Tree

```
import pandas
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# The dataset
dataset=pandas.read_csv('salaryData.csv')
x=dataset['YearsExperience'].values
```

```
y=dataset['Salary'].values
X=x.reshape(len(x),1)
Y=y.reshape(len(y),1)
xTrain,xTest,yTrain,yTest=train_test_split(X,Y,test_size=1/3)

# Creating a Decision Tree Regressor
regressor=DecisionTreeRegressor()
regressor.fit(xTrain,yTrain)


# Predicting the test set results
yPrediction=regressor.predict(xTest)

#  Displaing input, output and errors
print('Testing Data Dependant Values:', end='\n')
print(yTest)
print('Predicted values:', end='\n')
print(yPrediction)
print('Mean Absolute
Error:',metrics.mean_absolute_error(yTest,yPrediction))
```

VI. Naïve Bayes Classifier

Example of Naïve Bayes Classifier

```
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from pandas import DataFrame
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# The dataset
iris = datasets.load_iris()
X = iris.data
Y = iris.target
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3)

# Training a Gaussian Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, Y_train)

# Predictions
model_predictions = model.predict(X_test)
print(model_predictions)
print(Y_test)

# Accuracy of prediction
accuracyScore = accuracy_score(Y_test, model_predictions)
print(accuracyScore)

# Creating a confusion matrix
cm=confusion_matrix(Y_test,model_predictions)
print(cm)
```

VII. Neural Network Models

Classification with Neural Network

```
from sklearn import datasets
import pandas

# The dataset
iris = datasets.load_iris()
X = iris.data
Y = iris.target

# Working with data
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

print('Scalled training data:', end='\n')
print(X_train)
print('Scalled testing data:', end='\n')
print(X_test)

# Training the classifier
#import warnings
#warnings.simplefilter("ignore")

from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier()
mlp.fit(X_train,Y_train)

# Predictions
predictions = mlp.predict(X_test)
print(predictions)
print(Y_test)

# Accuracy
from sklearn.metrics import accuracy_score
accuracyScore = accuracy_score(Y_test, predictions)
print(accuracyScore)
```

VIII. k-means Clustering using Scikit-Learn

Simple Example

```
from pandas import DataFrame
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

```
Data = {'x':
[25,34,22,27,33,33,31,22,35,34,67,54,57,43,50,57,59,52,65,47,49,48,35,33,
44,45,38,43,51,46],
'y':
[79,51,53,78,59,74,73,57,69,75,51,32,40,47,53,36,35,58,59,50,25,20,14,12,
20,5,29,27,8,7]
        }

df = DataFrame(Data,columns=['x','y'])

plt.title("Data before clustering")
plt.scatter(df['x'], df['y'])
plt.show()

cn =int(input("How many clusters? :"))
kmeans = KMeans(n_clusters=cn).fit(df)
centroids = kmeans.cluster_centers_
print(centroids)

plt.title("Data after clustering")
plt.scatter(df['x'], df['y'], c= kmeans.labels_)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red')
plt.show()
```

Upgraded Example

```
import matplotlib.pyplot as plt
import pandas
from sklearn.cluster import KMeans
from pandas import DataFrame

dataset = pandas.read_csv('ClusteringData.csv')
print(dataset.shape) # number of rows and columns
for col in dataset.columns:
    print(col)

x = dataset['Quiz'].values
y = dataset['Final Exam'].values
print(x)
print(y)
for i in range(len(y)):
    if(y[i]=='Abs'):
        y[i]= 0.0
    else:
        y[i] = float(y[i])
print(x)
print(y)

plt.title("Data before clustering")
plt.xlabel('Class Test Marks')
plt.ylabel('Final Exam Marks')
plt.scatter(x, y, color='BLUE')
plt.show()

cn = int(input("How many clusters? :"))
```

```
table={'X':x,'Y':y}
data=DataFrame(table,columns=['X','Y'])
kmeans = KMeans(n_clusters=cn).fit(data)
centroids = kmeans.cluster_centers_
print(centroids)

plt.title("Data after clustering")
plt.xlabel('Class Test Marks')
plt.ylabel('Final Exam Marks')
plt.scatter(data['X'], data['Y'], c= kmeans.labels_)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red')
plt.show()
```

IX. Cross Validation

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
import warnings
warnings.simplefilter("ignore")

iris_data = load_iris()

x = iris_data.data
y = iris_data.target

kf = KFold(n_splits=5)

print("\nK nearest neighbor:")
i=1
for train_index, test_index in kf.split(x):
    x_train, x_test = x[train_index], x[test_index]
    #print('Fold No.:', i,'**********')
    #print(x_train)
    #print(x_test)
    #i=i+1
    y_train, y_test = y[train_index], y[test_index]
    knn = KNeighborsClassifier()
    knn.fit(x_train,y_train)
    prediction = knn.predict(x_test)
    print(accuracy_score(y_test, prediction))
```

MID TERM EXAMINATION

There will be a 40-minutes written mid-semester examination on the materials covered in the sessions conducted during the first half of the semester.

TERM FINAL EXAMINATION

There will be a 40-minutes written end-of-semester examination on the materials covered in the sessions conducted during the second half of the semester.

END