Ahsanullah University of Science and Technology (AUST)

Department of Computer Science and Engineering

LABORATORY MANUAL

Course No. : CSE3208
Course Title: Introduction to Artificial Intelligence Lab

For the students of 3rd Year, 2nd semester of
B.Sc. in Computer Science and Engineering program

Table of Contents

Course Outcomes

Course Outcomes for the course are -

1. Explain the fundamental topics of Artificial Intelligence
2. Implement proven Artificial Intelligence techniques
3. Analyze AI problems for alternative solutions
4. Evaluate related AI approaches to efficient solution

Preferred Tools

1. SWI-Prolog
2. Python IDLE
3. Jupyter Notebook
4. Google Colab

Reference Books

1. *"Artificial Intelligence - A Modern Approach"* by S. J. Russell & P. Norvig, 4th Edition
2. *"Data Mining: Concepts and Techniques"* by Jiawei Han, Micheline Kamber and Jian Pei, 3rd Edition

Administrative Policy of the Laboratory

1. Class assessment tasks must be performed by students individually, without help of others.
2. Viva for each program will be taken and considered as a performance.
3. Plagiarism is strictly forbidden.

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.
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).
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)

# 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
```

1	2	3
8		4
7	6	5

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.
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

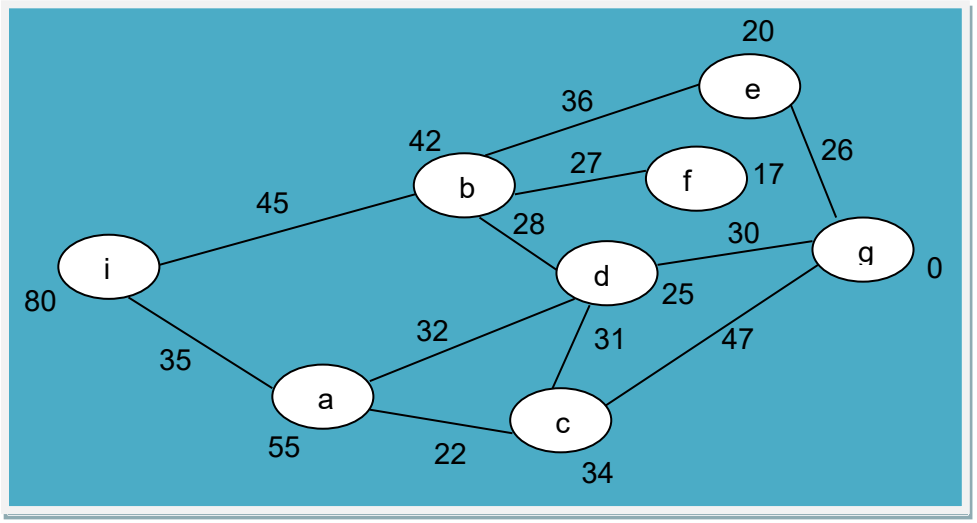
I. OBJECTIVES

- 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

A. Greedy Best-First Search

Consider a problem instance given in the following graph.



Nod	Neighbo	Distanc
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]).
```

B. **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

5) Explore thoroughly the supplementary material provided for this session.
6) Run and analyze the codes demonstrated in this session.
7) 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.
8) 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

A. 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) **First-Choice Hill- climbing:** Choose the first randomly generated successor that is better than the parent.

B. 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 also 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.
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 Python.

Session 5 & 6: Classification and Learning

I. OBJECTIVES

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

II. DEMONSTRATION OF USEFUL RESOURCES

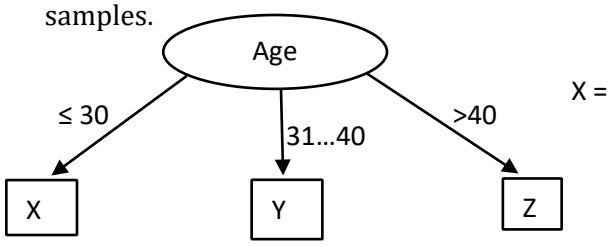
Machine Learning is an application of artificial intelligence that provides systems the ability to improve from experience.

- ✔ 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 to 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 Forest, 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.

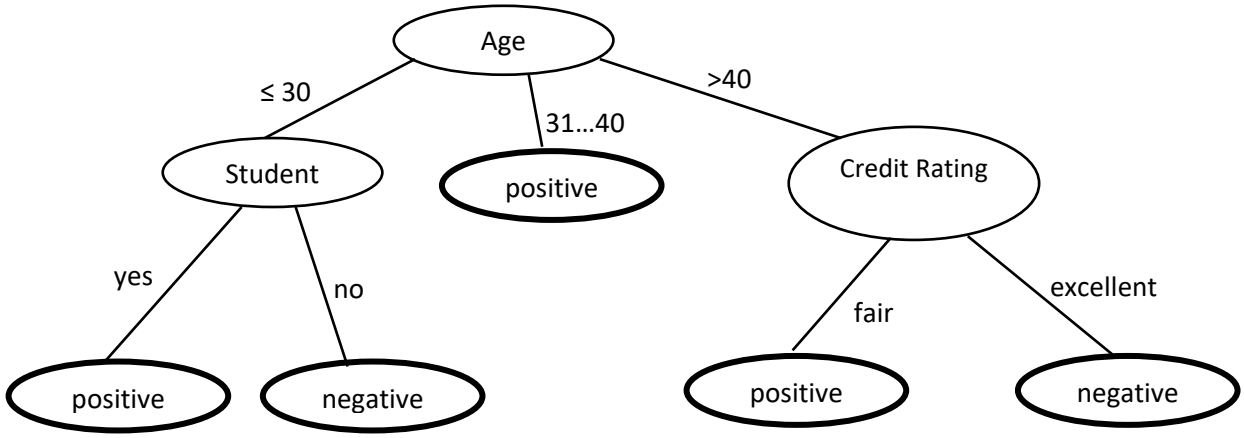
1) Learning Decision Trees

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

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



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').

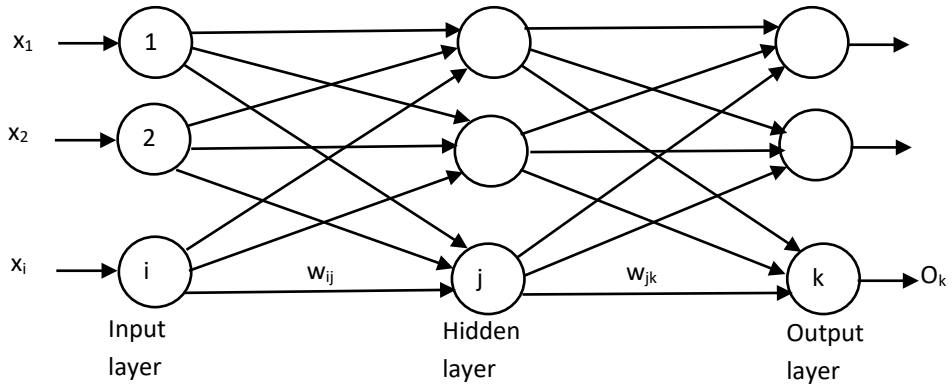
2) Naïve Bayes Classifier

- ✔ We take the same data set and apply the following simplified forms of Bayes' theorem.
- ✔ For an unknown sample, $X = (x_1, x_2, \dots, x_n)$, classifier should predict that X belongs to one of m classes, C_i with highest posterior probability
  $P(C_i \mid X) > P(C_j \mid X)$, $1 \le j \le m$ & $j \ne i$. [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(age = '<=30' | C_1) = 2/9 = 0.222
  ….
- ✔ We obtain,
  $P(X \mid C_1) P(C_1) = 0.044 \times 0.643 =$ **0.028**
  $P(X \mid C_2) P(C_2) = 0.019 \times 0.357 = 0.007$

- 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, … , 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 $x_1, x_2, \dots, x_s$, y' - of $y_1, y_2, \dots, y_s$, given sample data points $(x_1, y_1), (x_2, y_2), \dots, (x_s, y_s)$.
- 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

III. LAB EXERCISE

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

Lab Examination

There will be a 20 marks examination containing different AI problems, simulations, coding, multiple-choice question etc.