AI BASED PROGRAMMING LAB (CS-634)

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(Artificial Intelligence)

2024-26

Submitted to

Dr. Robin Singh Bhadoria

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY, HAMIRPUR

NIT, HAMIRPUR
HIMACHAL PRADESH - 177005

DEC 2024

TABLE OF CONTENTS

Program 1. Python Program on Data and Variables. **20-Aug-2024**

Write a Python program to perform the following operations on a dataset:
1. Load the dataset from a CSV file.
2. Display basic statistics, such as mean, median, standard deviation, and missing values for each column.
3. Handle missing data by either filling it with mean/median values or dropping rows with missing values.
4. Perform data normalization (scaling values between 0 and 1).
5. Group the dataset by a specific column and calculate aggregate statistics (mean, sum, count) for other columns.
6. Visualize the relationships between variables using a pair plot or correlation heatmap.
Use Pandas, NumPy, and Matplotlib/Seaborn libraries to implement the above operations.

Code--

```
import pandas as pd
df = pd.read_csv("/content/drive/MyDrive/annual-enterprise-survey-2023-financial-year-provisional.csv")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
df.head()
```

	Year	Industry_aggregation_NZSIOC	Industry_code_NZSIOC	Industry_name_NZSIOC	Units	Variable_code	Variable_name	Variable_c
0	2023	Level 1	99999	All industries	Dollars (millions)	H01	Total income	F perf
1	2023	Level 1	99999	All industries	Dollars (millions)	H04	Sales, government funding, grants and subsidies	F perf
2	2023	Level 1	99999	All industries	Dollars (millions)	H05	Interest, dividends and donations	F perf
3	2023	Level 1	99999	All industries	Dollars (millions)	H07	Non-operating income	F perf

print(df)

```
       Year Industry_aggregation_NZSIOC Industry_code_NZSIOC  \
0      2023                     Level 1                99999   
1      2023                     Level 1                99999   
2      2023                     Level 1                99999   
3      2023                     Level 1                99999   
4      2023                     Level 1                99999   
...     ...                         ...                  ...   
50980  2013                     Level 3                 ZZ11   
50981  2013                     Level 3                 ZZ11   
50982  2013                     Level 3                 ZZ11   
50983  2013                     Level 3                 ZZ11   
50984  2013                     Level 3                 ZZ11   

               Industry_name_NZSIOC               Units Variable_code  \
0                    All industries  Dollars (millions)           H01   
1                    All industries  Dollars (millions)           H04   
2                    All industries  Dollars (millions)           H05   
3                    All industries  Dollars (millions)           H07   
4                    All industries  Dollars (millions)           H08   
...                             ...                 ...           ...   
50980  Food product manufacturing          Percentage           H37   
50981  Food product manufacturing          Percentage           H38   
50982  Food product manufacturing          Percentage           H39   
50983  Food product manufacturing          Percentage           H40   
50984  Food product manufacturing          Percentage           H41   

                                           Variable_name      Variable_category  \
0                                           Total income  Financial performance   
1      Sales, government funding, grants and subsidies  Financial performance   
2                     Interest, dividends and donations  Financial performance   
3                                  Non-operating income  Financial performance   
4                                     Total expenditure  Financial performance   
...                                                  ...                    ...   
50980                                        Quick ratio       Financial ratios   
50981              Margin on sales of goods for resale       Financial ratios   
50982                                   Return on equity       Financial ratios   
50983                               Return on total assets       Financial ratios   
50984                                Liabilities structure       Financial ratios   

        Value                               Industry_code_ANZSIC06  
0      930995  ANZSIC06 divisions A-S (excluding classes K633...  
1      821630  ANZSIC06 divisions A-S (excluding classes K633...  
2       84354  ANZSIC06 divisions A-S (excluding classes K633...  
3       25010  ANZSIC06 divisions A-S (excluding classes K633...  
4      832964  ANZSIC06 divisions A-S (excluding classes K633...  
...       ...                                                ...  
50980      52  ANZSIC06 groups C111, C112, C113, C114, C115, ...  
50981      40  ANZSIC06 groups C111, C112, C113, C114, C115, ...  
50982      12  ANZSIC06 groups C111, C112, C113, C114, C115, ...  
50983       5  ANZSIC06 groups C111, C112, C113, C114, C115, ...  
50984      46  ANZSIC06 groups C111, C112, C113, C114, C115, ...  

[50985 rows x 10 columns]
```

df.tail()

	Year	Industry_aggregation_NZSIOC	Industry_code_NZSIOC	Industry_name_NZSIOC	Units	Variable_code	Variable_name	Vari
50980	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H37	Quick ratio	
50981	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H38	Margin on sales of goods for resale	
50982	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H39	Return on equity	

50983	2013	Level 3	ZZ11	Food product	Percentage	H40	Return on total

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50985 entries, 0 to 50984
Data columns (total 10 columns):
 #   Column                       Non-Null Count  Dtype
-----  ------------------------   --------------  -----------
 0   Year                         50985 non-null  int64
 1   Industry_aggregation_NZSIOC  50985 non-null  object
 2   Industry_code_NZSIOC         50985 non-null  object
 3   Industry_name_NZSIOC         50985 non-null  object
 4   Units                        50985 non-null  object
 5   Variable_code                50985 non-null  object
 6   Variable_name                50985 non-null  object
 7   Variable_category            50985 non-null  object
 8   Value                        50985 non-null  object
 9   Industry_code_ANZSIC06       50985 non-null  object
dtypes: int64(1),
object(9)memory usage:
3.9+ MB
```

df.describe()

	Year
count	50985.000000
mean	2018.000000
std	3.162309
min	2013.000000
25%	2015.000000
50%	2018.000000
75%	2021.000000
max	2023.000000

df.columns()

```
Index(['Year', 'Industry_aggregation_NZSIOC', 'Industry_code_NZSIOC',
       'Industry_name_NZSIOC', 'Units', 'Variable_code',
       'Variable_name','Variable_category', 'Value',
       'Industry_code_ANZSIC06'],
      dtype='object')
```

df.columns.values

```
array(['Year', 'Industry_aggregation_NZSIOC', 'Industry_code_NZSIOC',
       'Industry_name_NZSIOC', 'Units', 'Variable_code',
       'Variable_name','Variable_category', 'Value',
       'Industry_code_ANZSIC06'],
      ditypes=object)
```

Program 2. Python Program on Prediction **27-Aug-2024**

Write a Python program to predict house prices using a machine learning model.
The program should:
1. Load the housing dataset (CSV file) containing features such as number of rooms, location, area, etc.
2. Preprocess the data by handling missing values, encoding categorical variables,and scaling numeric features.
3. Split the data into training and testing sets.
4. Train a regression model (e.g., Linear Regression, Decision Tree, or SVR) on the training data.
5. Predict the prices of houses from the test set and compare them to the actual prices.
6. Visualize the results by plotting predicted vs. actual prices.

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
df = pd.read_csv("/content/drive/MyDrive/Housing.csv")
```

```
Mounted at /content/drive
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

```
df.isnull().sum()
```

	0
price	0
area	0
bedrooms	0
bathrooms	0
stories	0
mainroad	0
guestroom	0
basement	0
hotwaterheating	0
airconditioning	0
parking	0
prefarea	0
furnishingstatus	0

```
df_area_price=df[['area ','price']]
print(df_area_price)
```

```
     area     price
0    7420  13300000
1    8960  12250000
2    9960  12250000
3    7500  12215000
4    7420  11410000
..    ...       ...
540  3000   1820000
541  2400   1767150
542  3620   1750000
543  2910   1750000
544  3850   1750000

[545 rows x 2 columns]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(df[['area']],df[['price']], test_size =
0.20, random_state = 42)
from sklearn import svm
from sklearn.svm import SVC
model_SVR = svm.SVR()
model_SVR.fit(X_train,Y_train)
Y_pred = model_SVR.predict(X_test) print(X_test,Y_pred)
```

```
     area
316  5900
77   6500
360  4040
90   5000
493  3960
..    ...
15   6000
357  6930
39   6000
54   6000
155  6100

[109 rows x 1 columns] [4291068.98955744 4291089.5215933  4290965.72765436 4291019.75280652
 4290962.19214473 4291093.91286443 4291083.53528354 4291019.14248896
 4290941.85393168 4290941.25821675 4291082.48048294 4290961.09167094
 4290953.00664623 4290940.51444481 4290962.62186104 4290950.15591473
 4290955.63616614 4291073.25297259 4291066.7414097  4291073.25297259
 4291020.97246442 4291095.42510977 4290950.72950496 4290955.81963026
 4291088.13116148 4291060.37359185 4290940.45163963 4290940.08048196
 4291035.35010525 4290940.08048196 4290963.93218271 4290940.65311372
 4291073.25297259 4291092.02658269 4291006.02222461 4290995.53258301
 4290998.25931194 4290941.17417542 4290946.17130355 4290940.5497816
 4291092.24530288 4290950.15591473 4291087.50485529 4290973.42627436
 4291094.38265149 4291068.32308962 4291073.25297259 4290997.90219386
 4291091.24426482 4290940.08048196 4291097.36479907 4290940.08048196
 4291094.88183254 4290989.707461   4290950.45590319 4290940.05254572
 4291062.02377763 4290945.74409002 4291088.28758918 4290955.81963026
 4290941.17417542 4290951.89952393 4291088.03020756 4291017.31006544
 4291041.21305213 4290954.38092232 4291091.7260467  4290940.08048196
 4291085.61636812 4291059.55954211 4290940.21945452 4291085.8440915
 4291085.61636812 4291054.02930696 4290941.41939442 4290976.50830046
 4290950.15591473 4291008.45412197 4291073.25297259 4291037.74064428
 4291085.8440915  4290967.57674046 4291049.07050938 4291093.06187824
 4290964.91005414 4291071.58560127 4290940.46883351 4290945.49202657
 4291097.63268949 4291073.25297259 4290958.45166018 4291091.24426482
 4291046.30731748 4290944.64810418 4291082.48048294 4290952.34403498
 4291065.72844119 4291064.41864297 4291049.07050938 4291091.7260467
 4290941.85393168 4290995.53258301 4290940.03350926 4290942.69336472
 4291073.25297259 4291096.57489399 4291073.25297259 4291073.25297259
 4291077.19298718]
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1183: DataConversionWarning: A column-vector y was passed when
  y = column_or_1d(y, warn=True)
```

Program 3. Write python programs to implement BFS & DFS Algorithms

27-Aug-2024

I. **DFS**

```
# Maze dimensions and
obstacles maze_size = 6
obstacles = [(0,1),(1,1),(3,2),(3,3),(3,4),(3,5),(0,4),(4,1),(4,2),(4,3)]
start = (0,0)
goal = (0,5)
# checks whether a given position of (x,y) is valid
to move or not def is_valid(x,y):
 return 0 <= x < maze_size and 0 <= y < maze_size and (x,y)
not in obstacles #Dfs function (Depth-first search)
def dfs (current, visited, path):
 x, y = current
 if current == goal:
  path.append(cur
  rent) return
  True
 visited.add(current)
 moves = [(x-1,y), (x+1, y), (x, y-
 1), (x, y+1)] for move in moves:
  if is_valid(*move) and move not
   in visited: if dfs(move, visited,
   path):
    path.append(cur
    rent) return
    True
 return False
#Call DFS function to find
the path visited = set()
path = []
if dfs(start, visited,
 path): path.reverse()
 print("Path
 found:") for
 position in
 path:
 print(position)
else:
  print("No path found!")
```

```
Path found:
(0, 0)
(1, 0)
(2, 0)
(3, 0)
(3, 1)
(2, 1)
(2, 2)
(1, 2)
(0, 2)
(0, 3)
(1, 3)
(2, 3)
(2, 4)
(1, 4)
(1, 5)
(0, 5)
```

II. **BFS**

```
from collections import deque


class GridProblem:
    def __init__(self, initial_state, goal_state, grid):
        # Initializes a grid problem instance with initial and goal states, and the grid layout
        self.initial_state = initial_state
        self.goal_state = goal_state
        self.grid = grid

    def is_goal(self, state):
        # Checks if the given state is the goal state
        return state == self.goal_state

    def is_valid_cell(self, row, col):
        # Checks if the given cell coordinates are within the grid boundaries and not
blocked
        return 0 <= row < len(self.grid) and 0 <= col < len(self.grid[0]) and
self.grid[col][row] == 0

    def expand(self, node):
        # Expands the given node by generating child nodes for valid adjacent cells
        row, col = node.state
        children = []
        for dr, dc in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            new_row, new_col = row + dr, col + dc
            if self.is_valid_cell(new_row, new_col):
                child_state = (new_row, new_col)
                child_node = Node(child_state, parent=node)
                children.append(child_node)
        return children


class Node:
    def __init__(self, state, parent=None, action=None):
        # Initializes a node with a state, parent node (optional), and action (optional)
        self.state = state
        self.parent = parent
        self.action = action


def breadth_first_search(problem):
    # Performs breadth-first search algorithm to find a solution for the given problem
    node = Node(problem.initial_state)
    if problem.is_goal(node.state):
        return node

    frontier = deque([node])
    reached = {problem.initial_state}

    while frontier:
        node = frontier.popleft()

        for child in problem.expand(node):
            state = child.state

            if problem.is_goal(state):
                return child
            if state not in reached:
                reached.add(state)
                frontier.append(child)
    return None


def reconstruct_path(node):
    # Reconstructs the path from the goal node back to the initial node
    path = []
    while node:
        path.append(node.state)
```

```
        node = node.parent
    return list(reversed(path))


def print_complete_path(path):
    # Prints the complete path from start to goal
    if path:
        for step, point in enumerate(path):
            print("Step {}: {}".format(step, point))
    else:
        print("No solution found")


# Example usage and grid definition
"""
    1 : Denotes the obstacles
    0 : Empty space or a non-obstacle cell in the grid
"""
grid = [
    [0, 1, 0, 0, 1, 0, 0],
    [0, 1, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 1, 0, 0],
    [0, 0, 1, 0, 1, 0, 0],
    [0, 0, 1, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0, 0]
]

# Define initial and goal states
initial_state = (0, 0)
goal_state = (6, 0)

# Define the problem instance
problem = GridProblem(initial_state, goal_state, grid)

# Perform breadth-first search to find a solution
solution_node = breadth_first_search(problem)

# Print solution if found
print('!! Reached the Goal!!' if solution_node else None)
if solution_node:
    print("Solution found!")
    solution_path = reconstruct_path(solution_node)
    print("Complete Path:")
    print_complete_path(solution_path)
else:
    print("No solution found")
```

```
!! Reached the Goal!!
Solution found!
Complete Path:
Step 0: (0, 0)
Step 1: (0, 1)
Step 2: (0, 2)
Step 3: (1, 2)
Step 4: (2, 2)
Step 5: (3, 2)
Step 6: (3, 3)
Step 7: (3, 4)
Step 8: (4, 4)
Step 9: (5, 4)
Step 10: (6, 4)
Step 11: (6, 3)
Step 12: (6, 2)
Step 13: (6, 1)
Step 14: (6, 0)
```

Program 4. Write a python program to implement Uniform Cost Search.

03-Sept-2024

UNIFORM COST SEARCH

```
# Python3 implementation of above approach
def uniform_cost_search(goal, start):
  # minimum cost upto
  # goal state from starting
  global graph,cost
  answer = []
  # create a priority queue
  queue = []
  # set the answer vector to max value
  for i in range(len(goal)):
    answer.append(10**8)
  # insert the starting index
  queue.append([0, start])
  # map to store visited node
  visited = {}
  # count
  count = 0
  # while the queue is not empty
  while (len(queue) > 0):
    # get the top element of the
    queue = sorted(queue)
    p = queue[-1]
    # pop the element
    del queue[-1]
    # get the original value
    p[0] *= -1
    # check if the element is part of
    # the goal list
    if (p[1] in goal):
      index = goal.index(p[1])
      # if a new goal is reached
      if (answer[index] == 10**8):
        count += 1
      # if the cost is less
      if (answer[index] > p[0]):
        answer[index] = p[0]
      # pop the element
      del queue[-1]
      queue = sorted(queue)
      if (count == len(goal)):
        return answer
    # check for the non visited nodes
    # which are adjacent to present node
    if (p[1] not in visited):
      for i in range(len(graph[p[1]])):
        # value is multiplied by -1 so that
        # least priority is at the top
        queue.append([(p[0] + cost[(p[1],
graph[p[1]][i])])* -1,graph[p[1]][i]])
    # mark as visited
    visited[p[1]] = 1
  return answer
# main function
if _name_ == '_main_':
  # create the graph
```

```
graph,cost = [[] for i in range(8)],{}
# add edge
graph[0].append(1)
graph[0].append(3)
graph[3].append(1)
graph[3].append(6)
graph[3].append(4)
graph[1].append(6)
graph[4].append(2)
graph[4].append(5)
graph[2].append(1)
graph[5].append(2)
graph[5].append(6)
graph[6].append(4)
# add the cost
cost[(0, 1)] = 2
cost[(0, 3)] = 5
cost[(1, 6)] = 1
cost[(3, 1)] = 5
cost[(3, 6)] = 6
cost[(3, 4)] = 2
cost[(2, 1)] = 4
cost[(4, 2)] = 4
cost[(4, 5)] = 3
cost[(5, 2)] = 6
cost[(5, 6)] = 3
cost[(6, 4)] = 7
# goal state
goal = []
# set the goal
# there can be multiple goal states
goal.append(6)
# get the answer
answer = uniform_cost_search(goal, 0)
# print the answer
print("Minimum cost from 0 to 6 is  =  ",answer[0])
```

```
Minimum cost from 0 to 6 is = 3
```

Program 5. Write a python program to implement A* Search. 03-Sept-2024

A* SEARCH

```
# Python program for A* Search Algorithm
import math
import heapq
# Define the Cell class
class Cell:
  def __init__(self):

    # Parent cell's row index
     self.parent_i = 0
   # Parent cell's column index
     self.parent_j = 0
 # Total cost of the cell (g + h)
     self.f = float('inf')
    # Cost from start to this cell
     self.g = float('inf')
    # Heuristic cost from this cell to destination
     self.h = 0
# Define the size of the grid
ROW = 9
COL = 10
# Check if a cell is valid (within the grid)
def is_valid(row, col):
 return (row >= 0) and (row < ROW) and (col >= 0) and (col < COL)
# Check if a cell is unblocked
def is_unblocked(grid, row, col):
 return grid[row][col] == 1
# Check if a cell is the destination
def is_destination(row, col, dest):
 return row == dest[0] and col == dest[1]
# Calculate the heuristic value of a cell (Euclidean distance to destination)
def calculate_h_value(row, col, dest):
 return ((row - dest[0]) ** 2 + (col - dest[1]) ** 2) ** 0.5
# Trace the path from source to destination
def trace_path(cell_details, dest):
 print("The Path is ")
 path = []
 row = dest[0]
 col = dest[1]
 # Trace the path from destination to source using parent cells
 while not (cell_details[row][col].parent_i == row and cell_details[row][col].parent_j == col):
  path.append((row, col))
  temp_row = cell_details[row][col].parent_i
  temp_col = cell_details[row][col].parent_j
  row = temp_row
  col = temp_col
 # Add the source cell to the path
 path.append((row, col))
 # Reverse the path to get the path from source to destination
 path.reverse()
 # Print the path
 for i in path:
  print("->", i, end=" ")
 print()
 # Implement the A* search algorithm
def a_star_search(grid, src, dest):
 # Check if the source and destination are valid
 if not is_valid(src[0], src[1]) or not is_valid(dest[0], dest[1]):
  print("Source or destination is invalid")
  return
 # Check if the source and destination are unblocked
 if not is_unblocked(grid, src[0], src[1]) or not is_unblocked(grid, dest[0], dest[1]):
```

```
  print("Source or the destination is blocked")
  return
 # Check if we are already at the destination
 if is_destination(src[0], src[1], dest):
  print("We are already at the destination")
  return
 # Initialize the closed list (visited cells)
 closed_list = [[False for _ in range(COL)] for _ in range(ROW)]
 # Initialize the details of each cell
 cell_details = [[Cell() for _ in range(COL)] for _ in range(ROW)]
 # Initialize the start cell details
 i = src[0]
 j = src[1]
 cell_details[i][j].f = 0
 cell_details[i][j].g = 0
 cell_details[i][j].h = 0
 cell_details[i][j].parent_i = i
 cell_details[i][j].parent_j = j
 # Initialize the open list (cells to be visited) with the start cell
 open_list = []
 heapq.heappush(open_list, (0.0, i, j))
 # Initialize the flag for whether destination is found
 found_dest = False
 # Main loop of A* search algorithm
 while len(open_list) > 0:
  # Pop the cell with the smallest f value from the open list
  p = heapq.heappop(open_list)
  # Mark the cell as visited
  i = p[1]
  j = p[2]
  closed_list[i][j] = True
  # For each direction, check the successors
  directions = [(0, 1), (0, -1), (1, 0), (-1, 0),(1, 1), (1, -1), (-1, 1), (-1, -1)]
  for dir in directions:
   new_i = i + dir[0]
   new_j = j + dir[1]
   # If the successor is valid, unblocked, and not visited
   if is_valid(new_i, new_j) and is_unblocked(grid, new_i, new_j) and not closed_list[new_i][new_j]:
    # If the successor is the destination
    if is_destination(new_i, new_j, dest):
     # Set the parent of the destination cell
     cell_details[new_i][new_j].parent_i = i
     cell_details[new_i][new_j].parent_j = j
     print("The destination cell is found")
     # Trace and print the path from source to destination
     trace_path(cell_details, dest)
     found_dest = True
     return
    else:
     # Calculate the new f, g, and h values
     g_new = cell_details[i][j].g + 1.0
     h_new = calculate_h_value(new_i, new_j, dest)
     f_new = g_new + h_new
     # If the cell is not in the open list or the new f value is smaller
     if cell_details[new_i][new_j].f == float('inf') or cell_details[new_i][new_j].f > f_new:
      # Add the cell to the open list
      heapq.heappush(open_list, (f_new, new_i, new_j))
      # Update the cell details
      cell_details[new_i][new_j].f = f_new
      cell_details[new_i][new_j].g = g_new
      cell_details[new_i][new_j].h = h_new
      cell_details[new_i][new_j].parent_i = i
      cell_details[new_i][new_j].parent_j = j
 # If the destination is not found after visiting all cells
 if not found_dest:
  print("Failed to find the destination cell")
# Driver Code
def main():
 # Define the grid (1 for unblocked, 0 for blocked)
 grid = [
  [1, 0, 1, 1, 1, 1, 0, 1, 1, 1],
  [1, 1, 1, 0, 1, 1, 1, 0, 1, 1],
  [1, 1, 1, 0, 1, 1, 0, 1, 0, 1],
  [0, 0, 1, 0, 1, 0, 0, 0, 0, 1],
  [1, 1, 1, 0, 1, 1, 1, 0, 1, 0],
  [1, 0, 1, 1, 1, 1, 0, 1, 0, 0],
  [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
  [1, 0, 1, 1, 1, 1, 0, 1, 1, 1],
  [1, 1, 1, 0, 0, 0, 1, 0, 0, 1]
 ]
```

```
# Define the source and destination
src = [8, 0]
dest = [0, 0]
# Run the A* search algorithm a_star_search(grid, src, dest)
if __name__ == "__main__":
  main()
```

```
The destination cell is found The Path is -> (8, 0) -> (7, 0) -> (6, 0) -> (5, 0) -
> (4, 1) -> (3, 2) -> (2, 1) -> (1, 0) -> (0, 0)
```

Program 6. Write a Python program to implement the AO* algorithm for solving an And-Or graph. The program should take an And-Or graph as input, search for the optimal solution

10-Sept-2024

```
# Cost to find the AND and
OR path def Cost(H,
condition, weight = 1):
 cost = {}
 if 'AND' in condition:
  AND_nodes = condition['AND']
  Path_A = ' AND '.join(AND_nodes)
  PathA = sum(H[node]+weight for node in
  AND_nodes)cost[Path_A] = PathA
 if 'OR' in condition:
  OR_nodes = condition['OR']
  Path_B =' OR '.join(OR_nodes)
  PathB = min(H[node]+weight for node in
  OR_nodes)cost[Path_B] = PathB
 return cost
# Update the cost
def update_cost(H, Conditions, weight=1):
 Main_nodes =
 list(Conditions.keys())
 Main_nodes.reverse()
 least_cost= {}
 for key in Main_nodes:
  condition = Conditions[key]
  print(key,':', Conditions[key],'>>>', Cost(H, condition,
  weight))c = Cost(H, condition, weight)
  H[key] = min(c.values())
  least_cost[key] = Cost(H, condition,
 weight)return least_cost

# Print the shortest path
def shortest_path(Start,Updated_cost, H):
 Path = Start
 if Start in Updated_cost.keys():
  Min_cost =
  min(Updated_cost[Start].values())key
  = list(Updated_cost[Start].keys())
  values =
  list(Updated_cost[Start].values())
  Index = values.index(Min_cost)
  # FIND MINIMIMUM PATH KEY
  Next =
  key[Index].split(
  )# ADD TO PATH
  FOR OR PATH
  if len(Next) == 1:
   Start =Next[0]
   Path += '<--' +shortest_path(Start,
  Updated_cost, H)# ADD TO PATH FOR AND PATH
  else:
   Path +='<--
   ('+key[Index]+') '
   Start = Next[0]
   Path += '[' +shortest_path(Start, Updated_cost, H)
   + ' + 'Start = Next[-1]
   Path += shortest_path(Start, Updated_cost,
 H) + ']'return Path

H = {'A': -1, 'B': 5, 'C': 2, 'D': 4, 'E': 7, 'F': 9, 'G': 3, 'H': 0, 'I':0, 'J':0}
Conditions = {
'A': {'OR': ['B'], 'AND':
['C', 'D']},'B': {'OR': ['E',
'F']},
'C': {'OR': ['G'], 'AND':
['H', 'I']},'D': {'OR': ['J']}
}
# weight
weight = 1
# Updated cost
print('Updated Cost :')
Updated_cost = update_cost(H, Conditions,
```

```
weight=1)print('*'*75)
print('Shortest Path :\n',shortest_path('A', Updated_cost,H))
```

```
Updated Cost :
D : {'OR': ['J']} >>> {'J': 1}
C : {'OR': ['G'], 'AND': ['H', 'I']} >>> {'H AND I': 2,
'G': 4}B : {'OR': ['E', 'F']} >>> {'E OR F': 8}
A : {'OR': ['B'], 'AND': ['C', 'D']} >>> {'C AND D': 5, 'B': 9}
*******************************************************************
********Shortest Path :
 A<--(C AND D) [C<--(H AND I) [H + I] + D<--J]
```

Program 7. Write a Python program to implement the Alpha-Beta Pruning algorithm for optimizing the Minimax decision-making process in a two-player game. 10-Sept-2024

```
# Python program to demonstrate working of Alpha-Beta Pruning # Initial values of Alpha and Beta
MAX, MIN = 1000, -1000
# Returns optimal value for current player #(Initially called for root and maximizer)
def minimax(depth, nodeIndex, maximizingPlayer, values, alpha, beta):
	# Terminating condition. i.e # leaf node is reached
	if depth == 3:
		return values[nodeIndex]
	if maximizingPlayer:
		best = MIN
		# Recur for left and right children
		for i in range(0, 2):
			val = minimax(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
			best = max(best, val)
			alpha = max(alpha, best) # Alpha Beta Pruning
			if beta <= alpha:
				break
		return best
	else:
		best = MAX
		# Recur for left and # right children
		for i in range(0, 2):
			val = minimax(depth + 1, nodeIndex * 2 + i, True, values, alpha, beta)
			best = min(best, val)
			beta = min(beta, best)
			# Alpha Beta Pruning
			if beta <= alpha:
				break
		return best

# Driver Code

if __name__ == "__main__":
	values = [3, 5, 6, 9, 1, 2, 0, -1]
	print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))

The optimal value is : 5
```

Program 8. Implement a K-Nearest Neighbours (KNN) classification model using the Iris dataset to predict the species of iris flowers. 17-Sept-2024

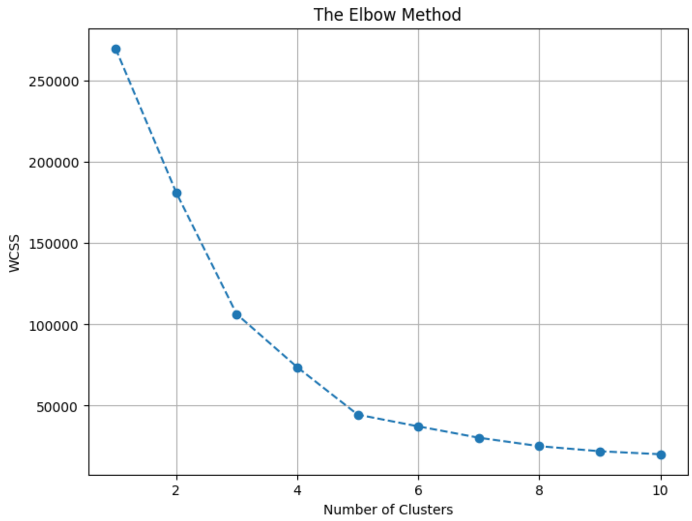
```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection
import train_test_split from
sklearn.datasets import load_iris
from sklearn.metrics import f1_score,
recall_score, accuracy_score # Loading data
irisData = load_iris()
# Create feature and
target arrays X =
irisData.data
y = irisData.target
# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42) # Initialize KNN classifier
knn =
KNeighborsClassifier(n_ne
ighbors=7) # Fit the model
knn.fit(X_train, y_train)
# Predict on dataset which model
has not seen before y_pred =
knn.predict(X_test)
# Print predictions
print("Predictions:", y_pred)
# Calculate and print F1 score,
recall, and accuracy f1 =
f1_score(y_test, y_pred,
average='weighted')
recall = recall_score(y_test, y_pred,
average='weighted') accuracy =
accuracy_score(y_test, y_pred)
print(f"F1
Score: {f1}")
print(f"Recall
: {recall}")
print(f"Accuracy: {accuracy}")
```

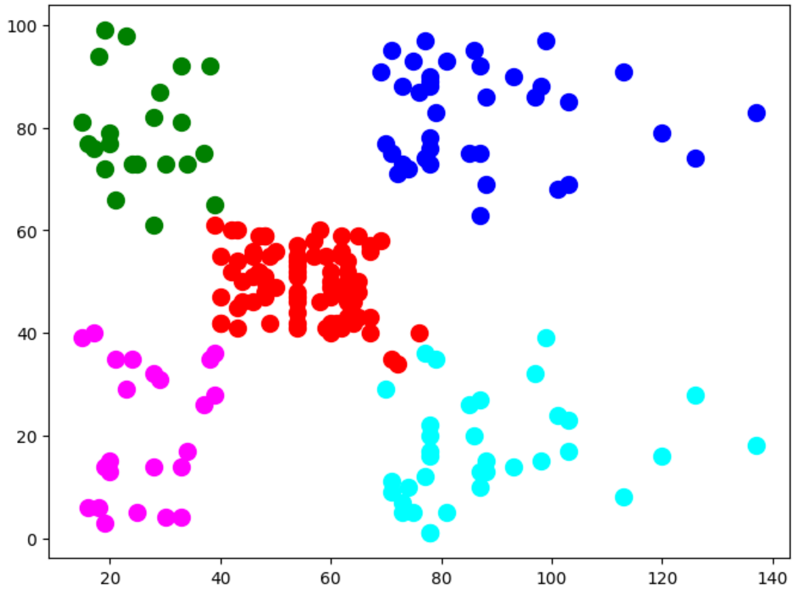
```
Predictions: [1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
F1 Score: 0.9664109121909632
Recall: 0.9666666666666667
Accuracy: 0.9666666666666667
```

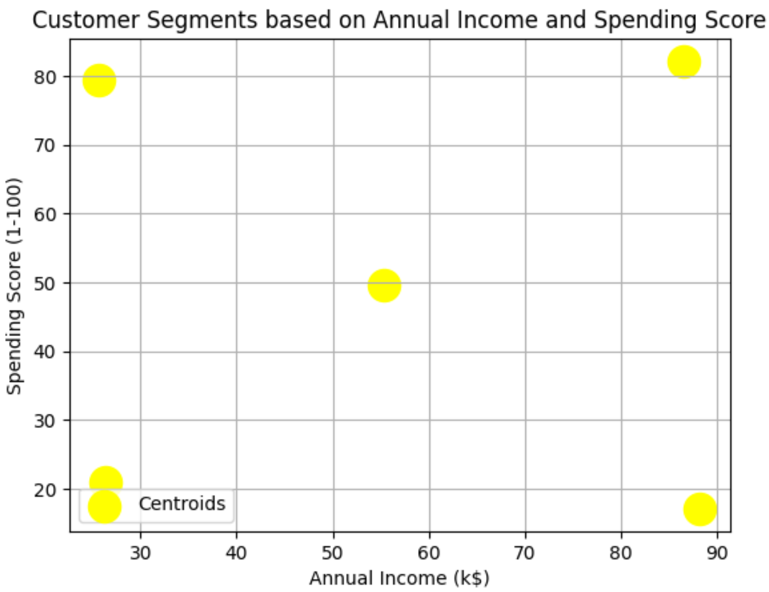
Program 9. Apply the K-Means clustering algorithm on a customer dataset to group customers according to their annual income and spending score. 17-Sept-2024

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load the customer dataset
data = pd.read_csv('Mall_Customers.csv')
data

data.columns.tolist()

X = data[['Annual Income (k$)', 'Spending Score (1-100)']].values

wcss = []  # Within-cluster sum of squares_24_mcs_105
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plot the Elbow method graph
plt.figure(figsize=(8,6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('The Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()

optimal_clusters = 5
kmeans = KMeans(n_clusters=optimal_clusters, init='k-means++', max_iter=300, n_init=10, random_state=42)
y_kmeans = kmeans.fit_predict(X)

plt.figure(figsize=(8,6))
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s=100, c='red', label='Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s=100, c='blue', label='Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s=100, c='green', label='Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s=100, c='cyan', label='Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s=100, c='magenta', label='Cluster 5')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='yellow', label='Centroids')
plt.title('Customer Segments based on Annual Income and Spending Score')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.grid(True)
plt.show()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive",
force_remount=True).
       CustomerID  Gender  Age  Annual Income (k$)  Spending Score
   (1-100)0    1    Male   19                 15                    39
   1           2    Male   21                 15                    81
   2           3  Female   20                 16                     6
   3           4  Female   23                 16                    77
   4           5  Female   31                 17                    40
 ['CustomerID', 'Gender', 'Age', 'Annual Income (k$)', 'Spending Score (1-100)']
```


The Elbow Method
250000
200000
150000
100000
50000
WCSS
2
4
6
8
10
Number of Clusters


100
80
60
40
20
0
20
40
60
80
100
120
140


Customer Segments based on Annual Income and Spending Score
80
70
60
50
40
30
20
Spending Score (1-100)
Centroids
30
40
50
60
70
80
90
Annual Income (k$)

Program 10. an example of a simple Python program that uses both NumPy and Pandas to generate a dataset, create a DataFrame, perform basic operations like adding a new column, sorting, filtering, and calculating statistics?
24-Sept-2024

```
import numpy as np
import pandas as pd
# Step 1: Create random data using NumPy
np.random.seed(42) # For reproducibility
data = np.random.randint(10, 100, size=(10, 3)) # 10 rows and 3 columns
# Step 2: Create a DataFrame using Pandas
df = pd.DataFrame(data, columns=['Math', 'Science', 'English'])
# Display the initial DataFrame
print("Initial DataFrame:")
print(df)
# Step 3: Add a new column for the average score of each student
df['Average'] = df.mean(axis=1)
# Display the DataFrame with the new column
print("\nDataFrame with Average scores:")
print(df)
# Step 4: Sort the DataFrame based on the Average column
df_sorted = df.sort_values(by='Average', ascending=False)
print("\nDataFrame sorted by Average scores (descending order):")
print(df_sorted)
# Step 5: Filter rows where Average score is greater than 50
df_filtered = df[df['Average'] > 50]
print("\nFiltered DataFrame with Average score > 50:")
print(df_filtered)
# Step 6: Calculate basic statistics for the 'Math' column
math_mean = df['Math'].mean()
math_std = df['Math'].std()
print(f"\nStatistics for Math scores:\nMean: {math_mean:.2f}, Standard Deviation: {math_std:.2f}")
```

```
 Initial DataFrame:
Math Science English
0       61      24      81
1       70      30      92
2       96      84      84
3       97      33      12
4       31      62      11
5       97      39      47
6       11      73      69
7       30      42      85
8       67      31      98
9       58      68      51

DataFrame with Average scores:
Math Science English            Average
0       61      24      81 55.333333
1       70      30      92 64.000000
2       96      84      84 88.000000
3       97      33      12 47.333333
4       31      62      11 34.666667
5       97      39      47 61.000000
6       11      73      69 51.000000
7       30      42      85 52.333333
8       67      31      98 65.333333
9       58      68      51 59.000000

DataFrame sorted by Average scores (descending order):
       Math  Science English Average
     2 96    84      84      88.000000
     8 67    31      98      65.333333
     1 70    30      92      64.000000
     5 97    39      47      61.000000
     9 58    68      51      59.000000
     0 61    24      81      55.333333
     7 30    42      85      52.333333
     6 11    73      69      51.000000
     3 97    33      12      47.333333
     4 31    62      11      34.666667
```

```
Filtered DataFrame with Average score > 50:
   Math  Science  English  Average
0  61    24       81       55.333333
1  70    30       92       64.000000
2  96    84       84       88.000000
5  97    39       47       61.000000
6  11    73       69       51.000000
7  30    42       85       52.333333
8  67    31       98       65.333333
9  58    68       51       59.000000

Statistics for Math scores:
Mean: 61.80, Standard Deviation: 30.36
```

Program 11. Write a Python program using NLTK to tokenize a sentence, filter out stopwords, and apply stemming?

12-Nov-2024

```
import nltk
from nltk.tokenize
import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
# Step 1: Download necessary NLTK data (run only once)
nltk.download('punkt')
nltk.download('stopwords')
# Step 2: Define a simple text
text = "Natural Language Processing with Python and NLTK is fun and interesting."
# Step 3: Tokenize the text into words
tokens = word_tokenize(text)
print("Tokenized words:", tokens)
# Step 4: Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
print("\nFiltered words (stopwords removed):", filtered_tokens)
# Step 5: Perform stemming using PorterStemmer ps = PorterStemmer()
stemmed_tokens = [ps.stem(word) for word in filtered_tokens]
print("\nStemmed words:", stemmed_tokens)
```

```
Tokenized words: ['Natural', 'Language', 'Processing', 'with', 'Python', 'and', 'NLTK', 'is', 'fun', 'and',

'interesting', '.']Filtered words (stopwords removed): ['Natural', 'Language', 'Processing', 'Python', 'NLTK', 'fun',

'interesting', '.']

Stemmed words: ['natur', 'languag', 'process', 'python', 'nltk', 'fun',
'interest', '.'][nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
/root/nltk_data...[nltk_data]  Package stopwords is
already up-to-date!
```

Program 12 . Implement a simple 3-layer neural network from scratch using Python and numpy. The network should have 2 input neurons, 2 hidden neurons, and 1 output neuron, using the sigmoid activation for both hidden and output layers. Train it on the XOR dataset using backpropagation for 100 epochs, printing the loss every 10 epochs. After training, print the predicted outputs for the XOR inputs. 15-Oct-2024

3- Layer Neural Network

Code 01

```
import numpy as np

# Sigmoid activation function
def sigmoid(x):
	return 1 / (1 + np.exp(-x))

# Derivative of sigmoid for backpropagation
def sigmoid_derivative(x):
	return x * (1 - x)

# Neural Network class
class SimpleNN:

	def __init__(self, input_size, hidden_size, output_size):

		# Initialize weights randomly
		self.weights_input_hidden = np.random.rand(input_size, hidden_size)
		self.weights_hidden_output = np.random.rand(hidden_size, output_size)

		# Initialize biases
		self.bias_hidden = np.random.rand(hidden_size)
		self.bias_output = np.random.rand(output_size)

	# Forward pass
	def forward(self, inputs):
		# Input to hidden layer
		self.hidden_layer_activation = np.dot(inputs, self.weights_input_hidden) +
		self.bias_hidden self.hidden_layer_output = sigmoid(self.hidden_layer_activation)

		# Hidden to output layer
		self.output_layer_activation = np.dot(self.hidden_layer_output,
		self.weights_hidden_output) + self.bias_output
		self.output = sigmoid(self.output_layer_activation)
		return self.output

	# Backward pass (backpropagation)
	def backward(self, inputs, expected_output, learning_rate):
		# Calculate output error
		error_output = expected_output - self.output
		d_output = error_output * sigmoid_derivative(self.output)
		# Calculate hidden layer error
		error_hidden = d_output.dot(self.weights_hidden_output.T)
		d_hidden = error_hidden * sigmoid_derivative(self.hidden_layer_output)
		# Update weights and biases using gradient descent
		self.weights_hidden_output += self.hidden_layer_output.T.dot(d_output) * learning_rate
		self.bias_output += np.sum(d_output, axis=0) * learning_rate
		self.weights_input_hidden += inputs.T.dot(d_hidden) * learning_rate
		self.bias_hidden += np.sum(d_hidden, axis=0) * learning_rate

	# Train the model
	def train(self, inputs, expected_output, epochs, learning_rate):
		for epoch in range(epochs):
```

```
            # Forward pass
            self.forward(inputs)
            # Backward pass and weights update
            self.backward(inputs, expected_output, learning_rate) # Print loss every 100
            epochs
            if epoch % 100 == 0:
                loss = np.mean(np.square(expected_output - self.output))
                print(f"Epoch {epoch} - Loss: {loss}")

# XOR Dataset (example binary classification task)
inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # XOR inputs
expected_output = np.array([[0], [1], [1], [0]]) # XOR outputs

# Hyperparameters
input_size = 2 # Number of features in the input layer
hidden_size = 2 # Number of neurons in the hidden layer
output_size = 1 # Binary classification (1 neuron in the output layer)
epochs = 10000 # Number of training iterations
learning_rate = 0.1 # Learning rate for gradient descent

# Initialize the neural network

nn = SimpleNN(input_size, hidden_size, output_size)

# Train the neural network
nn.train(inputs, expected_output, epochs, learning_rate)

# Test the neural network
print("\nFinal Output after training:")
for i in range(len(inputs)):
    print(f"Input: {inputs[i]} => Predicted Output: {nn.forward(inputs[i])}")
```

```
Epoch 4800 - Loss: 0.2389717293167135
Epoch 4900 - Loss: 0.23650996180684464
Epoch 5000 - Loss: 0.23356503744226392
Epoch 5100 - Loss: 0.23009249514957203
Epoch 5200 - Loss: 0.2260781973462792
Epoch 5300 - Loss: 0.22155499126019743
Epoch 5400 - Loss: 0.21661067519423138
Epoch 5500 - Loss: 0.2113785955489492
Epoch 5600 - Loss: 0.2060102728930146
Epoch 5700 - Loss: 0.20064011835383688
Epoch 5800 - Loss: 0.19535597447337405
Epoch 5900 - Loss: 0.19018331330130778
Epoch 6000 - Loss: 0.1850820092745647
Epoch 6100 - Loss: 0.17994962284243815
Epoch 6200 - Loss: 0.17462560905954277
Epoch 6300 - Loss: 0.16889477834812883
Epoch 6400 - Loss: 0.16249369706971994
Epoch 6500 - Loss: 0.15512935714774995
Epoch 6600 - Loss: 0.14652318785381507
Epoch 6700 - Loss: 0.13649014855242383
Epoch 6800 - Loss: 0.12504485230934304
Epoch 6900 - Loss: 0.11249186228359834
Epoch 7000 - Loss: 0.09942605154513279
Epoch 7100 - Loss: 0.08659494845485854
Epoch 7200 - Loss: 0.07467784538617003
Epoch 7300 - Loss: 0.06411466613377548
Epoch 7400 - Loss: 0.055067453776417585
Epoch 7500 - Loss: 0.047486742293285034
Epoch 7600 - Loss: 0.04120794900431174
Epoch 7700 - Loss: 0.03602778936440759
Epoch 7800 - Loss: 0.031748569867997664
Epoch 7900 - Loss: 0.028197555053161016
Epoch 8000 - Loss: 0.025231903609445383
Epoch 8100 - Loss: 0.022736850694123265
Epoch 8200 - Loss: 0.02062147710636897
Epoch 8300 - Loss: 0.01881413809769125
Epoch 8400 - Loss: 0.01725838076982395
Epoch 8500 - Loss: 0.015909583012417905
Epoch 8600 - Loss: 0.014732295735133884
Epoch 8700 - Loss: 0.013698183717287711
Epoch 8800 - Loss: 0.012784445427901359
Epoch 8900 - Loss: 0.011972604280204108
Epoch 9000 - Loss: 0.011247583392884699
```

```
Epoch 9100 - Loss: 0.010596995188292924
Epoch 9200 - Loss: 0.01001059345668271
Epoch 9300 - Loss: 0.009479848422769856
Epoch 9400 - Loss: 0.008997615231420075
Epoch 9500 - Loss: 0.008557873699201503
Epoch 9600 - Loss: 0.008155522716156187
Epoch 9700 - Loss: 0.0077862167947832335
Epoch 9800 - Loss: 0.007446235316958131
Epoch 9900 - Loss: 0.007132377301425352

Final Output after training:
Input: [0 0] => Predicted Output: [0.08340633]
Input: [0 1] => Predicted Output: [0.92038212]
Input: [1 0] => Predicted Output: [0.92036973]
Input: [1 1] => Predicted Output: [0.08792555]
```

Program 13 . Write a Python program using Keras to build a deep learning model with 4 layers for binary classification. The model should have 2 hidden layers with 32 and 16 neurons respectively, each using the ReLU activation function. The output layer should use a sigmoid activation function. Use the XOR dataset as input, and train the model for 50 epochs. After training, print the final accuracy and the predicted output for each input. (1 point)

15-Oct-2024

4- Layer Deep Learning

Code 02

```
import numpy as np
from tensorflow.keras.models import
Sequentialfrom
tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Create a simple dataset (e.g., binary
classification)# Example: XOR dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # Inputs
y = np.array([[0], [1], [1], [0]]) # Outputs

# Define the model
model = Sequential()

# Input layer (2 input neurons, corresponding to 2
features in X)# Hidden layer 1: 64 neurons, activation:
ReLU
model.add(Dense(64, input_dim=2, activation='relu'))

# Hidden layer 2: 32 neurons,
activation: ReLUmodel.add(Dense(32,
activation='relu'))

# Hidden layer 3: 16 neurons,
activation: ReLUmodel.add(Dense(16,
activation='relu'))

# Hidden layer 4: 8 neurons,
activation: ReLUmodel.add(Dense(8,
activation='relu'))

# Output layer: 1 neuron (binary
classification)model.add(Dense(1,
activation='sigmoid'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X, y, epochs=1000, verbose=0)

# Evaluate the model performance
loss, accuracy = model.evaluate(X, y,
verbose=0)print(f'Final Accuracy:
{accuracy * 100:.2f}%')

# Make predictions
predictions =
model.predict(X)
print("\nPredictions:
")
for i in range(len(X)):
  print(f"Input: {X[i]}, Predicted Output: {predictions[i][0]:.4f}")
```

```
Final Accuracy: 100.00%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step

Predictions:
Input: [0 0], Predicted Output: 0.0000
Input: [0 1], Predicted Output: 1.0000
Input: [1 0], Predicted Output: 1.0000
Input: [1 1], Predicted Output: 0.0000
```

Program 14. Setup single node Hadoop cluster and Apply HDFS Commands on Single Node Hadoop Cluster

22-Oct-2024

Setup single node Hadoop cluster and Apply HDFS Commands on Single Node Hadoop Cluster.

- **Prequisites**

1. Java 8 runtime environment (JRE)
2. Apache Hadoop 3.3.6

Download Hadoop binary package

The first step is to download Hadoop binaries from the official website(https://hadoop.apache.org/releases.html). The binary packagesize is about 696 MB.

Unpack the package

After finishing the file download, we should unpack the package using 7zip or WinRar.

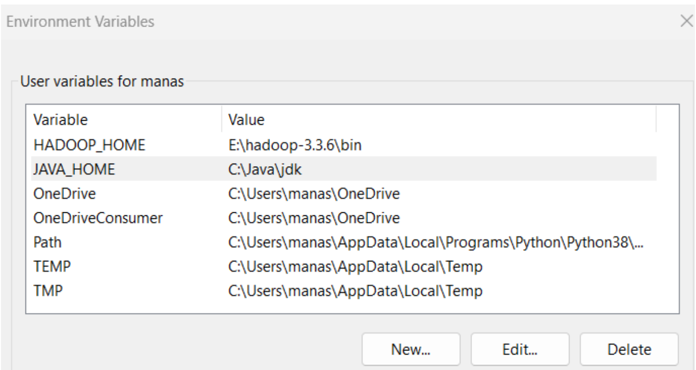
Java installation

Java is required to run Hadoop. If java is not installed , We have to install it. After finishing the file download we open a new command prompt, we should unpack the package.

Configure environment variables

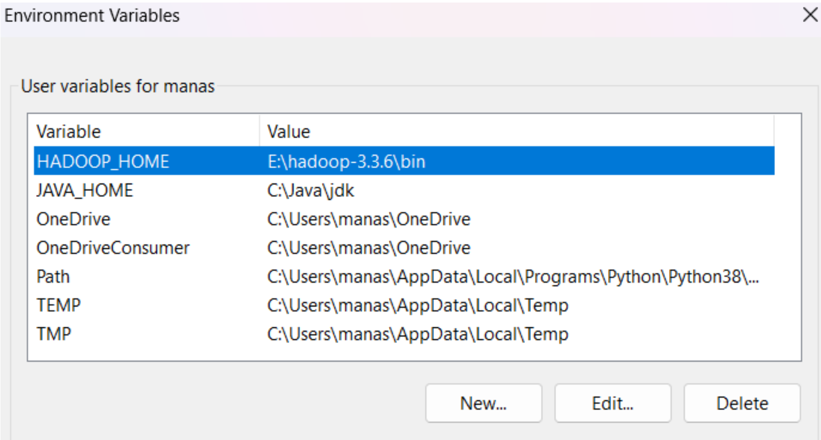
Now we've downloaded and unpacked all the files we need to configure two importantenvironment variables.

We configure **JAVA_HOME** environment variable by adding new environment variable.Variable name : JAVA_HOME Variable value: Java path



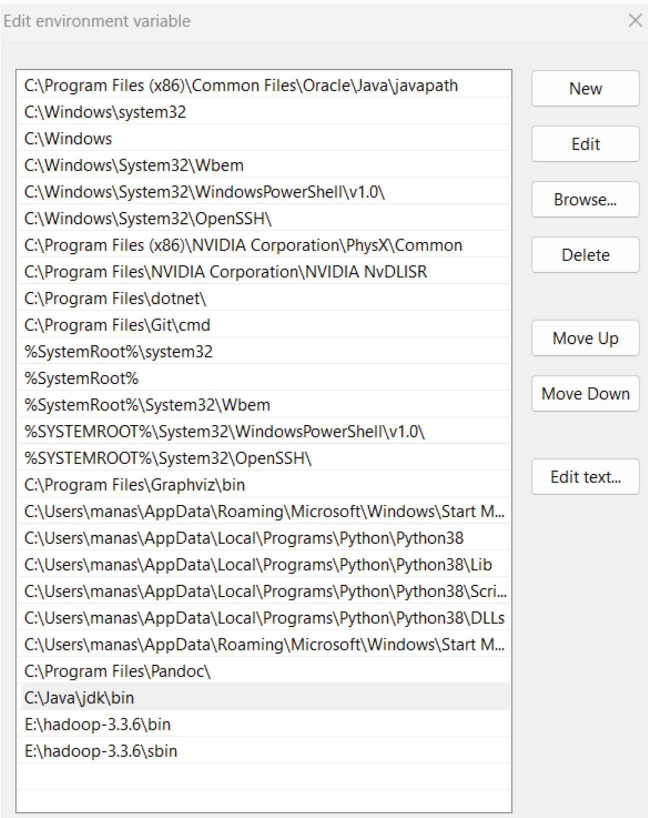
The same with **HADOOP_HOME** environment variable-

Variable name : HADOOP_HOME Variable value: C:\Hadoop\hadoop-3.3.6



- **Configure PATH environment variable**

Once we finish setting up the above two environment variables, we need to add the bin folders to the PATH environment variable.



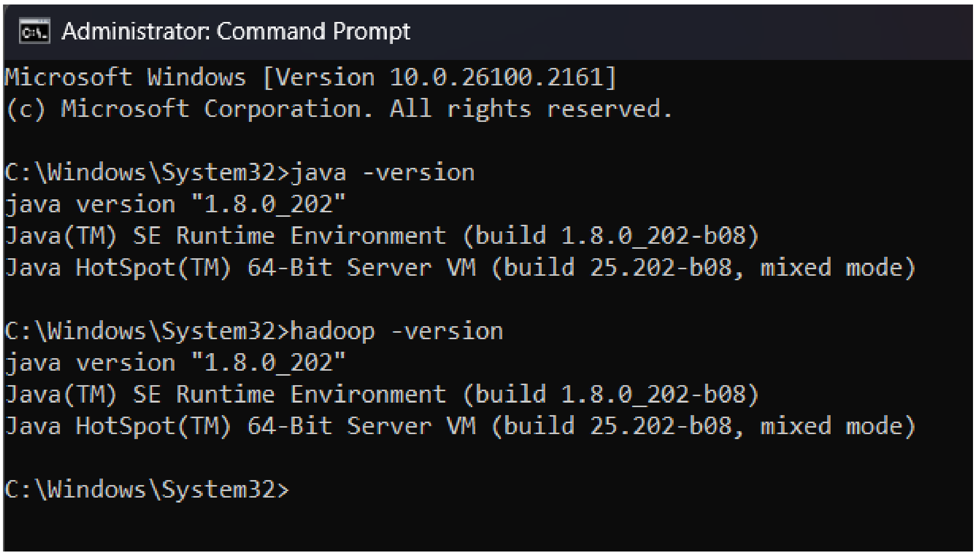
Verification of Installation

Once We complete the installation, Close terminal window and open a new one and run the following command to verify:

java -version

We can also be able to run the following command:

hadoop –version

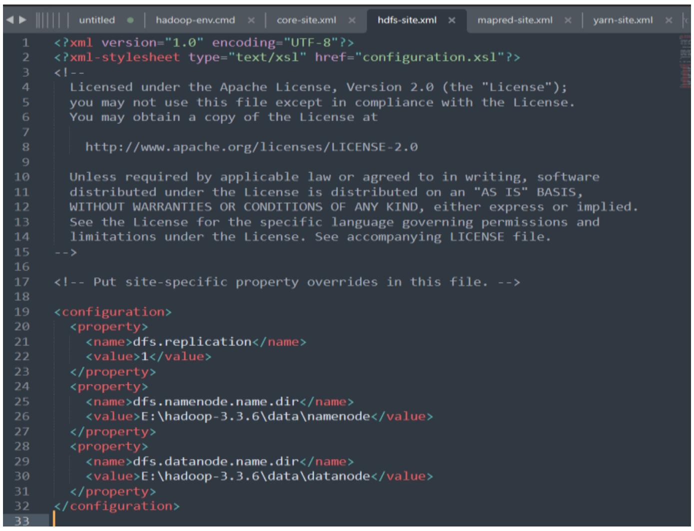


```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.26100.2161]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>java -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.202-b08, mixed mode)

C:\Windows\System32>hadoop -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.202-b08, mixed mode)

C:\Windows\System32>
```

Configure Hadoop

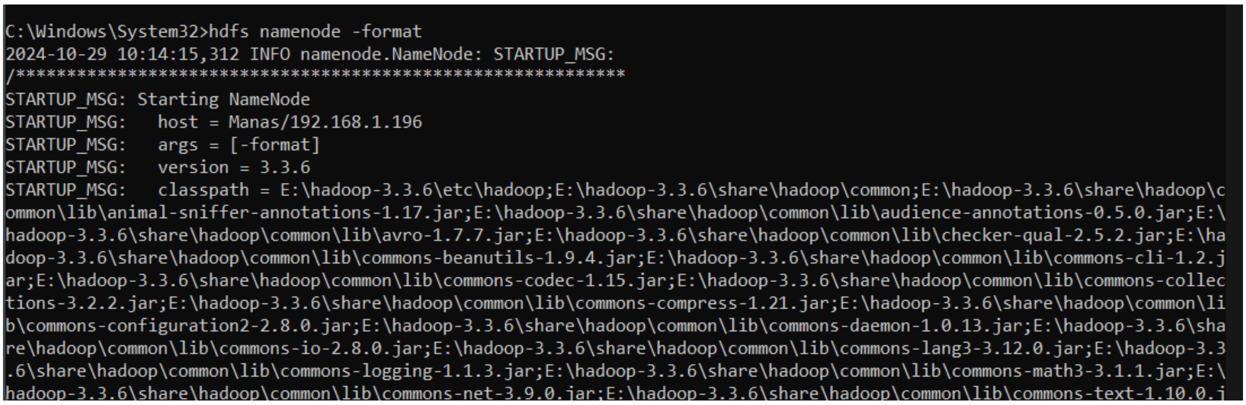
Now we are ready to configure the most important part - Hadoop configurations which involves Core, YARN, MapReduce, HDFS configuration

Initialize HDFS



```
untitled | hadoop-env.cmd | core-site.xml | hdfs-site.xml | mapred-site.xml | yarn-site.xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>E:\hadoop-3.3.6\data\namenode</value>
  </property>
  <property>
    <name>dfs.datanode.name.dir</name>
    <value>E:\hadoop-3.3.6\data\datanode</value>
  </property>
</configuration>
```

Run the following command in Command Prompt hdfs namenode –format

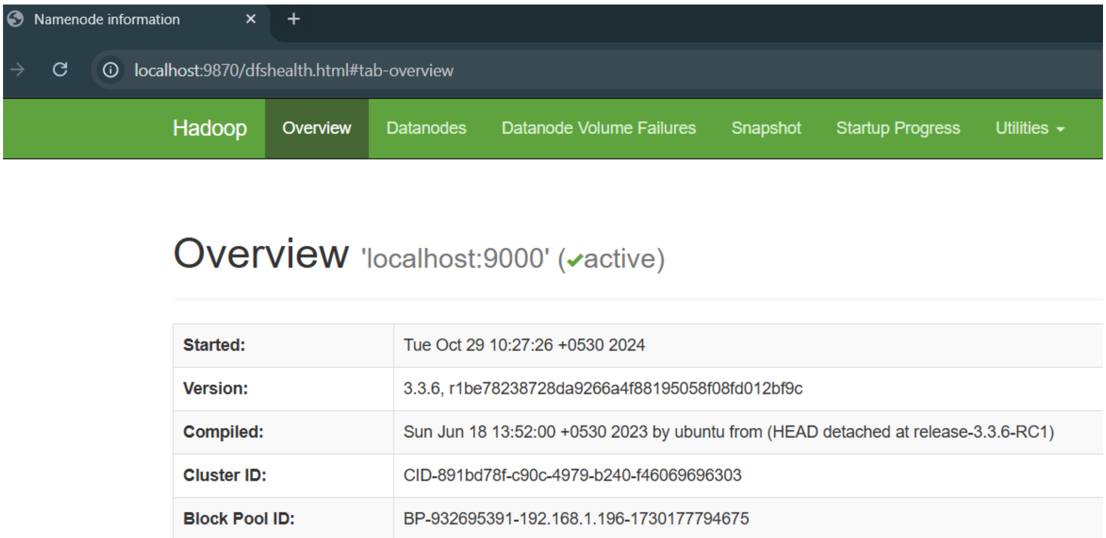


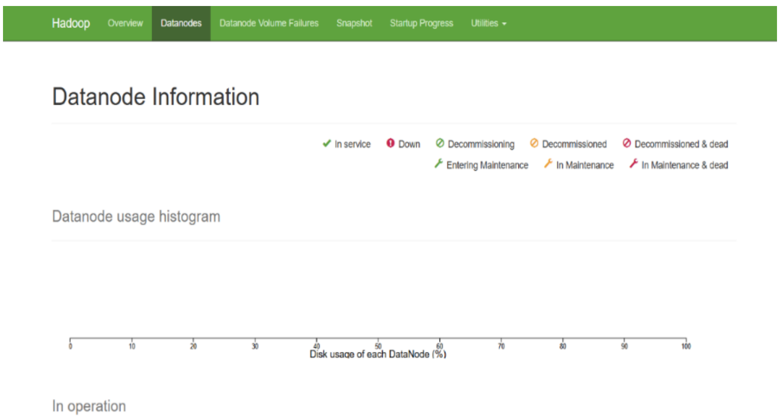
Start HDFS daemons

Run the following command to start HDFS daemons in Command Prompt:

start-all.cmd

Verify HDFS web portal UI through this link http://localhost:9870/dfshealth.html#tab-overview.





Start YARN daemons

Run the following command in an elevated Command Prompt window (Run as administrator) to start YARN daemons:

%HADOOP_HOME%\sbin\start-yarn.cmd
You can verify YARN resource manager UI when all services are started successfully.

http://localhost:8088

```
C:\Windows\System32>JPS
22164 Jps
22884 ResourceManager
24020 NodeManager
23816 NameNode

C:\Windows\System32>
```

Program 15. Write a Python program to implement a Naive Bayes classifier for text classification using a dataset with text and label columns. Use CountVectorizer to preprocess the text data, train the model on a training set, and evaluate it on a test set with accuracy and F1-score metrics.

12-Nov-2024

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, f1_score
# Load the dataset (ensure it has 'text' and 'label' columns)
# You can replace this with the path to your dataset file
df = pd.read_csv('/content/drive/MyDrive/df_file.csv') # Ensure 'text' and 'label' columns
# Split the dataset into features (X) and labels (y)
X = df['text'] # Assuming the text data is in the 'text' column
y = df['label']
# Assuming the labels are in the 'label' column
# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize CountVectorizer for text preprocessing
vectorizer = CountVectorizer(stop_words='english')
# Fit the vectorizer on the training data and transform both training and testing data X_train_vec =
vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
# Initialize and train the Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_vec, y_train)
# Predict on the test set
y_pred = nb_classifier.predict(X_test_vec)
# Evaluate the model using accuracy and F1-score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')
# Use 'weighted' for multi-class classification
# Print the results
print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"F1-Score (Weighted): {f1:.2f}")
```

```
Accuracy: 97.30%
F1-Score (Weighted): 0.97
```

Program 16. Write a Python program to calculate the Information Gain (IG) for a dataset and determine the root node for building a decision tree. The dataset contains attributes and a target column with binary values (e.g., "yes" or"no"). SEE ATTACHMENT to this Assignment.

19-Nov-2024

Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

```
import numpy as np
import pandas as pd
ANAND = pd.read_csv('/content/drive/MyDrive/golf-dataset.csv')


ANAND.columns

Index(['Outlook', 'Temp', 'Humidity', 'Windy', 'Play Golf'], dtype='object')


def entropy(target_col):
    elements, counts = np.unique(target_col, return_counts=True)
    entropy = np.sum([(-counts[i]/np.sum(counts)) * np.log2(counts[i]/np.sum(counts)) for i
    in range(len(elements))])return entropy

def information_gain(data, split_attribute_name,
    target_name="class"):# Calculate the entropy of the
    total dataset
    total_entropy = entropy(data[target_name])

    # Calculate the values and the corresponding counts for
    the split attributevals, counts =
    np.unique(data[split_attribute_name], return_counts=True)

    # Calculate the weighted entropy
    weighted_entropy = np.sum([(counts[i]/np.sum(counts)) * entropy(data.where(data[split_attribute_name] ==
    vals[i]).dropna()[target_nam

    # Calculate the information gain
    information_gain = total_entropy -
    weighted_entropyreturn information_gain
```

```
print("Information Gain for 'Outlook':", information_gain(ANAND, 'Outlook', 'Play Golf'))
```

```
Information Gain for 'Outlook': 0.24674981977443933
```

```
print("Information Gain for 'Temp':", information_gain(ANAND, 'Temp', 'Play Golf'))
```

```
Information Gain for 'Temp': 0.02922256565895487
```

```
print("Information Gain for 'Humidity':", information_gain(ANAND, 'Humidity', 'Play Golf'))
```

```
Information Gain for 'Humidity': 0.15183550136234159
```

```
print("Information Gain for 'Humidity':", information_gain(ANAND,
'Windy', 'Play Golf')) Information Gain for 'Humidity':
0.04812703040826949
```