1. **A\* search algorithm pip install heuristicsearch**

from heuristicsearch.a\_star\_search import AStar

graph\_nodes = { 'A': [('B', 1), ('C', 3), ('D', 7)], 'B': [('D', 5)], 'C': [('D', 12)] } heuristics = {'A':1, 'B':1, 'C':1, 'D':1}

graph= AStar(graph\_nodes,heuristics) graph.apply\_a\_star(start='A',stop='D')

1. **AO\* search algorithm from heuristicsearch.ao\_star import AOStar**

print("Graph - 1")

heuristic = { 'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7 }

graph\_nodes = { 'A': [[('B', 1), ('C', 1)], [('D', 1)]], 'B': [[('G', 1)], [('H', 1)]],

# 'C': [[('J', 1)]], 'D': [[('E', 1), ('F', 1)]], # 'G': [[('I', 1)]] }

graph = AOStar(graph\_nodes, heuristic, 'A') graph.applyAOStar()

## 3. CandidateEliminationLab3

import csv

with open("trainingexamples.csv") as f: csv\_file = csv.reader(f)

data = list(csv\_file) specific = data[1][:-1]

general = [['?' for i in range(len(specific))] for j in range(len(specific))]

for i in data:

if i[-1] == "Yes":

for j in range(len(specific)): if i[j] != specific[j]:

specific[j] = "?"

general[j][j] = "?"

elif i[-1] == "No":

for j in range(len(specific)): if i[j] != specific[j]:

general[j][j] = specific[j] else:

general[j][j] = "?"

print("\nStep " + str(data.index(i)+1) + " of Candidate Elimination Algorithm") print(specific)

print(general)

gh = [] # gh = general Hypothesis for i in general:

for j in i:

if j != '?': gh.append(i)

break

print("\nFinal Specific hypothesis:\n", specific)

print("\nFinal General hypothesis:\n", gh)

## Lab 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)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) # maximum of X array longitudinally y = y/100

#Sigmoid Function def sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function def derivatives\_sigmoid(x):

return x \* (1 - x)

#Variable initialization

epoch=5000 #Setting training iterations lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set hiddenlayer\_neurons = 3 #number of hidden layers neurons output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons)) bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons)) bout=np.random.uniform(size=(1,output\_neurons))

#draws a random range of numbers uniformly of dim x\*y for i in range(epoch):

#Forward Propogation hinp1=np.dot(X,wh) hinp=hinp1 + bh hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout) outinp= outinp1+ bout

output = sigmoid(outinp)

#Backpropagation EO = y-output

outgrad = derivatives\_sigmoid(output) d\_output = EO\* outgrad

EH = d\_output.dot(wout.T)

#how much hidden layer wts contributed to error hiddengrad = derivatives\_sigmoid(hlayer\_act) d\_hiddenlayer = EH \* hiddengrad

# dotproduct of nextlayererror and currentlayerop wout += hlayer\_act.T.dot(d\_output) \*lr

wh += X.T.dot(d\_hiddenlayer) \*lr

print("Input: \n" + str(X)) print("Actual Output: \n" + str(y)) print("Predicted Output: \n" ,output)

# Lab: 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.

import matplotlib.pyplot as plt from sklearn import datasets

from sklearn.cluster import KMeans import pandas as pd

import numpy as np

# import some data to play with iris = datasets.load\_iris()

X = pd.DataFrame(iris.data)

X.columns = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width'] y = pd.DataFrame(iris.target)

y.columns = ['Targets']

# Build the K Means Model model = KMeans(n\_clusters=3)

model.fit(X) # model.labels\_ : Gives cluster no for which samples belongs to

# # Visualise the clustering results plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications using Petal features plt.subplot(1, 3, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40) plt.title('Real Clusters')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

# Plot the Models Classifications plt.subplot(1, 3, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40) plt.title('K-Means Clustering')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

# General EM for GMM

from sklearn import preprocessing

# transform your data such that its distribution will have a # mean value 0 and standard deviation of 1.

scaler = preprocessing.StandardScaler() scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns) from sklearn.mixture import GaussianMixture gmm = GaussianMixture(n\_components=40)

gmm.fit(xs) plt.subplot(1, 3, 3)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[0], s=40) plt.title('GMM Clustering')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')

# Lab 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.model\_selection import train\_test\_split from sklearn.neighbors import KNeighborsClassifier from sklearn import datasets

iris=datasets.load\_iris() print("Iris Data set loaded...")

x\_train, x\_test, y\_train, y\_test = train\_test\_split(iris.data,iris.target,test\_size=0.1) #random\_state=0

for i in range(len(iris.target\_names)): print("Label", i , "-",str(iris.target\_names[i]))

classifier = KNeighborsClassifier(n\_neighbors=2) classifier.fit(x\_train, y\_train) y\_pred=classifier.predict(x\_test)

print("Results of Classification using K-nn with K=1 ") for r in range(0,len(x\_test)):

print(" Sample:", str(x\_test[r]), " Actual-label:", str(y\_test[r])," Predicted-label:", str(y\_pred[r]))

print("Classification Accuracy :" , classifier.score(x\_test,y\_test));

# Lab 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 numpy as np

import matplotlib.pyplot as plt

def local\_regression(x0, X, Y, tau): x0 = [1, x0]

X = [[1, i] for i in X] X = np.asarray(X)

xw = (X.T) \* np.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau)) beta = np.linalg.pinv(xw @ X) @ xw @ Y @ x0

return beta

def draw(tau):

prediction = [local\_regression(x0, X, Y, tau) for x0 in domain] plt.plot(X, Y, 'o', color='black')

plt.plot(domain, prediction, color='red') plt.show()

X = np.linspace(-3, 3, num=1000) domain = X

Y = np.log(np.abs(X \*\* 2 - 1) + .5)

draw(10) draw(0.1) draw(0.01) draw(0.001)

lab 4 and 6 is hard