# THE OXFORD COLLEGE OF ENGINEERING

#### BOMMANAHALLI, HOSUR ROAD, BENGALURU-560068.

**(Affiliated to Visvesvaraya Technological University, Belgaum)**

**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING**



**LAB MANUAL**

**Subject Name : Artificial Intelligence and Machine Learning Lab Subject Code : 18CSL76**

**Semester : VII Academic Year: 2023-24**

**Prepared by**

**Ms.VISALINI S**

**Ms.C A BINDYASHREE**

### Table of Contents

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sl.**  **No.** | **Content** | | | | | |
| 1 | AIML Lab Syllabus | | | | | |
| 2 | CO-PO, PSO Mapping | | | | | |
| 3 | Anaconda Installation Steps | | | | | |
| 4 | **Program 1:** Implement A\* Search algorithm. | | | | | |
| 5 | **Program 2:** Implement AO\* Search algorithm. | | | | | |
| 6 | **Program 3:**  algorithm. | Implement | and | demonstrate | the | Candidate-Elimination |
| 7 | **Program 4:** Program to demonstrate the working of the decision tree based ID3 algorithm. | | | | | |
| 8 | **Program 5:** Build an Artificial Neural Network by implementing the Backpropagation algorithm. | | | | | |
| 9 | **Program 6:** Program to implement the naïve Bayesian classifier. | | | | | |
| 10 | **Program 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. | | | | | |
| 11 | **Program 8:** Program to implement k-Nearest Neighbor algorithm | | | | | |
| 12 | **Program 9:** Implement the non-parametric Locally Weighted Regression algorithm. | | | | | |
| 13 | Viva Questions | | | | | |

Children’s Education Society ®

### DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

**THE OXFORD COLLEGE OF ENGINEERING**

Hosur Road, Bommanahalli, Bengaluru-560 068

Website**:**[www.theoxford.edu](http://www.theoxford.edu/) Email : [enghodcse@theoxford.edu](mailto:enghodcse@theoxford.edu)

**(Approved by AICTE, New Delhi, Accredited by NBA, NAAC, New Delhi & Affiliated to VTU, Belgaum)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 algorithmto 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 toclassify a new sample. | | | |
| 5. | Build an Artificial Neural Network by implementing the Backpropagation 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 Regressionalgorithm 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)*  1. For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 =   100 Marks   1. For laboratories having PART A and PART B    1. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks    2. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks | | | | |



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

THE OXFORD COLLEGE OF ENGINEERING

Hosur Road, Bommanahalli, Bengaluru-560 068

Website**:**[www.theoxford.edu](http://www.theoxford.edu/) Email : [engprincipal@theoxford.edu](mailto:engprincipal@theoxfor.edu)

**(Approved by AICTE, New Delhi, Accredited by NBA, New Delhi & Affiliated to VTU, Belgaum)**

**Subject: Artificial Intelligence and Machine Learning Laboratory Sub Code:18CSL76**

**Course Outcomes (COs)**

**C416.1:** Understand the implementation procedures for the Artificial Intelligence and Machine Learning algorithms.

**C416.2:** Design and evaluate Python programs for various Learning algorithms.

**C416.3:** Apply appropriate data sets to the Artificial Intelligence and Machine Learning algorithms.

**C416.4:** Identify and apply Artificial Intelligence and Machine Learning algorithms to solve real world problems.

### CO-PO Mapping

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PO CO** | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **P8** | **P9** | **P10** | **P11** | **P12** |
| **C416.1** | 2 | 1 | 3 | 3 | 2 | - | - | - | - | - | - | 1 |
| **C416.2** | 2 | 1 | 3 | 3 | 2 | - | - | - | - | - | - | 1 |
| **C416.3** | 2 | 1 | 3 | 3 | 2 | - | - | - | - | - | - | 1 |
| **C416.4** | 2 | 1 | 3 | 3 | 2 | - | - | - | - | - | - | 1 |

**CO-PSO Mapping**

|  |  |  |
| --- | --- | --- |
| **CO** | **PSO1** | **PSO2** |
| **C416.1** | 3 | 2 |
| **C416.2** | 3 | 2 |
| **C416.3** | 3 | 2 |
| **C416.4** | 3 | 2 |

### Installing Anaconda on Windows

This tutorial will demonstrate how you can install Anaconda, a powerful package manager, on Microsoft Windows.

Anaconda is a package manager, an environment manager, and Python distribution that contains a collection of many open source packages. This is advantageous as when you are working on a data science project, you will find that you need many different packages (numpy, scikit-learn, scipy, pandas to name a few), which an installation of Anaconda comes preinstalled with. If you need additional packages after installing Anaconda, you can use Anaconda's package manager, conda, or pip to install those packages. This is highly advantageous as you don't have to manage dependencies between multiple packages yourself. Conda even makes it easy to switch between Python 2 and 3 (you can learn more [here](https://towardsdatascience.com/environment-management-with-conda-python-2-3-b9961a8a5097)). In fact, an installation of Anaconda is also

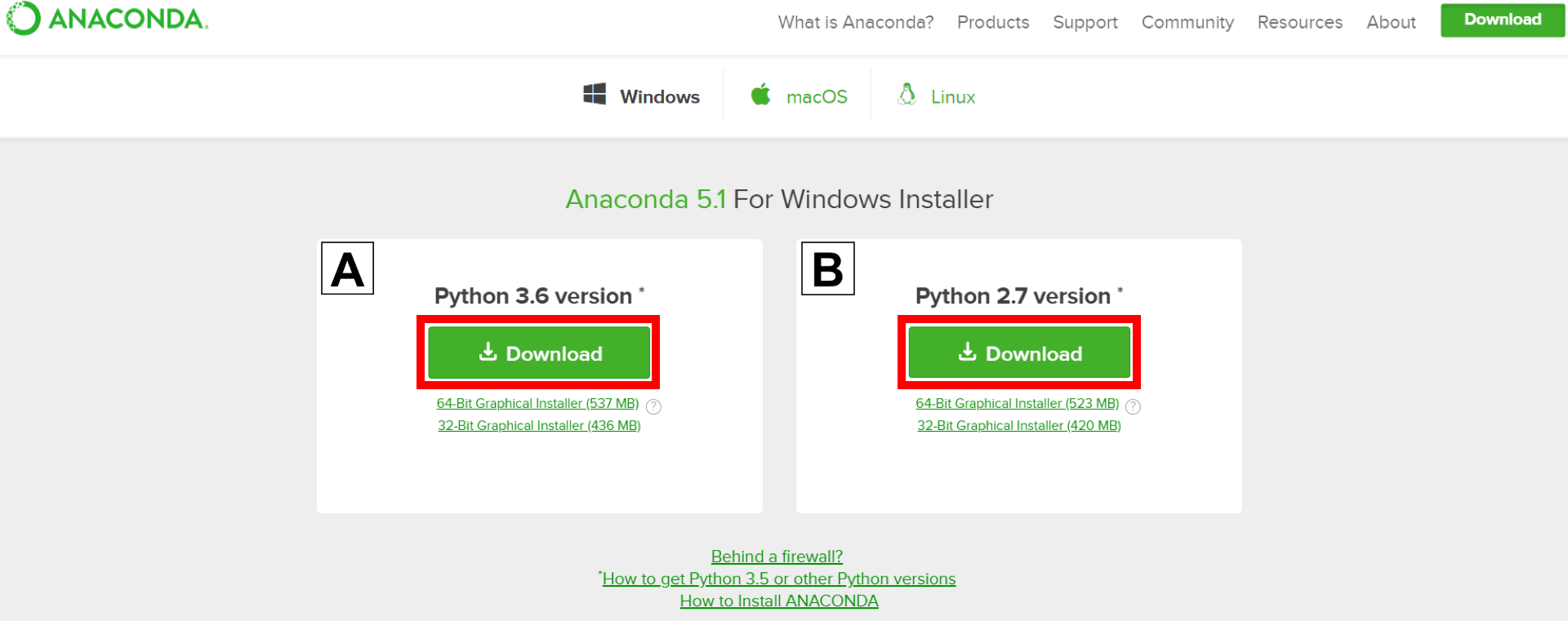
the [recommended way to install Jupyter Notebooks](http://jupyter.org/install.html) which you can learn more about [here](https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook) on the DataCamp community.

This tutorial will include:

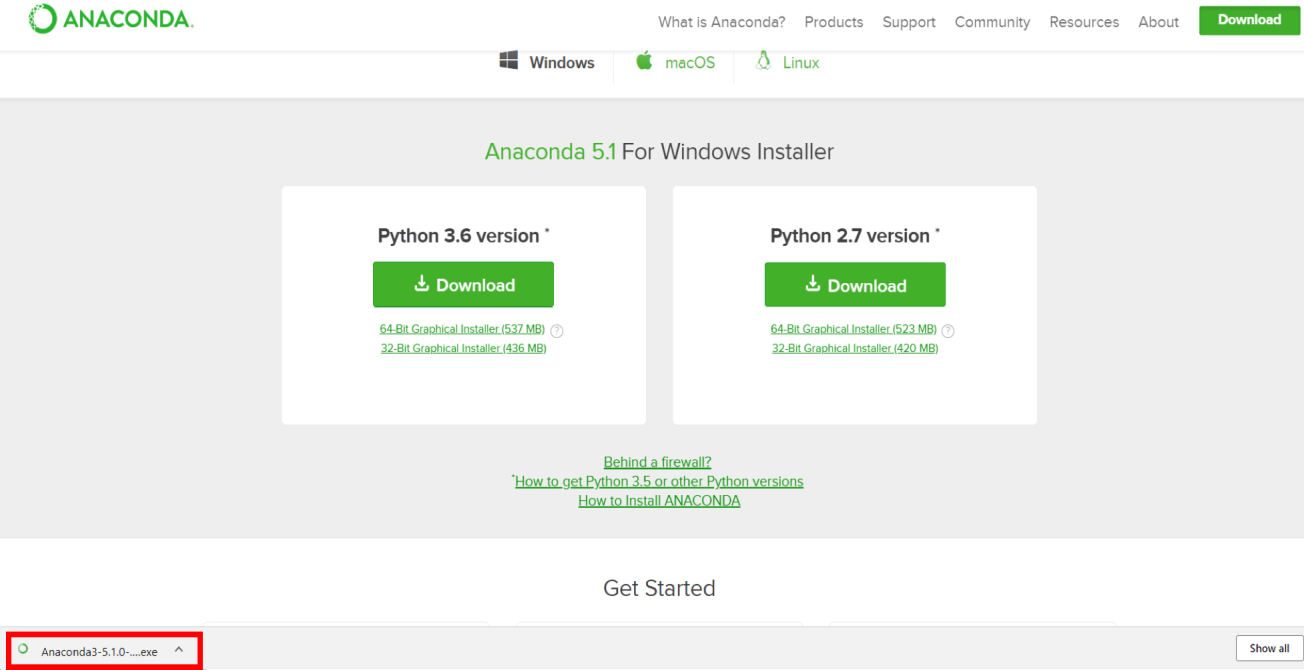
* [How to Install Anaconda on Windows](https://www.datacamp.com/community/tutorials/installing-anaconda-windows#install)
* [How to test your installation and fix common installation issues](https://www.datacamp.com/community/tutorials/installing-anaconda-windows#test)
* [What to do after installing Anaconda.](https://www.datacamp.com/community/tutorials/installing-anaconda-windows#after) With that, let's get started!

**Download and Install Anaconda**

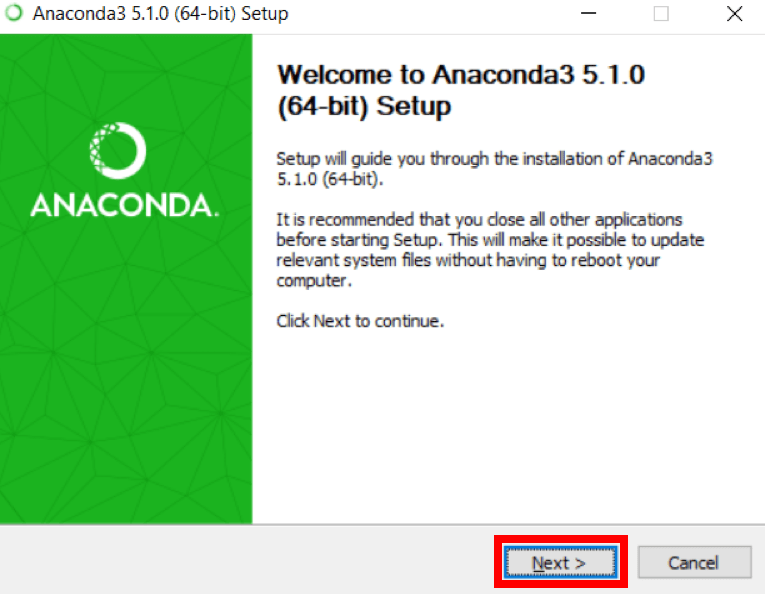
1. Go to the [Anaconda Website](https://www.anaconda.com/download/#windows) and choose a Python 3.x graphical installer (A) or a Python 2.x graphical installer (B). If you aren't sure which Python version you want to install, choose Python 3. Do not choose both.



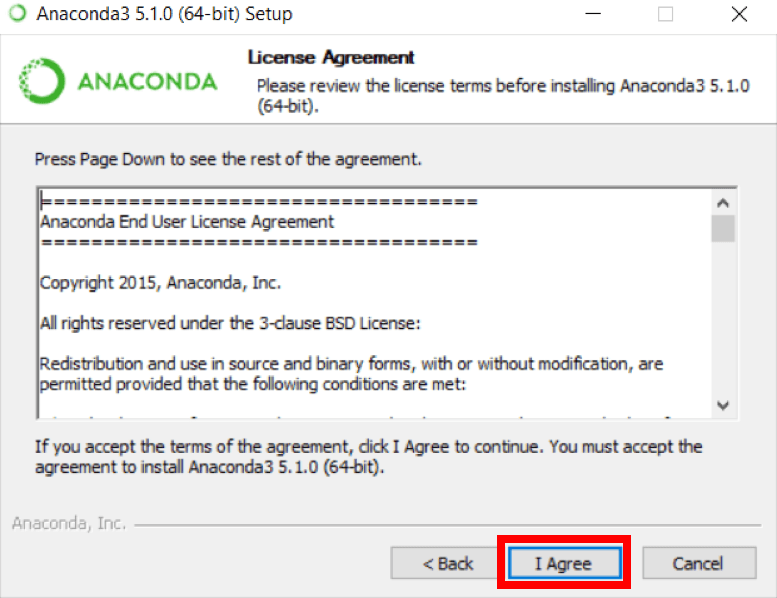
## Locate your download and double click it.



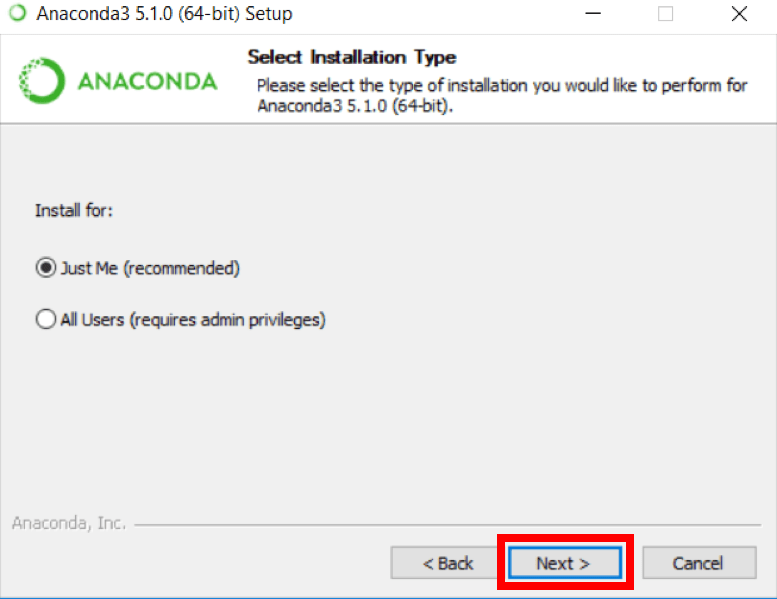
When the screen below appears, click on Next.

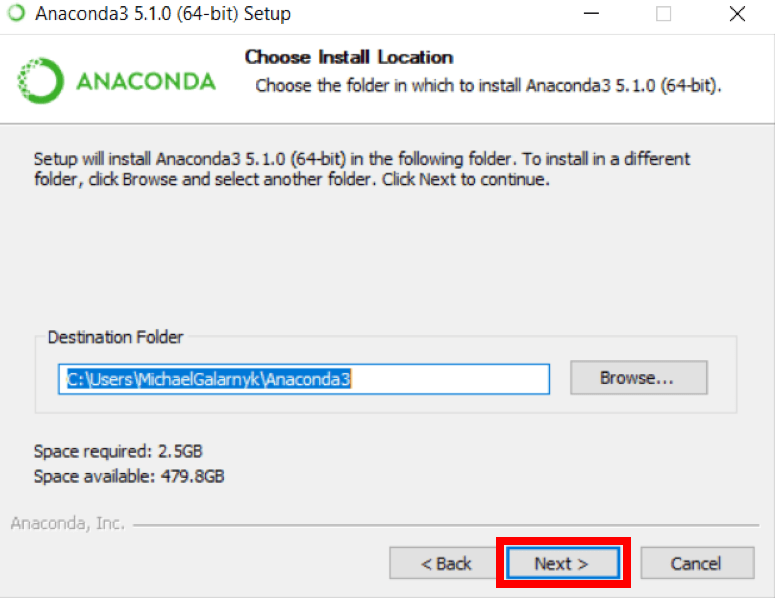


1. Read the license agreement and click on I Agree.



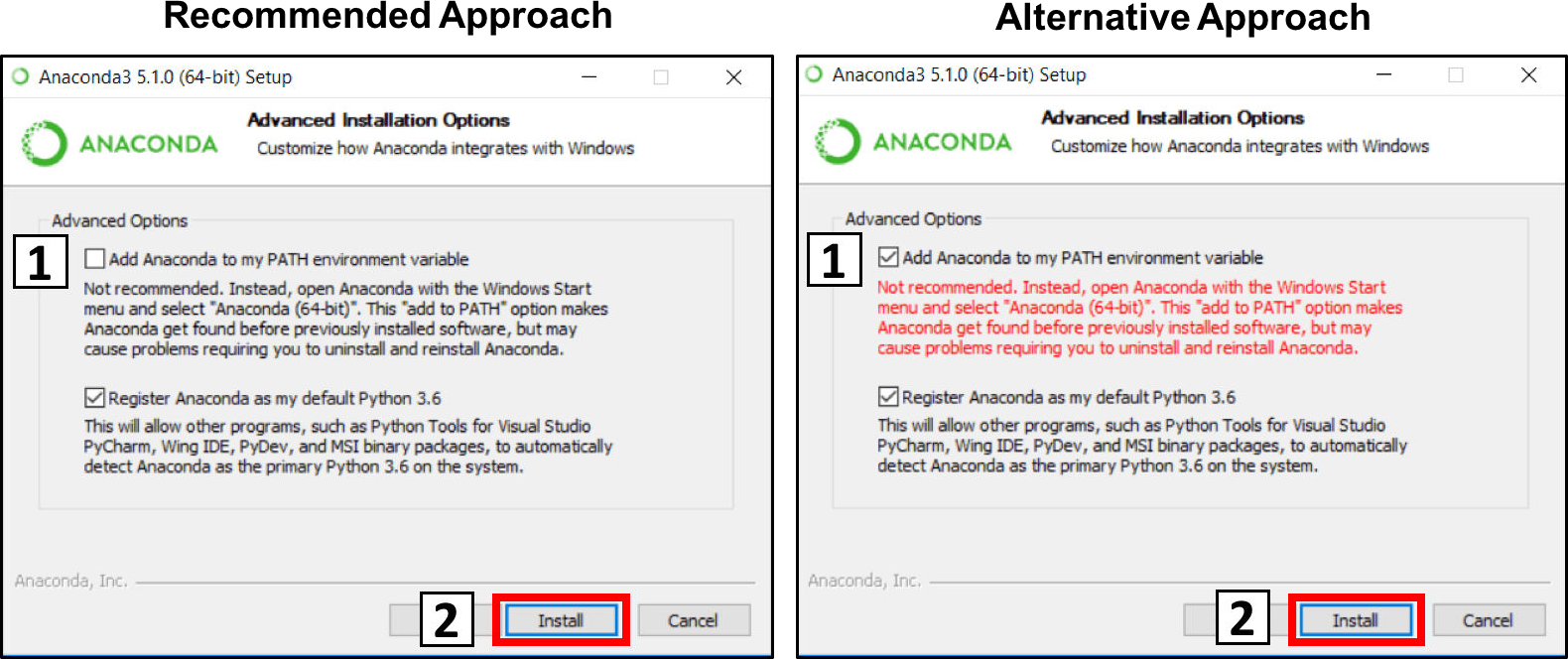
1. Click on Next.



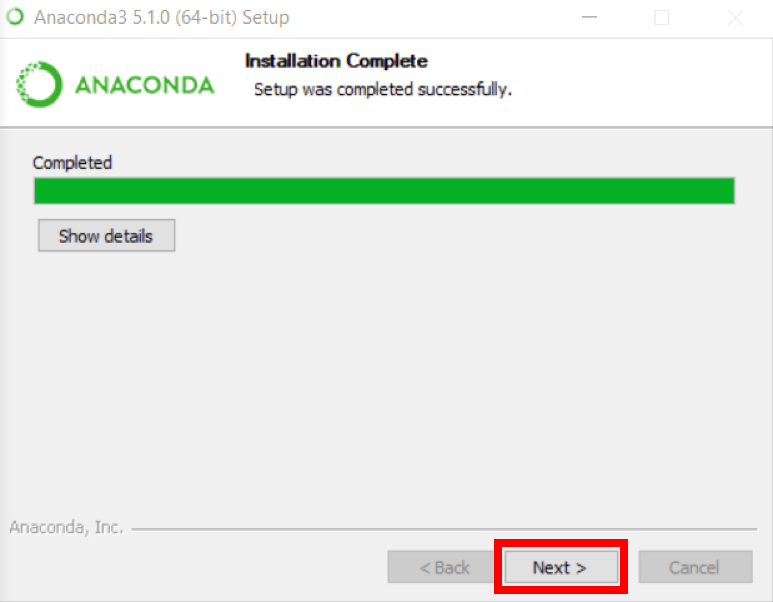
1. Note your installation location and then click Next.
2. This is an important part of the installation process. The recommended approach is to not check the box to add Anaconda to your path. This means you will have to use Anaconda

Navigator or the Anaconda Command Prompt (located in the Start Menu under "Anaconda") when you wish to use Anaconda (you can always add Anaconda to your PATH later if you don't check the box). If you want to be able to use Anaconda in your command prompt (or git

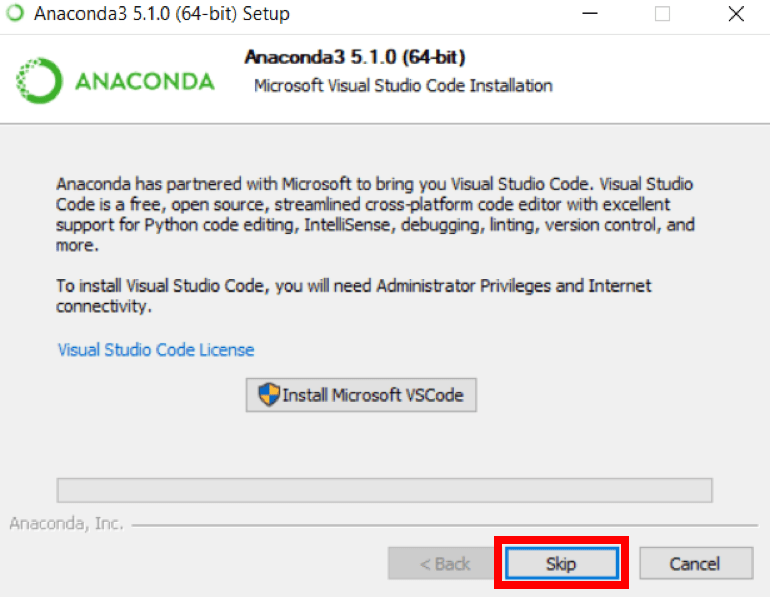
bash, [cmder,](http://cmder.net/) powershell etc), please use the alternative approach and check the box.



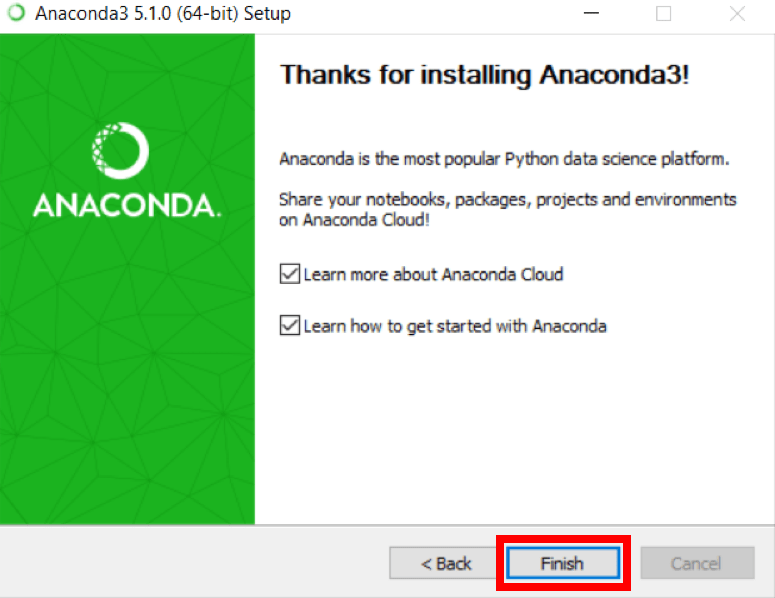
## Click on Next.



1. You can install Microsoft VSCode if you wish, but it is optional.

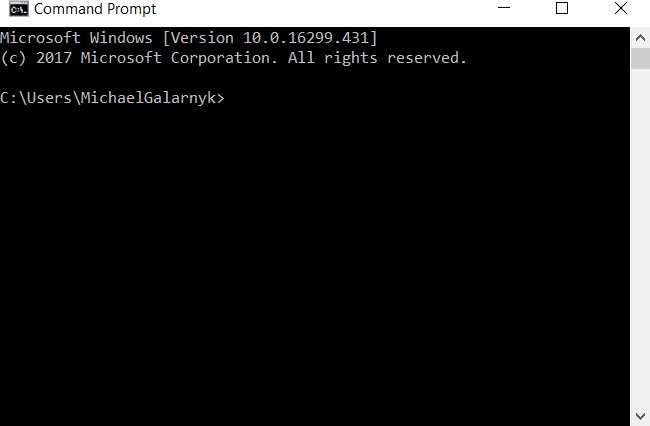


## Click on Finish.



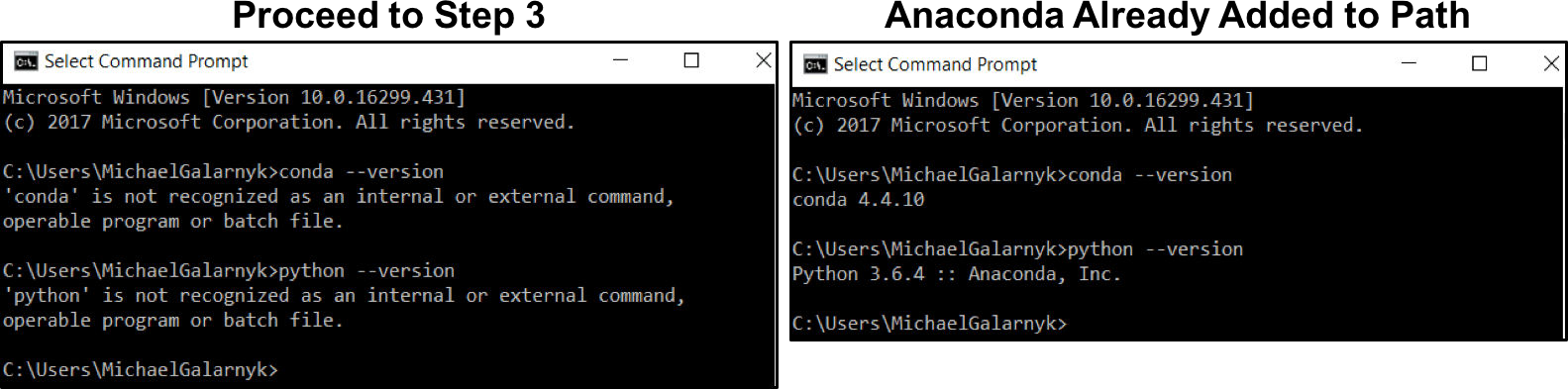
### Add Anaconda to Path (Optional)

This is an **optional** step. This is for the case where you didn't check the box in step 6 and now want to add Anaconda to your Path. The advantage of this is that you will be able to use Anaconda in your Command Prompt, Git Bash, cmder etc.

1. Open a Command Prompt.
2. Check if you already have Anaconda added to your path. Enter the commands below into your Command Prompt. This is checking if you already have Anaconda added to your path. If you get a command **not recognized** error like in the left side of the image below, proceed to step
3. If you get an output similar to the right side of the image below, you have already added Anaconda to your path.

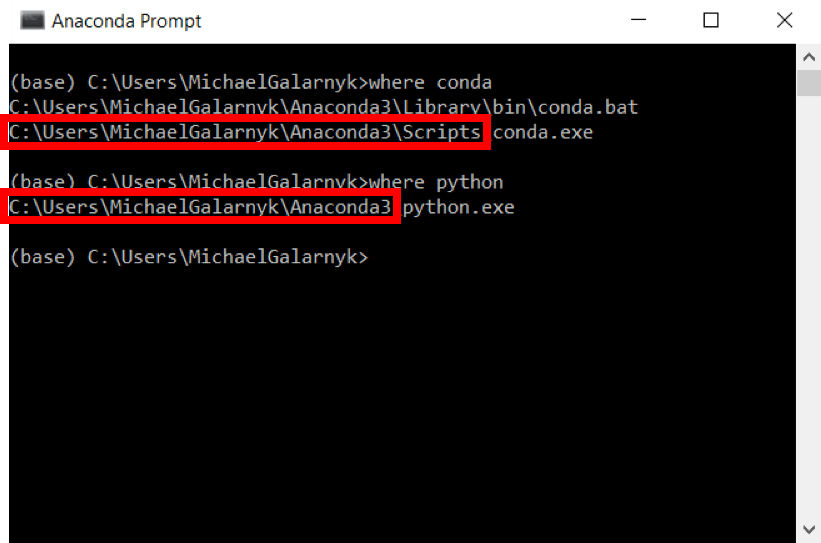
conda --version

python --version

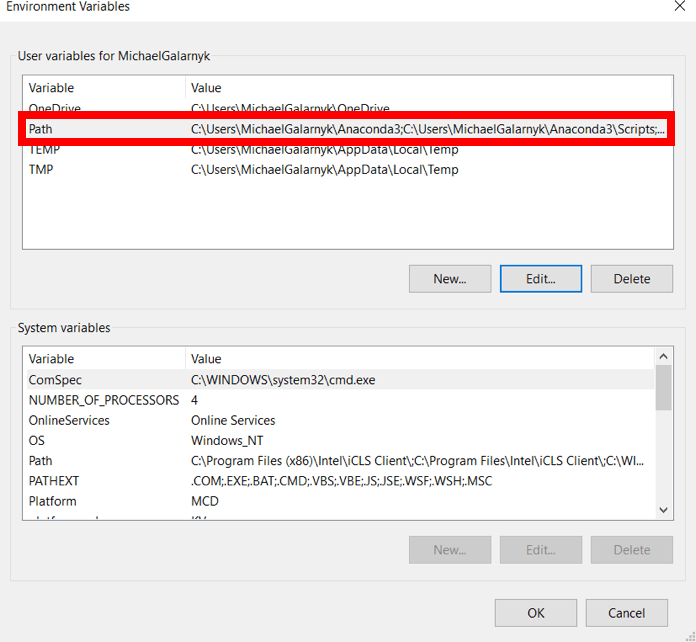


1. If you don't know where your conda and/or python is, open an **Anaconda Prompt** and type in the following commands. This is telling you where conda and python are located on your computer.

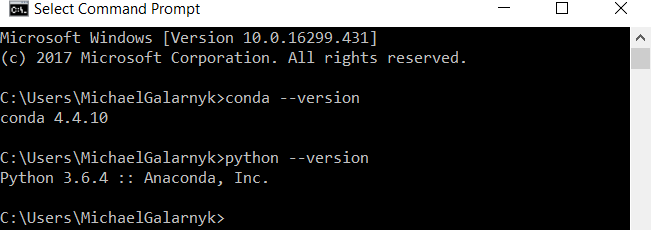
where conda where python



1. Add conda and python to your PATH. You can do this by going to your Environment Variables and adding the output of step 3 (enclosed in the red rectangle) to your path. If you are having issues, here is a short [video](https://youtu.be/mf5u2chPBjY?t=15m45s) on adding conda and python to your PATH.



1. Open a **new Command Prompt**. Try typing conda --version and python --version into the **Command Prompt** to check to see if everything went well.



1. Implement A\* Search algorithm

**Algorithm:**

* 1. Start with OPEN containing only the initial state (node). Set that node’s g value 0 its h’ value to whatever it is and its f’ value h’+ 0 or h’. Set CLOSED to the empty list.
  2. Until a goal node is found repeat the following procedure: If there are no nodes on OPEN, report failure. Otherwise pick the node on OPEN with lowest f’ value. CALL it BESTNODE. Remove from OPEN. Place it on CLOSED. If BESTNODE is the goal node, exit and report a solution. Otherwise, generate the successors of BESTNODE.

For each successor, do the following

* + 1. Set successors to point back to BESTNODE. These backwards links will make possible to recover the path once a solution is found.
    2. Compute
    3. If successor is already existed in OPEN call that node as OLD and we must decide whether OLD’ s parent link should reset to point to BESTNODE (graphs exist in this case). If OLD is cheaper then we need do nothing. If successor is cheaper then reset OLD’s parent link to point to BESTNODE. Record the new cheaper path in g(OLD) and update f’(OLD).
    4. If SUCCESSOR was not on OPEN, see if it is on CLOSED. If so, call node on CLOSED OLD and add OLD to the list of BESTNODE successors. Calculate all the g, f’ and h’ values for successors of that node which is better then move that. So, to propagate the new cost downward, do a depth first traversal of the tree starting at OLD, changing each nodes value (and thus also its f’ value), terminating each branch when you reach either a node with no successor or a node which an equivalent or better path has already been found.
    5. If successor was not already on either OPEN or CLOSED, then put it on OPEN and add it to the list of BESTNODE successors. Compute

In [4]: 

**def** aStarAlgo(start\_node , stop\_node): open\_set **=** set(start\_node)

closed\_set **=** set() g **=** {}

parents **=** {}

g[start\_node] **=** 0

parents[start\_node] **=** start\_node

**while** len(open\_set)**>**0: n **= None**

**for** v **in** open\_set:

**if** n**==None or** g[v]**+**heuristic(v) **<** g[n]**+**heuristic(n): n **=** v

**if** n**==**stop\_node **or** Graph\_nodes[n]**==None**:

###### pass else:

**for** (m,weight) **in** get\_neighbours(n):

**if** m **not in** open\_set **and** m **not in** closed\_set: open\_set.add(m)

parents[m] **=** n

g[m] **=** g[n] **+** weight

###### else:

**if** g[m]**>**g[n]**+**weight: g[m] **=** g[n] **+** weight parents[m] **=** n

**if** m **in** closed\_set:

closed\_set.remove(m) open\_set.add(m)

###### if n==None:

print('Path not found')

**return None if** n**==**stop\_node:

path **=** []

**while** parents[n]**!=**n: path.append(n) n **=** parents[n]

path.append(start\_node) path.reverse()

print('Path found : {}'.format(path))

**return** path

open\_set.remove(n) closed\_set.add(n)

print("Path doesn't exist")

###### return None

**def** get\_neighbours(v):

**if** v **in** Graph\_nodes:

**return** Graph\_nodes[v]

###### else:

**return None**

**def** heuristic(n): H\_dist **=** {

|  |  |
| --- | --- |
| 'A' : | 11, |
| 'B' : | 6, |
| 'C' : | 99, |
| 'D' : | 1, |
| 'E' : | 7, |
| 'G' : | 0 |
| } |  |

**return** H\_dist[n]

Graph\_nodes **=** {

'A' : [('B',2),('E',3)] ,

'B' : [('C',1),('G',9)] ,

'C' : **None** ,

'E' : [('D',6)] ,

'D' : [('G',1)]

}

aStarAlgo('A','G')

Path found : ['A', 'E', 'D', 'G']

Out[4]:

['A', 'E', 'D', 'G']

1. Implement AO\* Search algorithm

#### Algorithm:

* **Input:** Weighted Directed Graph (G) with Heuristics(h) pre-computed, Start node.
* **Output:** Optimal path and cost in the graph

**Step-1:** Create an initial graph with a single node (start node).

**Step-2:** Transverse the graph following the current path, accumulating node that has not yet been expanded or solved.

**Step-3:** Select any of these nodes and explore it. If it has no successors then call this value- FUTILITY else calculate f'(n) for each of the successors.

**Step-4:** If **f'(n)=0**, then mark the node as **SOLVED**.

**Step-5:** Change the value of f'(n) for the newly created node to reflect its successors by backpropagation.

**Step-6:** Whenever possible use the most promising routes, If a node is marked as SOLVED then mark the parent node as SOLVED.

**Step-7:** If the starting node is SOLVED or value is greater than **FUTILITY** then stop else repeat from Step-2.

In [2]: 

**def** recAOStar(n):

**global** finalPath

print('Expanding node:',n) and\_nodes **=** []

or\_nodes **=** []

**if** n **in** allNodes:

**if** 'AND' **in** allNodes[n]:

and\_nodes **=** allNodes[n]['AND']

**if** 'OR' **in** allNodes[n]:

or\_nodes **=** allNodes[n]['OR']

**if** len(and\_nodes)**==**0 **and** len(or\_nodes)**==**0:

###### return

solvable **= False**

marked **=** {}

**while not** solvable:

**if** len(marked)**==**len(and\_nodes)**+**len(or\_nodes):

min\_cost\_least, min\_cost\_group\_least **=** least\_cost\_grop(and\_nodes, or\_nodes, {}) solvable **= True**

change\_heuristic(n, min\_cost\_least)

optimal\_child\_group[n] **=** min\_cost\_group\_least

###### continue

min\_cost, min\_cost\_group **=** least\_cost\_group(and\_nodes, or\_nodes,marked) is\_expanded **= False**

**if** len(min\_cost\_group)**>**1:

**if** min\_cost\_group[0] **in** allNodes: is\_expanded **= True**

recAOStar(min\_cost\_group[0])

**if** min\_cost\_group[1] **in** allNodes: is\_expanded **= True**

recAOStar(min\_cost\_group[1])

###### else:

**if** min\_cost\_group **in** allNodes: is\_expanded **= True**

recAOStar(min\_cost\_group)

**if** is\_expanded:

min\_cost\_verify, min\_cost\_group\_verify **=** least\_cost\_group(and\_nodes, or\_nodes,

**if** min\_cost\_group**==**min\_cost\_group\_verify: solvable **= True**

change\_heuristic(n, min\_cost\_verify)

optimal\_child\_group[n] **=** min\_cost\_group

###### else:

solvable **= True**

change\_heuristic(n, min\_cost)

optimal\_child\_group[n] **=** min\_cost\_group marked[min\_cost\_group] **=** 1

**return** heuristic(n)

**def** least\_cost\_group(and\_nodes, or\_nodes, marked): node\_wise\_cost **=** {}

**for** node\_pair **in** and\_nodes:

**if not** node\_pair[0]**+**node\_pair[1] **in** marked: cost **=** 0

cost **=** cost **+** heuristic(node\_pair[0]) **+** heuristic(node\_pair[1]) **+** 2 node\_wise\_cost[node\_pair[0]**+**node\_pair[1]] **=** cost

**for** node **in** or\_nodes:

**if not** node **in** marked: cost **=** 0

cost **=** cost **+** heuristic(node) **+** 1 node\_wise\_cost[node] **=** cost

min\_cost **=** 999999

min\_cost\_group **= None**

**for** costKey **in** node\_wise\_cost:

**if** node\_wise\_cost[costKey]**<**min\_cost: min\_cost **=** node\_wise\_cost[costKey] min\_cost\_group **=** costKey

**return** [min\_cost, min\_cost\_group]

**def** heuristic(n):

**return** H\_dist[n]

**def** change\_heuristic(n,cost):

H\_dist[n] **=** cost

###### return

**def** print\_path(node):

print(optimal\_child\_group[node], end**=**"") node **=** optimal\_child\_group[node]

**if** len(node)**>**1:

**if** node[0] **in** optimal\_child\_group: print("->",end**=**"")

print\_path(node[0])

**if** node[1] **in** optimal\_child\_group: print("->",end**=**"")

print\_path(node[1])

###### else:

**if** node **in** optimal\_child\_group: print("->",end**=**"")

print\_path(node)

H\_dist **=** {'A':**-**1, 'B':4, 'C':2, 'D':3, 'E':6, 'F':8, 'G':2, 'H':0, 'I':0, 'J':0} allNodes **=** {'A' :{'AND':[('C','D')], 'OR':['B']} ,

'B' :{'OR':['E','F']} ,

'C' :{'OR':['G'], 'AND':[('H','I')]} , 'D' :{'OR':['J']}

}

optimal\_child\_group **=** {}

optimal\_cost **=** recAOStar('A')

print('Nodes which give optimal cost are:') print\_path('A')

print("\nOptimal Cost is : ",optimal\_cost)

Expanding node: A Expanding node: B Expanding node: C Expanding node: D

Nodes which give optimal cost are: CD->HI->J

Optimal Cost is : 5

In [ ]: 

1. 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.

#### Algorithm

Initialize G to the set of maximally general hypotheses in H Initialize S to the set of maximally specific hypotheses in H For each training example d, do

* If d is a positive example
  + Remove from G any hypothesis inconsistent with d
  + For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      * h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S
* If d is a negative example
  + Remove from S any hypothesis inconsistent with d
  + For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      * h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

In [8]: 

**import** numpy **as** np

**import** pandas **as** pd

data **=** pd.DataFrame(data**=**pd.read\_csv('Training.csv')) print(data)

concepts **=** np.array(data.iloc[:,0:**-**1]) target **=** np.array(data.iloc[:,**-**1])

**def** learn(concepts, target):

specific\_h **=** concepts[0].copy()

print("\nInitialization of specific\_h and general\_h") print("\n",specific\_h)

general\_h **=** [["?" **for** i **in** range(len(specific\_h))] **for** i **in** range(len(specific\_h))] print("\n",general\_h)

**for** i, h **in** enumerate(concepts):

**if** target[i] **==** "Yes":

**for** x **in** range(len(specific\_h)):

**if** h[x] **!=** specific\_h[x]: specific\_h[x] **=** '?' general\_h[x][x] **=** '?'

**if** target[i] **==** "No":

**for** x **in** range(len(specific\_h)):

**if** h[x] **!=** specific\_h[x]:

general\_h[x][x] **=** specific\_h[x]

###### else:

general\_h[x][x] **=** '?'

print(" \nsteps of Candidate Elimination Algorithm",i**+**1) print("\nSpecific\_h ",i**+**1,"\n ")

print(specific\_h)

print("\ngeneral\_h ", i**+**1, "\n ") print(general\_h)

indices **=** [i **for** i, val **in** enumerate(general\_h) **if** val **==** ['?', '?', '?', '?', '?', '?'

**for** i **in** indices:

general\_h.remove(['?', '?', '?', '?', '?', '?'])

**return** specific\_h, general\_h

s\_final, g\_final **=** learn(concepts, target)

print("\nFinal Specific\_h:", s\_final, sep**=**"\n") print("\nFinal General\_h:", g\_final, sep**=**"\n")

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sky | AirTemp | Humidity | Wind | Water | Forecast | EnjoySport |
| 0 | sunny | warm | normal | strong | warm | same | Yes |
| 1 | sunny | warm | high | strong | warm | same | Yes |
| 2 | cloudy | cold | high | strong | warm | change | No |
| 3 | sunny | warm | high | strong | cool | change | Yes |

Initialization of specific\_h and general\_h

['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',

'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

steps of Candidate Elimination Algorithm 1 Specific\_h 1

['sunny' 'warm' 'normal' 'strong' 'warm' 'same'] general\_h 1

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',

'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

steps of Candidate Elimination Algorithm 2 Specific\_h 2

['sunny' 'warm' '?' 'strong' 'warm' 'same'] general\_h 2

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',

'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

steps of Candidate Elimination Algorithm 3 Specific\_h 3

['sunny' 'warm' '?' 'strong' 'warm' 'same'] general\_h 3

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

steps of Candidate Elimination Algorithm 4 Specific\_h 4

['sunny' 'warm' '?' 'strong' '?' '?'] general\_h 4

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',

'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific\_h:

['sunny' 'warm' '?' 'strong' '?' '?']

Final General\_h:

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

1. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

#### Algorithm

***ID3(Examples, Target\_attribute, Attributes)***

**Examples** are the training examples.

**Target\_attribute** is the attribute whose value is to be predicted by the tree. **Attributes** is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

* Create a Root node for the tree
* If all Examples are positive, Return the single-node tree Root, with label = +
* If all Examples are negative, Return the single-node tree Root, with label = -
* If Attributes is empty, Return the single-node tree Root, with label = most common value of Target\_attribute in Examples
* Otherwise Begin

A ← the attribute from Attributes that best\* classifies Examples The decision attribute for Root ← A

For each possible value, *vi*, of A,

Add a new tree branch below *Root*, corresponding to the test A = *vi* Let *Examples vi*, be the subset of Examples that have value *vi* for *A* If *Examples vi* , is empty

Then below this new branch add a leaf node with label = most common value of Target\_attribute in Examples

Else below this new branch add the subtree ID3(*Examples vi*, Targe\_tattribute, Attributes – {A}))

End

Return Root

In [11]: 

**import** math

**def** dataset\_split(data, arc, val): newData **=** []

**for** rec **in** data:

**if** rec[arc] **==** val:

reducedSet **=** list(rec[:arc]) reducedSet.extend(rec[arc**+**1:]) newData.append(reducedSet)

**return** newData

**def** calc\_entropy(data): entries **=** len(data) labels **=** {}

**for** rec **in** data:

label **=** rec[**-**1]

**if** label **not in** labels.keys(): labels[label] **=** 0

labels[label] **+=** 1

entropy **=** 0.0

**for** key **in** labels:

prob **=** float(labels[key])**/**entries entropy **-=** prob **\*** math.log(prob, 2)

**return** entropy

**def** attribute\_selection(data): features **=** len(data[0]) **-** 1

baseEntropy **=** calc\_entropy(data) max\_InfoGain **=** 0.0

bestAttr **= -**1

**for** i **in** range(features):

AttrList **=** [rec[i] **for** rec **in** data] uniqueVals **=** set(AttrList)

newEntropy **=** 0.0

attrEntropy **=** 0.0

**for** value **in** uniqueVals:

newData **=** dataset\_split(data, i, value) prob **=** len(newData)**/**float(len(data))

newEntropy **=** prob **\*** calc\_entropy(newData) attrEntropy **+=** newEntropy

infoGain **=** baseEntropy **-** attrEntropy

**if** infoGain **>** max\_InfoGain: max\_InfoGain **=** infoGain bestAttr **=** i

**return** bestAttr

**def** decision\_tree(data, labels):

classList **=** [rec[**-**1] **for** rec **in** data]

**if** classList.count(classList[0]) **==** len(classList):

**return** classList[0]

maxGainNode **=** attribute\_selection(data) treeLabel **=** labels[maxGainNode]

theTree **=** {treeLabel: {}}

**del**(labels[maxGainNode])

nodeValues **=** [rec[maxGainNode] **for** rec **in** data] uniqueVals **=** set(nodeValues)

**for** value **in** uniqueVals: subLabels **=** labels[:]

theTree[treeLabel][value] **=** decision\_tree(dataset\_split(data, maxGainNode, value),

**return** theTree

**def** print\_tree(tree, level):

**if** tree **==** 'yes' **or** tree **==** 'no': print(' '**\***level, 'd=', tree) **return**

**for** key,value **in** tree.items(): print(' ' **\***level, key)

print\_tree(value, level**\***2)

**with** open('tennis.csv', 'r') **as** csvfile:

fdata **=** [line.strip() **for** line **in** csvfile] metadata **=** fdata[0].split(',')

train\_data **=** [x.split(',') **for** x **in** fdata[1:]]

tree **=** decision\_tree(train\_data, metadata) print\_tree(tree, 1)

print(tree)

Outlook

overcast d= yes

rain

Wind

sunny

weak

strong

d= yes d= no

Humidity

high

normal

d= no d= yes

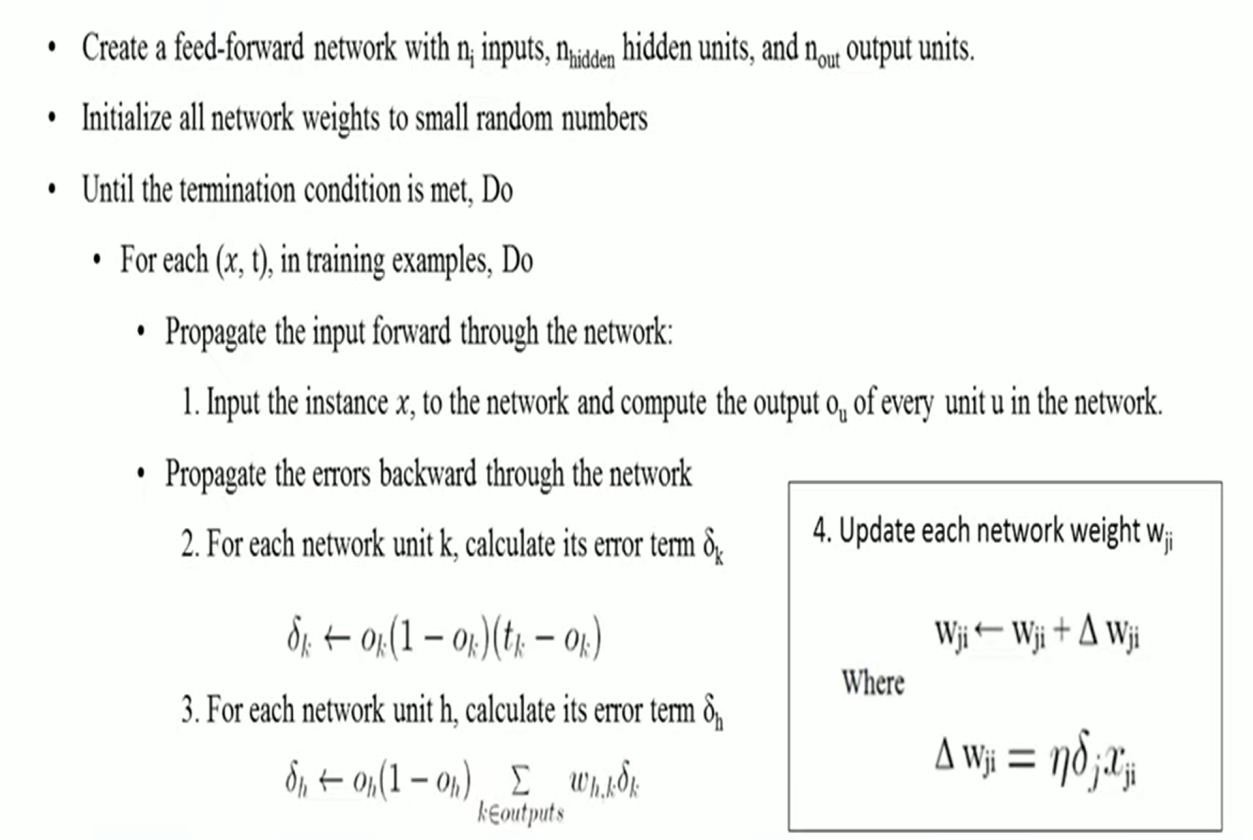
{'Outlook': {'overcast': 'yes', 'rain': {'Wind': {'weak': 'yes', 'strong':

'no'}}, 'sunny': {'Humidity': {'high': 'no', 'normal': 'yes'}}}}

In [ ]: 

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

#### Algorithm



In [2]: 

**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)

**def** sigmoid(x):

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

**def** der\_sigmoid(x):

**return** x **\*** (1 **-** x)

epoch **=** 5000

lr **=** 0.01

neurons\_i **=** 2

neurons\_h **=** 3

neurons\_o **=** 1

weight\_h **=** np.random.uniform(size**=**(neurons\_i, neurons\_h)) bias\_h **=** np.random.uniform(size**=**(1, neurons\_h))

weight\_o **=** np.random.uniform(size**=**(neurons\_h, neurons\_o)) bias\_o **=** np.random.uniform(size**=**(1, neurons\_o))

**for** i **in** range(epoch):

inp\_h **=** np.dot(X, weight\_h) **+** bias\_h out\_h **=** sigmoid(inp\_h)

inp\_o **=** np.dot(out\_h, weight\_o) **+** bias\_o out\_o **=** sigmoid(inp\_o)

err\_o **=** y **-** out\_o

grad\_o **=** der\_sigmoid(out\_o) delta\_o **=** err\_o **\*** grad\_o

err\_h **=** delta\_o.dot(weight\_o.T) grad\_h **=** der\_sigmoid(out\_h)

delta\_h **=** err\_h **\*** grad\_h

weight\_o **+=** out\_h.T.dot(delta\_o) **\*** lr weight\_h **+=** X.T.dot(delta\_h) **\*** lr

print('Input: ', X)

print('Actual: ', y)

print('Predicted: ', out\_o)

Input: [[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual: [[0.92]

[0.86]

[0.89]]

Predicted: [[0.89077146]

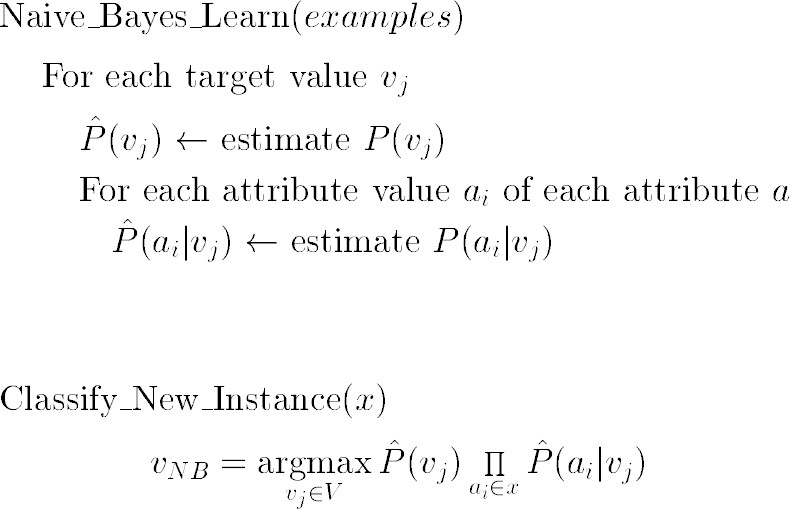
[0.8744389 ]

[0.89555458]]

In [ ]: 

1. 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.

#### Algorithm:



In [2]: 

**import** pandas **as** pd

**import** numpy **as** np

mush **=** pd.read\_csv('mushrooms.csv') mush **=** mush.replace('?',np.nan)

mush.dropna(axis**=**1,inplace**=True**) target **=** 'class'

features **=** mush.columns[mush.columns**!=**target] target\_classes**=**mush[target].unique()

test **=** mush.sample(frac **=** .3) mush **=** mush.drop(test.index) cond\_probs **=** {}

target\_class\_prob **=** {}

**for** t **in** target\_classes:

mush\_t **=** mush[mush[target]**==**t][features]

target\_class\_prob[t] **=** float(len(mush\_t)**/**len(mush)) class\_prob **=** {}

**for** col **in** mush\_t.columns: col\_prob **=** {}

**for** val,cnt **in** mush\_t[col].value\_counts().iteritems(): pr **=** cnt**/**len(mush\_t)

col\_prob[val] **=** pr

class\_prob[col] **=** col\_prob cond\_probs[t] **=** class\_prob

**def** calc\_probs(x): probs **=** {}

**for** t **in** target\_classes:

p **=** target\_class\_prob[t]

**for** col,val **in** x.iteritems():

**try**:

p **\*=** cond\_probs[t][col][val]

###### except:

p **=** 0 probs[t] **=** p

**return** probs

**def** classify(x):

probs **=** calc\_probs(x) max **=** 0

max\_class **=** ' '

**for** cl,pr **in** probs.items():

**if** pr**>**max:

max **=** pr

max\_class **=** cl

**return** max\_class

b **=** []

**for** i **in** mush.index:

b.append(classify(mush.loc[i,features]) **==** mush.loc[i,target]) print(sum(b)," correct of ",len(mush))

print('Accuracy : ',sum(b)**/**len(mush))

b **=** []

**for** i **in** test.index:

b.append(classify(test.loc[i,features]) **==** test.loc[i,target]) print(sum(b)," correct of ",len(test))

print('Accuracy : ',sum(b)**/**len(test))

5669 correct of 5687

Accuracy : 0.9968348865834359

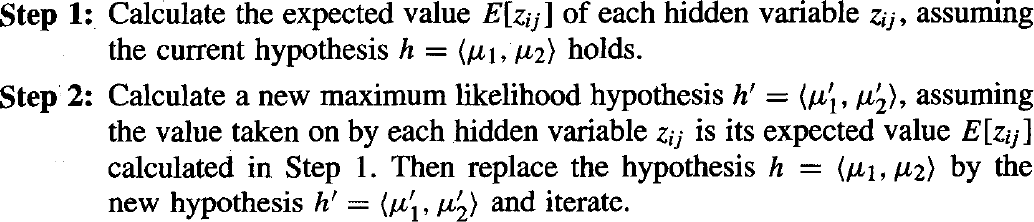
2433 correct of 2437

Accuracy : 0.9983586376692655

In [ ]: 

1. 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.

#### Algorithm



In [4]: 

**import** numpy **as** np

**import** pandas **as** pd

**from** matplotlib **import** pyplot **as** plt

**from** sklearn.mixture **import** GaussianMixture

**from** sklearn.cluster **import** KMeans data **=** pd.read\_csv('ex.csv')

f1 **=** data['V1'].values f2 **=** data['V2'].values

X **=** np.array(list(zip(f1,f2))) print("x: ",X)

print("Graph for whole dataset") plt.scatter(f1,f2,c**=**'black')

plt.show()

KMeans **=**KMeans(2)

labels **=** KMeans.fit(X).predict(X) print("labels for KMeans:",labels)

print('Graph using KMeans Algorithm') plt.scatter(f1,f2,c **=** labels)

centroids **=** KMeans.cluster\_centers\_ print("centroids: ",centroids)

plt.scatter(centroids[:,0],centroids[:,1],marker **=**'\*',c**=**'red') plt.show()

gmm**=**GaussianMixture(2)

Labels**=**gmm.fit(X).predict(X)

print("Labels for GMM: ",labels) print('Graph using EM Algorithm') plt.scatter(f1,f2,c**=**labels)

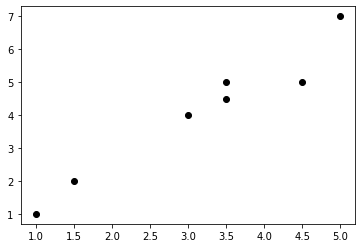
plt.show()

x: [[1. 1. ]

|  |  |  |
| --- | --- | --- |
| [1.5 | 2. | ] |
| [3. | 4. | ] |
| [5. | 7. | ] |
| [3.5 | 5. | ] |
| [4.5 | 5. | ] |

[3.5 4.5]]

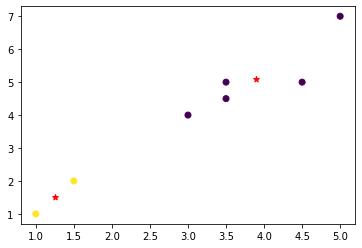
Graph for whole dataset



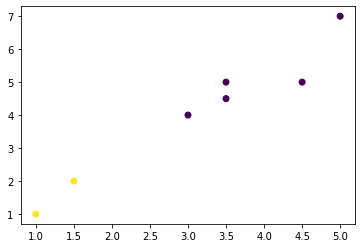
labels for KMeans: [1 1 0 0 0 0 0]

Graph using KMeans Algorithm centroids: [[3.9 5.1 ]

[1.25 1.5 ]]



Labels for GMM: [1 1 0 0 0 0 0] Graph using EM Algorithm



In [ ]: 

1. Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

#### Algorithm

* ***Step 1:*** First, Find the distance.
* ***Step 2:*** Find the rank
* ***Step 3:*** Find the nearest neighbor.

In [2]: 

**from** sklearn.datasets **import** load\_iris

**from** sklearn.neighbors **import** KNeighborsClassifier

**import** numpy **as** np

**from** sklearn.model\_selection **import** train\_test\_split iris\_dataset **=** load\_iris()

targets **=** iris\_dataset.target\_names print('class : number')

**for** i **in** range(len(targets)): print(targets[i]," : ",i)

X\_train, X\_test, Y\_train, Y\_test **=** train\_test\_split(iris\_dataset['data'],iris\_dataset['targ kn **=** KNeighborsClassifier(1)

kn.fit(X\_train,Y\_train)

**for** i **in** range(len(X\_test)):

x\_new **=** np.array([X\_test[i]])

prediction **=** kn.predict(x\_new)

print("Actual:[{0}][{1}],Predicted:{2} {3}".format(Y\_test[i],targets[Y\_test[i]],predict print("\nAccuracy:",kn.score(X\_test,Y\_test))

class : number setosa : 0



versicolor : 1

virginica : 2

Actual:[1][versicolor],Predicted:[2] ['virginica'] Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[2][virginica],Predicted:[2] ['virginica']

Actual:[2][virginica],Predicted:[2] ['virginica'] Actual:[0][setosa],Predicted:[0] ['setosa']

Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[0][setosa],Predicted:[0] ['setosa']

Actual:[0][setosa],Predicted:[0] ['setosa']

Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[0][setosa],Predicted:[0] ['setosa']

Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[2][virginica],Predicted:[2] ['virginica']

Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[2][virginica],Predicted:[2] ['virginica']

Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[2][virginica],Predicted:[2] ['virginica']

Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[2][virginica],Predicted:[2] ['virginica']

Actual:[0][setosa],Predicted:[0] ['setosa']

Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[2][virginica],Predicted:[2] ['virginica']

Actual:[0][setosa],Predicted:[0] ['setosa']

Actual:[2][virginica],Predicted:[2] ['virginica'] Actual:[0][setosa],Predicted:[0] ['setosa']

Actual:[0][setosa],Predicted:[0] ['setosa']

Actual:[2][virginica],Predicted:[2] ['virginica']

Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[0][setosa],Predicted:[0] ['setosa']

Actual:[2][virginica],Predicted:[2] ['virginica'] Actual:[2][virginica],Predicted:[2] ['virginica']

Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[2][virginica],Predicted:[2] ['virginica']



Actual:[2][virginica],Predicted:[2] ['virginica']

Actual:[1][versicolor],Predicted:[1] ['versicolor'] Actual:[2][virginica],Predicted:[2] ['virginica']

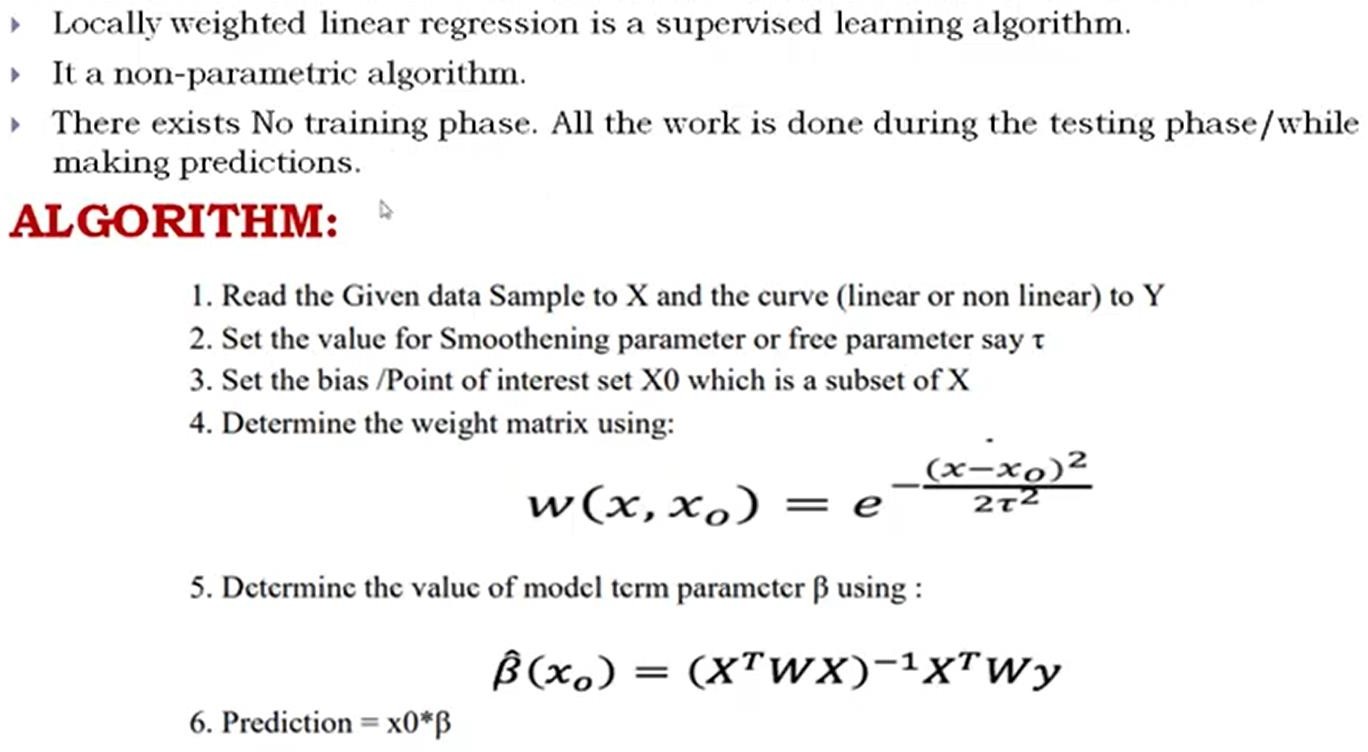
Actual:[1][versicolor],Predicted:[1] ['versicolor'] Accuracy: 0.9736842105263158

**rogram 9**

**P**

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

#### Algorithm



In [5]: 

**from** math **import** ceil

**import** numpy **as** np

**from** scipy **import** linalg

**def** lowess(x, y, f**=**2.**/**3., iter**=**3): n **=** len(x)

r **=** int(ceil(f**\***n))

h **=** [np.sort(np.abs(x**-**x[i]))[r] **for** i **in** range(n)]

w **=** np.clip(np.abs((x[:,**None**]**-**x[**None**,:])**/**h) , 0.0 , 1.0) w **=** (1**-** w**\*\***3) **\*\*** 3

yest **=** np.zeros(n) delta **=** np.ones(n)

**for** iteration **in** range(iter):

**for** i **in** range(n):

weights **=** delta**\***w[:,i]

b **=** np.array([np.sum(weights**\***y) , np.sum(weights**\***y**\***x)]) A **=** np.array([[np.sum(weights) , np.sum(weights**\***x)],

[np.sum(weights**\***x),np.sum(weights**\***x**\***x)]]) beta **=** linalg.solve(A,b)

yest[i] **=** beta[0] **+** beta[1]**\***x[i] residuals **=** y **-** yest

s **=** np.median(np.abs(residuals))

delta **=** np.clip(residuals**/**(6.0**\***s),**-**1,1) delta **=** (1 **-** delta**\*\***2) **\*\*** 2

**return** yest

**if** name **==**' main ':

**import** math n **=** 100

x **=** np.linspace(0 , 2**\***math.pi , n)

y **=** np.sin(x) **+** 0.3 **\*** np.random.randn(n) f **=** 0.25

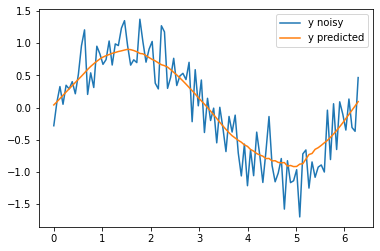
yest **=** lowess(x,y,f,3)

**import** pylab **as** pl pl.clf()

pl.plot(x,y,label**=**'y noisy')

pl.plot(x,yest,label**=**'y predicted') pl.legend()

pl.show()



### VIVA Questions

* 1. What is Artificial Intelligence?
  2. Explain A\* Search algorithm.
  3. Explain AO\* Search algorithm.
  4. How is AO\* search different from A\* search algorithm.
  5. What is machine learning?
  6. Define supervised learning
  7. Define unsupervised learning
  8. Define semi supervised learning
  9. Define reinforcement learning
  10. What do you mean by hypotheses?
  11. What is classification?
  12. What is clustering?
  13. Define precision, accuracy and recall
  14. Define entropy
  15. Define regression
  16. How KNN is different from K-Means clustering
  17. What is concept learning?
  18. Define specific boundary and general boundary
  19. Define target function
  20. Define decision tree
  21. What is ANN
  22. Explain gradient descent approximation
  23. State Bayes theorem
  24. Define Bayesian belief networks
  25. Differentiate hard and soft clustering
  26. Define variance
  27. What is inductive machine learning?
  28. Why K Nearest Neighbor algorithm is lazy learning algorithm
  29. Why naïve Bayes is naïve
  30. Mention classification algorithms
  31. Define pruning
  32. Differentiate Clustering and classification
  33. Mention clustering algorithms
  34. Define Bias
  35. What is learning rate? Why it is needed