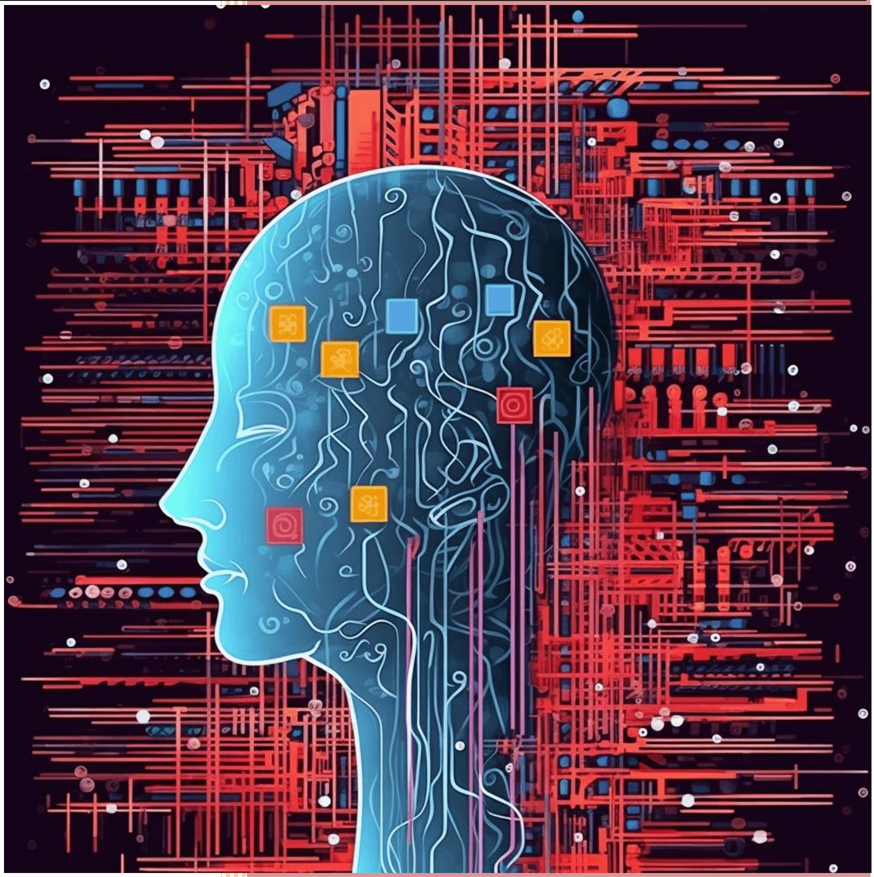
Version 1.3

CS4811: Artificial Intelligence Lab Manual



Capital University of Science & Technology

Department of Computer Science

Lab Course Development Team

Supervision and Coordination	**Dr. M. Abdul Qadir** Professor/ Dean Faculty of Computing
Lab Designer	**Ms. Syeda Amna Rizwan** Associate Lecturer Faculty of Computing **Ms. Snober Naseer** Associate Lecturer Faculty of Computing
Lab Reviewer	**Mr. Salman Ahmed** Senior Lecturer Faculty of Computing

Course Outline

Lab Manual for Artificial Intelligence

Lab-1: Python for Beginners

1. Introduction to Python

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently.

Python is:

- **Interpreted:** it can execute at runtime, and changes in a program are instantly perceptible. To be very technical, Python has a compiler. The difference when compared to Java or C++ is how transparent and automatic it is. With Python, we don't have to worry about the compilation step as it's done in real-time. The tradeoff is that interpreted languages are usually slower than compiled ones.

- **Semantically Dynamic:** you don't have to specify types for variables and there is nothing that makes you do it.

- **Object-Oriented:** everything in Python is an object. But you can choose to write code in an object-oriented, procedural, or even functional way.

- **High level:** you don't have to deal with low-level machine details.
  This lab is a course on the introduction of Python language. The lab allows you to learn the core of the language in a matter of hours instead of weeks.

1.1. Use of Python

Python has been growing a lot recently partly because of its many uses in the following areas:

- **System Scripting**: it's a great tool to automate everyday repetitive tasks.

- **Data Analysis**: it is a great language to experiment with and has tons of libraries and tools to handle data, create models, visualize results and even deploy solutions. This is used in areas like Finance, E-commerce, and Research.
- **Web Development**: frameworks like Django and Flask allow the development of web applications, API's, and websites.
- **Machine Learning**: Tensorflow and Pytorch are some of the libraries that allow scientists and the industry to develop and deploy Artificial Intelligence solutions in Image Recognition, Health, Self-driving cars, and many other fields.

1.2. Syntax

1.2.1. Semicolons

Python doesn't use semicolons to finish lines. A new line is enough to tell the interpreter that a new command is beginning.

The print () method will display something.

In this example, we have two commands that will display the messages inside the single quotes.

```
print('First command')
print('Second command')
```

But the following is wrong due to the semicolons in the end:

```
print('First command');
print('Second command');
```

1.2.2. Indentation

Many languages use curly-brackets to define scope.

Python's interpreter uses only indentation to define when a scope ends and another one starts.

This means you have to be aware of white spaces at the beginning of each line -- they have meaning and might break your code if misplaced.

This definition of a function works:

```
def my_function():
    print('First command')
```

This doesn't work because the indentation of the second line is missing and will throw an error:

```
def my_function():
print('First command')
```

1.2.3. Case Sensitivity and Variables

This Python is case sensitive. So the variables name and Name are not the same thing and store different values.

```
name = 'Renan'
Name = 'Moura'
```

As you can see, variables are easily created by just assigning values to them using the = symbol.

This means name stores 'Renan' and Name stores 'Moura'.

1.2.4. Comments

Finally, to comment something in your code, use the hash mark #.

The commented part does not influence the program flow.

```python
# this function prints something
def my_function():
    print('First command')
```

1.2.5. Types

To store data in Python you need to use a variable. And every variable has its type depending on the value of the data stored. Python has dynamic typing, which means you don't have to explicitly declare the type of your variable -- but if you want to, you can. Lists, Tuples, Sets, and Dictionaries are all data types and have dedicated sections later on with more details, but we'll look at them briefly here.

1.2.6. User Input

If you need to interact with a user when running your program in the command line (for example, to ask for a piece of information), you can use the `input ()` function.

```python
country = input("What is your country? ") #user enters 'Brazil'

print(country)
```

1.2.7. Operators

In a programming language, operators are special symbols that you can apply to your variables and values in order to perform operations such as arithmetic/mathematical and comparison. Python has lots of operators that you can apply to your variables and I will demonstrate the most used ones.

1.2.7.1. Arithmetic Operators

Arithmetic operators are the most common type of operators and also the most recognizable ones. They allow you to perform simple mathematical operations. They are:

- +: Addition
- -: Subtraction
- *: Multiplication
- /: Division
- **: Exponentiation
- //: Floor Division, rounds down the result of a division
- %: Modulus, gives you the remainder of a division

Let's see a program that shows how each of them is used.

```
print('Addition:', 5 + 2)
print('Subtraction:', 5 - 2)
print('Multiplication:', 5 * 2)
print('Division:', 5 / 2)
print('Floor Division:', 5 // 2)
print('Exponentiation:', 5 ** 2)
print('Modulus:', 5 % 2)
```

1.2.7.2. Concatenation

Concatenation is when you have two or more strings and you want to join them into one. This is useful when you have information in multiple variables and want to combine them.

For instance, in this next example I combine two variables that contain my first name and my last name respectively to have my full name.

The + operator is used to concatenate.

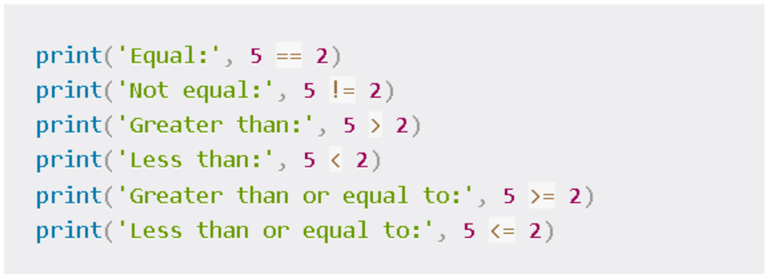
```
first_name = 'Renan '
last_name = 'Moura'

print(first_name + last_name)
```

1.2.7.3. Comparison Operators

Use comparison operators to compare two values. These operators return either True or False. They are:

- ==: Equal
- !=: Not equal
- >: Greater than
- <: Less than
- >=: Greater than or equal to
- <=: Less than or equal to

Let's see a program that shows how each of them is used.



```python
print('Equal:', 5 == 2)
print('Not equal:', 5 != 2)
print('Greater than:', 5 > 2)
print('Less than:', 5 < 2)
print('Greater than or equal to:', 5 >= 2)
print('Less than or equal to:', 5 <= 2)
```

1.2.7.4. Logical Operators

Logical operators are used to combine statements applying Boolean algebra. They are:

- **and:** True only when both statements are true
- **or:** False only when both x and y are false
- **not:** The not operator simply inverts the input, True becomes False and vice versa.

Let's see a program that shows how each one is used.

```
x = 5
y = 2

print(x == 5 and y > 3)

print(x == 5 or y > 3)

print(not (x == 5))
```

1.2.8. Conditionals

Conditionals are one of the cornerstones of any programming language. They allow you to control the program flow according to specific conditions you can check.

1.2.8.1. if statement

The way you implement a conditional is through the `if` statement. The general form of an `if` statement is:

```
if expression:
    statement
```

The `expression` contains some logic that returns a Boolean, and the `statement` is executed only if the return is `True`. A simple example:

```
bob_age = 32
sarah_age = 29

if bob_age > sarah_age:
    print('Bob is older than Sarah')
```

1.2.8.2. if else or elif statement

In our last example, the program only does something if the condition returns `True`.

But we also want it to do something if it returns `False` or even check a second or third condition if the first one wasn't met.

In this example, we swapped Bob's and Sarah's age. The first condition will return `False` since Sarah is older now, and then the program will print the phrase after the `else` instead.

```
bob_age = 29
sarah_age = 32

if bob_age > sarah_age:
    print('Bob is older than Sarah')
else:
    print('Bob is younger than Sarah')
```

Now, consider the example below with the `elif`.

```
bob_age = 32
sarah_age = 32

if bob_age > sarah_age:
    print('Bob is older than Sarah')
elif bob_age == sarah_age:
    print('Bob and Sarah have the same age')
else:
    print('Bob is younger than Sarah')
```

1.3. Objective of the Experiment

After completing this lab, the students should be able to:

- Understand Python and its syntax in detail.

1.4. Practice Task

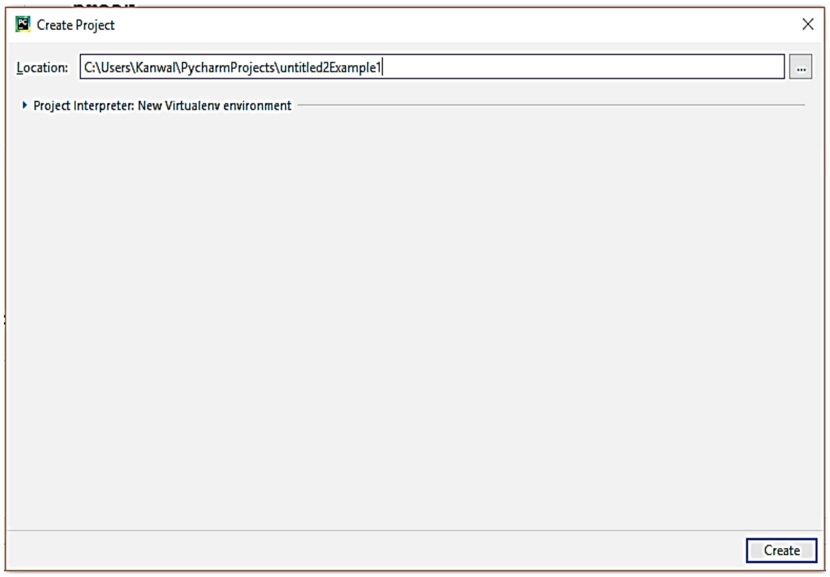
1. Go to the URL https://www.jetbrains.com/pycharm/download/?section=windows#section=windows . To download Pycharm python IDE and install it.

2. For Python installation, go to the URL

https://www.python.org/downloads/release/python-373/ and install **Windows x86-64 executable installer.**

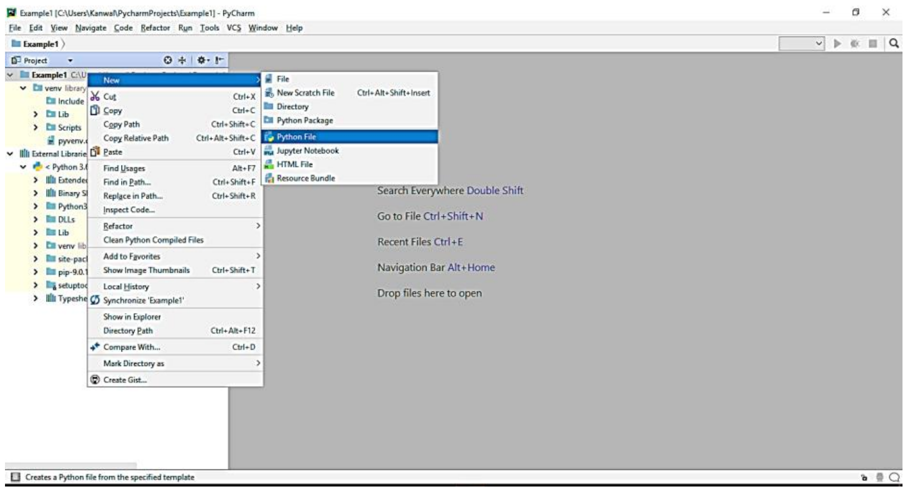
3. Open Pycharm and create new project named **Example1** by clicking on **file** in menu bar and then on **new project.**

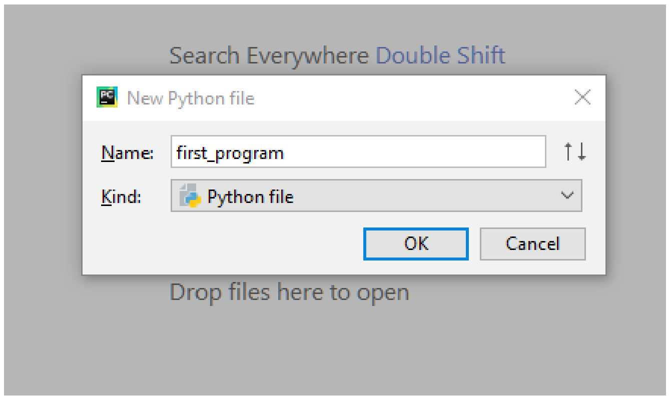
4. After creating new project right click on project name and create a **python file** name first program as shown in figures below.



Location: C:\Users\Kanwal\PycharmProjects\untitled2Example1

Project Interpreter: New Virtualenv environment

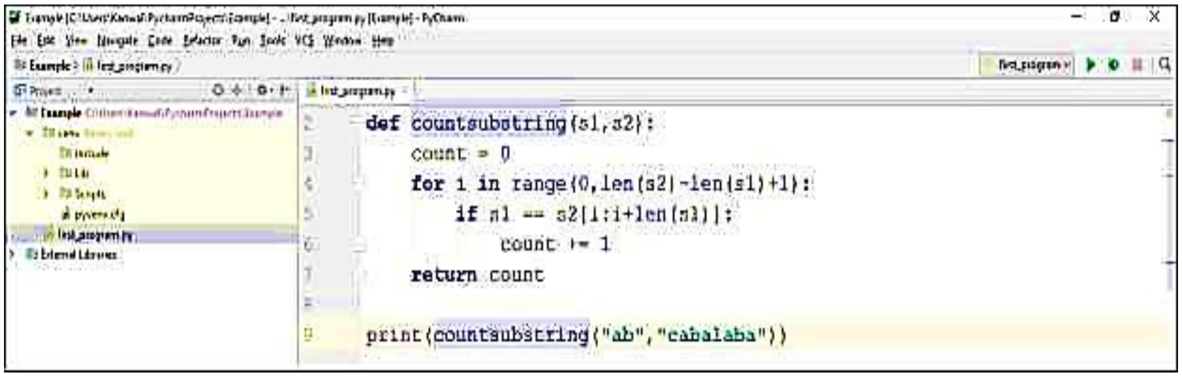

Example1 [C:\Users\Kanwal\PycharmProjects\Example1] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Example1
Project
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy Relative Path Ctrl+Alt+Shift+C
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Inspect Code...
Refactor
Clean Python Compiled Files
Add to Favorites
Show Image Thumbnails Ctrl+Shift+T
Local History
Synchronize 'Example1'
Show in Explorer
Directory Path Ctrl+Alt+F12
Compare With... Ctrl+D
Mark Directory as
Create Gist...
File
New Scratch File Ctrl+Alt+Shift+Insert
Directory
Python Package
Python File
Jupyter Notebook
HTML File
Resource Bundle
Search Everywhere Double Shift
Go to File Ctrl+Shift+N
Recent Files Ctrl+E
Navigation Bar Alt+Home
Drop files here to open
Creates a Python file from the specified template



Name: first_program

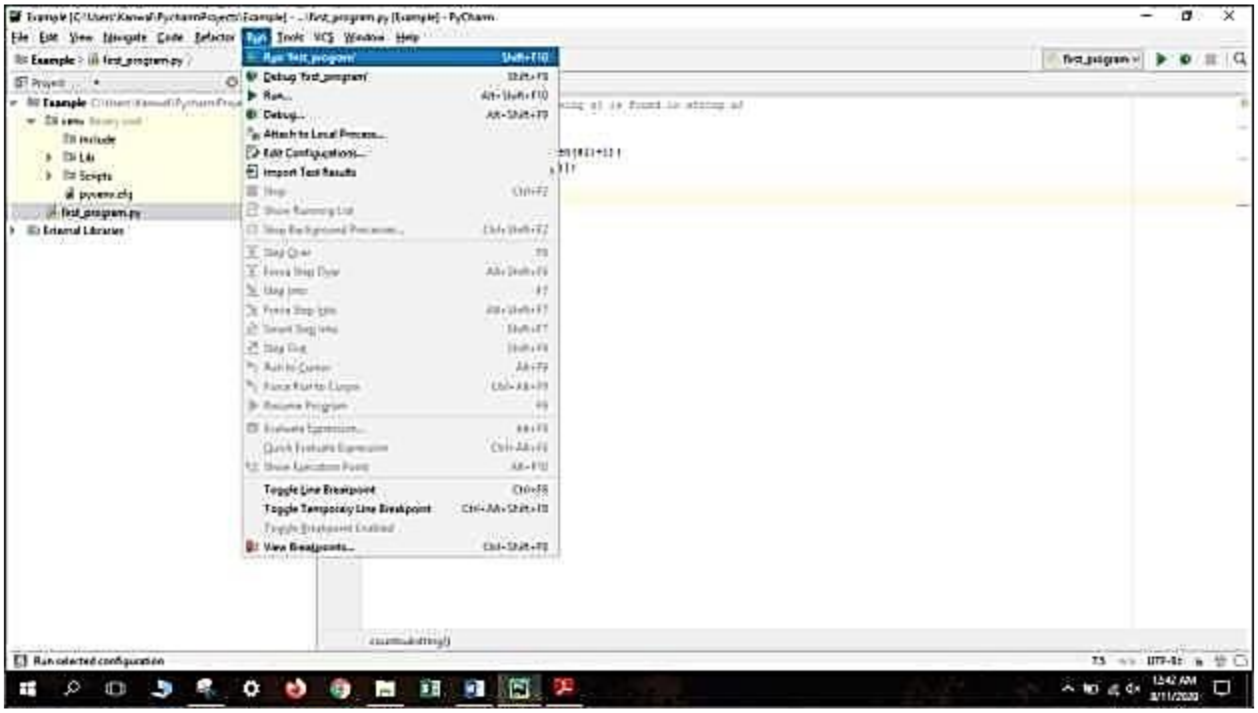
Kind: Python file

5. When file created write following code

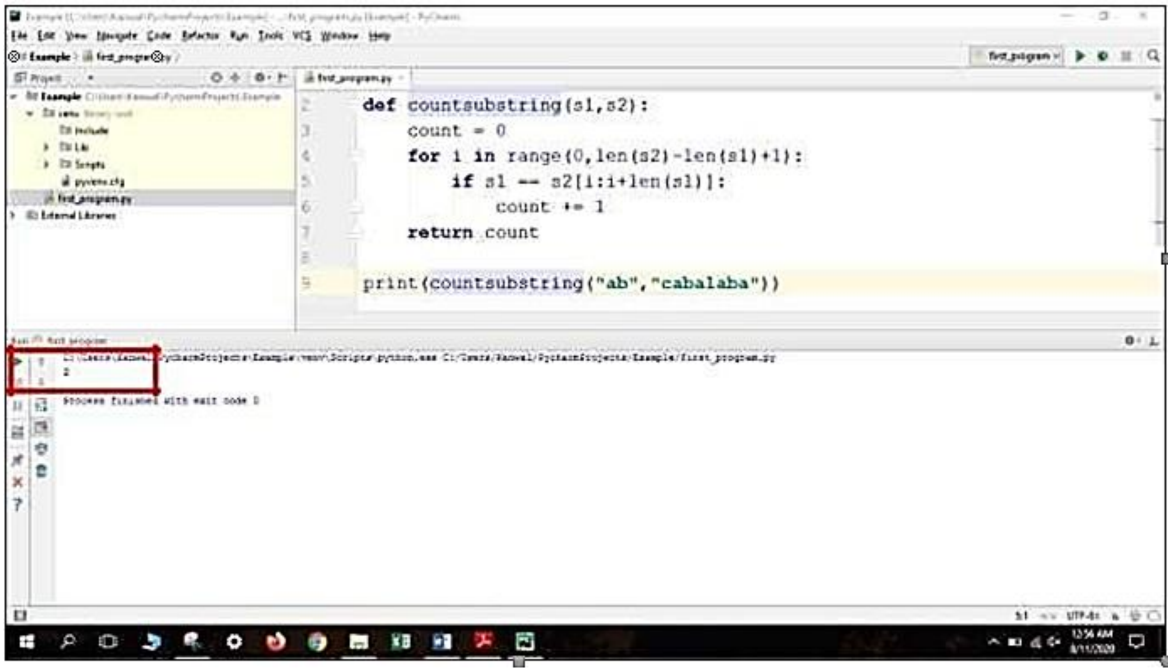


```
def countsubstring(s1,s2):
    count = 0
    for i in range(0,len(s2)-len(s1)+1):
        if s1 == s2[i:i+len(s1)]:
            count += 1
    return count

print(countsubstring("ab","cabalaba"))
```

6. Run file by clicking on Run in menu bar


File Edit View Navigate Code Refactor Run Tools VCS Window Help
Run 'first_program' Shift+F10
Debug 'first_program' Shift+F9
Run... Alt+Shift+F10
Debug... Alt+Shift+F9
Attach to Local Process...
Edit Configurations...
Import Test Results
Toggle Line Breakpoint Ctrl+F8
Toggle Temporary Line Breakpoint Ctrl+Alt+Shift+F8
View Breakpoints... Ctrl+Shift+F8
Run selected configuration

7. After that you can see output in output box. Function returned 2.



```
def countsubstring(s1,s2):
    count = 0
    for i in range(0,len(s2)-len(s1)+1):
        if s1 == s2[i:i+len(s1)]:
            count += 1
    return count

print(countsubstring("ab","cabalaba"))
```

```
C:\Users\Kanwal\PycharmProjects\Example\venv\Scripts\python.exe C:/Users/Kanwal/PycharmProjects/Example/first_program.py
2

Process finished with exit code 0
```

1.5. Evaluation Tasks

1.5.1. Task 1 (Marks 10)

Write a python program to print the multiplication table for a number and range given by a user?

For example: the number is 2 and the range is 11 to 13 and the output will be:

```
2        x        11        =        22
2        x        12        =        24
2        x        13        =        26
```

1.5.2. Task 2 (Marks 10)

Write a python program using **function** to check whether the positive integer is prime or not. The number will be given by the user that lies between 0 to 1000. For example, for input 5 the output will be displayed as “prime number”. For input 6, the output will be “not a prime number”.

1.5.3. Task 3 (Marks 10)

Write a Python program that accepts a positive integer (n) and computes the sum of squares from 1 to n. For example: n is 3 so the sum will be 14 and for n is 5 the sum will be 55.

1.6. Evaluation using Rubrics

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Criteria	**Level 0 (0%)**	**Level 1 (1-50)%**	**Level 2 (51-65)%**	**Level 3 (66-80)%**	**Level 4 (81-100)%**	**Total Score**
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

1.7. Out comes

After completing this lab, students will be able to learn Python syntax and implement different Python concepts in detail.

1.8. Further Readings

1.8.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

1.8.2. Links

https://pynative.com/python-basic-exercise-for-beginners/

https://www.w3resource.com/python-exercises/

Lab Manual for Artificial Intelligence

Lab-2: Python Libraries & Practices

2. Introduction to Python Libraries

A library is a collection of pre-written code that you can use to perform specific tasks. Libraries are often used to reduce the amount of code a programmer needs to write by providing reusable functions or classes that can be called upon as needed.

All libraries are divided into two classifications. All libraries are either Static or Dynamic.

- **Static:** These are compiled into a program's executable code, making it self-contained and independent of external changes. Static libraries are linked with a program at compile time, resulting in a single executable file that includes all of the code from the library. As a result, static libraries are often used to improve the performance of a program by avoiding the overhead of loading a library at runtime.

- **Dynamic:** Dynamic libraries are loaded at runtime rather than linked with a program at compile time. This allows programs to use the library without having to be re-compiled if the library is updated. In addition, dynamic libraries are often used to provide platform-specific functionality or to allow multiple programs to share a single copy of the library in memory.

You can write libraries in most programming languages, and they can be used by programs written in that same language or a different language. **For example**, a library written in C++ could be used by a program written in Python.

Libraries can perform various tasks, such as file input and output, data manipulation, network communication, and more. They can also provide access to system resources, such as the file system or hardware devices.

2.1. Python Libraries in AI

Python libraries play a significant role in making AI the technology they are today. Here are some of the well-known Python libraries that are leveraged in the development of Artificial Intelligence applications:

- **Numpy:** It is a popular Python library used to handle multi-dimensional data and complex mathematical functions being used on the data. The NumPy library powers the speed of computation of mathematical expressions and execution of complex functions working on arrays. Discretion of Fourier transformations, Statistical operations, and linear algebra, Support for n-dimensional arrays, Data cleaning and manipulation, and Random simulations.
- **Pandas:** Pandas is a prominent Python library generally used for Machine Learning concepts. It is basically a data analysis library that analyses and manipulates the data. Pandas make it easier for the developers to work with structured multidimensional data and time series concepts and produce efficient results, Data alignment and handling of missing data, merging and joining of datasets, Dataset reshaping and pivoting, data filtration, data manipulation and analysis, indexing of the data.
- **Matplotlib:** It is a data visualization library used for designing plots and graphs. The library itself is an extension of SciPy and handles complex data models of Pandas as well as NumPy data structures. Matplotlib offer features such as Basemap, GTK tools, Cartopy, Mplot4d, etc., that help in generating image plots, 3D plots, contour plots, and more. High-quality diagrams, plots, histograms, graphs, etc,Intuitive and easy to use,GUI toolkit support,Map projections,Recognition of data patterns.
- **SciPy:** It is a Python library that originates from NumPy. SciPy is leveraged by Python development services to perform technical and scientific computing on large sets of data. The library has built-in array optimization and linear algebra modules that help in

scientific analysis and engineering. Array manipulation subroutines, User-friendliness, Data visualization and manipulation, Scientific and technical analysis.

- **Scikit-learn**: It is a powerful Python library that was originally generated to serve the purpose of data modelling and building machine learning algorithms. It has a simple, engaging, and consistent interface that is exceptionally user-friendly, making it easy to use and share data. Data modelling, End-to-end Machine Learning algorithms, Model selection, Classification of data, Dimensionality reduction, and Pre-processing of the data.
- **TensorFlow**: It is an open-source ML library used for reaching data and production purposes. The library is offered by Google and can be used to make ML model building easier. TensorFlow offers a flexible framework and architecture that enables it to run on various computational platforms. However, it has its own tensor processing unit (TPU) through which the best results can be obtained. Creating deep learning models, Natural Language Processing, Abstraction capabilities, Managing deep neural networks, Image, text, and speech recognition.
- **Keras:** It is an open-source Python neural network library that is offered as an extension to the TensorFlow library. Keras is designed for building and evaluating neural networks within machine learning and deep learning models. The library is flexible, modular, and extensible and can be integrated with objectives, optimizers, layers, and activation functions. Activation and cost functions, Data pooling, developing neural layers, Batch normalization, and Building deep learning models.
- **PyTorch:** It is a highly popular production-ready Python library generally used in Machine Learning concepts. The library supports GPU acceleration and provides performance optimization to deep neural networks. PyTorch is mostly used for boosting the performance of deep learning frameworks and is supported by a vast Python community. Development of deep learning models, Matrix-vector multiplication,

Statistical distribution and operations, Greater control over datasets, and Python libraries are highly beneficial when working with AI technologies.

2.2. Objective of the Experiment

After completing this lab, the students should be able to:

- Understand different Python libraries, its usage and practical implementation.

2.3. Evaluation Tasks

2.3.1. Task 1 (Marks 10)

Write a code to create a basic line plot using matplotlib.

2.3.2. Task 2 (Marks 10)

Write a code to read a CSV file into a pandas data Frame?

Note: You can take any CSV file of your choice from the given link ⇩

https://www.kaggle.com/datasets?fileType=csv

2.4. Evaluation using Rubrics

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Criteria	**Level 0 (0%)**	**Level 1 (1-50)%**	**Level 2 (51-65)%**	**Level 3 (66-80)%**	**Level 4 (81-100)%**	**Total Score**
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

2.5. Out comes

After completing this lab, students will be able to learn Python syntax and implement different Python concepts in detail.

2.6. Further Readings

2.6.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

2.6.2. Links

https://www.edureka.co/blog/python-libraries/

Lab Manual for Artificial Intelligence

Lab-3: **Uninformed Search:** Breadth First Search, Depth First Search

3. Introduction

The objective of this lab is to introduce the concept of uninformed search algorithms and their usage. Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search. Following are some of the types of uninformed search algorithms:

- Breadth-first Search
- Depth-first Search

3.1. Breadth First Search (BFS)

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level. It uses a queue (First in First Out) instead of a stack and it checks whether a vertex has been discovered before enqueueing the vertex rather than delaying this check until the vertex is dequeued from the queue.

3.2. Depth First Search (DFS)

Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally print the nodes in the path.

3.3. Relevant Lecture Readings

Revise Lecture Uninformed search

3.4. Objectives of the experiment

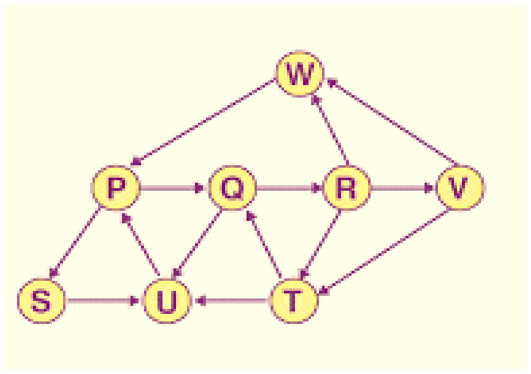
- To get knowledge of implementing Breadth First Search, Depth First Search and Uniform Cost Search

- To know the difference between the above mentioned uninformed search algorithms

3.5. Concept Map

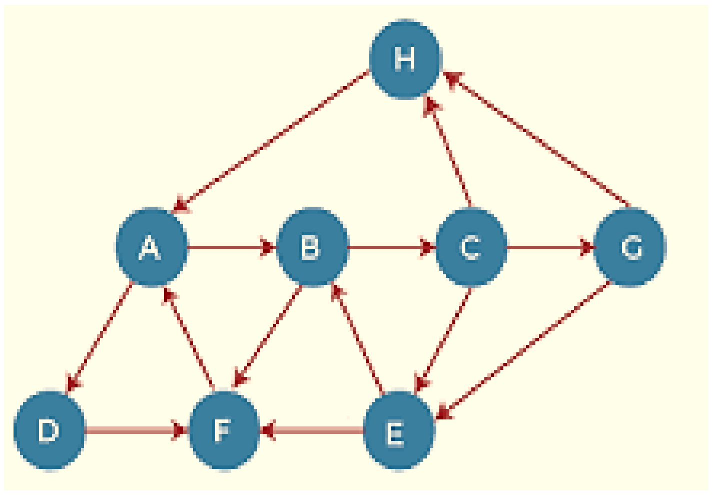
This concept map will help students to understand the main concepts of topics covered in the lab. In this lab, students will be provided with a detailed content to learn that how built in libraries can be used to implement concept of BFS, DFS and Uniform Cost Search in python.

3.6. Breadth First Search (BFS) Pseudo Code



```
BFS (G, s)    //Where G is the graph and s is the source node

   let Q be queue.

   Q.enqueue(s) //Inserting s in queue until all its neighbor vertices are marked.
   mark s as visited.
   while (Q is not empty)
//Removing that vertex from queue, whose neighbor will be visited
  now
   v = Q.dequeue( )//processing all the neighbours of v for all neighbours w of v in
Graph G if w is not visited
   Q.enqueue( w )//Stores w in Q to further visit its neighbor mark w as visited.
```

3.7. Depth First Search (DFS) Implementation



```
DFS-iterative (G, s)//Where G is graph and s is source vertex let S be
stack
        S.push(s )//Inserting s in stack mark s as visited.
        while (S is not empty):
        //Popa vertex from stack to visit next
        v= S.top( )
        S.pop( )
        //Push all the neighbours of v in stack that are not visited for
        all neighbours w of v in Graph G:
        if w is not visited :
        S.push( w ) //mark w as visited
        DFS-recursive(G, s) //mark s as visited
        for all neighbours w of s in Graph G: if w is not visited:
        DFS-recursive(G, w)
```

3.8. Homework before Lab

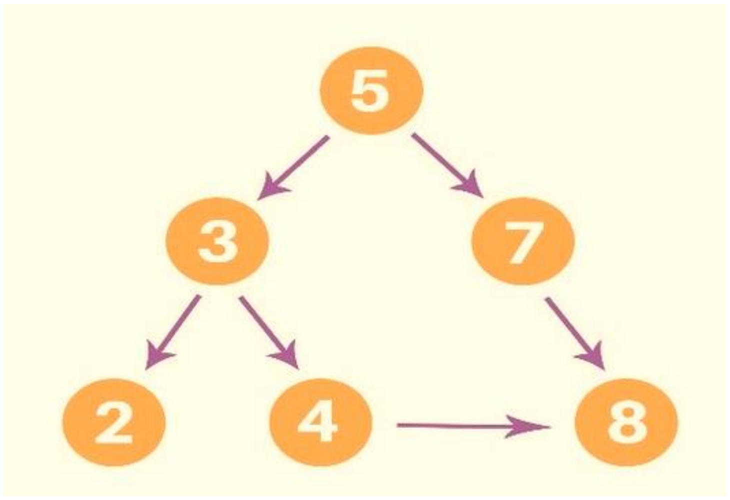
Revise the concepts of BFS and DFS.

3.9. Tools

Pycharm

3.10. Practice Task

Implementation of BFS in Python.



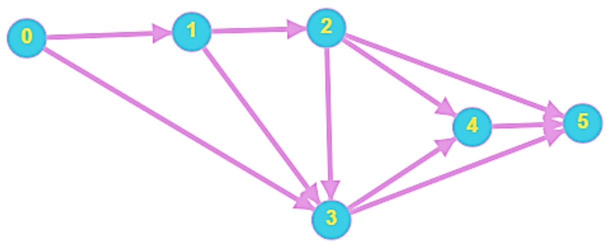
```python
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')    # function calling
```

3.11. Evaluation Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time.

3.11.1. Task 1 **(Marks 10)**

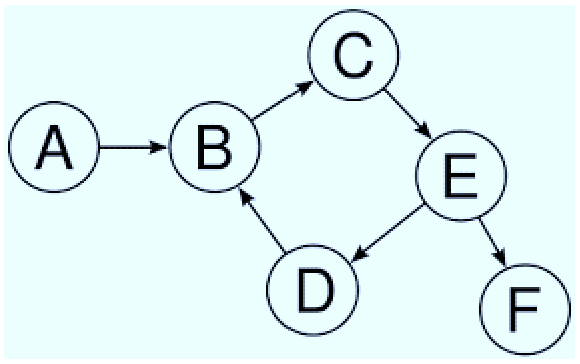
Write a Python function to perform topological sorting of a directed acyclic graph (DAG) using BFS. You are required to write a Python function to count the number of connected components in an undirected graph using BFS.



Test Case 1

3.11.2. Task 2 (Marks 10)

You are given the following tree whose starting node is **A** and Goal node is **F.**



You are required to implement the Depth First Search on the given graph in Python.

3.11.3. Task 3 (Marks 10)

Implement Water Jug Problem using the **DFS** in Python, initial stage (0,0) and final stage (2,Y). A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3- gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?

3.12. Evaluation using Rubrics

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Criteria	**Level 0 (0%)**	**Level 1 (1-50)%**	**Level 2 (51-65)%**	**Level 3 (66-80)%**	**Level 4 (81-100)%**	**Total Score**
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

3.12.1. Out comes

After completing this lab, students will know implementation of uninformed search i-e., BFS and DFS in Python.

3.13. Further Readings

3.13.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

3.13.2. Links

https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

https://pythoninwonderland.wordpress.com/2017/03/18/how-to-implement-breadth-first-search-in-python/

https://en.wikipedia.org/wiki/Depth-first_search

https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/

Lab Manual for Artificial Intelligence

Lab-04: A* Search

4. Introduction

In this lab you will learn about the concepts of A* Search. This algorithm is one of the best and popular technique used in path-finding and graph traversals. The idea is avoid expanding nodes which are expansive.

4.1. Relevant Lecture Readings

Revise Lectures A* Search Algorithm, Admissibility and Monotonicity conditions

4.2. Objective of the Experiment

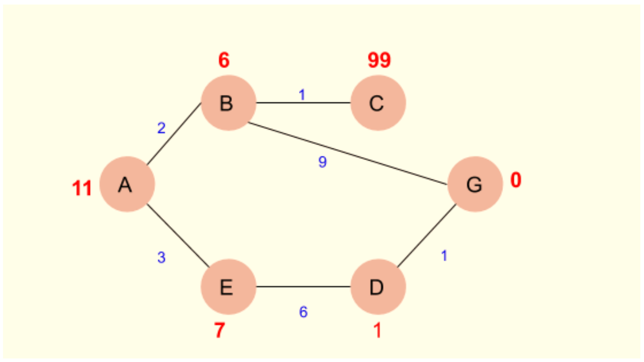
After completing this lab, the student should be able to:

- Clearly understand difference between uninformed searches and informed searches.
- Better understanding of heuristic function.
- Understand search mechanism where cost is involved.

4.3. Concept Map

This concept map will help students to understand the main concepts of topics covered in the lab. In this lab, students will be provided with a detailed content to learn that how built in libraries can be used to implement concept of A* search algorithm in python.

4.4. A* Search



1. Numbers written on edges represent the distance between nodes. Numbers written on nodes represent the heuristic value.

2. Given the graph, find the cost-effective path from A to G. That is A is the source node and G is the goal node.

3. Now from A, we can go to point B or E, so we compute f(x) for each of them,

4. $A \rightarrow B = g(B) + h(B) = 2 + 6 = 8$

5. $A \rightarrow E = g(E) + h(E) = 3 + 7 = 10$

6. Since the cost for $A \rightarrow B$ is less, we move forward with this path and compute the f(x) for the children nodes of B.

7. Now from B, we can go to point C or G, so we compute f(x) for each of them,

8. $A \rightarrow B \rightarrow C = (2 + 1) + 99= 102$

9. $A \rightarrow B \rightarrow G = (2 + 9) + 0 = 11$

10. Here the path $A \rightarrow B \rightarrow G$ has the least cost but it is still more than the cost of $A \rightarrow E$, thus we explore this path further.

11. Now from E, we can go to point D, so we compute f(x),

12. $A \rightarrow E \rightarrow D = (3 + 6) + 1 = 10$

13. Comparing the cost of $A \rightarrow E \rightarrow D$ with all the paths we got so far and as this cost is least of all we move forward with this path.

14. Now compute the f(x) for the children of D

15. A → E → D → G = (3 + 6 + 1) +0 = 10

16. Now comparing all the paths that lead us to the goal, we conclude that **A → E → D → G** is the most cost-effective path to get from A to

4.5. Homework before Lab

Attempt the following questions. You are required to bring this task with you and submit to the lab instructor.

4.5.1. Problem description

Design the solution of the following 8 puzzle problem given in figure 4.2. Your solution must contain all steps until goal state not find as described the solution design in section 4.1.

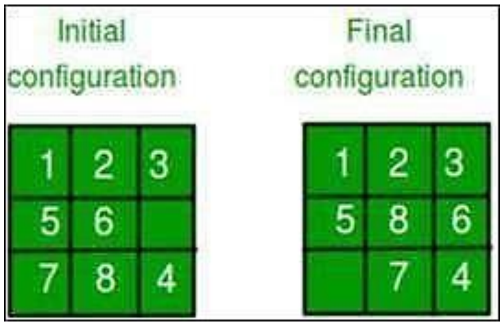


Figure 4.2: 8 Puzzle Problem

4.6. Procedure & Tools

4.6.1. Tools

Pycharm

4.7. Practice Task

This is a direct implementation of A* on a graph structure given below in Figure 4.3. The heuristic function is defined as 1 for all nodes for the sake of simplicity and brevity. The graph is represented with an adjacency list, where the keys represent graph nodes, and the values contain a list of edges with the corresponding neighboring nodes.

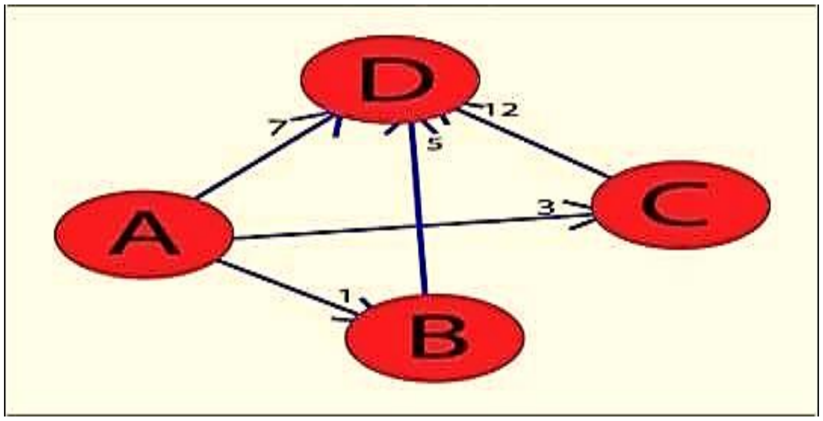
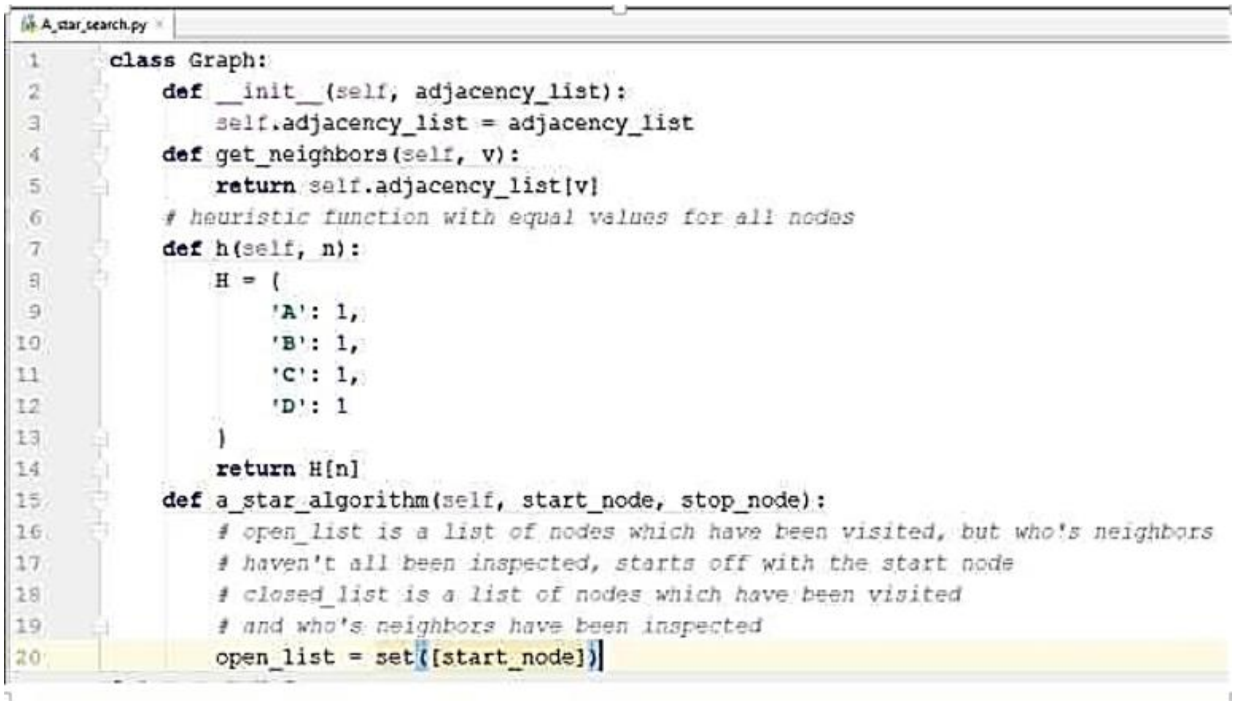
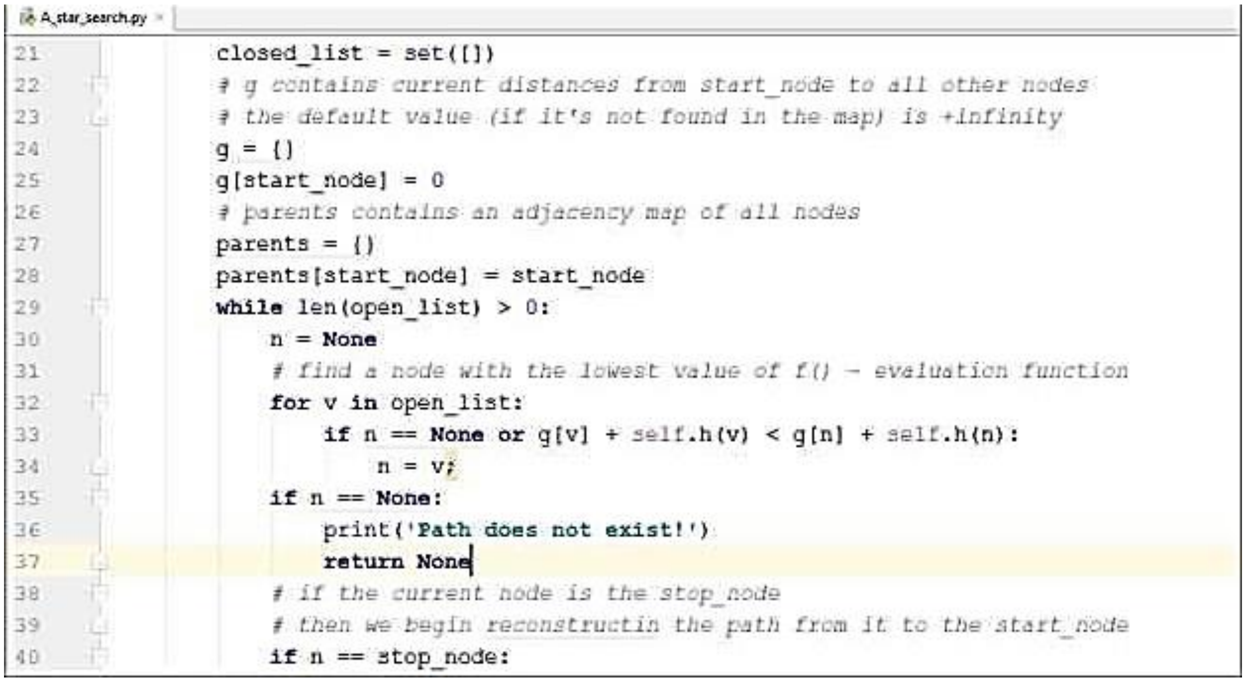


Figure 4.3

Write the given code in .py file in Pycharm and run it.

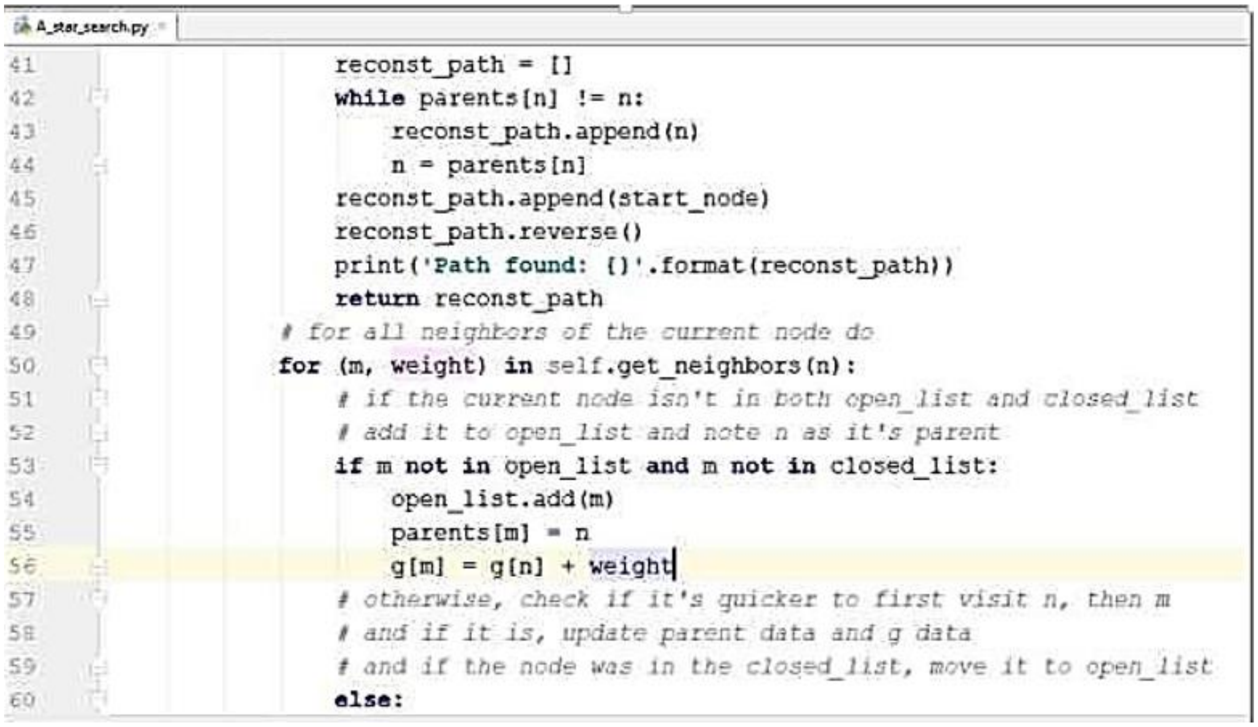


```
class Graph:
    def __init__(self, adjacency_list):
        self.adjacency_list = adjacency_list
    def get_neighbors(self, v):
        return self.adjacency_list[v]
    # heuristic function with equal values for all nodes
    def h(self, n):
        H = {
            'A': 1,
            'B': 1,
            'C': 1,
            'D': 1
        }
        return H[n]
    def a_star_algorithm(self, start_node, stop_node):
        # open_list is a list of nodes which have been visited, but who's neighbors
        # haven't all been inspected, starts off with the start node
        # closed_list is a list of nodes which have been visited
        # and who's neighbors have been inspected
        open_list = set([start_node])
```



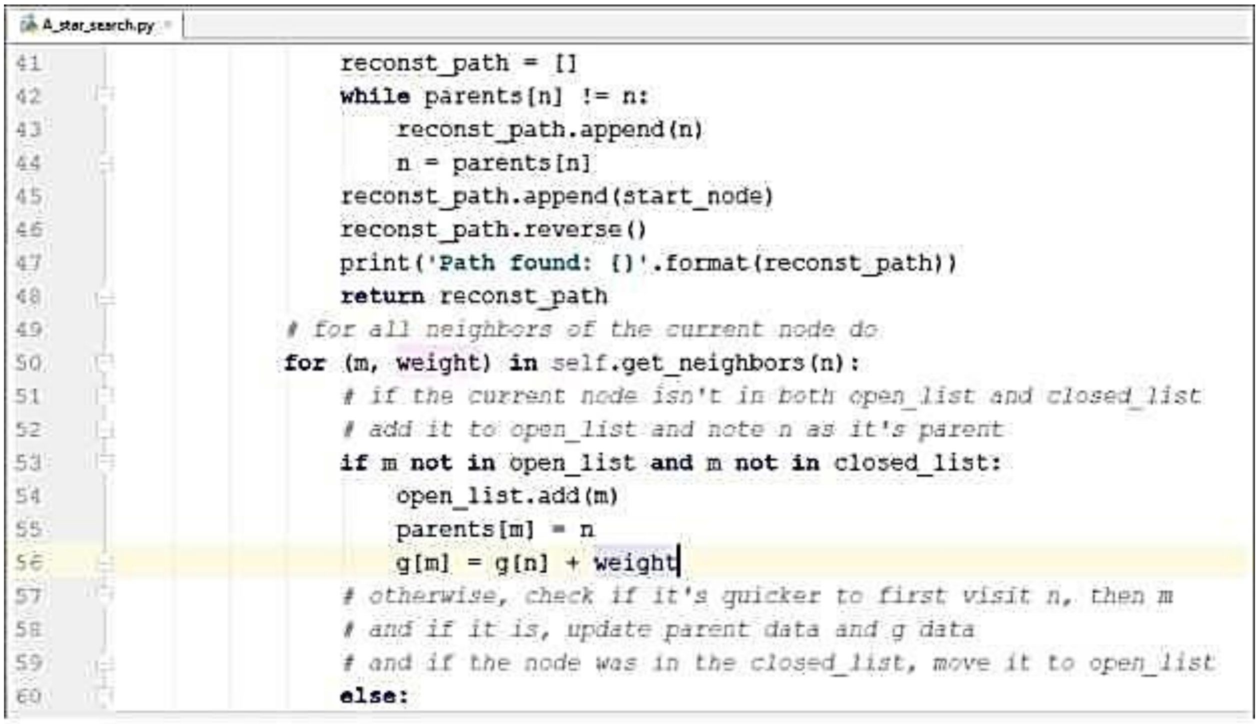
A_star_search.py

```
        closed_list = set([])
        # g contains current distances from start_node to all other nodes
        # the default value (if it's not found in the map) is +infinity
        g = {}
        g[start_node] = 0
        # parents contains an adjacency map of all nodes
        parents = {}
        parents[start_node] = start_node
        while len(open_list) > 0:
            n = None
            # find a node with the lowest value of f() - evaluation function
            for v in open_list:
                if n == None or g[v] + self.h(v) < g[n] + self.h(n):
                    n = v;
            if n == None:
                print('Path does not exist!')
                return None
            # if the current node is the stop_node
            # then we begin reconstructin the path from it to the start_node
            if n == stop_node:
```

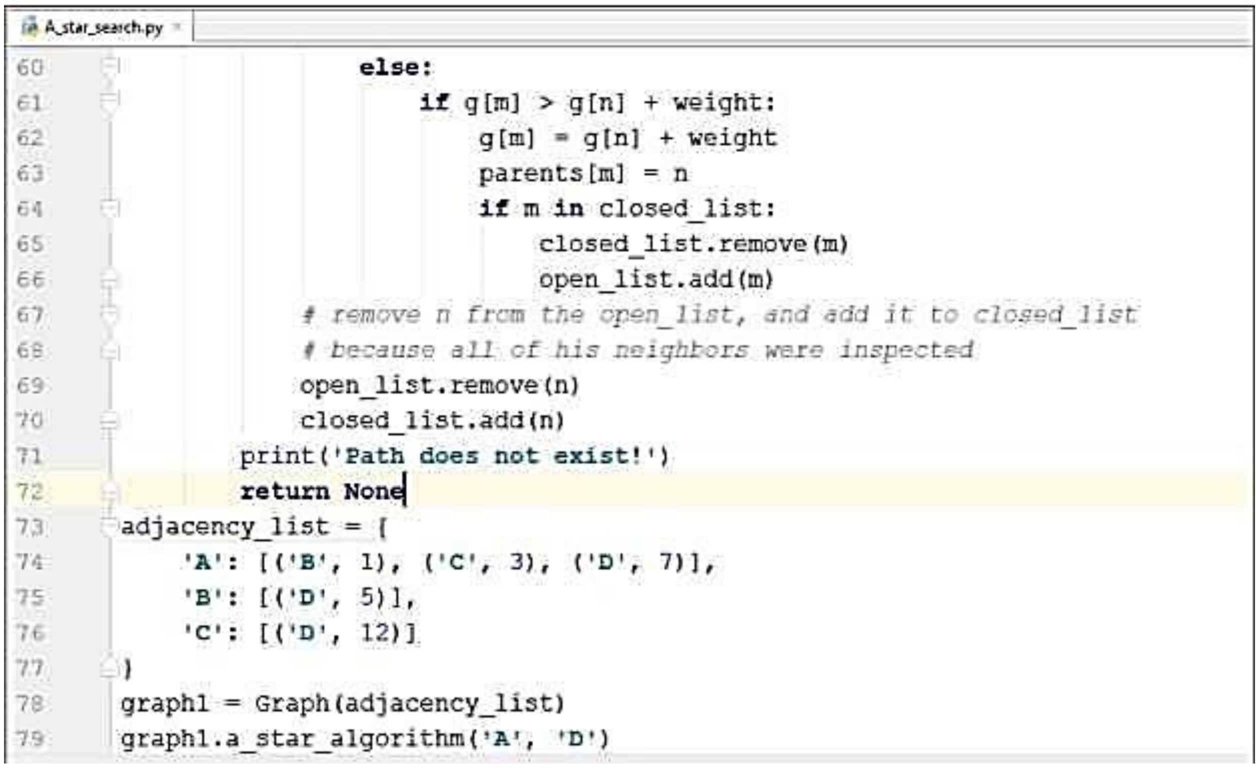


A_star_search.py

```
                reconst_path = []
                while parents[n] != n:
                    reconst_path.append(n)
                    n = parents[n]
                reconst_path.append(start_node)
                reconst_path.reverse()
                print('Path found: {}'.format(reconst_path))
                return reconst_path
            # for all neighbors of the current node do
            for (m, weight) in self.get_neighbors(n):
                # if the current node isn't in both open_list and closed_list
                # add it to open_list and note n as it's parent
                if m not in open_list and m not in closed_list:
                    open_list.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                # otherwise, check if it's quicker to first visit n, then m
                # and if it is, update parent data and g data
                # and if the node was in the closed_list, move it to open_list
                else:
```

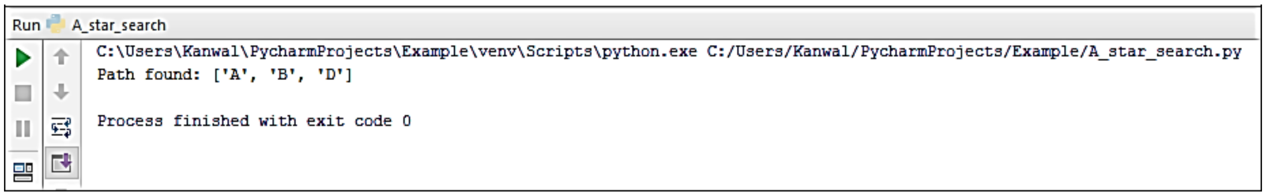

A_star_search.py

```
                reconst_path = []
                while parents[n] != n:
                    reconst_path.append(n)
                    n = parents[n]
                reconst_path.append(start_node)
                reconst_path.reverse()
                print('Path found: {}'.format(reconst_path))
                return reconst_path
            # for all neighbors of the current node do
            for (m, weight) in self.get_neighbors(n):
                # if the current node isn't in both open_list and closed_list
                # add it to open_list and note n as it's parent
                if m not in open_list and m not in closed_list:
                    open_list.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                # otherwise, check if it's quicker to first visit n, then m
                # and if it is, update parent data and g data
                # and if the node was in the closed_list, move it to open_list
                else:
```


A_star_search.py

```
                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n
                        if m in closed_list:
                            closed_list.remove(m)
                            open_list.add(m)
            # remove n from the open_list, and add it to closed_list
            # because all of his neighbors were inspected
            open_list.remove(n)
            closed_list.add(n)
        print('Path does not exist!')
        return None
adjacency_list = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')
```

OUTPUT



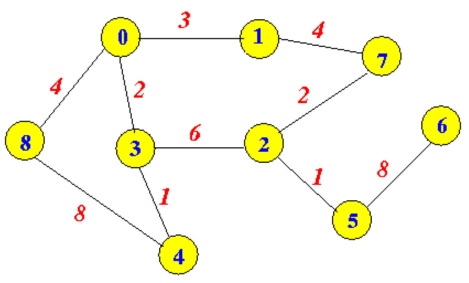
```
C:\Users\Kanwal\PycharmProjects\Example\venv\Scripts\python.exe C:/Users/Kanwal/PycharmProjects/Example/A_star_search.py
Path found: ['A', 'B', 'D']

Process finished with exit code 0
```

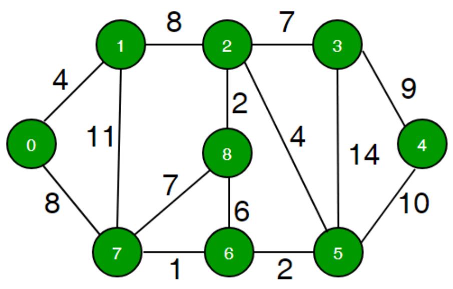
4.8. Evaluation Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time.

4.8.1. Task 1 (Marks 10)

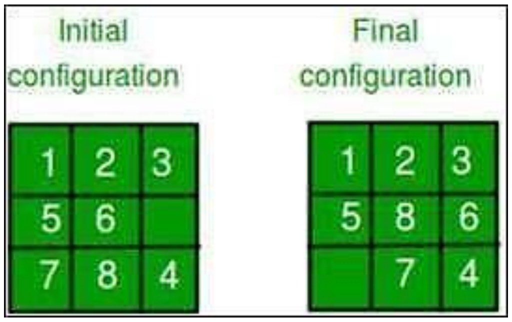
Implement the A* code in python on the following test cases (for help see the walkthrough task). In test case 1, the starting node is 8 and the last node is 6 whereas in test case 2 the starting node is 0 and the ending node is 4.





4.8.2. Task 2 (Marks 10)

Implement the 8 Puzzle Problem using A* search in Python.



The puzzle consists of an area divided into a grid, 3 by 3 for the 8-puzzle. On each grid square is a tile, expect for one square which remains empty. Thus, there are eight tiles in the 8-puzzle. A tile that is next to the empty grid square can be moved into the empty space, leaving its previous position empty in turn. Tiles are numbered, 1 thru 8 for the 8-puzzle, so that each tile can be uniquely identified. Heuristic for 8-puzzle problem is Number of Misplaced Tiles. You can design your own heuristic.

4.8.3. Out comes

After completing this lab, students will to know implementation of A* search algorithm.

4.9. Evaluation Criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Criteria	**Level 0 (0%)**	**Level 1 (1-50)%**	**Level 2 (51-65)%**	**Level 3 (66-80)%**	**Level 4 (81-100)%**	**Total Score**
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

4.10. Further Readings

4.10.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

4.10.2. Links

https://www.geeksforgeeks.org/a-search-algorithm/

https://www.annytab.com/a-star-search-algorithm-in-python/#:~:text=The%20A*%20search%20algorithm%20uses,heuristic%20to%20guide%20the%20search.

Lab Manual for Artificial Intelligence

Lab-05: Min-max Algorithm

5. Introduction

Games have always been an important application area for heuristic algorithms. In playing games whose state space may be exhaustively delineated, the primary difficulty is in accounting for the actions of the opponent. This can be handled easily by assuming that the opponent uses the same knowledge of the state space as us and applies that knowledge in a consistent effort to win the game. Minmax implements game search under referred to as MIN and MAX.

5.1. Relevant Lecture Readings

Revise Lectures Game Playing and Mini-max algorithm

5.2. Objective of the Experiment

After completing this lab, the student should be able to:

- Clearly understand the concept of Min-Max algorithm and able to use an optimization technique for min-max algorithm (Alpha-Beta).

5.3. Concept map

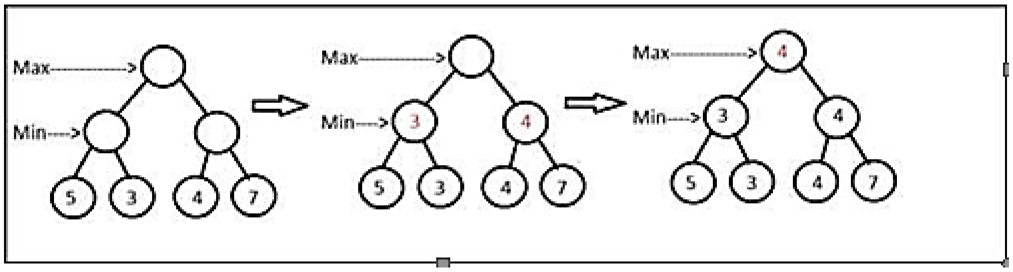
In this lab, we're going to discuss Minimax algorithm and its applications in Artificial Intelligence. As it's a game theory algorithm, we'll implement a simple game using it.

5.4. Min-Max Algorithm

Min-max is a decision-making algorithm, typically used in a turn-based, two player games. The goal of the algorithm is to find the optimal next move. In the algorithm, one player is called the maximizer, and the other player is a minimizer. If we assign an evaluation score to the game board, one player tries to choose a game state with the maximum score, while the other chooses a state with the minimum score.

In other words, the maximizer works to get the highest score, while the minimizer tries get the lowest score by trying to counter moves. Examples of such games are chess, poker, checkers, tic-tac-toe.

Our goal is to find the best move for the player. To do so, we can just choose the node with best evaluation score. To make the process smarter, we can also look ahead and evaluate potential opponent's moves. For each move, we can look ahead as many moves as our computing power allows. The algorithm assumes that the opponent is playing optimally. Technically, we start with the root node and choose the best possible node. We evaluate nodes based on their evaluation scores. In our case, evaluation function can assign scores to only result nodes (leaves). Therefore, we recursively reach leaves with scores and back propagate the scores. Consider the below game tree:



Maximizer starts with the root node and chooses the move with the maximum score. Unfortunately, only leaves have evaluation scores with them, and hence the algorithm has to reach leaf nodes recursively. In the given game tree, currently it's the minimizer's turn to choose a move from the leaf nodes, so the nodes with minimum scores (here, node 3 and 4) will get selected. It keeps picking the best nodes similarly, till it reaches the root node.

5.5. Home Task

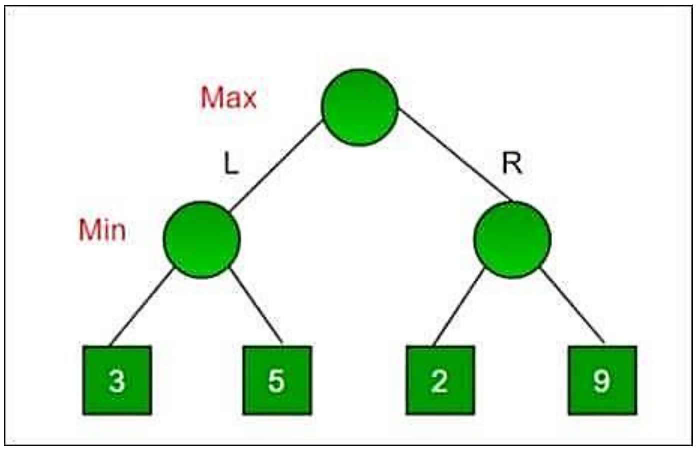
Read and understand the concept of minmax algorithm. Try to design the solution of any example on paper and bring it in lab.

5.6. Procedure & Tools

Pycharm

5.7. Practice Task

Consider a game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at next level.



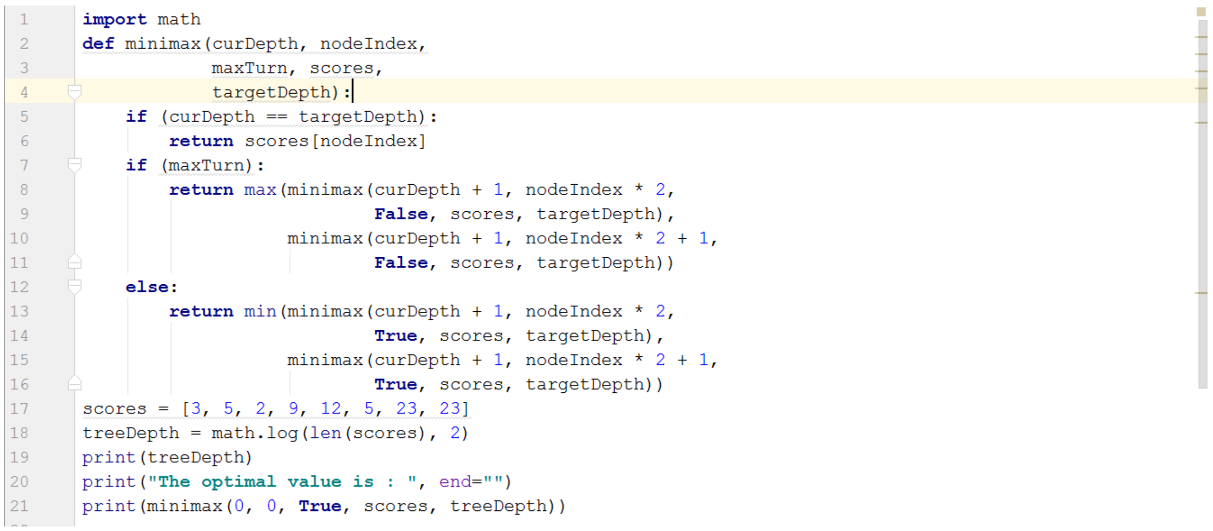
Since this is a backtracking based algorithm, it tries all possible moves, then backtracks and makes a decision.

- Maximizer goes LEFT: It is now the minimizers turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will definitely choose the least among both, that is 3
- Maximizer goes RIGHT: It is now the minimizers turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values.

Being the maximizer you would choose the larger value that is 3. Hence the optimal move for the maximizer is to go LEFT and the optimal value is 3.

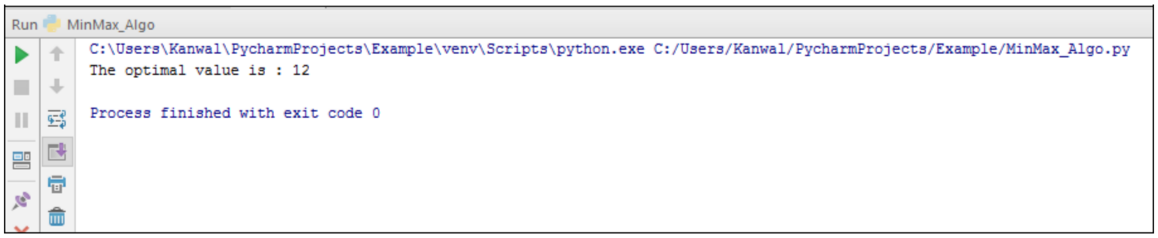
Now the game tree looks like the above figure, and the above tree shows two possible scores when maximizer makes left and right moves.

Note: Even though there is a value of 9 on the right sub tree, the minimizer will never pick that. We must always assume that our opponent plays optimally. Below is the implementation for the same.



```
import math
def minimax(curDepth, nodeIndex,
            maxTurn, scores,
            targetDepth):
    if (curDepth == targetDepth):
        return scores[nodeIndex]
    if (maxTurn):
        return max(minimax(curDepth + 1, nodeIndex * 2,
                           False, scores, targetDepth),
                   minimax(curDepth + 1, nodeIndex * 2 + 1,
                           False, scores, targetDepth))
    else:
        return min(minimax(curDepth + 1, nodeIndex * 2,
                           True, scores, targetDepth),
                   minimax(curDepth + 1, nodeIndex * 2 + 1,
                           True, scores, targetDepth))
scores = [3, 5, 2, 9, 12, 5, 23, 23]
treeDepth = math.log(len(scores), 2)
print(treeDepth)
print("The optimal value is : ", end="")
print(minimax(0, 0, True, scores, treeDepth))
```

Output:



```
C:\Users\Kanwal\PycharmProjects\Example\venv\Scripts\python.exe C:/Users/Kanwal/PycharmProjects/Example/MinMax_Algo.py
The optimal value is : 12

Process finished with exit code 0
```

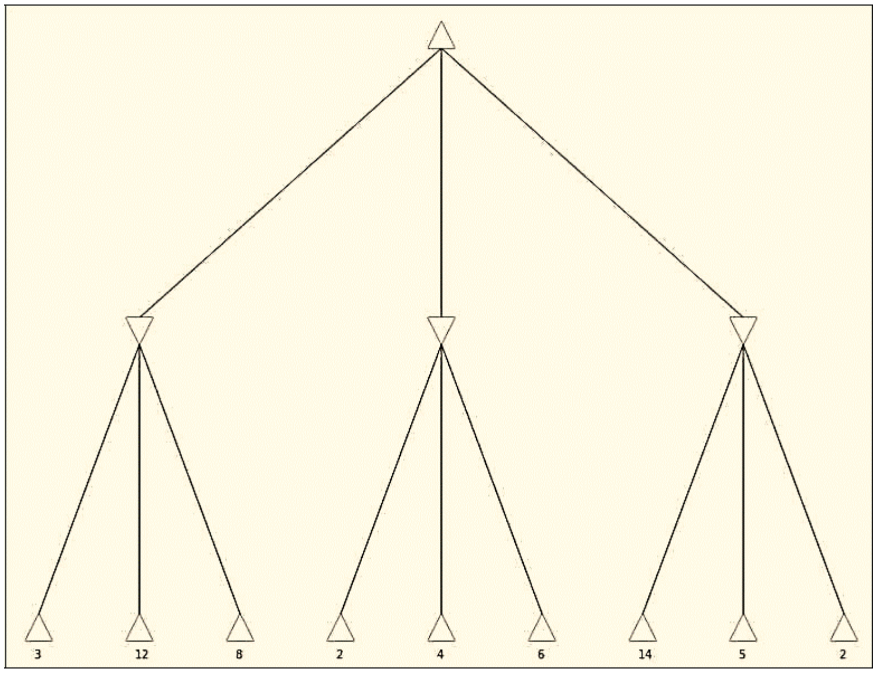
5.8. Evaluation Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time.

5.8.1. Task 1 **(Marks 10)**

Apply the following algorithms on the tree below (use python).

- Min-Max



5.9.1. Task 2 (Marks 10)

Implement tic-tac-toe game in python using min-max algorithm.

5.9.2. Out comes

After completing this lab, students will to know about Minimax algorithm and its applications in Artificial Intelligence.

5.10. Evaluation Criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Criteria	**Level 0 (0%)**	**Level 1 (1-50)%**	**Level 2 (51-65)%**	**Level 3 (66-80)%**	**Level 4 (81-100)%**	**Total Score**
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

5.11. Further Readings

5.11.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

5.11.2. Links

https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html

https://stackabuse.com/minimax-and-alpha-beta-pruning-in-python/

Lab Manual for Artificial Intelligence

Lab-06: Alpha-beta Pruning

6. Introduction

Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm. The basic idea behind this modification to the min-max search algorithm is the following. During the process of searching for the next move, not every move *(i.e. every node in the search tree)* needs to consider in order to reaching a correct decision. Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.

The two-parameter can be defined as:

- **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.

- **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.

The main condition which required for alpha-beta pruning is: **α>=β**

6.1. Relevant Lecture Readings

Revise Lecture alpha-beta pruning

6.2. Objective of the Experiment

After completing this lab, the student should be able to:

- Clearly understand the concept of Min-Max algorithm and able to use an optimization technique for min-max algorithm (Alpha-Beta).

6.3. Concept map

In this lab, we're going to the advantages of using the algorithm and see how it can be improved it using Alpha-Beta Pruning. As it's a game theory algorithm, we'll implement a simple game using it. We'll also discuss.

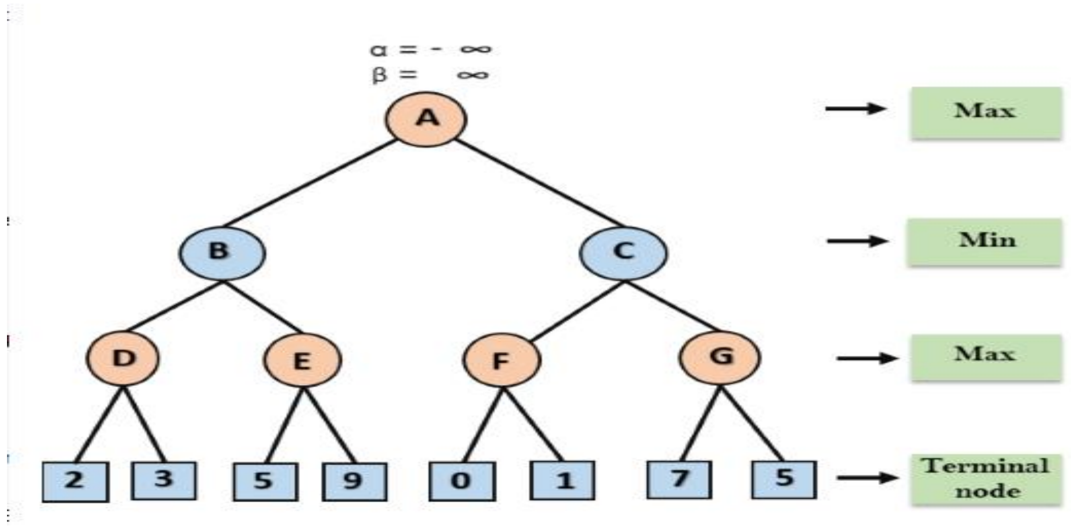
6.4. Alpha Beta Pseudo Code

```
function minimax(node, depth, alpha, beta, maximizingPlayer) is
if depth ==0 or node is a terminal node then
return static evaluation of node

if MaximizingPlayer then      // for Maximizer Player
   maxEva= -infinity
   for each child of node do
   eva= minimax(child, depth-1, alpha, beta, False)
  maxEva= max(maxEva, eva)
  alpha= max(alpha, maxEva)
   if beta<=alpha
 break
 return maxEva

else                     // for Minimizer player
   minEva= +infinity
   for each child of node do
   eva= minimax(child, depth-1, alpha, beta, true)
   minEva= min(minEva, eva)
   beta= min(beta, eva)
    if beta<=alpha
  break
 return minEva
```
.

6.5. Working of Alpha-Beta Pruning

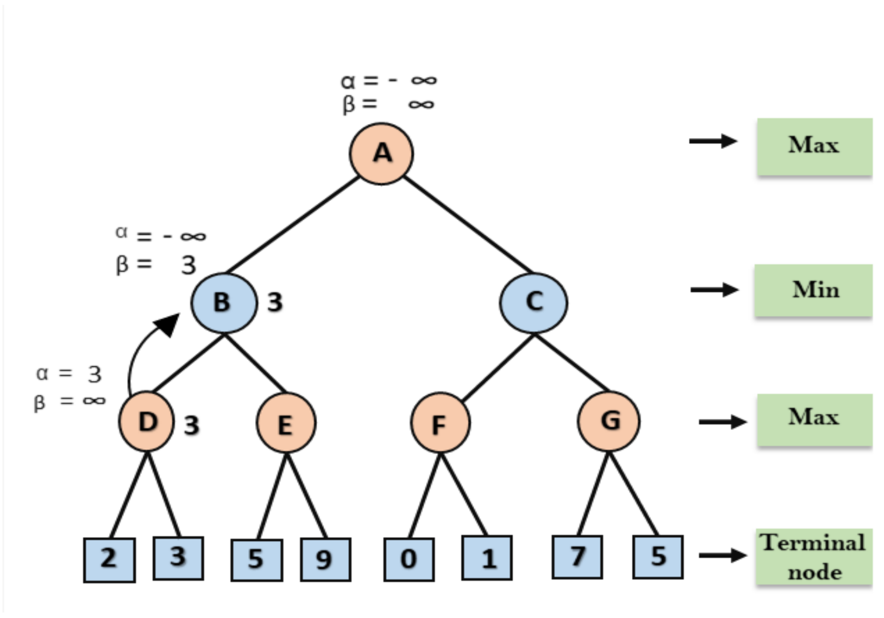
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning.

Step 1: At the first step the Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



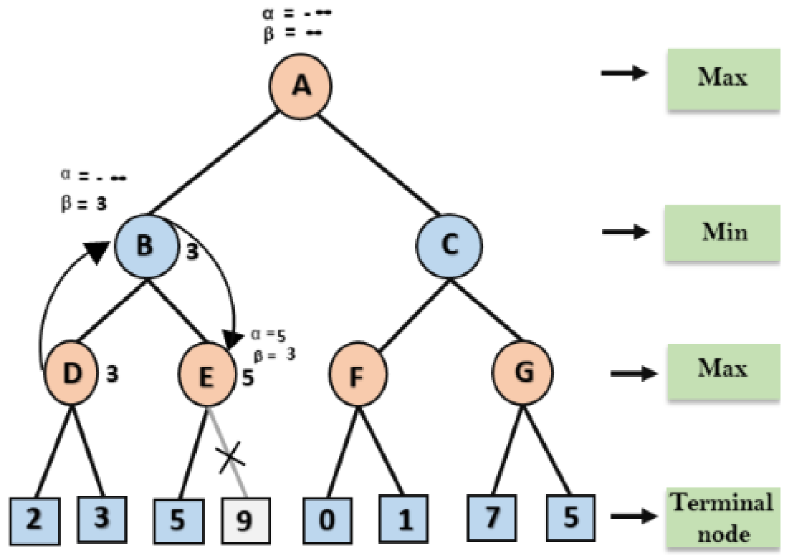
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

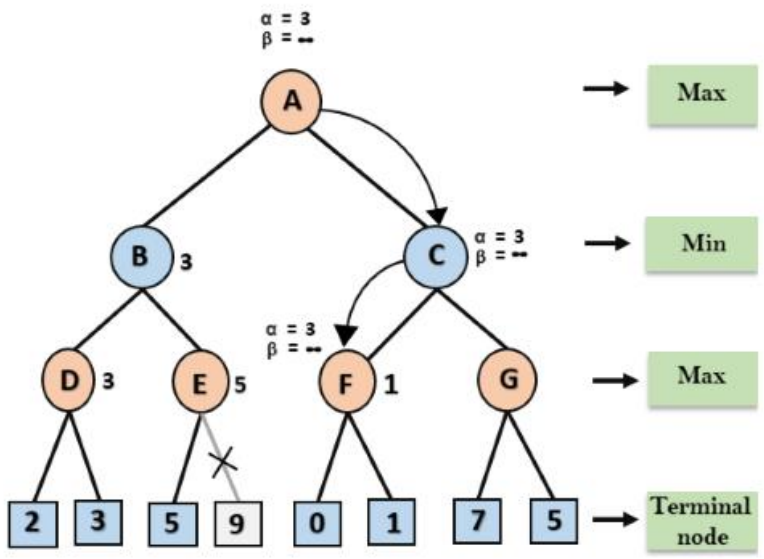
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



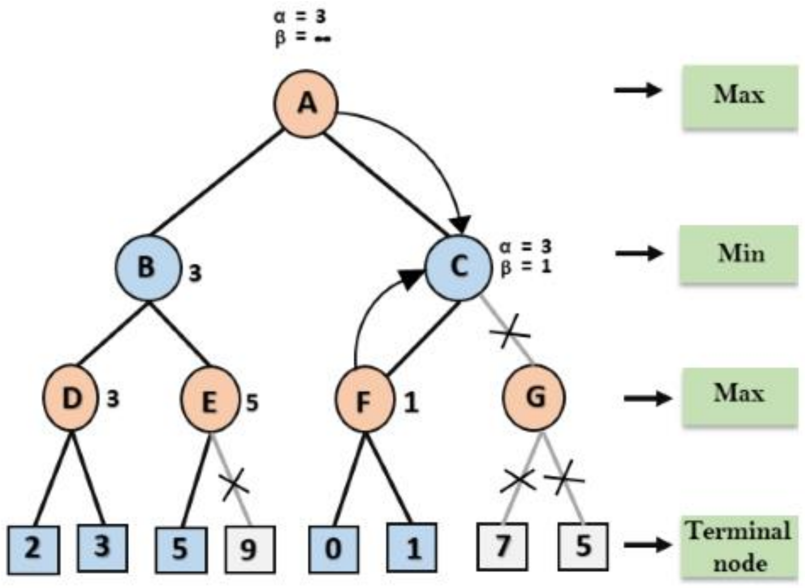
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.

6.6. Home Task

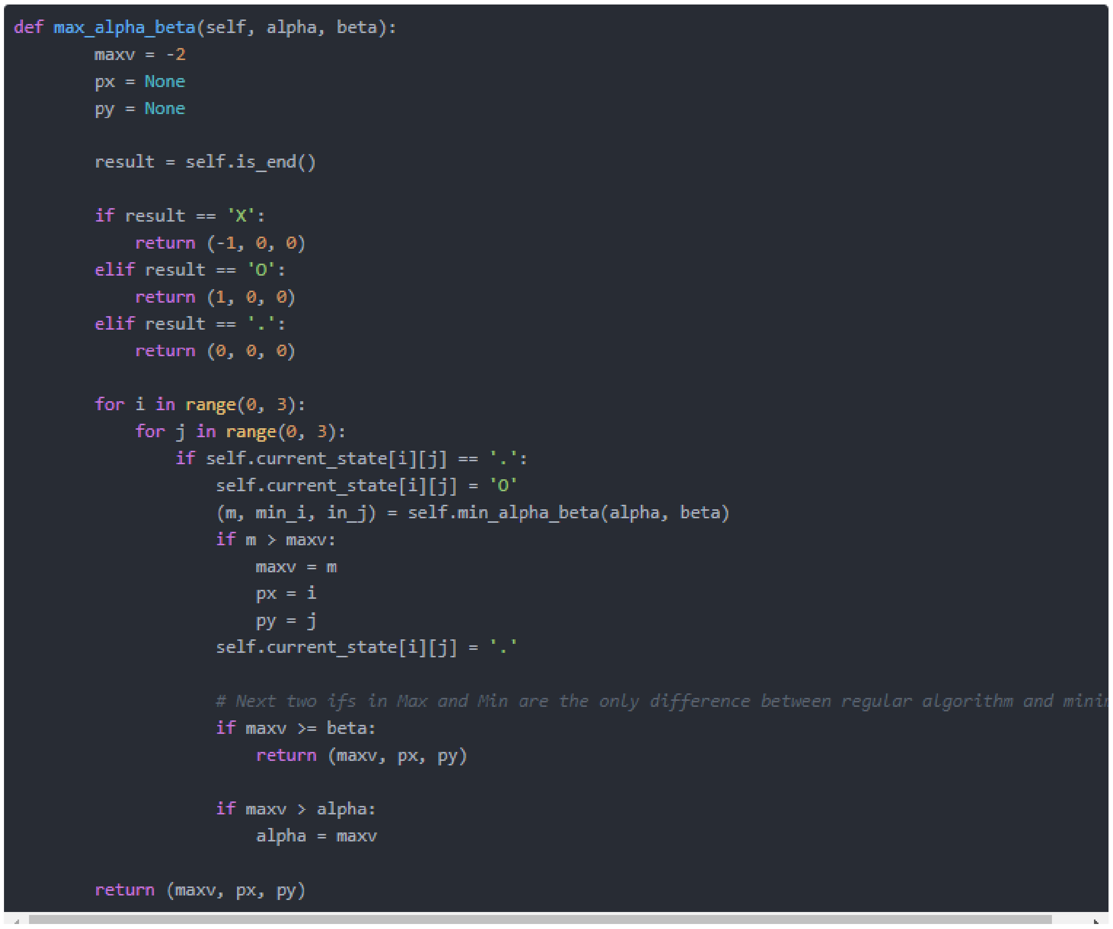
Read and understand the concept of alpha beta pruning. Try to design the solution of any example on paper and bring it in lab.

6.7. Procedure & Tools

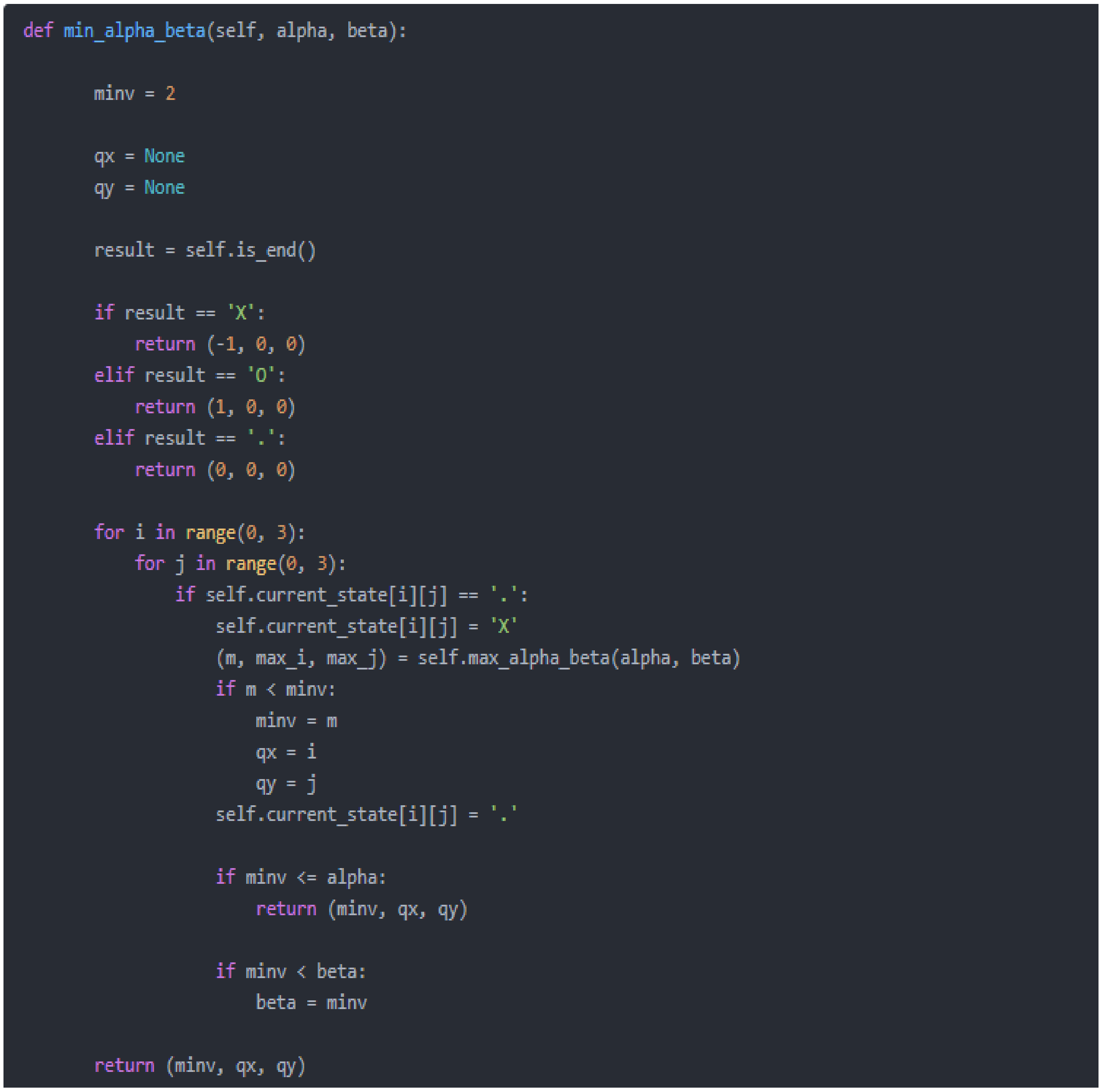
Pycharm

6.8. Practice Task

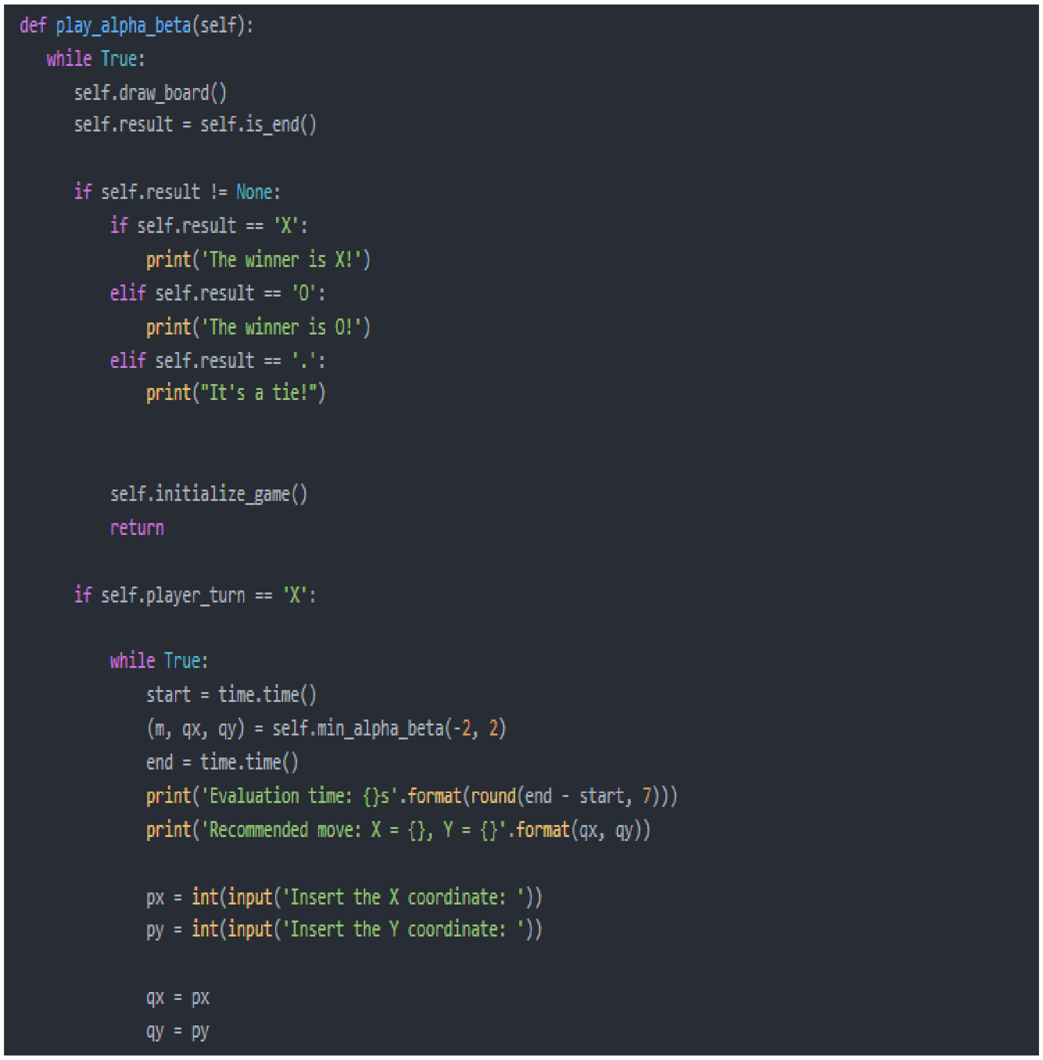
Here is the implementation of alpha beta pruning.



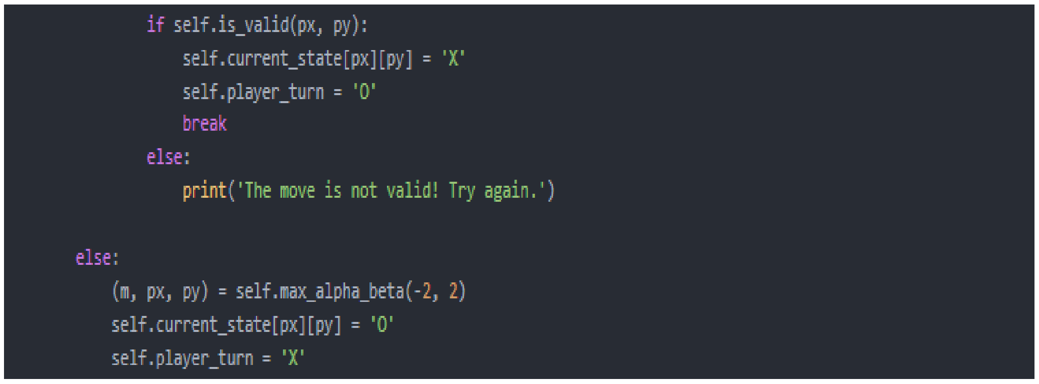
```python
def max_alpha_beta(self, alpha, beta):
        maxv = -2
        px = None
        py = None

        result = self.is_end()

        if result == 'X':
            return (-1, 0, 0)
        elif result == 'O':
            return (1, 0, 0)
        elif result == '.':
            return (0, 0, 0)

        for i in range(0, 3):
            for j in range(0, 3):
                if self.current_state[i][j] == '.':
                    self.current_state[i][j] = 'O'
                    (m, min_i, in_j) = self.min_alpha_beta(alpha, beta)
                    if m > maxv:
                        maxv = m
                        px = i
                        py = j
                    self.current_state[i][j] = '.'

                    # Next two ifs in Max and Min are the only difference between regular algorithm and minin
                    if maxv >= beta:
                        return (maxv, px, py)

                    if maxv > alpha:
                        alpha = maxv

        return (maxv, px, py)
```



```python
def min_alpha_beta(self, alpha, beta):

       minv = 2

       qx = None
       qy = None

       result = self.is_end()

       if result == 'X':
           return (-1, 0, 0)
       elif result == 'O':
           return (1, 0, 0)
       elif result == '.':
           return (0, 0, 0)

       for i in range(0, 3):
           for j in range(0, 3):
               if self.current_state[i][j] == '.':
                   self.current_state[i][j] = 'X'
                   (m, max_i, max_j) = self.max_alpha_beta(alpha, beta)
                   if m < minv:
                       minv = m
                       qx = i
                       qy = j
                   self.current_state[i][j] = '.'

                   if minv <= alpha:
                       return (minv, qx, qy)

                   if minv < beta:
                       beta = minv

       return (minv, qx, qy)
```

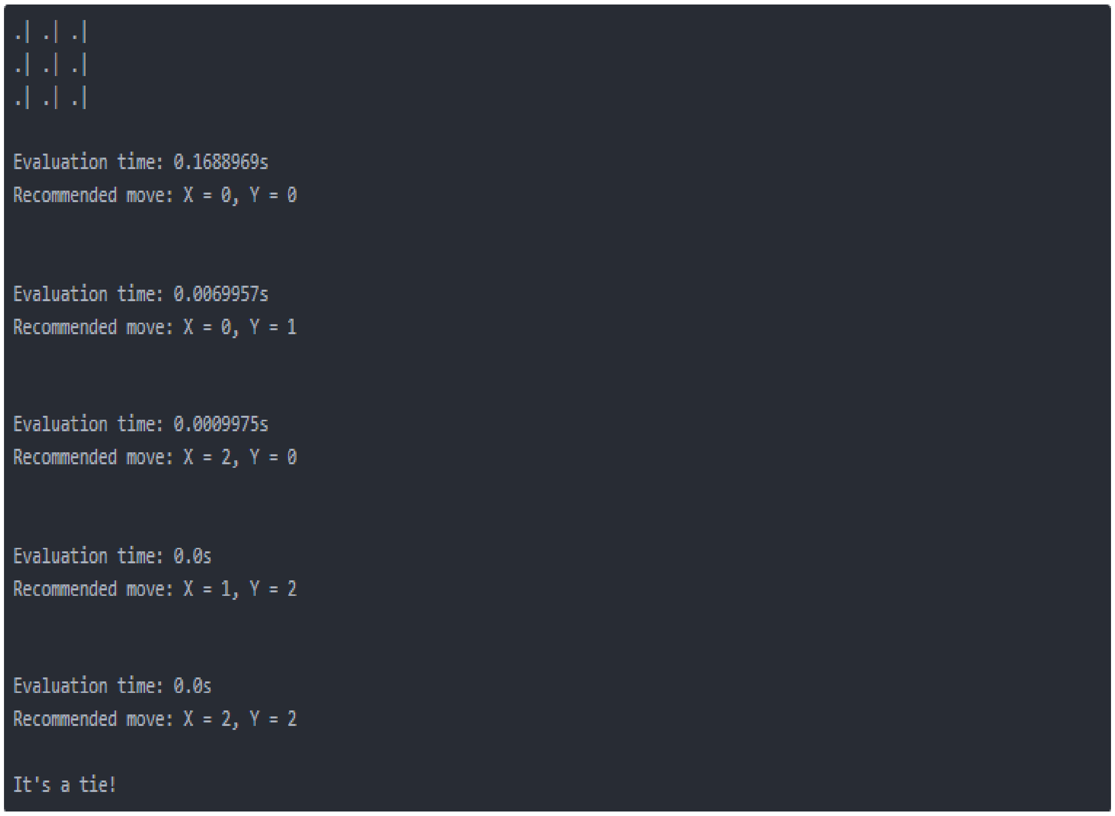


```python
def play_alpha_beta(self):
    while True:
        self.draw_board()
        self.result = self.is_end()

        if self.result != None:
            if self.result == 'X':
                print('The winner is X!')
            elif self.result == 'O':
                print('The winner is O!')
            elif self.result == '.':
                print("It's a tie!")


            self.initialize_game()
            return

        if self.player_turn == 'X':

            while True:
                start = time.time()
                (m, qx, qy) = self.min_alpha_beta(-2, 2)
                end = time.time()
                print('Evaluation time: {}s'.format(round(end - start, 7)))
                print('Recommended move: X = {}, Y = {}'.format(qx, qy))

                px = int(input('Insert the X coordinate: '))
                py = int(input('Insert the Y coordinate: '))

                qx = px
                qy = py
```



```python
                if self.is_valid(px, py):
                    self.current_state[px][py] = 'X'
                    self.player_turn = 'O'
                    break
                else:
                    print('The move is not valid! Try again.')

        else:
            (m, px, py) = self.max_alpha_beta(-2, 2)
            self.current_state[px][py] = 'O'
            self.player_turn = 'X'
```

Output:



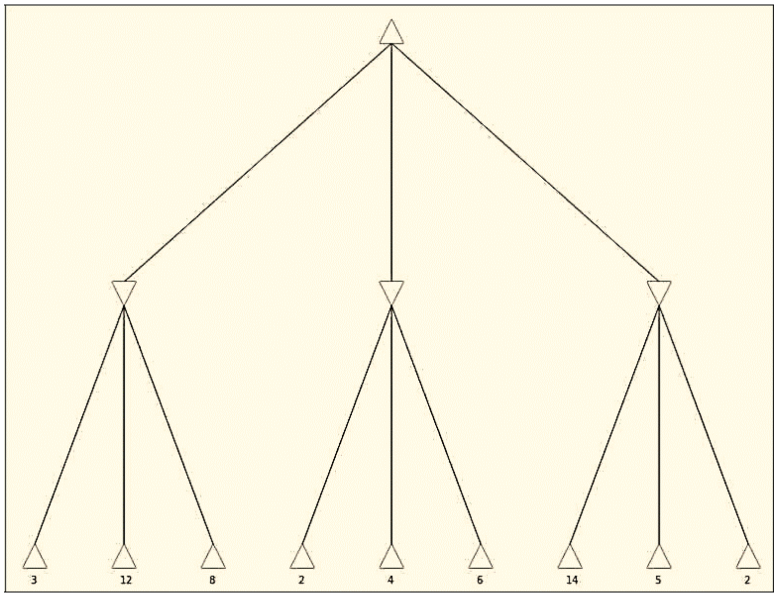
```
.| .| .|
.| .| .|
.| .| .|

Evaluation time: 0.1688969s
Recommended move: X = 0, Y = 0


Evaluation time: 0.0069957s
Recommended move: X = 0, Y = 1


Evaluation time: 0.0009975s
Recommended move: X = 2, Y = 0


Evaluation time: 0.0s
Recommended move: X = 1, Y = 2


Evaluation time: 0.0s
Recommended move: X = 2, Y = 2

It's a tie!
```

6.9. Evaluation Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time.

6.9.1. Task 1 (Marks 10)

Apply the following algorithm on the tree below (use python).



6.9.2. Task 2 (Marks 10)

How can you represent a game tree in Python for a game like Connect Four or Checkers? Implement the alpha-beta pruning algorithm for one of these games.

6.9.3. Outcome

After completing this lab, students will to know about Minimax algorithm and its applications in Artificial Intelligence

6.10. Evaluation Criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Criteria	Level 0 (0%)	Level 1 (1-50)%	Level 2 (51-65)%	Level 3 (66-80)%	Level 4 (81-100)%	Total Score
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

6.11. Further Readings

6.11.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

6.11.2. Links

https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html

https://stackabuse.com/minimax-and-alpha-beta-pruning-in-python/

Lab Manual for Artificial Intelligence

LAB-07: Hill Climbing

7. Introduction

In the field of AI, many complex algorithms have been used. It is also important to find out an optimal solution. Hill climbing algorithm is one such optimization algorithm used in the field of Artificial Intelligence. It is a mathematical method which optimizes only the neighboring points and is considered to be heuristic. A heuristic method is one of those methods which does not guarantee the best optimal solution. This algorithm belongs to the local search family.

Local search algorithms are used on complex optimization problems where it tries to find out a solution that maximizes the criteria among candidate solutions. A candidate solution is considered to be the set of all possible solutions in the entire functional region of a problem.

7.1. Relevant Lecture Readings

Revise Lectures Hill Climbing Algorithm (Simple hill Climbing, Steepest-Ascent hill climbing)

7.2. Objective of the Experiment

After completing this lab, the student should be able to:

- Clearly understand Hill climbing problems in artificial intelligence.

7.3. Concept Map

This concept map will help students to understand the main concepts of topics covered in the lab. In this lab, students will be provided with a detailed content to learn that what the concept of hill climbing is and how to implement it in python.

7.4. Overview

Hill Climbing is a heuristic search used for mathematical optimization problems in the field of Artificial Intelligence.

Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum.

7.5. Working Process

This algorithm works on the following steps in order to find an optimal solution.

- It tries to define the current state as the state of starting or the initial state.
- It generalizes the solution to the current state and tries to find an optimal solution. The solution obtained may not be the best.
- It compares the solution which is generated to the final state also known as the goal state.
- It will check whether the final state is achieved or not. If not achieved, it will try to find another solution.

7.6. Types of Hill Climbing

There are various types of Hill Climbing which are-

- Simple Hill climbing
- Steepest-Ascent Hill climbing
- Stochastic Hill climbing

7.7. Applications of Hill climbing technique

7.7.1. Robotics

Hill climbing is mostly used in robotics which helps their system to work as a team and maintain coordination.

7.7.2. Marketing

The algorithm can be helpful in team management in various marketing domains where hill climbing can be used to find an optimal solution. The travelling time taken by a sale member or the place he visited per day can be optimized using this algorithm.

7.8. Homework before Lab

Attempt the following questions. You are required to bring this task with you and submit to the lab instructor.

7.8.1. Problem description

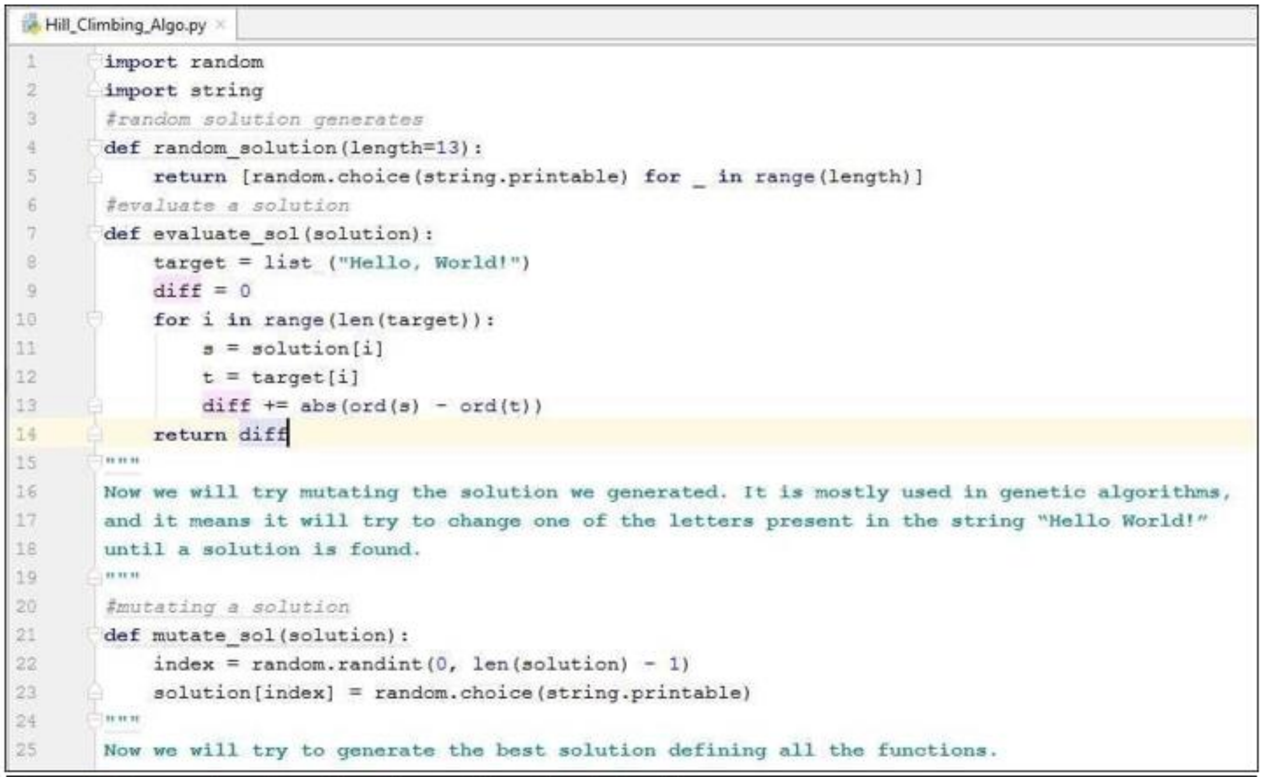
Read and under the detailed concepts of hill climbing algorithm and their usage in AI field.

7.9. Procedure & Tools

Pycharm

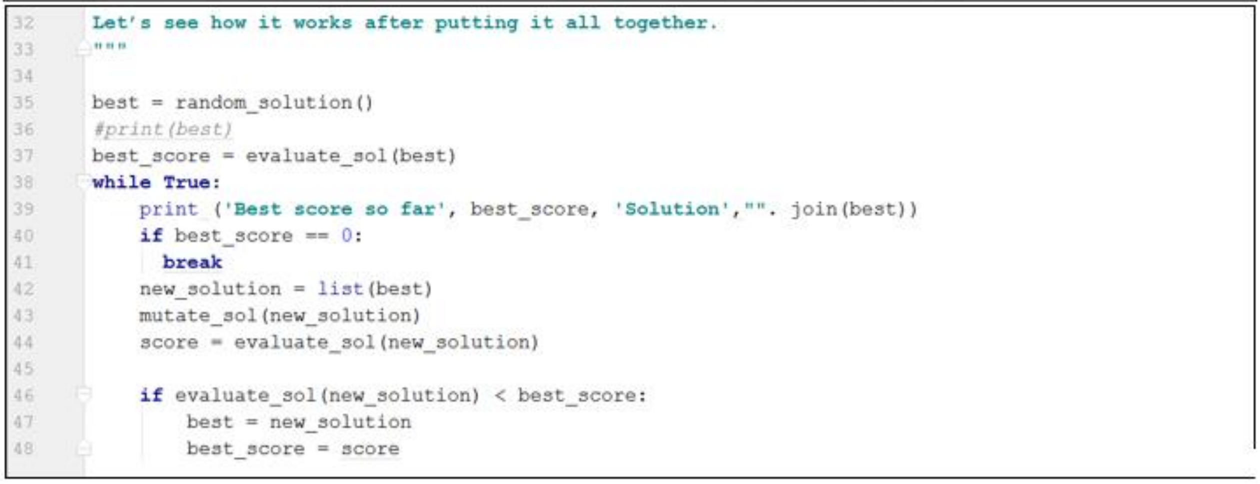
7.10. Practice Task

We will perform a simple study in Hill Climbing on a greeting “Hello World!”. We will see how the hill climbing algorithm works on this. We will generate random solutions and evaluate our solution. If the solution is the best one, our algorithm stops; else it will move forward to the next step. Write the given code in .py file and run it.



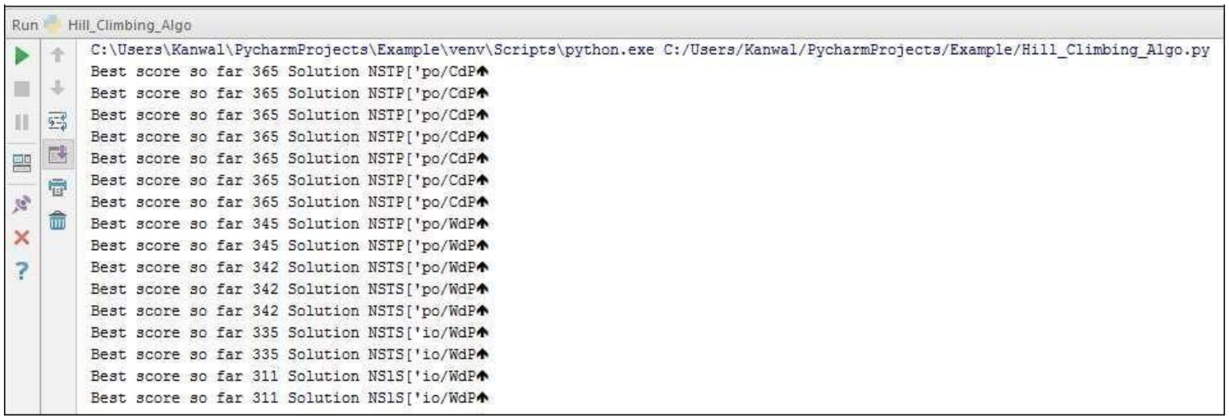
Hill_Climbing_Algo.py

```
import random
import string
#random solution generates
def random_solution(length=13):
    return [random.choice(string.printable) for _ in range(length)]
#evaluate a solution
def evaluate_sol(solution):
    target = list ("Hello, World!")
    diff = 0
    for i in range(len(target)):
        s = solution[i]
        t = target[i]
        diff += abs(ord(s) - ord(t))
    return diff
"""
Now we will try mutating the solution we generated. It is mostly used in genetic algorithms,
and it means it will try to change one of the letters present in the string "Hello World!"
until a solution is found.
"""
#mutating a solution
def mutate_sol(solution):
    index = random.randint(0, len(solution) - 1)
    solution[index] = random.choice(string.printable)
"""
Now we will try to generate the best solution defining all the functions.
```



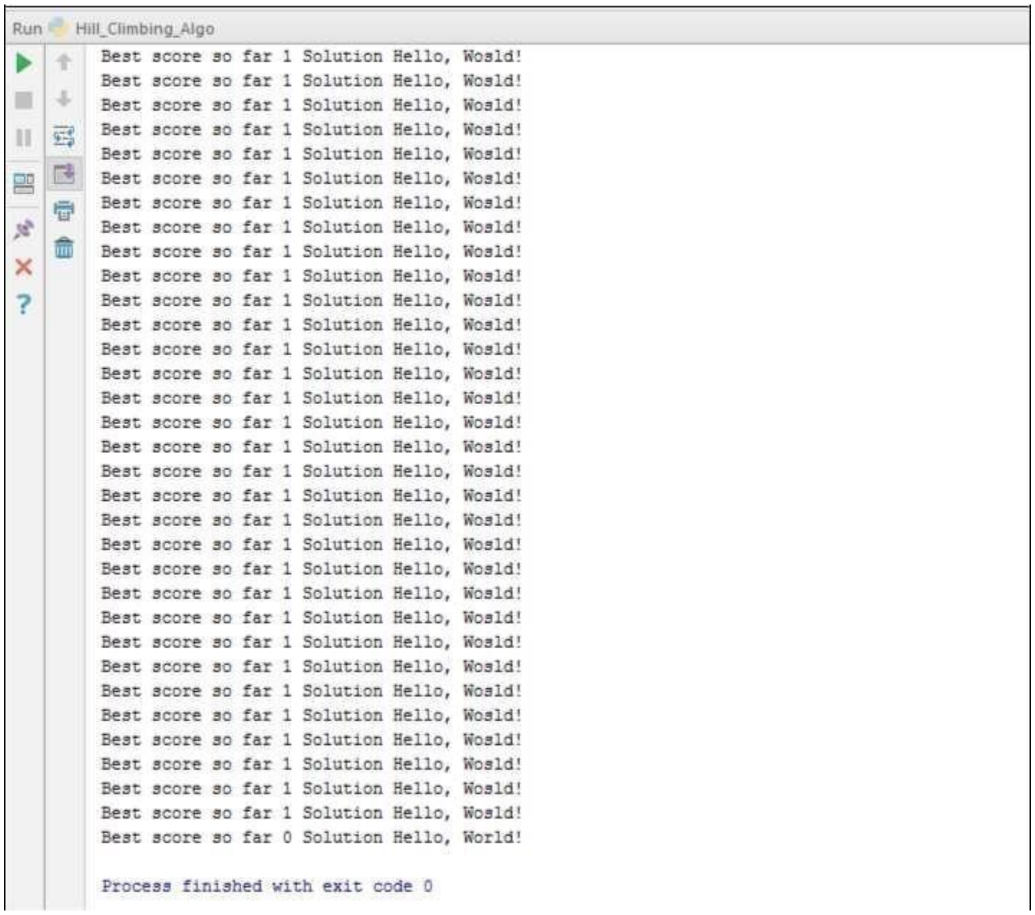
```
Let's see how it works after putting it all together.
"""

best = random_solution()
#print(best)
best_score = evaluate_sol(best)
while True:
    print ('Best score so far', best_score, 'Solution',"". join(best))
    if best_score == 0:
        break
    new_solution = list(best)
    mutate_sol(new_solution)
    score = evaluate_sol(new_solution)

    if evaluate_sol(new_solution) < best_score:
        best = new_solution
        best_score = score
```

Output:



Run Hill_Climbing_Algo

```
C:\Users\Kanwal\PycharmProjects\Example\venv\Scripts\python.exe C:/Users/Kanwal/PycharmProjects/Example/Hill_Climbing_Algo.py
Best score so far 365 Solution NSTP['po/CdP♠
Best score so far 365 Solution NSTP['po/CdP♠
Best score so far 365 Solution NSTP['po/CdP♠
Best score so far 365 Solution NSTP['po/CdP♠
Best score so far 365 Solution NSTP['po/CdP♠
Best score so far 365 Solution NSTP['po/CdP♠
Best score so far 365 Solution NSTP['po/CdP♠
Best score so far 345 Solution NSTP['po/WdP♠
Best score so far 345 Solution NSTP['po/WdP♠
Best score so far 342 Solution NSTS['po/WdP♠
Best score so far 342 Solution NSTS['po/WdP♠
Best score so far 342 Solution NSTS['po/WdP♠
Best score so far 335 Solution NSTS['io/WdP♠
Best score so far 335 Solution NSTS['io/WdP♠
Best score so far 311 Solution NS1S['io/WdP♠
Best score so far 311 Solution NS1S['io/WdP♠
```



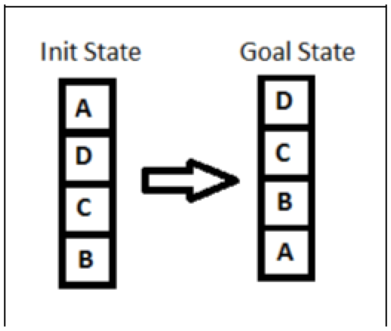
```
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 1 Solution Hello, Wosld!
Best score so far 0 Solution Hello, World!

Process finished with exit code 0
```

7.11. Evaluation Tasks

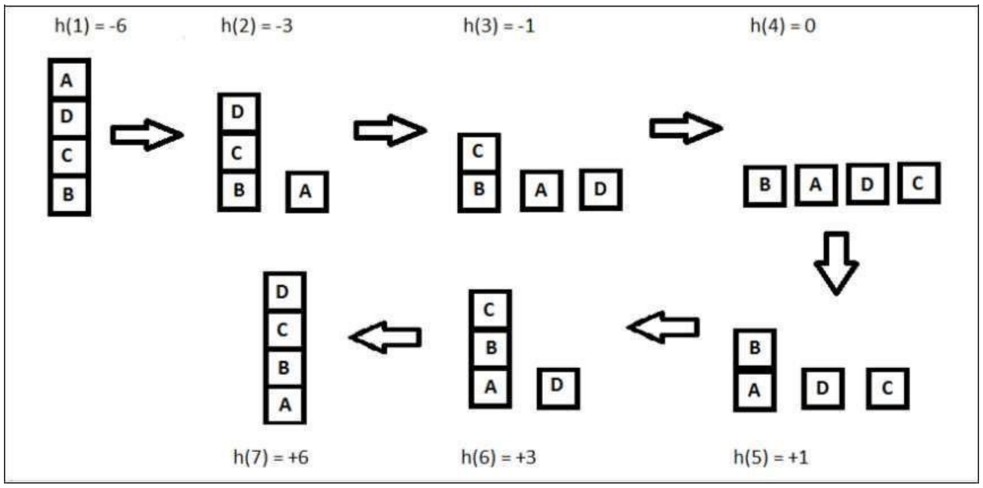
This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time.

7.11.1. Task 1 **(Marks 10)**

Implement the following goal state finding problem using hill climbing algorithm.

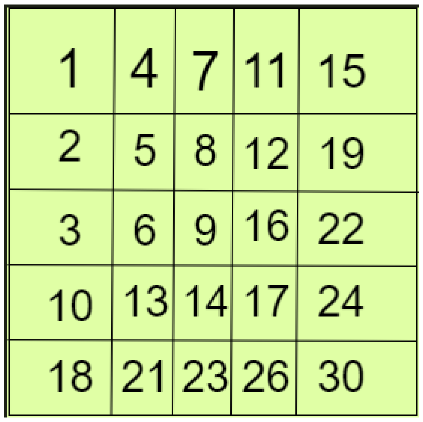


Hint: Key point while solving any hill-climbing problem is to choose an appropriate heuristic function.



7.11.2. Task 2 (Marks 10)

Consider a 2D grid with a starting point (0, 0) and a target point (5, 5). You are allowed to move either up, down, left, or right by one unit at a time. Implement the Hill Climbing algorithm in Python to find the shortest path from the starting point to the target point.



1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

Remember to consider edge cases, handle obstacles if any, and optimize your implementation for efficiency.

7.11.3. Out comes

After completing this lab, students will to know how to deal with Hill climbing problem.

7.12. Evaluation Criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Criteria	**Level 0 (0%)**	**Level 1 (1-50)%**	**Level 2 (51-65)%**	**Level 3 (66-80)%**	**Level 4 (81-100)%**	**Total Score**
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

7.13. Further Readings

7.13.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

7.13.2. Links

https://www.mygreatlearning.com/blog/an-introduction-to-hill-climbing-algorithm/

https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

https://www.baeldung.com/java-hill-climbing-algorithm

Lab Manual for Artificial Intelligence

Lab-08: Constraint Satisfaction Problems

8. Introduction

A **constraint satisfaction problem** (CSP) consists of

- A set of variables,
- A domain for each variable, and
- A set of constraints.

The aim is to choose a value for each variable so that the resulting possible world satisfies the constraints; we want a model of the constraints.

8.1. Relevant Lecture Readings

Revise Lecture Constraint Satisfaction Problem

8.2. Objective of the Experiment

After completing this lab, the student should be able to:

- Clearly understand constraint satisfaction problems in artificial intelligence.

8.3. Concept Map

This concept map will help students to understand the main concepts of topics covered in the lab. In this lab, students will be provided with a detailed content to learn that how built in libraries can be used to implement concept of CSP in python.

8.4. Overview

Formally speaking, a constraint satisfaction problem (or CSP) is defined by a set of variables, {X1; X2; : : : ; Xn}, and a set of constraints, {C1; C2; : : : ; Cm}. Each variable Xi has a nonempty domain Di of possible values. Each constraint Ci involves some subset of the variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables,{Xi=vi; Xj=vj; : : :}. An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints.

8.5. Varieties of constraints

8.5.1. Unary Constraints

This is the simplest type of constraint, e.g. unary constraint can be. South Australia can't be assigned green color.

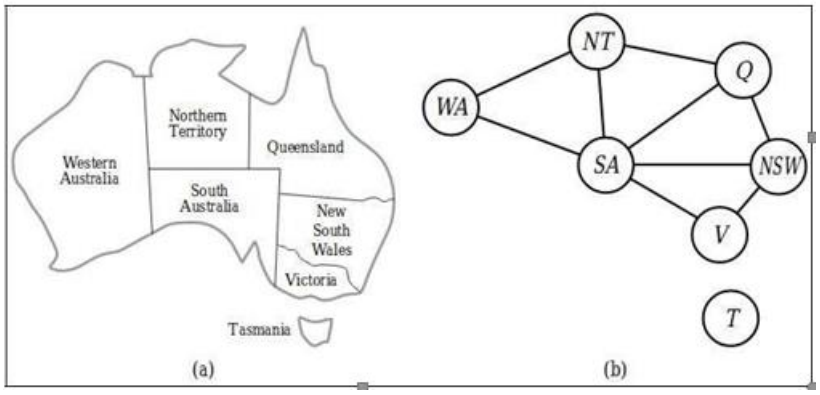
SA ≠ green or **WA = red**

8.5.2. Binary Constraints

Binary constraint relates two variables, e.g. SA ≠ NSW is a binary constraint that involves two pairs of variables.

8.5.3. Higher-order Constraints

Higher-order constraints involves two or more variables e.g. **SA ≠ NSW ≠ Q.** We will see detailed examples of this constraint in following map coloring problem.



We are looking at the map of Australia showing each of its states and territories as shown in above figure (a), we are given the task of coloring each region either red, green or blue in such a way that no neighboring regions have the same color.

To formulate this CSP, we define the **variables** to the regions: WA, NT, Q, NSW, V, SA, T.

The **domain** of each variable is the set {red, green, blue}.

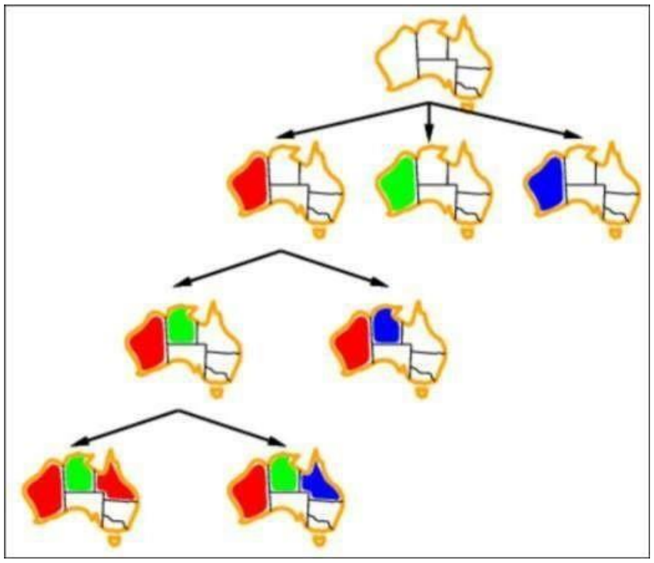
The **constraints** require **neighboring regions to have distinct colors**, e.g. if WA is given red color then NT and SA can't have red color, their domain will be limited to {green, blue}.

It is helpful to visualize a CSP as a constraint graph as shown in figure (b). The nodes of the graph correspond to variables of the problem.

We can give incremental formulation as a standard search problem as follows:

- **Initial State:** the empty assignment {}, in which all variables are unassigned.
- **Successor Function:** a value can be assigned to any unassigned variable, provided that it doesn't conflict with previously assigned variables.
- **Goal Test:** the current assignment is complete.
- **Path Cost:** a constraint cost e.g 1 for every step.

To solve the above problem we use backtracking. Backtracking search is used for a depth first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.



Assign domain (colors) to one variable at a time. Continue…

…

But backtracking is not efficient yet. We can improve efficiency of backtracking using some General-Purpose methods. Most Constraint Variable

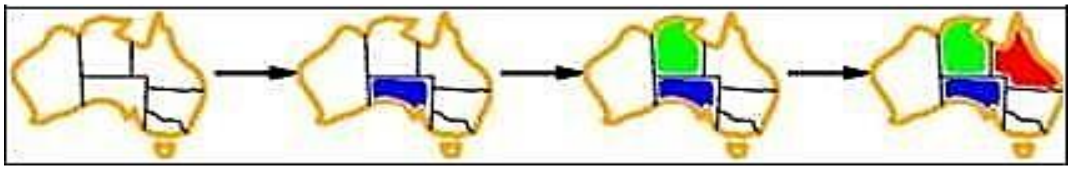
1. Most Constraining Variable
2. Least Constraint Value

8.5.4. Most Constraint Variable

Pick the variable with Minimum Remaining Values (MRV). E.g. WA can have values {red, green, blue}, and NT can have values {red, green}, so you have to choose NT first.

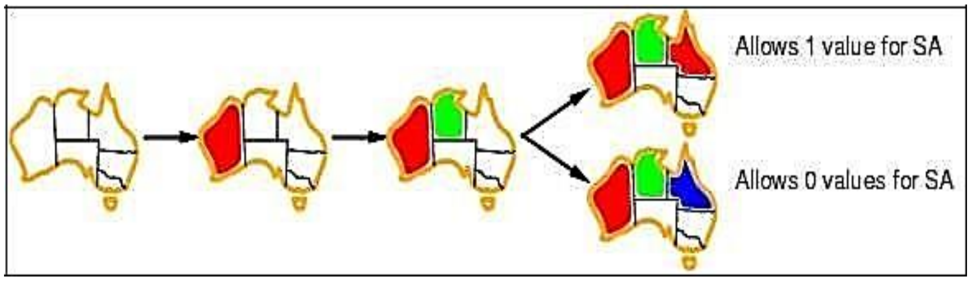
8.5.5. Most Constraining Variable

Choose the variable with the most constraints on remaining variables (most edges in graph)



8.5.6. Least Constraint Value

LSV guides the choice of which value to assign next. Choose the least constraining value:



Now let's perform the general rules on the above problem.

	WA	NT	SA	Q	NSW	V	T
Initial	R,G,B	R,G,B	R,G,B	R,G,B	R,G,B	R,G,B	R,G,B
Domain							
SA = Red	G,B	G,B	R	G,B	G,B	G,B	R,G,B
NT = Blue	G	B	R	G	B	G	R,G,B

T = Red	G	B	R	G	B	G	R

8.6. Homework before Lab

Attempt the following questions. You are required to bring this task with you and submit to the lab instructor.

8.6.1. Problem description

Design the solution of the following CSP also make table to show the process of coloring on paper given in figure 5.1. Your solution must contain all steps until the whole map is not colored as described in the solution design in section 4.2.3.

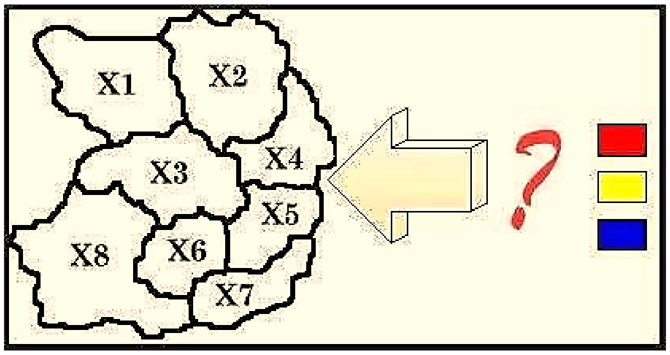


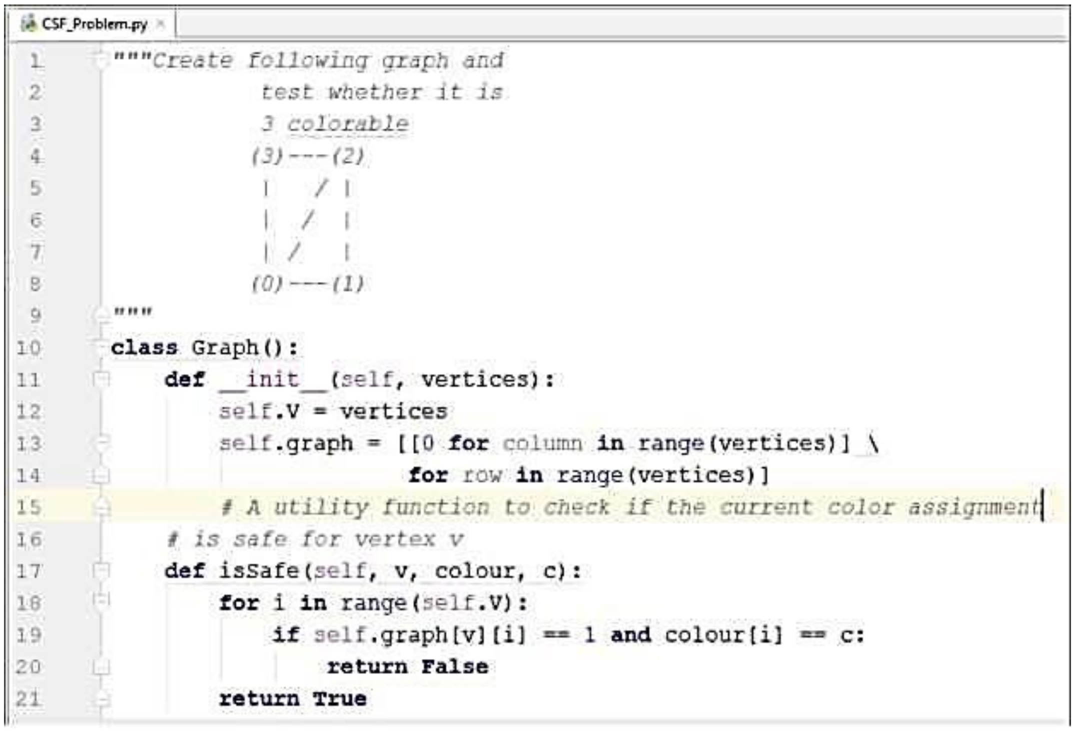
Figure 10.1: CSF Problem

8.7. Procedure & Tools

Pycharm

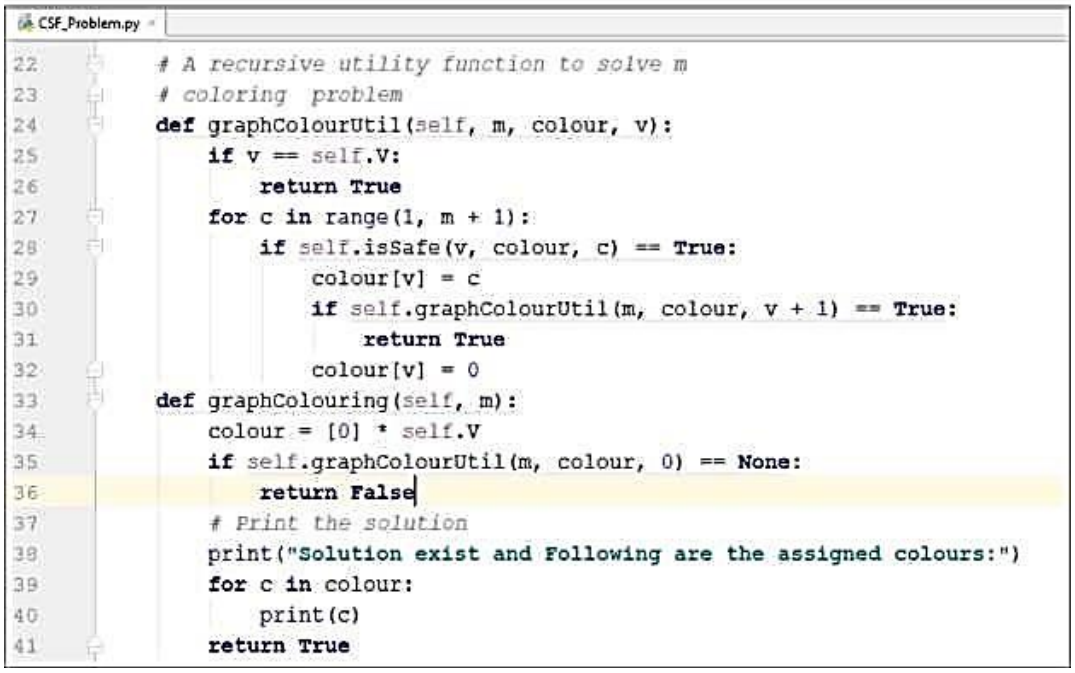
8.8. Practice Task

Write the following code in .py file and run it.



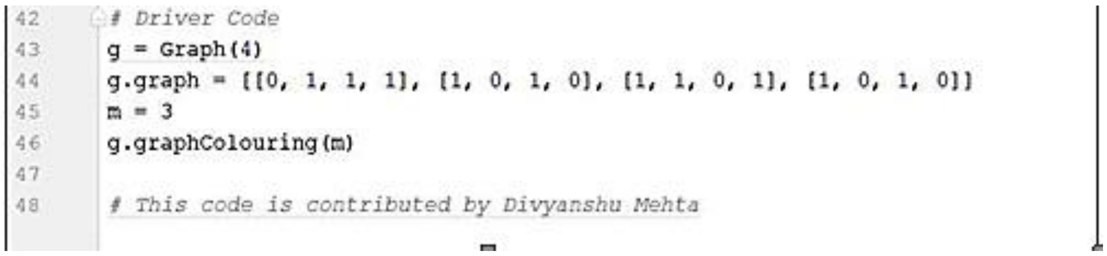
CSF_Problem.py

```
"""Create following graph and
            test whether it is
            3 colorable
           (3)---(2)
            |   / |
            |  /  |
            | /   |
           (0)---(1)
"""
class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)] \
                      for row in range(vertices)]
        # A utility function to check if the current color assignment
    # is safe for vertex v
    def isSafe(self, v, colour, c):
        for i in range(self.V):
            if self.graph[v][i] == 1 and colour[i] == c:
                return False
        return True
```



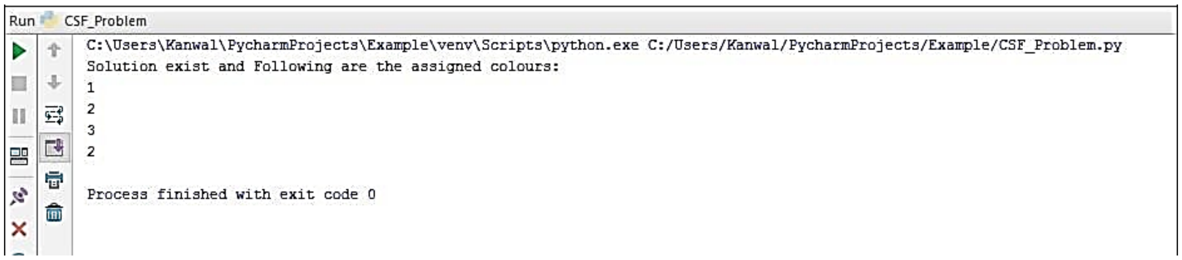
CSF_Problem.py

```
    # A recursive utility function to solve m
    # coloring  problem
    def graphColourUtil(self, m, colour, v):
        if v == self.V:
            return True
        for c in range(1, m + 1):
            if self.isSafe(v, colour, c) == True:
                colour[v] = c
                if self.graphColourUtil(m, colour, v + 1) == True:
                    return True
                colour[v] = 0
    def graphColouring(self, m):
        colour = [0] * self.V
        if self.graphColourUtil(m, colour, 0) == None:
            return False
        # Print the solution
        print("Solution exist and Following are the assigned colours:")
        for c in colour:
            print(c)
        return True
```



```
# Driver Code
g = Graph(4)
g.graph = [[0, 1, 1, 1], [1, 0, 1, 0], [1, 1, 0, 1], [1, 0, 1, 0]]
m = 3
g.graphColouring(m)

# This code is contributed by Divyanshu Mehta
```

Output:


Run CSF_Problem

```
C:\Users\Kanwal\PycharmProjects\Example\venv\Scripts\python.exe C:/Users/Kanwal/PycharmProjects/Example/CSF_Problem.py
Solution exist and Following are the assigned colours:
1
2
3
2

Process finished with exit code 0
```

8.9. Evaluation Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time.

8.9.1. Task 1 (Marks 10)

Implement the home task problem in python.

8.9.2. Task 2 (Marks 10)

The eight queen's puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other; thus, a solution requires no two queens share the same row, column, or diagonal. Formulate this problem as a CSP problem.

8.9.3. Out comes

After completing this lab, students will to know how to deal with CSP.

8.10. Evaluation Criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Criteria	**Level 0 (0%)**	**Level 1 (1-50)%**	**Level 2 (51-65)%**	**Level 3 (66-80)%**	**Level 4 (81-100)%**	**Total Score**
Procedural Awareness						
Practical Implementation						
Program Correctness						

Use of Software Tool						
Sub Total Marks						

8.11. Further Readings

8.11.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

8.11.2. Links

https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/

https://freecontent.manning.com/constraint-satisfaction-problems-in-python/

Lab Manual for Artificial Intelligence

Lab-09: First Order Logic (FOL) inference by using Python Libraries

9. Introduction

First-order logic, often abbreviated as FOL or FOL, is a formal system used in mathematics, philosophy, computer science, and artificial intelligence to represent and reason about statements and relationships between objects in a structured and precise manner. It's also known as first-order predicate logic, first-order predicate calculus, or simply predicate logic.

First-order logic is a fundamental logic system that extends propositional logic (which deals with simple true/false statements) to handle more complex statements involving objects, relations, and quantifiers. Here are some key components and concepts of first-order logic:

1. **Syntax:** In first-order logic, you work with symbols to represent various elements. These symbols include variables, constants, predicates, functions, and logical connectives.
2. **Variables:** Symbols that can represent any object, like 'x', 'y', 'z'.
3. **Constants:** Symbols representing specific objects, like 'John', 'Mary'.
4. **Predicates:** Symbols representing relationships or properties, like 'IsRed', 'IsParentOf'.
5. **Functions:** Symbols representing operations that return objects, like 'Add', 'Multiply'.
6. **Logical Connectives:** Symbols for logical operations, including 'and' ($\wedge$), 'or' ($\vee$), 'not' ($\neg$), 'implies' ($\rightarrow$), and 'if and only if' ($\leftrightarrow$).
7. **Quantifiers:** Symbols like $\forall$ (for all) and $\exists$ (there exists) used to make statements about entire sets of objects.
8. **Semantics:** First-order logic provides a way to interpret these symbols and assign meaning to statements through a formal semantics. It defines how predicates, functions, and quantifiers behave in terms of truth values.
9. **Formulas:** Sentences or statements in first-order logic are called formulas. These can be simple, like "IsRed(Apple)", or complex, involving quantifiers, conjunctions, disjunctions, and implications.
10. **Quantifiers:** Quantifiers are a fundamental aspect of first-order logic. They allow you to make statements about sets of objects.

- The universal quantifier (∀) represents "for all" and asserts that a statement holds true for all objects in a given set.
- The existential quantifier (∃) represents "there exists" and asserts that there is at least one object in a given set for which a statement holds true.

11. **Axioms and Inference Rules:** First-order logic is typically accompanied by a set of axioms and inference rules that govern how to derive new statements from existing ones. Common inference rules include modus ponens and universal generalization.
12. **Soundness and Completeness:** First-order logic is both sound and complete. Soundness means that if a conclusion is derived using valid inference rules, it is necessarily true. Completeness means that any statement that is logically valid within the language of first-order logic can be proven using the rules of the system.

First-order logic is a powerful tool for formal reasoning and forms the basis for various AI systems, knowledge representation languages, and automated theorem provers. It provides a rigorous framework for expressing and reasoning about a wide range of real-world problems and concepts.

9.1. Relevant Lecture Readings

Revise Lectures First Order Logic and its components, Inference in FOL, Reasoning in AI using FOL.

9.2. Objective of the Experiment

The lab aims to equip students with the knowledge and skills to work effectively with First-Order Logic and use it as a tool for formal reasoning and problem-solving.

9.3. Concept Map

This concept map will help students to understand the main concepts of topics covered in the lab. In this lab, students will be provided with a detailed content to learn that how built in libraries can be used for First Order Logic implementation.

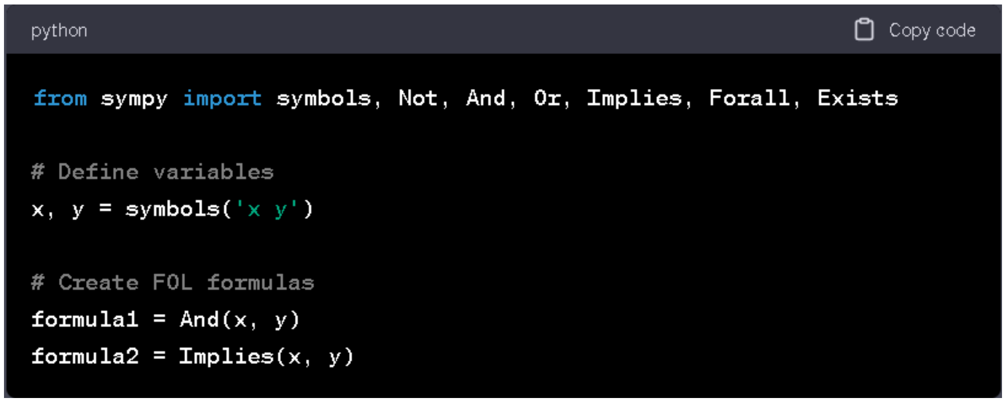
9.4. Libraries

If you want to work with First-Order Logic (FOL) in Python, you can use various libraries and tools to help you represent, manipulate, and reason with FOL statements. Here are some popular libraries and tools for working with FOL in Python:

1. **SymPy:**

- SymPy is a powerful Python library for symbolic mathematics.
- It can be used to create and manipulate symbolic expressions, including those in FOL.
- SymPy provides support for logic operations, quantifiers, and proof techniques.

Example: You can use SymPy to create FOL formulas and perform logical reasoning.



```python
from sympy import symbols, Not, And, Or, Implies, Forall, Exists

# Define variables
x, y = symbols('x y')

# Create FOL formulas
formula1 = And(x, y)
formula2 = Implies(x, y)
```

2. **Pyke:**

- Pyke is a knowledge-based inference engine (expert system) that uses FOL for knowledge representation and reasoning.
- It's designed for building expert systems and rule-based applications.
- Pyke can be used to define knowledge bases and rules using FOL.

3. **Prover9/Mace4:**

- Prover9 and Mace4 are theorem proving tools that can handle FOL.
- While they are standalone programs, you can use Python to interact with them and automate theorem proving tasks.

4. **NetworkX** is a Python library for creating and analyzing complex networks.

- It can be used to model and work with graphs and networks, which can be a useful representation for certain FOL applications.

5. **Pyke Knowledge Engine:**

- Pyke Knowledge Engine is a Python library that provides an integrated environment for rule-based programming and knowledge representation using FOL.

6. **Natural Language Processing Libraries:**

- Libraries like spaCy, NLTK, and Stanford NLP can be used to extract FOL-like representations from natural language text, which can be a useful starting point for FOL reasoning.

When choosing a library or tool for your FOL coding needs in Python, consider your specific requirements and whether you need basic FOL manipulation, complex theorem proving, expert system development, or natural language understanding capabilities. Your choice may also depend on the complexity of the FOL tasks you need to accomplish.

9.5. Tools

Prolog is a popular programming language and logic programming environment used for implementing and working with logic-based systems. There are several Prolog implementations (also called Prolog compilers or Prolog systems) available, each with its own features and advantages. Here are some commonly used Prolog implementations:

1. SWI-Prolog:

- SWI-Prolog is a widely used and actively maintained Prolog system.
- It is open-source and available on multiple platforms, including Windows, macOS, and Linux.
- SWI-Prolog has a rich set of libraries and extensions, making it suitable for various applications.
- It includes a built-in development environment with a graphical debugger and a web-based IDE called SWISH.

2. GNU Prolog:

- GNU Prolog is an open-source Prolog compiler designed to be compatible with the ISO standard for Prolog.
- It is available on various platforms, including Unix-like systems, Windows, and macOS.
- GNU Prolog focuses on being a lightweight and efficient Prolog system.

3. YAP (Yet another Prolog):

- YAP is a high-performance Prolog compiler that emphasizes efficiency.
- It is available on Unix-like systems, Windows, and macOS.
- YAP offers various extensions and libraries for advanced Prolog programming.

4. **SICStus Prolog:**

- SICStus Prolog is a commercial Prolog system known for its high-performance and advanced features.
- It offers a wide range of libraries and tools for applications like constraint logic programming and natural language processing.
- SICStus Prolog is available for Windows, Linux, and macOS.

5. **ECLiPSe:**

- ECLiPSe is a versatile constraint logic programming system that extends Prolog.
- It is often used for constraint satisfaction problems and optimization tasks.
- ECLiPSe supports several programming paradigms, including constraint logic programming (CLP).

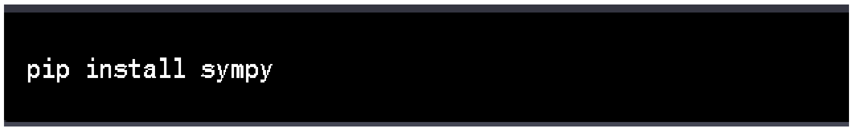
The choice of Prolog implementation depends on factors such as your specific project requirements, platform compatibility, performance considerations, and whether you prefer open-source or commercial solutions. SWI-Prolog and GNU Prolog are often recommended for beginners due to their accessibility and robust community support.

9.6. Practice Task

In this example, we will use the SymPy library, which provides excellent support for symbolic mathematics and logic. Here's how to implement FOL using SymPy:

1. **Step 1: Install SymPy**

   If you haven't already installed SymPy, you can do so using pip:

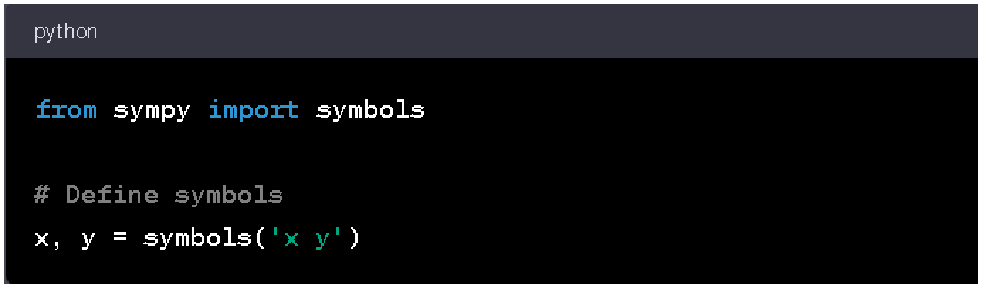


```
pip install sympy
```

2. **Step 2: Define Symbols**

   In SymPy, you first need to define symbols to represent variables and constants in your

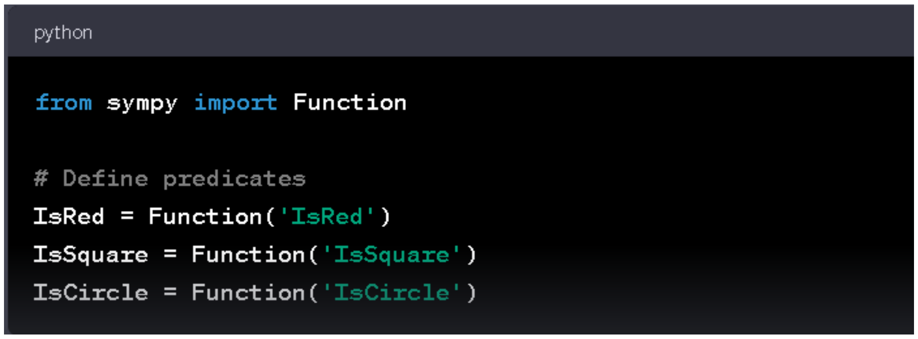
FOL statements:



```python
from sympy import symbols

# Define symbols
x, y = symbols('x y')
```

3. **Step 3: Define Predicates**

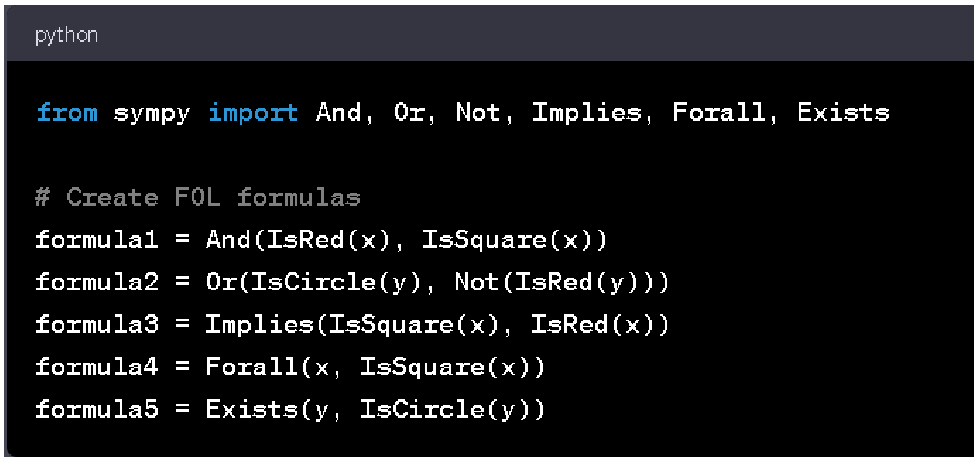
   Predicates represent relationships or properties. You can define predicates as functions of symbols:



```python
from sympy import Function

# Define predicates
IsRed = Function('IsRed')
IsSquare = Function('IsSquare')
IsCircle = Function('IsCircle')
```

4. **Step 4: Create FOL Formulas**

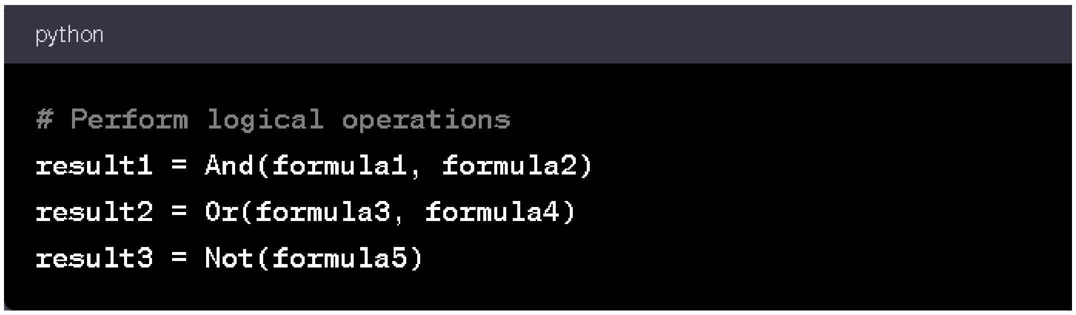
   You can create FOL formulas using these predicates, logical connectives, and quantifiers:



```python
from sympy import And, Or, Not, Implies, Forall, Exists

# Create FOL formulas
formula1 = And(IsRed(x), IsSquare(x))
formula2 = Or(IsCircle(y), Not(IsRed(y)))
formula3 = Implies(IsSquare(x), IsRed(x))
formula4 = Forall(x, IsSquare(x))
formula5 = Exists(y, IsCircle(y))
```

5. **Step 5: Perform Logical Operations**

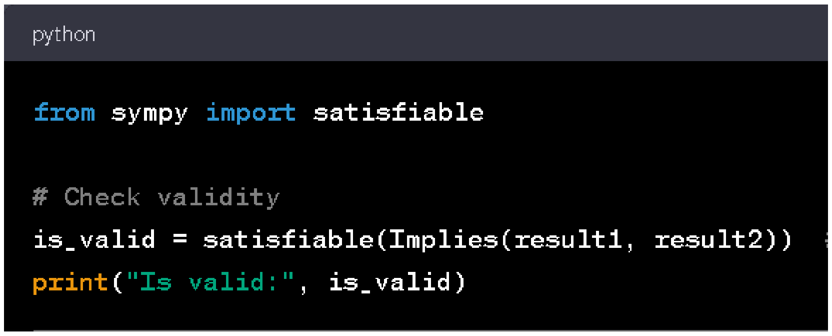
   You can use SymPy's logical operators to perform operations on FOL formulas:



```python
# Perform logical operations
result1 = And(formula1, formula2)
result2 = Or(formula3, formula4)
result3 = Not(formula5)
```

6. **Step 6: Check Validity**

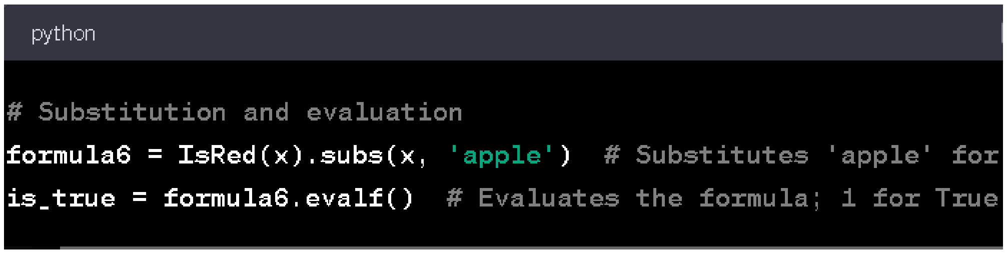
   SymPy can be used to check the validity of FOL formulas:



```python
from sympy import satisfiable

# Check validity
is_valid = satisfiable(Implies(result1, result2))
print("Is valid:", is_valid)
```

7. **Step 7: Substitution and Evaluation**

   You can substitute values into FOL formulas and evaluate them:



```python
# Substitution and evaluation
formula6 = IsRed(x).subs(x, 'apple')  # Substitutes 'apple' for
is_true = formula6.evalf()  # Evaluates the formula; 1 for True
```

9.7. Evaluation Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time.

9.7.1. Task 1 (Marks 10)

Translate the following English sentence into FOL and implement it in Python using SymPy:

1. "Every cat is an animal."

2. "If it is raining, then the ground is wet."

3. "For all x, if x is a person, then there exists a y such that y is the parent of x."

9.7.2. Task 2 (Marks 10)

To write the negation of the FOL statement "Some birds can fly" in Python using the SymPy library, you can use the Not operator to negate the entire statement.

9.7.3. Out comes

After completing this lab, students will know the practical implementation of FOL using Python libraries.

9.8. Evaluation Criteria

The evaluation criteria for this lab will be based on the completion of the given tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Criteria	Level 0 (0%)	Level 1 (1-50)%	Level 2 (51-65)%	Level 3 (66-80)%	Level 4 (81-100)%	Total Score
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

9.9. Further Readings

9.9.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

9.9.2. Links

https://avigad.github.io/lamr/implementing_first_order_logic.html

Lab Manual for Artificial Intelligence

Lab-10: Deep Learning Inference

10. Introduction

Deep learning inference refers to the process of using a trained deep learning model to make predictions or decisions based on new, unseen data. In deep learning, models are typically composed of multiple layers of artificial neurons (like neural networks) that have been trained on large datasets to learn complex patterns and relationships within the data.

Here's a step-by-step explanation of deep learning inference:

1. **Training:** Initially, a deep learning model is trained on a labeled dataset. During training, the model learns to recognize patterns and make predictions based on the input data and associated labels. This involves adjusting the model's internal parameters (weights and biases) through a process called backpropagation and optimization techniques such as gradient descent.

2. **Model Deployment:** Once the model is trained and its parameters are optimized, it can be deployed for inference. Deployment can occur on various platforms, including cloud servers, edge devices (like smartphones or IoT devices), or even embedded systems.

3. **Inference:** During inference, the trained model is presented with new, unseen data (input). The model processes this input data through its layers of neurons, and the output is generated based on what the model has learned during training. This output can take different forms, depending on the nature of the task. For example, in image classification, the output might be a label identifying the object in the image.

4. **Decision-Making:** The output generated by the model during inference is used for decision-making or prediction. In applications like image recognition, natural language processing, and autonomous driving, deep learning models are used to make decisions based on the input data. For instance, a deep learning model in a self-driving car might use inference to detect obstacles and make decisions about steering and braking.

5. **Feedback Loop:** In some cases, the output of the model during inference can be used to improve the model itself. This can be done by collecting feedback data, adjusting model parameters, and retraining the model periodically to keep it up-to-date and accurate.

   Deep learning inference is a critical component of many modern applications, including image and speech recognition, natural language understanding, recommendation systems, and autonomous systems, among others. It enables these systems to make real-time decisions and predictions based on the patterns and knowledge they have learned from training data.

10.1. Relevant Lecture Readings

Revise Lecture Probabilistic Reasoning in AI and Bayes Theorem in AI.

10.2. Objective of the Experiment

After completing this lab, the student should be able to:

- Clearly understand deep learning models in artificial intelligence.

10.3. Concept Map

This concept map will help students to understand the main concepts of topics covered in the lab. In this lab, students will be provided with a detailed content to learn that what the concept of deep learning is and how to implement these models in python.

10.4. Types of Deep Learning Models

There are several types of deep learning models, each designed for specific tasks and data types. Here are some common types:

1. **Feedforward Neural Networks (FNN):** Also known as multi-layer perceptron (MLPs), these are the foundational deep learning models composed of input, hidden, and output layers. They're used for various tasks like classification and regression.

2. **Convolutional Neural Networks (CNN):** Primarily designed for image recognition tasks, CNNs utilize specialized layers like convolutional and pooling layers to automatically learn and recognize patterns in images.

3. **Recurrent Neural Networks (RNN):** Suited for sequence data, RNNs have connections that loop back on themselves, allowing them to process sequences in a time-dependent manner. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are popular RNN variants.

4. **Transformer Models:** Introduced by the "Attention Is All You Need" paper, transformers have revolutionized natural language processing tasks. They use attention mechanisms to process sequences of data in parallel, enabling efficient processing of long-range dependencies. Models like BERT, GPT, and T5 are based on transformers.

5. **Auto encoders:** Used for unsupervised learning, auto encoders learn to encode data into a compressed representation and then decode it back to the original data. They're used for tasks like data denoising, dimensionality reduction, and anomaly detection.

6. **Generative Adversarial Networks (GAN):** GANs consist of two networks—the generator and the discriminator—competing in a game-like setting. They're used for generating new data instances that resemble a given dataset, leading to applications in image and text generation.

7. **Variational Auto encoders (VAE):** VAEs combine auto encoders with probabilistic modeling, enabling the generation of new data instances while exploring the underlying data distribution.

8. **Siamese Networks:** These networks are used for tasks like similarity and distance measurement. They have two or more identical subnetworks, sharing weights, and are often used for tasks like face recognition and signature verification.

9. **Graph Neural Networks (GNN):** Designed to work with graph-structured data, GNNs can capture information from nodes and their connections in graphs, making them useful

for applications like social network analysis and recommendation systems.

10. **Attention-Based Models:** Beyond transformers, attention mechanisms are used in various contexts to weigh the importance of different elements in a sequence, such as in image captioning and visual question answering.

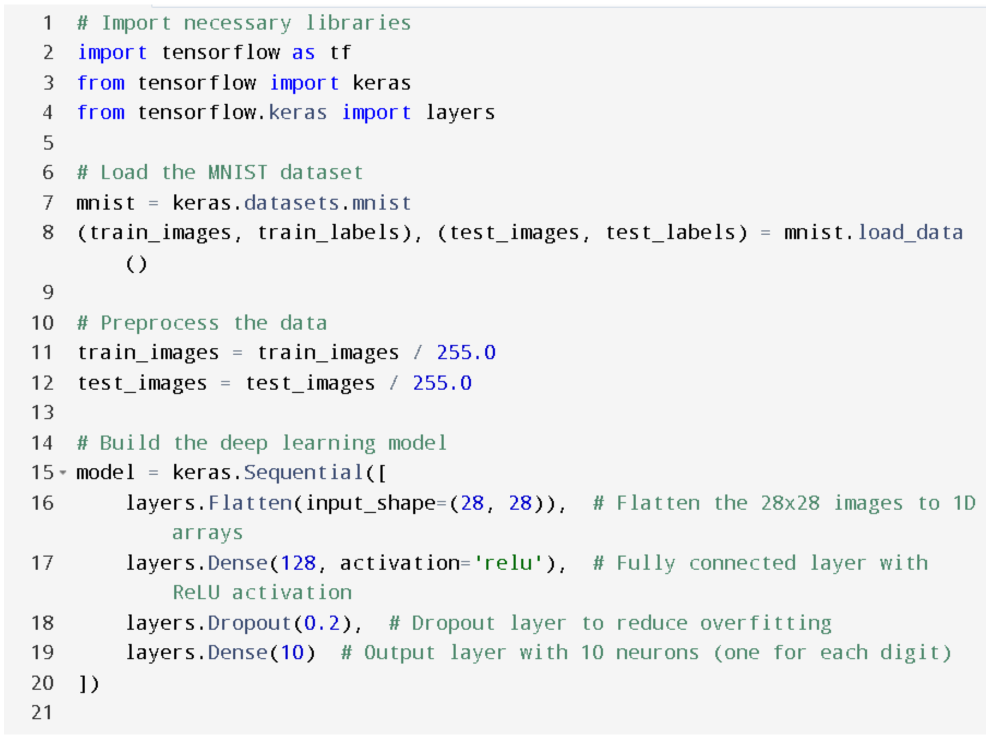
These are just a few examples of the diverse range of deep learning models. Each type is designed to address specific types of data and tasks, and researchers continue to develop new architectures to tackle increasingly complex problems.

10.5. Procedure & Tools

Pycharm

10.6. Practice Task

We will perform a simple practice task to build and train a deep learning model in Python using TensorFlow and Keras. In this example, we'll create a feedforward neural network for image classification using the famous MNIST dataset of handwritten digits.



```python
# Import necessary libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Load the MNIST dataset
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Preprocess the data
train_images = train_images / 255.0
test_images = test_images / 255.0

# Build the deep learning model
model = keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),  # Flatten the 28x28 images to 1D arrays
    layers.Dense(128, activation='relu'),  # Fully connected layer with ReLU activation
    layers.Dropout(0.2),  # Dropout layer to reduce overfitting
    layers.Dense(10)  # Output layer with 10 neurons (one for each digit)
])

```

```
# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy
                  (from_logits=True),
              metrics=['accuracy'])

# Train the model
model.fit(train_images, train_labels, epochs=10, validation_split=0.2)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print("\nTest accuracy:", test_acc)
```

In this code:

1. We import TensorFlow and Keras.
2. We load the MNIST dataset of handwritten digits, which consists of 28x28 pixel images.
3. We preprocess the data by scaling pixel values to the range [0, 1].
4. We build a feedforward neural network model using Keras. This model has an input layer that flattens the 28x28 images into 1D arrays, a hidden layer with ReLU activation, a dropout layer to reduce overfitting, and an output layer with 10 neurons (one for each digit) using softmax activation.
5. We compile the model, specifying the optimizer ('adam'), loss function ('sparse_categorical_crossentropy' for multiclass classification), and evaluation metric ('accuracy').
6. We train the model on the training data for 10 epochs with a validation split of 20%.
7. Finally, we evaluate the model's accuracy on the test data and print the result.

The output will show the test accuracy of the trained model, indicating how well it performs on classifying handwritten digits in the MNIST dataset.

10.7. Evaluation Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time.

10.7.1. Task 1 (Marks 10)

Load a dataset of your choice (e.g., MNIST, CIFAR-10, Oxford Flowers 102) using Python and a CNN deep learning framework. Preprocess the data, including normalization, splitting into training and testing sets, and data augmentation if applicable.

10.7.2. Task 2 (Marks 10)

Train the neural network on the training data. Monitor training progress by visualizing training and validation loss and accuracy. Set an appropriate number of epochs and batch size. For example: You are required to train a CNN model for image classification. The dataset you are using is Oxford Flowers 102, epoch's ranges from 10 to 100 and batch size ranges from 32 to 256.

10.7.3. Out comes

After completing this lab, students will to know how to deal with different deep learning models.

10.8. Evaluation Criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Criteria	**Level 0 (0%)**	**Level 1 (1-50)%**	**Level 2 (51-65)%**	**Level 3 (66-80)%**	**Level 4 (81-100)%**	**Total Score**
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

10.9. Further Readings

10.9.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

10.9.2. Links

https://colab.research.google.com/drive/19YwLLREPmE0ZFoneEKQpSkXRIIF9pPP

https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

https://www.mathworks.com/discovery/deep-learning.html

https://www.projectpro.io/article/deep-learning-projects-for-beginners/441

Lab Manual for Artificial Intelligence

Lab-11: Basic ML and Naïve Bayes

11. Introduction

Machine learning is a subfield of artificial intelligence (AI) that focuses on developing algorithms and models that enable computers to learn from and make predictions or decisions based on data, without being explicitly programmed to perform a specific task. In other words, it's about creating systems that can improve their performance on a given task through experience (data) rather than through explicit programming.

Here are some key aspects of machine learning:

- **Data-Driven Learning:** Machine learning algorithms learn from data. You provide a machine learning model with a dataset that includes examples (input data) and their corresponding outcomes (labels or target values), and the model learns patterns and relationships within the data.

- **Generalization:** A primary goal of machine learning is to create models that can generalize well to new, unseen data. This means that the model should be able to make accurate predictions or decisions on data it hasn't encountered during training.

- **Automated Learning:** Machine learning models automatically adapt and improve their performance as they receive more data. This adaptability is a key feature, as it allows models to handle changing and evolving environments.

- **Applications:** Machine learning is used in a wide range of applications, including image and speech recognition, natural language processing, recommendation systems, fraud detection, autonomous vehicles, medical diagnosis, and much more.

11.1. Use of Machine Learning

- **Complex Tasks**: Machine learning excels at tasks that are difficult for humans to program explicitly due to their complexity or the need to process vast amounts of data. For example, it's challenging to write precise rules for recognizing objects in images, but machine learning models can learn to do this from labeled image data.

- **Pattern Recognition:** Machine learning is excellent at recognizing patterns and making predictions based on those patterns. This ability is valuable in fields like finance for predicting stock prices, in healthcare for diagnosing diseases, and in marketing for predicting customer behavior.

- **Automation:** Machine learning can automate repetitive tasks, allowing humans to focus on more creative and complex aspects of their work. For example, in customer support, chatbots powered by machine learning can handle routine inquiries, freeing up human agents for more complex customer interactions.

- **Personalization:** Machine learning is used to create personalized experiences, such as personalized recommendations on streaming platforms, personalized marketing messages, and even personalized medicine.

In summary, machine learning is a powerful tool for solving complex problems and making predictions or decisions based on data. Its ability to learn from data and generalize to new situations makes it a valuable technology in a wide range of fields and applications.

11.2. Naïve Bayes

The Bayesian Classification represents a supervised learning method as well as a statistical method for classification. Assumes an underlying probabilistic model and it allows us to capture uncertainty about the model in a principled way by determining probabilities of the outcomes. It can solve diagnostic and predictive problems. This Classification is named after Thomas Bayes (1702-1761), who proposed the Bayes Theorem. Bayesian classification provides practical learning algorithms and prior knowledge and observed data can be combined.

Bayesian Classification provides a useful perspective for understanding and evaluating many learning algorithms. It calculates explicit probabilities for hypothesis and it is robust to noise in input data.

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Bayes theorem can be written as:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

11.2.1. Types of Naive Bayes Classifier

- **Multinomial Naive Bayes**

This is mostly used for document classification problem, i.e. whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

- **Bernoulli Naive Bayes**

This is similar to the multinomial Naive Bayes but the predictors are Boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

- **Gaussian Naive Bayes**

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a Gaussian distribution. Naive Bayes algorithms are mostly used in sentiment analysis, spam filtering, and recommendation systems etc. They are fast and easy to implement.

11.3. Relevant Lecture Readings

Revise Lecture No. Bayes Theorem in AI and Introduction to ML

11.4. Objective of the Experiment

After completing this lab, the student should be able to:

- Different concepts of machine learning and implementation of Naïve Bayes theorem.

11.5. Concept Map

This concept map will help students to understand the main concepts of topics covered in the lab. In this lab, students will be provided with a detailed content to learn that what the concept of machine learning is and how to implement different ML algorithms in python.

11.6. Procedure & Tools

Pycharm

11.7. Homework before Lab

Attempt the following questions. You are required to bring this task with you and submit to the lab instructor.

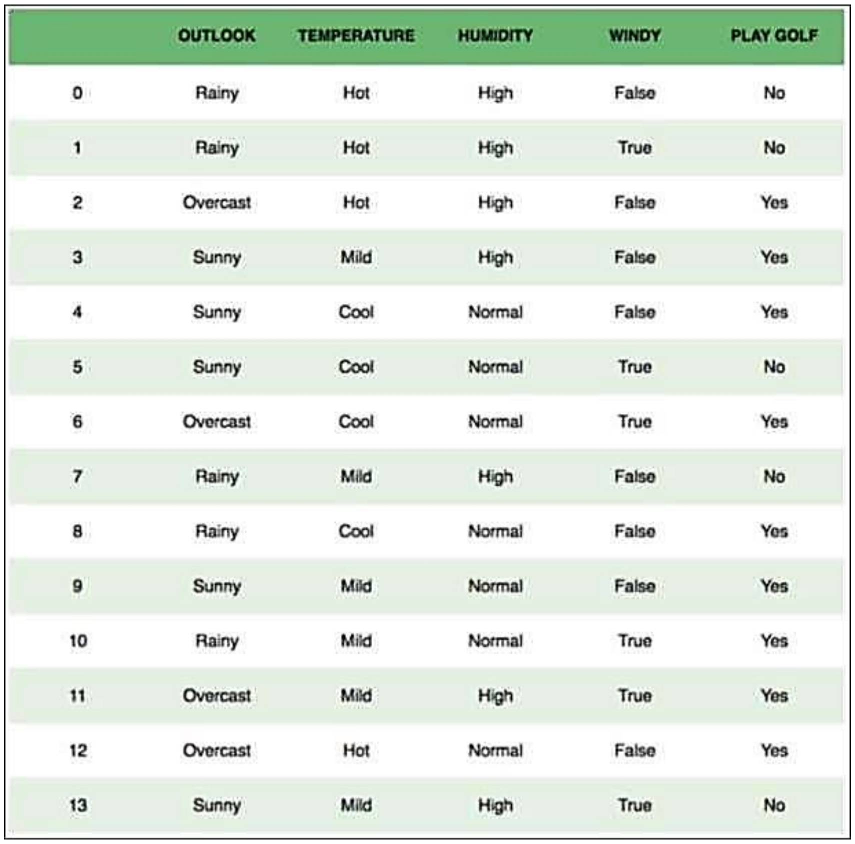
11.7.1. Problem description

Apply the given formula on the given dataset Figure 1 to classify whether the day is suitable for playing golf or not. The columns represent these features and the rows represent individual entries. If we take the first row of the dataset, we can observe that is not suitable for playing golf if the outlook is rainy, temperature is hot, humidity is high and it is not windy. We make two assumptions here, one as stated above we consider that these predictors are independent. That is, if the temperature is hot, it does not necessarily

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

mean that the humidity is high. Another assumption made here is that all the predictors have an equal effect on the outcome. That is, the day being windy does not have more importance in deciding to play golf or not. According to this example, Bayes theorem can be rewritten as:

The variable y is the class variable (play golf), which represents if it is suitable to play golf or not given the conditions. Variable X represent the parameters/features.



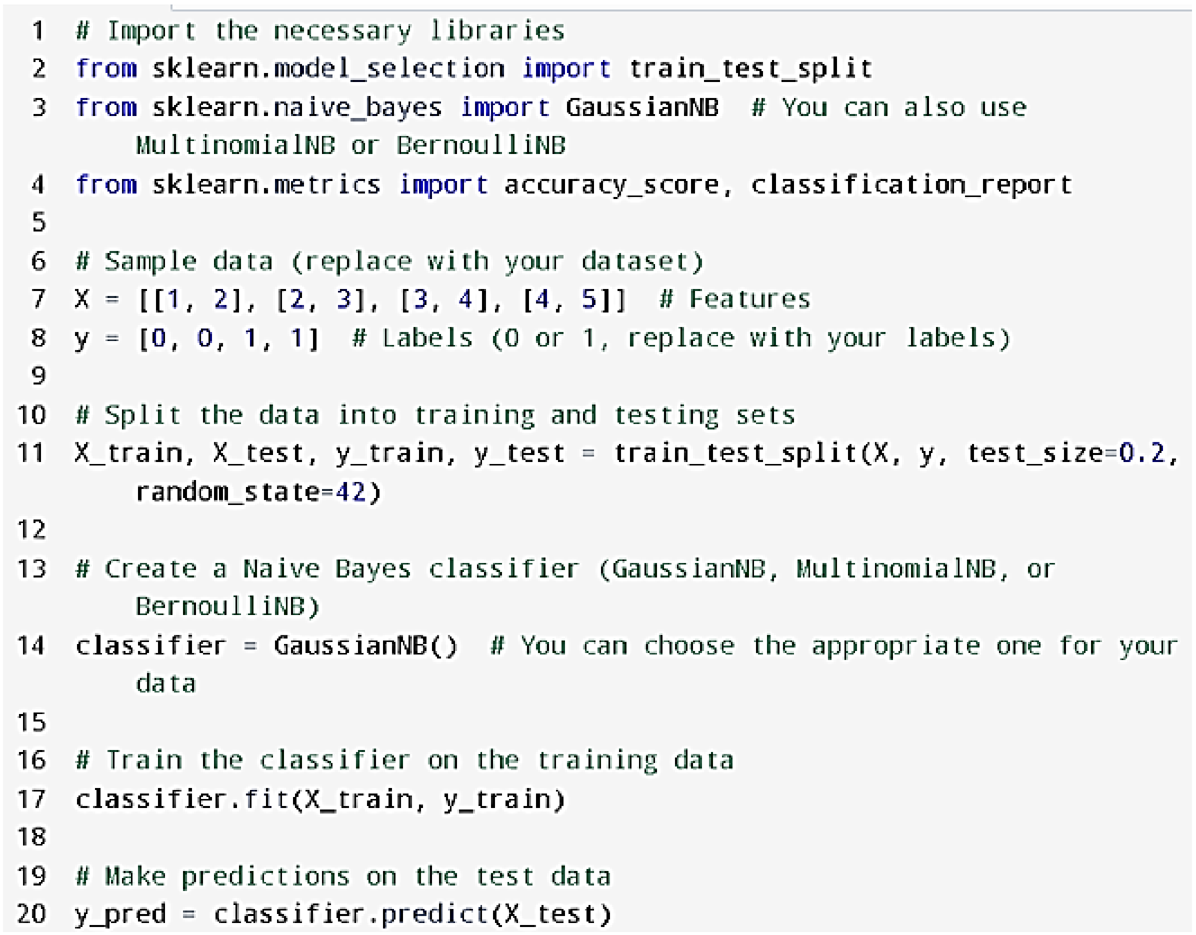
	OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY GOLF
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

Figure 2: Dataset

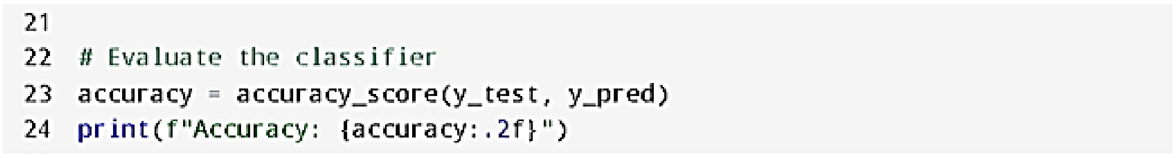
11.8. Practice Task

Implementing a Naive Bayes classifier in Python is relatively straightforward using popular libraries like Scikit-Learn (sklearn). Scikit-Learn provides a simple and efficient implementation of various Naive Bayes algorithms, including Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive Bayes, which are suitable for different types of data.

Here's a basic example of how to implement Naive Bayes classification using Scikit-Learn:



```python
# Import the necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB  # You can also use MultinomialNB or BernoulliNB
from sklearn.metrics import accuracy_score, classification_report

# Sample data (replace with your dataset)
X = [[1, 2], [2, 3], [3, 4], [4, 5]]  # Features
y = [0, 0, 1, 1]  # Labels (0 or 1, replace with your labels)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Naive Bayes classifier (GaussianNB, MultinomialNB, or BernoulliNB)
classifier = GaussianNB()  # You can choose the appropriate one for your data

# Train the classifier on the training data
classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = classifier.predict(X_test)
```



```python

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

Output:

```
Accuracy: 95.67
```

11.9. Evaluation Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time.

11.9.1. Task 1 (Marks 10)

You can use a dataset of emails labeled as spam or not spam (ham).

Build a Naive Bayes classifier to classify emails as spam or not spam based on their content (e.g., subject, body). You may need to preprocess the text data, extract relevant features, and represent the data appropriately.

11.9.2. Task 2 (Marks 10)

Create your own dataset for a classification task of your choice. It could be related to a topic or domain you are interested in, such as sports, movies, or food.

Design the dataset, gather data, and define classes or labels. Then, implement a Naive Bayes classifier to perform the classification task.

11.9.3. Out comes

After completing this lab, students will to know how to deal with Naïve Bayes classifier.

11.10. Evaluation Criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s)

Criteria	**Level 0 (0%)**	**Level 1 (1-50)%**	**Level 2 (51-65)%**	**Level 3 (66-80)%**	**Level 4 (81-100)%**	**Total Score**
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

11.11. Further Readings

11.11.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

11.11.2. Links

https://www.geeksforgeeks.org/naive-bayes-classifiers/

https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

Lab Manual for Artificial Intelligence

Lab-12: Supervised & Unsupervised Learning

12. Introduction

Supervised learning and unsupervised learning are two fundamental approaches in machine learning, and they differ in terms of their objectives and how they use labeled or unlabeled data. Here's a comparison of the two with examples:

1. Supervised Learning:

- **Objective:** In supervised learning, the goal is to learn a mapping from input data to known output labels. It involves training a model to make predictions or classifications based on labeled training data.
- **Labeled Data:** Supervised learning requires a dataset with both input features and corresponding output labels.

Examples:

- **Classification:** Classifying emails as spam or not spam. Given a dataset of emails, each labeled as spam or not spam, a supervised learning algorithm learns to classify new, unlabeled emails.
- **Regression:** Predicting house prices based on features like square footage, number of bedrooms, and location. The model learns to predict the price (a continuous value) from the input features.
- **Image Classification:** Identifying objects in images. Given a dataset of images with labeled objects (e.g., cats, dogs, cars), the model learns to classify new images.

2. Unsupervised Learning:

- **Objective:** Unsupervised learning aims to discover patterns or structure within unlabeled data. It does not have access to explicit output labels and often involves grouping or clustering similar data points.

- **Unlabeled Data:** Unsupervised learning uses data without any predefined output labels.

Examples:

- **Clustering:** Grouping similar data points together. For instance, in customer segmentation, you might have data on customer behavior but not their segment. Unsupervised learning can cluster customers based on their behavior to identify different market segments.
- **Dimensionality Reduction:** Reducing the number of features while retaining important information. Principal Component Analysis (PCA) is an example used to reduce the dimensionality of data for visualization or compression.
- **Anomaly Detection:** Identifying unusual or rare data points in a dataset. This can be used in fraud detection, where unusual financial transactions need to be flagged.

Example Comparison:

Imagine you have a dataset containing customer purchase history, and you want to analyze it:

Supervised Learning: If you have labeled data with information about which customers made a purchase (output label) and other customer attributes (input features), you can use supervised learning to build a model to predict which new customers are likely to make a purchase. This would be a classification task.

Unsupervised Learning: If you only have the purchase history without labels indicating whether each customer made a purchase or not, you can use unsupervised learning to cluster customers based on their purchase behavior. This might reveal distinct customer segments, even though you don't know what those segments represent in advance.

In summary, supervised learning is used when you have labeled data and want to predict or classify based on known outcomes, while unsupervised learning is used when you want to uncover patterns or structure within unlabeled data. Each approach is valuable in different scenarios and serves distinct machine learning purposes.

12.1. Supervised and Unsupervised Learning Algorithms

12.1.1. Supervised Learning Algorithms

Here are some common supervised learning algorithms:

1. **Linear Regression:** Linear regression is used for predicting a continuous target variable (a real number) based on one or more input features. It fits a linear equation to the data and is particularly useful for regression tasks.

2. **Logistic Regression:** Unlike linear regression, logistic regression is used for binary classification problems, where the goal is to predict one of two classes. It models the probability that a given input belongs to a particular class.

3. **Decision Trees:** Decision trees are versatile algorithms that can be used for both classification and regression tasks. They partition the data into subsets based on the values of input features and make decisions at each node to classify or predict the target variable.

4. **Random Forest:** Random Forest is an ensemble learning method that combines multiple decision trees to improve predictive performance and reduce overfitting. It is commonly used for both classification and regression tasks.

5. **Support Vector Machines (SVM):** SVM is a classification algorithm that finds the optimal hyperplane to separate data into different classes. It works well for linear and non-linear classification tasks.

6. **K-Nearest Neighbors (KNN):** KNN is a simple yet effective algorithm for classification and regression. It makes predictions based on the majority class or average value of its k nearest neighbors in the training data.

7. **Naive Bayes:** Naive Bayes is a probabilistic algorithm based on Bayes' theorem. It is

often used for text classification and spam filtering but can be applied to other classification tasks as well.

8. **Neural Networks:** Neural networks, particularly deep learning models, have gained popularity for their ability to handle complex data and tasks. They consist of interconnected layers of neurons and are used for a wide range of tasks, including image and speech recognition, natural language processing, and more.

12.1.2. Unsupervised Learning Algorithms

Here are some common unsupervised learning algorithms:

1. **Clustering Algorithms:**

❖ **K-Means Clustering:** This algorithm partitions the data into 'K' clusters based on similarity. It assigns data points to clusters by minimizing the distance between data points and the cluster centroids.

❖ **Hierarchical Clustering:** Hierarchical clustering builds a tree-like structure of clusters, where data points are grouped together based on their similarity.

❖ **Gaussian Mixture Models (GMM):** GMM models data as a mixture of multiple Gaussian distributions and can capture complex patterns in the data.

2. **Dimensionality Reduction Algorithms:**

❖ **Principal Component Analysis (PCA):** PCA reduces the dimensionality of data by projecting it onto a lower-dimensional subspace while preserving the most important information.

❖ **Autoencoders:** Autoencoders are neural network-based models that learn to encode and decode data, effectively reducing its dimensionality. They are used for tasks like feature extraction and denoising.

3. **Generative Models:**

- ❖ Generative Adversarial Networks (GANs): GANs consist of two neural networks, a generator, and a discriminator, which are trained adversarial. GANs can generate new data instances that resemble the training data.

12.2. Relevant Lecture Readings

Revise Lectures Supervised and Unsupervised Learning.

12.3. Objective of the Experiment

After completing this lab, the student should be able to:

- Different concepts of supervised and unsupervised learning.

12.4. Concept Map

This concept map will help students to understand the main concepts of topics covered in the lab. In this lab, students will be provided with a detailed content to learn that what the concept of supervised and unsupervised learning is and how to implement different algorithms in python.

12.5. Procedure & Tools

Pycharm

12.6. Homework before Lab

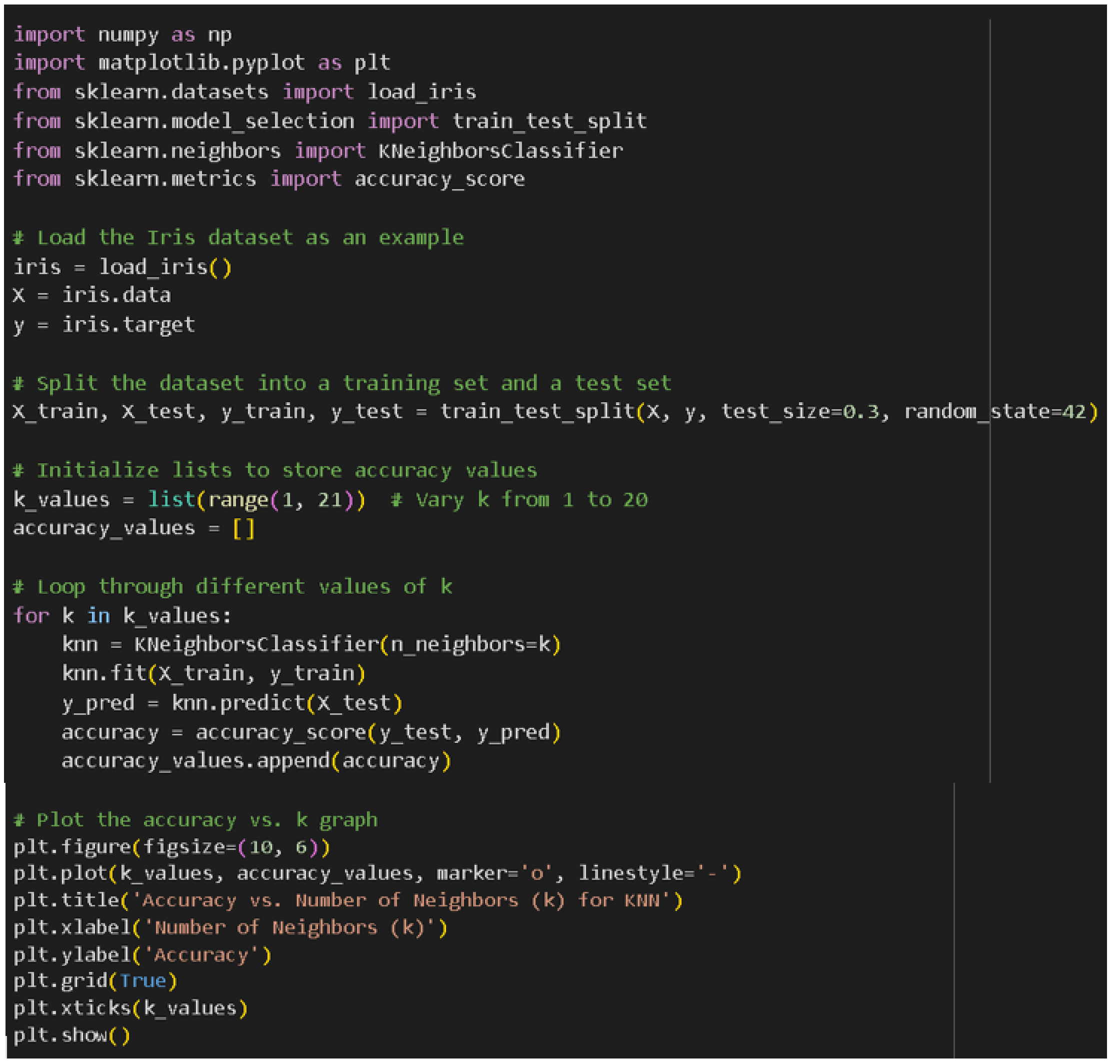
Attempt the following questions. You are required to bring this task with you and submit to the lab instructor.

12.6.1. Problem description

- ✓ Suppose you have a dataset with historical temperature data and energy consumption for a building. How can you apply supervised learning to predict energy consumption based on temperature?
- ✓ Given a dataset of customer purchase histories, how can you use unsupervised learning to segment customers into distinct groups based on their buying behavior? Provide an example of an algorithm you might use.

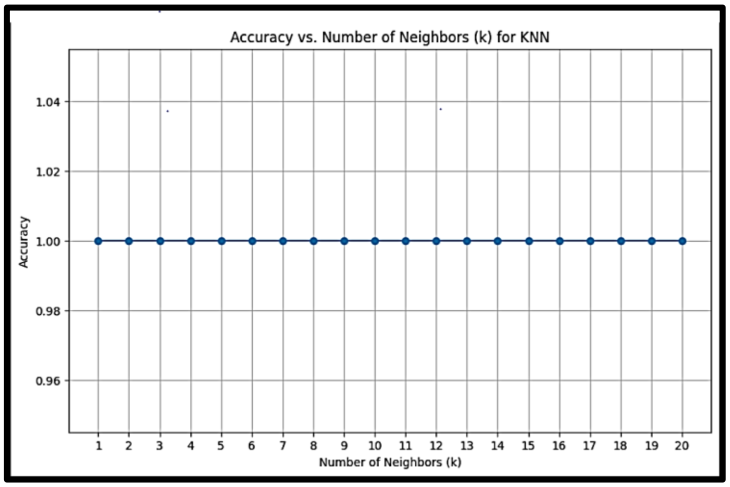
12.7. Practice Task

Here's an example of implementing the k-Nearest Neighbors (KNN) classification algorithm in Python using scikit-learn, and then plotting the accuracy results with a graph.



```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset as an example
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize lists to store accuracy values
k_values = list(range(1, 21))  # Vary k from 1 to 20
accuracy_values = []

# Loop through different values of k
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_values.append(accuracy)

# Plot the accuracy vs. k graph
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracy_values, marker='o', linestyle='-')
plt.title('Accuracy vs. Number of Neighbors (k) for KNN')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.xticks(k_values)
plt.show()
```

Output:



12.8. Evaluation Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time.

12.8.1. Task 1 (Marks 10)

Implement linear regression in Python using scikit-learn to predict housing prices. Load a dataset, split it into training and testing sets, fit the model, and evaluate its performance using mean squared error.

Datasets links: https://www.kaggle.com/datasets/yasserh/housing-prices-dataset

12.8.2. Task 2 (Marks 10)

Apply Principle Component Analysis (PCA) to reduce the dimensionality of a high-dimensional dataset of stock market features. Show the cumulative explained variance and visualize the reduced dataset.

12.8.3. Out comes

After completing this lab, students will to know how to deal with different supervised and unsupervised learning algorithms.

12.9. Evaluation Criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s)

Criteria	**Level 0 (0%)**	**Level 1 (1-50)%**	**Level 2 (51-65)%**	**Level 3 (66-80)%**	**Level 4 (81-100)%**	**Total Score**
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

12.10. Further Readings

12.10.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

12.10.2. Links

https://colab.research.google.com/github/RPI-DATA/course-intro-ml-app/blob/master/content/notebooks/08-intro-modeling/04-knn.ipynb#scrollTo=nhXd2F5ewXPo

Lab Manual for Artificial Intelligence

LAB-13: Performance Metrics in Machine Learning

13. Introduction

Performance metrics in machine learning are used to evaluate the quality and effectiveness of a machine learning model's predictions. These metrics help you understand how well your model is performing on a given task, such as classification, regression, or clustering. The choice of performance metrics depends on the specific problem you are trying to solve and the nature of your data. Here we only discuss some performance metrics for classification and regressions metrics of machine learning tasks:

Classification Metrics:

1. **Accuracy:** This is one of the most basic metrics and measures the ratio of correctly predicted instances to the total instances in the dataset. However, it may not be suitable for imbalanced datasets.

   Example: If a binary classification model correctly predicts 800 out of 1000 instances, the accuracy is 80%.

2. **Precision:** Precision measures the proportion of true positive predictions out of all positive predictions. It is useful when false positives are costly.

   Example: Out of 100 positive predictions, 85 were true positives, so the precision is 85%.

$$Precision = \frac{TP}{TP + FP}$$

3. **Recall (Sensitivity or True Positive Rate):** Recall calculates the proportion of true positive predictions out of all actual positives. It is important when false negatives are costly.

   Example: There were 100 actual positive instances, and the model correctly predicted 85 of them, so the recall is 85%.

$$recall = \frac{TP}{TP + FN}$$

4. **F1-Score:** The F1-Score is the harmonic mean of precision and recall, providing a balanced measure between the two.

   Example: If precision is 0.85 and recall is 0.80, the F1-Score is 0.825.

$$f1_{score} = \frac{precision \times recall}{precision + recall}$$

5. **ROC AUC (Receiver Operating Characteristic Area under Curve):** ROC AUC quantifies the model's ability to distinguish between classes across different thresholds. It's especially useful when you need to control the trade-off between true positive and false positive rates.

   Example: A ROC AUC score of 0.90 indicates good discrimination between classes.

6. **Confusion Matrix:** Not a single metric but a table that summarizes the performance of a classification algorithm, showing true positives, true negatives, false positives, and false negatives.

Regression Metrics:

1. **Mean Absolute Error (MAE):** MAE measures the average absolute difference between predicted and actual values. It is robust to outliers but not differentiable.

   Example: If the MAE is 5, it means, on average, predictions are off by 5 units from actual values.

2. **Mean Squared Error (MSE):** MSE measures the average squared difference between predicted and actual values. It punishes larger errors more than MAE.

   Example: If the MSE is 25, it means, on average, predictions are off by 25 squared units from actual values.

3. **Root Mean Squared Error (RMSE):** RMSE is the square root of MSE and provides an interpretable metric in the same units as the target variable.

   Example: If RMSE is 5, it means, on average, predictions are off by 5 units from actual values.

4. **R-squared (Coefficient of Determination):** R-squared measures the proportion of the variance in the target variable explained by the model. It ranges from 0 to 1, with higher values indicating better fit.

   Example: An R-squared of 0.75 means 75% of the variance in the target variable is explained by the model.

13.1. Relevant Lecture Readings

Revise Lecture Performance Metrics

13.2. Objective of the Experiment

After completing this lab, the student should be able to:

- Understand the concept of different performance metrics such as precision, recall, F1 score etc.

13.3. Concept map

This concept map will help students to understand the main concepts of topics covered in the lab. In this lab, students will be provided with a detailed content to learn about different performance metrics and their implementation in classification and regression metrics.

13.4. Homework before Lab

Attempt the following questions. You are required to bring this task with you and submit to the lab instructor.

13.4.1. Problem description

1. You have a classifier that predicts whether an email is spam or not. Out of 1000 emails, it correctly identifies 400 spam emails and incorrectly classifies 50 non-spam emails as spam. Calculate precision, recall, and F1-Score for this classifier.

2. You have a regression model for predicting house prices. The model has an RMSE of 10. What does this RMSE value signify in the context of your model's

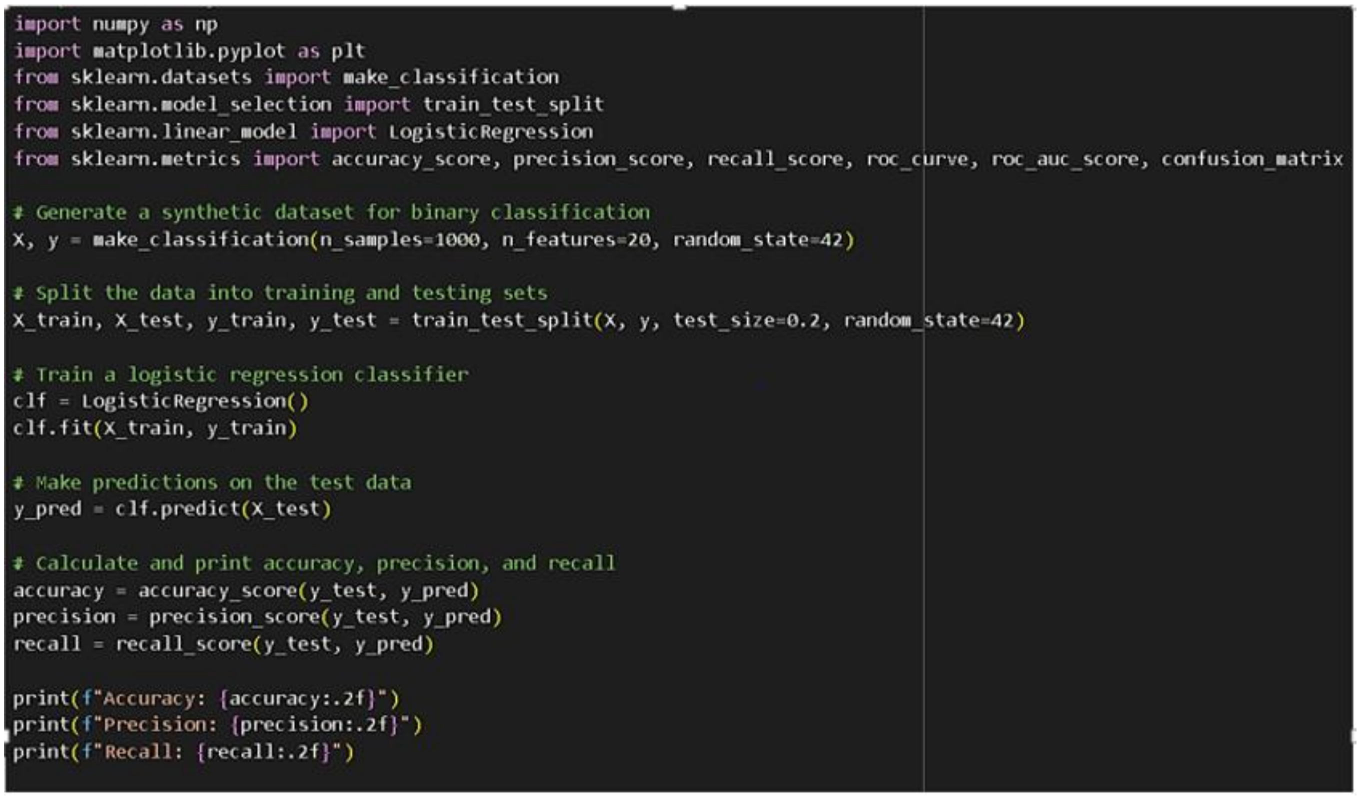
predictions?

13.5. Procedure & Tools

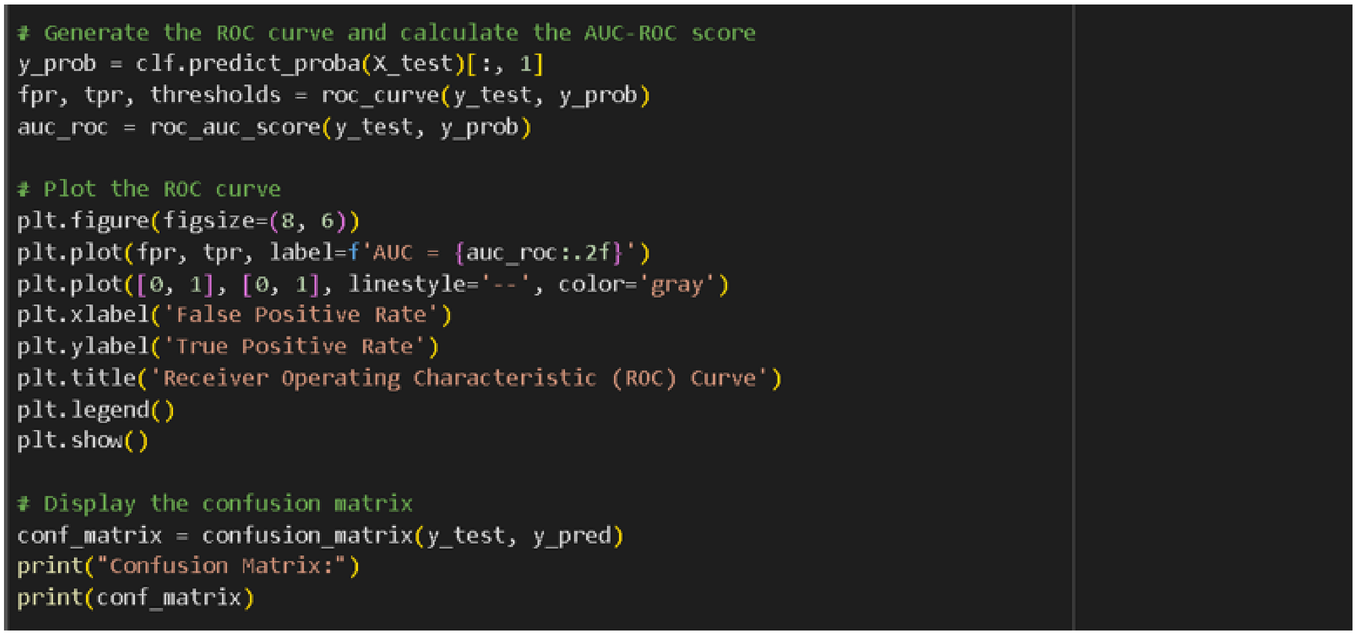
Pycharm
Google Colab

13.6. Practice Task

Certainly! Here's a practice task in Python that involves using performance metrics for a classification problem. In this task, you'll evaluate the performance of a binary classification model using common metrics such as accuracy, precision, recall, and the ROC curve. You can use libraries like scikit-learn to simplify the process.

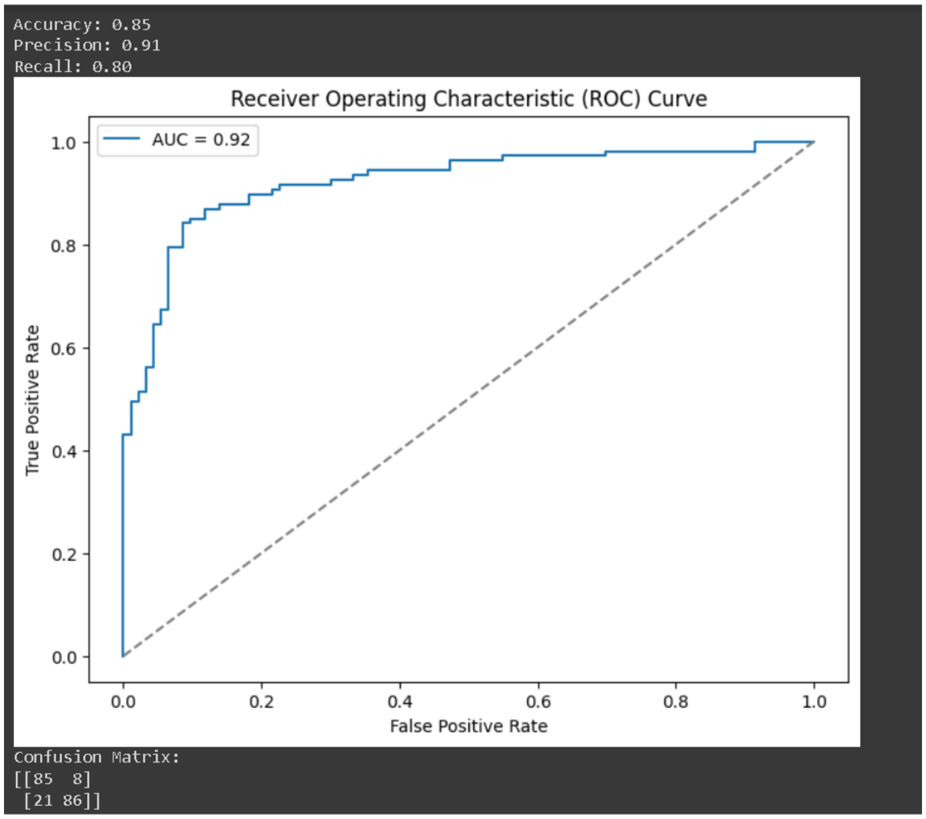


```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve, roc_auc_score, confusion_matrix

# Generate a synthetic dataset for binary classification
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a logistic regression classifier
clf = LogisticRegression()
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate and print accuracy, precision, and recall
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```



```python
# Generate the ROC curve and calculate the AUC-ROC score
y_prob = clf.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
auc_roc = roc_auc_score(y_test, y_prob)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'AUC = {auc_roc:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()

# Display the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

Output:



```
Accuracy: 0.85
Precision: 0.91
Recall: 0.80
```

```
Confusion Matrix:
[[85  8]
 [21 86]]
```

13.7. Evaluation Tasks

This section will provide more evaluation exercises which you need to finish during the lab. You need to finish the tasks in the required time.

13.7.1. Task 1

Write Python code to calculate and print the accuracy, precision, recall, and F1-Score of a binary classification model using scikit-learn. Assume you have the true labels (y_true) and predicted labels (y_pred).

13.7.2. Task 2

Implement a Python function to compute and print the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2) for a regression model using scikit-learn. Assume you have the true target values (`y_true`) and predicted values (`y_pred`).

13.7.3. Out comes

After completing this lab, students will be able to compute different performance metrics.

13.8. Evaluation Criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s)

Table 3: Evaluation of the Lab

Criteria	Level 0 (0%)	Level 1 (1-50)%	Level 2 (51-65)%	Level 3 (66-80)%	Level 4 (81-100)%	Total Score
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

13.9. Further Readings

13.9.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

13.9.2. Links

https://www.javatpoint.com/performance-metrics-in-machine-learning

Lab Manual for Artificial Intelligence

LAB-14: Natural Language Processing (NLP)

14. Introduction

Natural Language Processing (NLP) is the development of applications or services that understand human language. Here are some practical examples of natural language processing (NLP), such as speech recognition, speech translation, understanding complete sentences, understanding synonyms of matching words, and generating grammatically correct complete sentences and paragraphs.

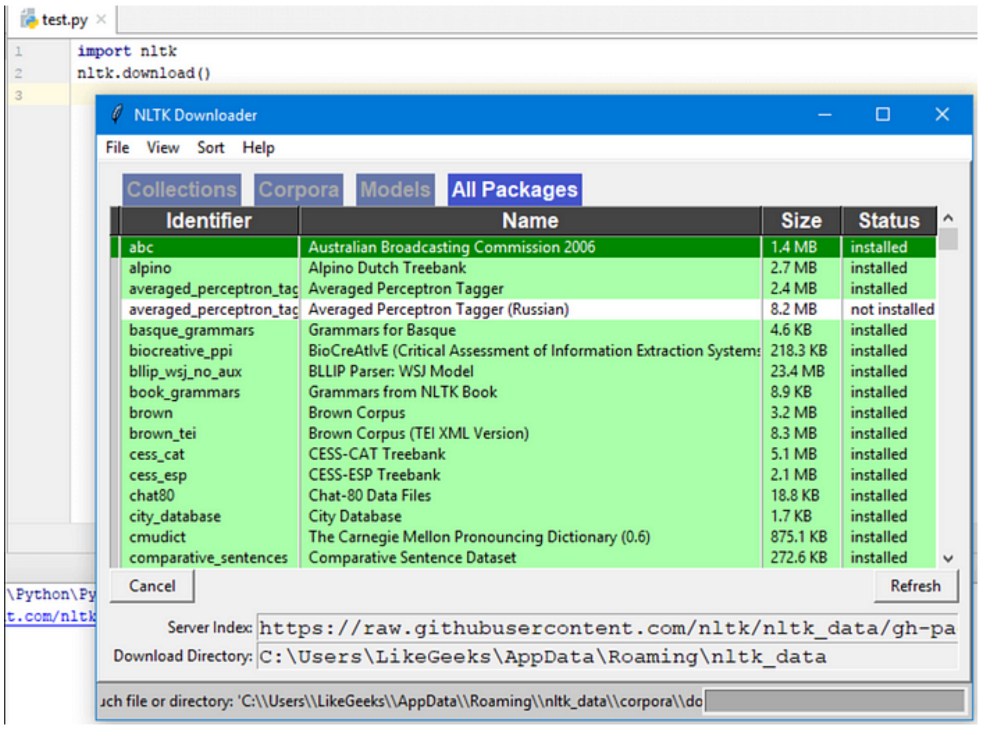
14.1. Practical Examples of NLP

1. **Search Engine:** such as Google, Yahoo, etc. Google search engine knows you are a technician, so it shows technology-related results.
2. **Social Network Push:** Facebook News Feed. If the News Feed algorithm knows that your interest is natural language processing, relevant ads and posts will be displayed.
3. **Speech Engine:** such as Apple's Siri.

14.2. Libraries of NLP

Here are some open-source natural language processing libraries (NLPs):

- **Natural language toolkit (NLTK):** The Natural Language Toolkit (NLTK) is the most popular Natural Language Processing Library (NLP), written in Python, and has very strong community support behind it. NLTK is also very easy to use and , it is the simplest natural language processing (NLP) library.For installation NLTK Windows/Linux/Mac, you can install NLTK using the **pip command**.To install NLTK for the first time, you need to install the NLTK extension package. This will bring up the NLTK download window to choose which packages need to be installed:



```
import nltk
nltk.download()
```

Identifier	Name	Size	Status
abc	Australian Broadcasting Commission 2006	1.4 MB	installed
alpino	Alpino Dutch Treebank	2.7 MB	installed
averaged_perceptron_tag	Averaged Perceptron Tagger	2.4 MB	installed
averaged_perceptron_tag	Averaged Perceptron Tagger (Russian)	8.2 MB	not installed
basque_grammars	Grammars for Basque	4.6 KB	installed
biocreative_ppi	BioCreAtIvE (Critical Assessment of Information Extraction Systems	218.3 KB	installed
bllip_wsj_no_aux	BLLIP Parser: WSJ Model	23.4 MB	installed
book_grammars	Grammars from NLTK Book	8.9 KB	installed
brown	Brown Corpus	3.2 MB	installed
brown_tei	Brown Corpus (TEI XML Version)	8.3 MB	installed
cess_cat	CESS-CAT Treebank	5.1 MB	installed
cess_esp	CESS-ESP Treebank	2.1 MB	installed
chat80	Chat-80 Data Files	18.8 KB	installed
city_database	City Database	1.7 KB	installed
cmudict	The Carnegie Mellon Pronouncing Dictionary (0.6)	875.1 KB	installed
comparative_sentences	Comparative Sentence Dataset	272.6 KB	installed

```
\Python\Py
t.com/nltk
```

Server Index: `https://raw.githubusercontent.com/nltk/nltk_data/gh-pa`

Download Directory: `C:\Users\LikeGeeks\AppData\Roaming\nltk_data`

```
ıch file or directory: 'C:\\Users\\LikeGeeks\\AppData\\Roaming\\nltk_data\\corpora\\do
```

- **SpaCy:** It is a modern and efficient NLP library that offers fast and accurate tokenization, part-of-speech tagging, named entity recognition, and dependency parsing. It is known for its speed and ease of use.

- **Scikit-learn:** It is a popular machine learning library that includes various NLP functionalities. It provides tools for text preprocessing, feature extraction, and classification algorithms that can be applied to NLP tasks.

- **TensorFlow and Keras:** TensorFlow is a powerful deep learning library, and Keras is a high-level neural networks API that runs on top of TensorFlow. They are commonly used for building and training deep learning models for NLP tasks like text classification, sentiment analysis, and machine translation.
- **PyTorch:** PyTorch is another popular deep learning library that provides dynamic computational graphs and a flexible framework for building neural networks. It is widely used for NLP tasks, including language modeling, text generation, and machine translation.

These libraries offer a wide range of functionalities and tools to handle various NLP tasks efficiently. They are constantly evolving and provide extensive documentation and community support, making them valuable resources for NLP practitioners and researchers

14.3. Relevant Lecture Readings

Revise Lecture Natural Language Processing and its parameters

14.4. Objective of the Experiment

After completing this lab, the student should be able to:

- Understand the concept of different parameters of NLP and its practical implementation.

14.5. Concept map

This concept map will help students to understand the main concepts of topics covered in the lab. In this lab, students will be provided with a detailed content to learn about different Natural language processing parameters and their implementation using neural networks.

14.6. Homework before Lab

Attempt the following questions. You are required to bring this task with you and submit to the lab instructor.

14.6.1. Problem description

1. Write Python code to tokenize a text document into words or tokens.
2. How can you use the CountVectorizer from the scikit-learn library to create a BoW representation of a collection of text documents? Provide an example.

14.7. Procedure & Tools

Pycharm
Google Colab

14.8. Practice Task

Implementing GloVe (Global Vectors for Word Representation) from scratch in Python is a complex task, and the full code can be quite lengthy. However, I can provide you with an outline of the steps involved and some key code snippets to get you started. To implement GloVe, you'll need a large text corpus and the following main steps:

- **Data Preparation:**

- Tokenize the corpus into words or tokens.
- Create a word co-occurrence matrix that captures how often words co-occur in the context of each other.

- **Initialize Word Vectors:**

- Initialize word vectors for each word in the vocabulary.
- Initialize bias terms for each word.

- **Define the Objective Function:**

- Define the loss function based on the word co-occurrence matrix and word vectors.

- **Optimization:**

- Use an optimization algorithm (e.g., stochastic gradient descent) to minimize the loss function.
- Update word vectors and bias terms iteratively.

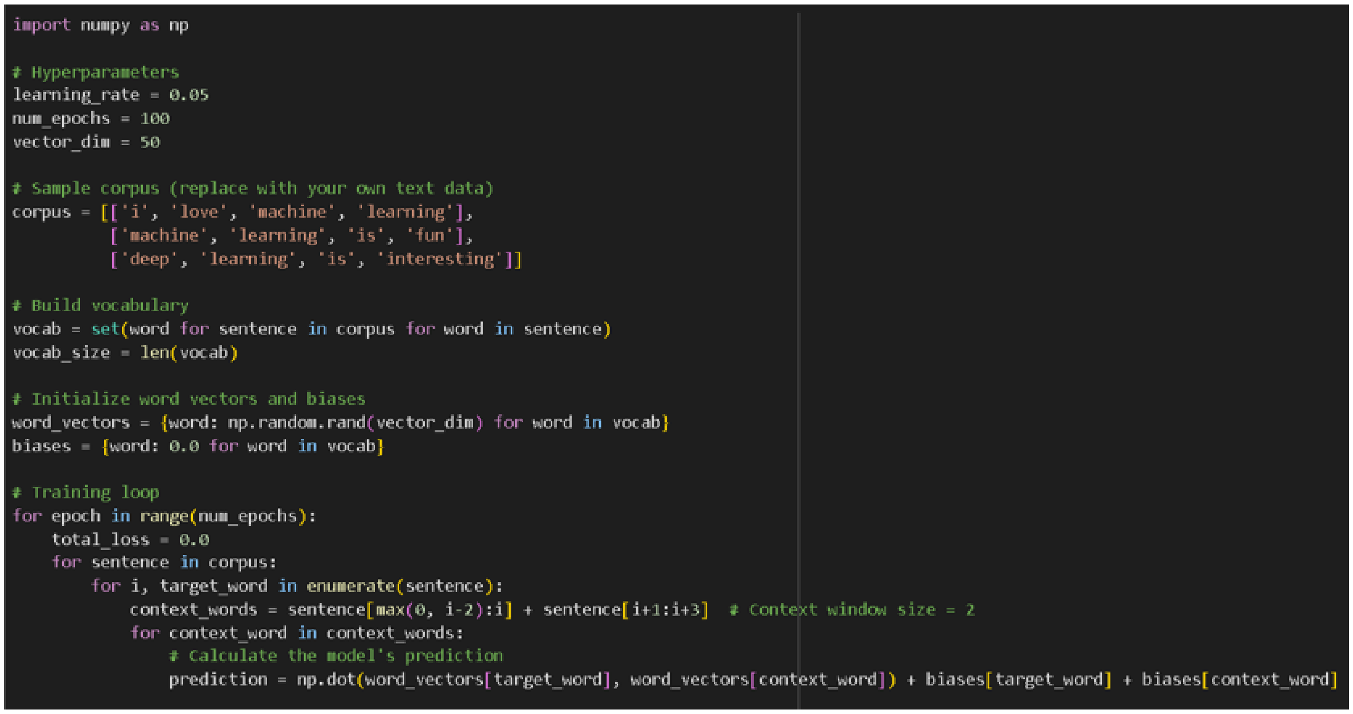
- **Training:**

- Train the model on the corpus for a specified number of epochs.

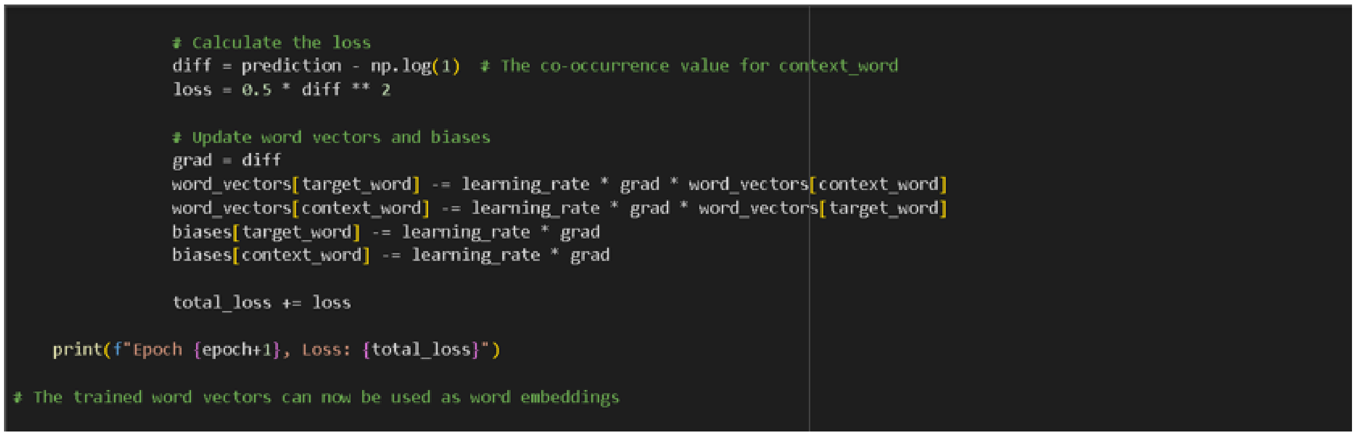
- **Obtain Word Embedding:**

- After training, the word vectors can be used as word embedding.

Here's a simplified code snippet to give you an idea of how you might start implementing GloVe using Python:

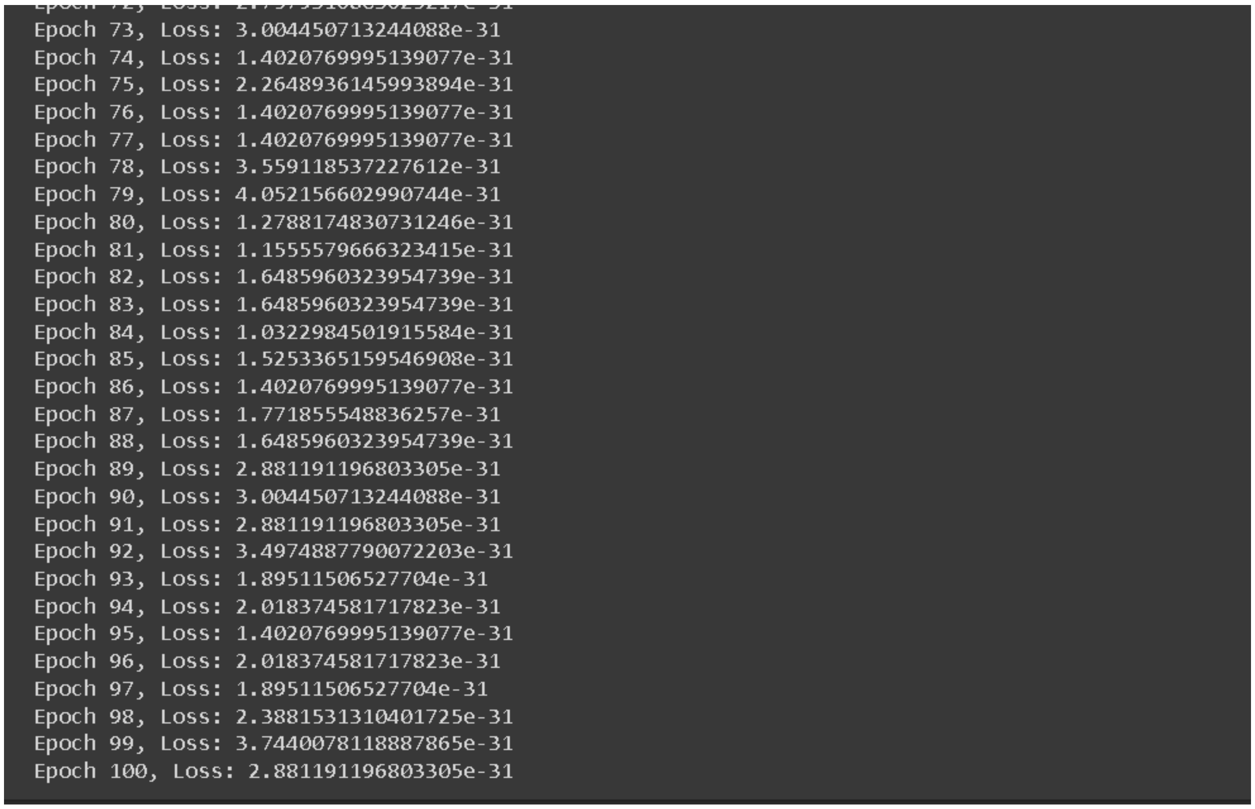


```python
import numpy as np

# Hyperparameters
learning_rate = 0.05
num_epochs = 100
vector_dim = 50

# Sample corpus (replace with your own text data)
corpus = [['i', 'love', 'machine', 'learning'],
          ['machine', 'learning', 'is', 'fun'],
          ['deep', 'learning', 'is', 'interesting']]

# Build vocabulary
vocab = set(word for sentence in corpus for word in sentence)
vocab_size = len(vocab)

# Initialize word vectors and biases
word_vectors = {word: np.random.rand(vector_dim) for word in vocab}
biases = {word: 0.0 for word in vocab}

# Training loop
for epoch in range(num_epochs):
    total_loss = 0.0
    for sentence in corpus:
        for i, target_word in enumerate(sentence):
            context_words = sentence[max(0, i-2):i] + sentence[i+1:i+3]  # Context window size = 2
            for context_word in context_words:
                # Calculate the model's prediction
                prediction = np.dot(word_vectors[target_word], word_vectors[context_word]) + biases[target_word] + biases[context_word]
```



```python
                # Calculate the loss
                diff = prediction - np.log(1)  # The co-occurrence value for context_word
                loss = 0.5 * diff ** 2

                # Update word vectors and biases
                grad = diff
                word_vectors[target_word] -= learning_rate * grad * word_vectors[context_word]
                word_vectors[context_word] -= learning_rate * grad * word_vectors[target_word]
                biases[target_word] -= learning_rate * grad
                biases[context_word] -= learning_rate * grad

                total_loss += loss

    print(f"Epoch {epoch+1}, Loss: {total_loss}")

# The trained word vectors can now be used as word embeddings
```

Output:



```
Epoch 73, Loss: 3.004450713244088e-31
Epoch 74, Loss: 1.4020769995139077e-31
Epoch 75, Loss: 2.2648936145993894e-31
Epoch 76, Loss: 1.4020769995139077e-31
Epoch 77, Loss: 1.4020769995139077e-31
Epoch 78, Loss: 3.559118537227612e-31
Epoch 79, Loss: 4.052156602990744e-31
Epoch 80, Loss: 1.2788174830731246e-31
Epoch 81, Loss: 1.1555579666323415e-31
Epoch 82, Loss: 1.6485960323954739e-31
Epoch 83, Loss: 1.6485960323954739e-31
Epoch 84, Loss: 1.0322984501915584e-31
Epoch 85, Loss: 1.5253365159546908e-31
Epoch 86, Loss: 1.4020769995139077e-31
Epoch 87, Loss: 1.771855548836257e-31
Epoch 88, Loss: 1.6485960323954739e-31
Epoch 89, Loss: 2.881191196803305e-31
Epoch 90, Loss: 3.004450713244088e-31
Epoch 91, Loss: 2.881191196803305e-31
Epoch 92, Loss: 3.4974887790072203e-31
Epoch 93, Loss: 1.89511506527704e-31
Epoch 94, Loss: 2.018374581717823e-31
Epoch 95, Loss: 1.4020769995139077e-31
Epoch 96, Loss: 2.018374581717823e-31
Epoch 97, Loss: 1.89511506527704e-31
Epoch 98, Loss: 2.3881531310401725e-31
Epoch 99, Loss: 3.7440078118887865e-31
Epoch 100, Loss: 2.881191196803305e-31
```

14.9. Evaluation Tasks

This section will provide more evaluation exercises which you need to finish during the lab. You need to finish the tasks in the required time.

14.9.1. Task 1 **(Marks 10)**

Implement text preprocessing steps such as lowercasing, removing punctuation, and stop word removal.

14.9.2. Practice Task 2 **(Marks 10)**

Load pre-trained word embedding (e.g., Word2Vec or GloVe) using python libraries and use them to find similar words for a given word?

14.9.3. Out comes

After completing this lab, students will be able to understand and implement Natural Language Processing.

14.10. Evaluation Criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s)

Table 3: Evaluation of the Lab

Criteria	**Level 0 (0%)**	**Level 1 (1-50)%**	**Level 2 (51-65)%**	**Level 3 (66-80)%**	**Level 4 (81-100)%**	**Total Score**
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

14.11. Further Readings

14.11.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

14.11.2. Links

http://www.dataversity.net/natural-language-processing/

https://www.wired.com/insights/2014/02/growing-importance-natural-language-processing/

https://radimrehurek.com/gensim/models/keyedvectors.html

Glossary

1. **Artificial Intelligence (AI):** The field of computer science that focuses on creating systems or algorithms capable of performing tasks that typically require human intelligence, such as understanding natural language, recognizing patterns, and making decisions.
2. **Machine Learning (ML):** A subfield of AI that involves the development of algorithms that enable computers to learn and improve their performance on a specific task through experience and data.
3. **Deep Learning:** A subset of machine learning that uses neural networks with multiple layers (deep neural networks) to automatically learn patterns and representations from data.
4. **Neural Network:** A computational model inspired by the structure and function of the human brain, used in machine learning and deep learning for tasks such as image recognition and natural language processing.
5. **Supervised Learning:** A type of machine learning in which an algorithm learns from labeled training data to make predictions or classify new, unseen data.
6. **Unsupervised Learning:** A type of machine learning in which an algorithm learns patterns and structures in data without explicit labels or supervision.
7. **Reinforcement Learning:** A machine learning paradigm where an agent learns to make decisions by interacting with an environment and receiving rewards or penalties based on its actions.
8. **Natural Language Processing (NLP):** The branch of AI that focuses on the interaction between computers and human language, enabling machines to understand, interpret, and generate human language.
9. **Feature Engineering:** The process of selecting, transforming, or creating relevant features from raw data to improve the performance of machine learning models.
10. **Data Preprocessing:** The tasks involved in cleaning, formatting, and transforming raw data into a suitable format for training machine learning models.
11. **Overfitting:** A common issue in machine learning where a model performs well on the training data but poorly on unseen data, indicating that it has learned noise rather than meaningful patterns.
12. **Bias-Variance Trade-off:** A fundamental concept in machine learning, where a model's ability to generalize is balanced between underfitting (high bias) and overfitting (high variance).
13. **TensorFlow:** An open-source machine learning framework developed by Google that is widely used for building and training deep learning models.
14. **PyTorch:** An open-source deep learning framework developed by Facebook's AI Research lab (FAIR) that is known for its flexibility and dynamic computation graph.
15. **Hyperparameter:** A configuration setting that is not learned from data but is set prior to training a machine learning model, such as learning rate or the number of hidden layers in a neural network.
16. **Algorithm:** A set of step-by-step instructions for solving a specific problem or task, often used interchangeably with "model" in the context of machine learning.
17. **Bias:** Systematic errors or inaccuracies in a model's predictions or decisions, often resulting from flaws in the data or the model's design.
18. **Variance:** The variability of a model's predictions when trained on different subsets of the data, indicating how sensitive the model is to changes in the training data.
19. **Cross-Validation:** A technique used to assess a model's performance by splitting the data into multiple subsets for training and testing to obtain a more robust evaluation.