Lab 1. Introduction to Artificial Intelligence

This lab sheet aims to help us understand more the basics of Google Colab, as well as its python console.

Task 1.1 The “Hello World” Program

Problem Description

For this task, we created a new file in Google Colab and learned about its mechanics.

So first, we are going to take a look at a simple “Hello World” program, which simply outputs the “Hello World” string, as well as the current time, in the output. We will also generate a plot of waveform, so we can take a look at some graphical output.

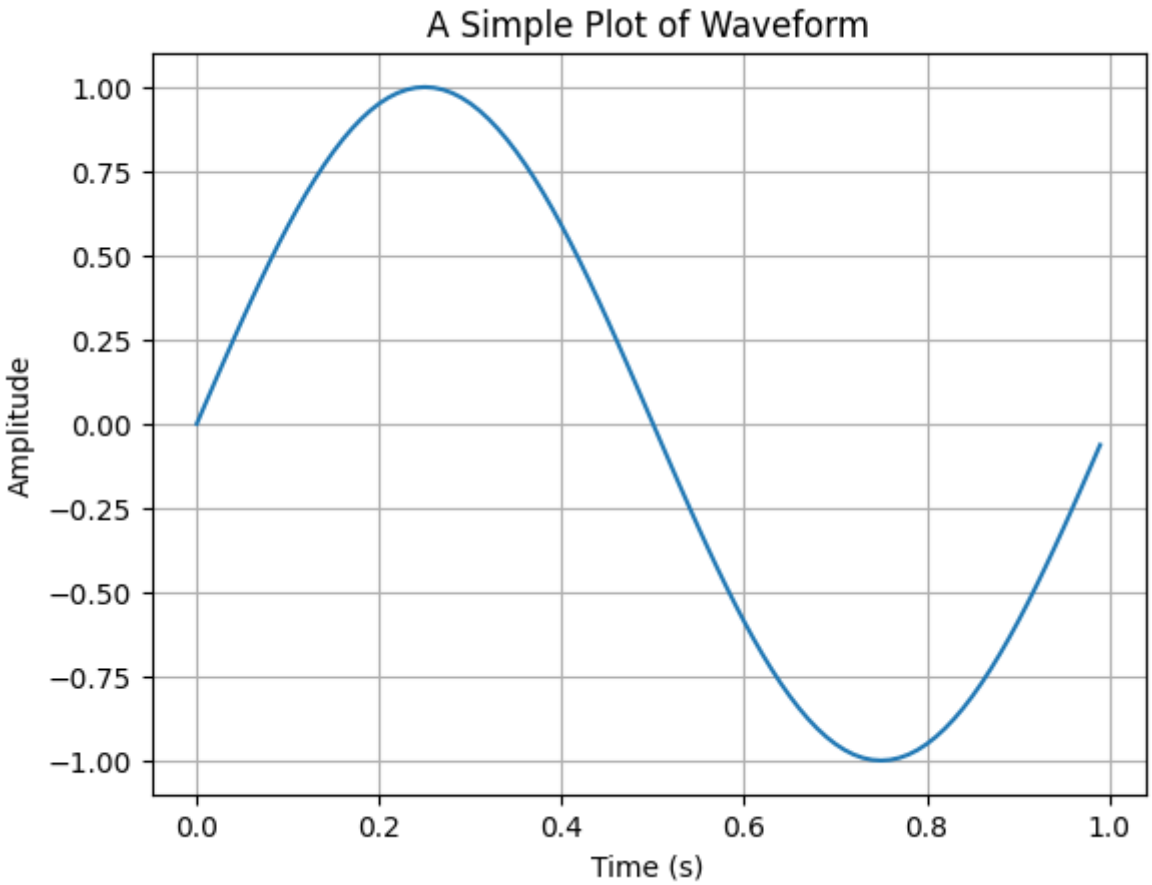
Implementation and results

This section shows the implemented code and its respective output.

In [ ]:
```
import time
print("Hello World!")
print("The current time is", time.ctime())
```

```
Hello World!
The current time is Thu Nov  9 23:31:15 2023
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
t = np.arange(0.0, 1.0, 0.01)
s = np.sin(2 * np.pi * t)
plt.plot(t,s)
plt.title('A Simple Plot of Waveform')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid()
```



Discussions

For this problem, we have analysed two simple programs, in which, as we can see by the resulting output, the first one printed the "Hello World" along with the current time, and the second one outputted a simple plot of waveform, with the adequate labels and graph title.

Conclusion

In conclusion, after completing this Lab work we learned about the fundamentals of Google Colab, along with some of its main features.

Lab 2. Search Algorithms

This lab sheet aims to help us understand that there are many different search algorithms and some of them may get us more optimal results than others, depending on the objectives of the problem. So in this lab report, we will analyse three different tasks:

1. Route planning problem
2. 8-puzzle problem
3. Maze solver

Each one of these presents a real-world problem, in which the appropriate solution will be met by resorting to some of these algorithms. The introduced searching algorithms during this report are the following:

1. A*
2. Breadth-first
3. Depth-first
4. Uniform-cost
5. Greedy

Task 2.1 Route Planning

Problem Description

The following task presents the problem visited in the previous lecture, as we got 6 distinctive cities, all numbered from 1 to 6, which can be seen in Figure 1. when directly connected, the distances between these cities are displayed on the graph edges.

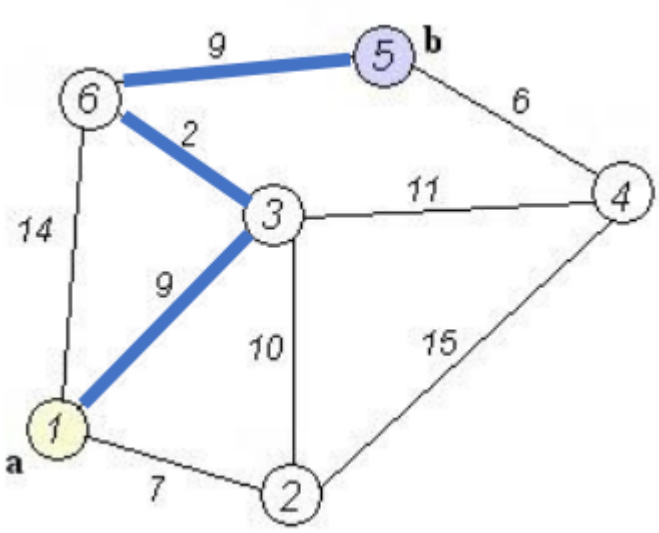


Figure 1

The objective is to write a code that can find an optimal route, which means (in the presented case) the shortest one, between a certain pair of cities. As an example, if we decide to go from

city 1 to 5, we should be able to see in the output the shortest route chosen and its respective total cost, which would be 20 in this particular test case.

In order to explain what a search problem really is, four topics were analysed during the lecture:

1. State: For each city, it was assigned a respective number from 1 to 6.
2. Action: The new (or current) state indicates the city we are in at the moment and the next state is the one we are about to travel to, in other words, our destination. For example, if we are in state 1 and the next one is number 3, we can say that 3 is the action, which means we travelled from 1 to 3.
3. Goal test: We now check if our current state matches the final destination expected. For example, if our current state is 5 and the final destination is also 5.
4. Path cost: This information is displayed in the graph shown above (Figure 1). We turn this into a 6x6 array, where a positive finite distance value takes place between any directly connected city pair, 0 for the current city and "inf" for unconnected pairs of cities.

Implementation and results

This section shows the implemented code and its respective output.

In [1]:

```
!pip install simpleai
from simpleai.search import SearchProblem, astar, greedy, breadth_first, depth_first
import math, time
```

```
Collecting simpleai
  Downloading simpleai-0.8.3.tar.gz (94 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 94.4/94.4 kB 2.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: simpleai
  Building wheel for simpleai (setup.py) ... done
  Created wheel for simpleai: filename=simpleai-0.8.3-py3-none-any.whl size=100983 sh
a256=90d097ae0abbdc65ec129e3b553562bd03c9914057f5daf04e57ae8ecc6028dc
  Stored in directory: /root/.cache/pip/wheels/91/0c/38/421d7910e7bc59b97fc54f490808b
db1097607d83d1a592865
Successfully built simpleai
Installing collected packages: simpleai
Successfully installed simpleai-0.8.3
/bin/bash: -c: line 1: syntax error near unexpected token `('
/bin/bash: -c: line 1: `[Figure 1](https://drive.google.com/file/d/1gcXSMc4hCQdJlnR-H
bDGjx9xA4fH-WyR/view?usp=sharing)'
```

In [ ]:

```
COSTS = [
    [0, 7, 9, 'inf', 'inf', 14],
    [7, 0, 10, 15, 'inf', 'inf'],
    [9, 10, 0, 11, 'inf', 2],
    ['inf', 15, 11, 0, 6, 'inf'],
    ['inf', 'inf', 'inf', 6, 0, 9],
    [14, 'inf', 2, 'inf', 9, 0]
]

class Route(SearchProblem):

    # initialise with the initial and goal states. Do "-1" because the nodes start f
    def __init__(self, initial, goal):
        self.initial = initial-1
        self.goal = goal-1
```

```
        super(Route, self).__init__(initial_state=self.initial)

    # add all connected nodes, i.e. those with a cost of not 0 or inf
    def actions(self, state):
        actions = []
        for action in range(len(COSTS[state])):
            if COSTS[state][action] not in ['inf', 0]:
                actions.append(action)
        return actions

    # the result state is just the action as defined
    def result(self, state, action):
        return action

    # check if goal is reached
    def is_goal(self, state):
        return state == self.goal

    # return the cost between two states
    def cost(self, state, action, state2):
        return COSTS[state][action]

if __name__ == "__main__":
  problem = Route(1, 5)
  init_time = time.time()

  #result_bf = breadth_first(problem, graph_search=True)
  #result_df = depth_first(problem, graph_search=True)
  result_uc = uniform_cost(problem)

  print("--- %s seconds ---" % (time.time() - init_time))

  # Print the results
  #path_bf = [x[1]+1 for x in result_bf.path()]
  #path_df = [x[1]+1 for x in result_df.path()]
  path_uc = [x[1]+1 for x in result_uc.path()]
  #print("Breath-first: the route is %s, and total cost is %s" %(path_bf, result_bf.
  #print("Depth-first: the route is %s, and total cost is %s" %(path_df, result_df.c
  print("Uniform-cost: the route is %s, and total cost is %s" %(path_uc, result_uc.c
```

```
--- 0.0009272098541259766 seconds ---
Uniform-cost: the route is [1, 3, 6, 5], and total cost is 20
```

Discussions

To solve this problem, 3 different search algorithms were used, such as:

1. Uniform-cost
2. Breadth-first
3. Depth-first

The Python package simpleai was used (https://pypi.org/project/simpleai/) for the python code implementation and its online documentation can be found at https://simpleai.readthedocs.io/en/latest/.

This table presents us the results outputted by all the algorithms mentioned above:

Algorithm	Path cost	Runtime (seconds)
Uniform-cost	20	0.00092

Algorithm	Path cost	Runtime (seconds)
Breadth-first	23	0.00008
Depth-first	23	0.00004

table 1

By analysing the results we obtained in the previous section when using these algorithms, we can see that both the breath-first and depth-first search calculated the same optimal path for the same test input, with approximately an equal runtime, as well as the same total cost. On the other hand, the uniform-cost algorithm calculated a different optimal route, by opting for one more city-state for its path, while displaying a higher runtime, nevertheless, it was able to get a lower total cost. Therefore, we can easily approve that the uniform-cost algorithm provided a better solution to this problem, probably due to the fact that this algorithm prioritizes state selection based on cost values (edge weights) in order to find the shortest path, which the other algorithms don't.

Task 2.2 8-Puzzle Problem

Problem Description

This next task consists in solving the 8-puzzle problem, which is better described in "Russell and Norvig 2021". So the presented problem, illustrated below (Figure 2), presents a 3x3 panel composed of 8 tiles, each one numbered from 1 to 8 and an empty one, where the objective is, given a initial panel configuration, to find a way to turn that initial state into the final one, by placing all the numbered tiles in its correct position and order. It is important to highlight that in order to make moves, we must only use the available empty space so that one of its adjacent tiles can occupy it, by making moves such as, below, above, left or right. This should be accomplished by making the minimum possible number of moves.

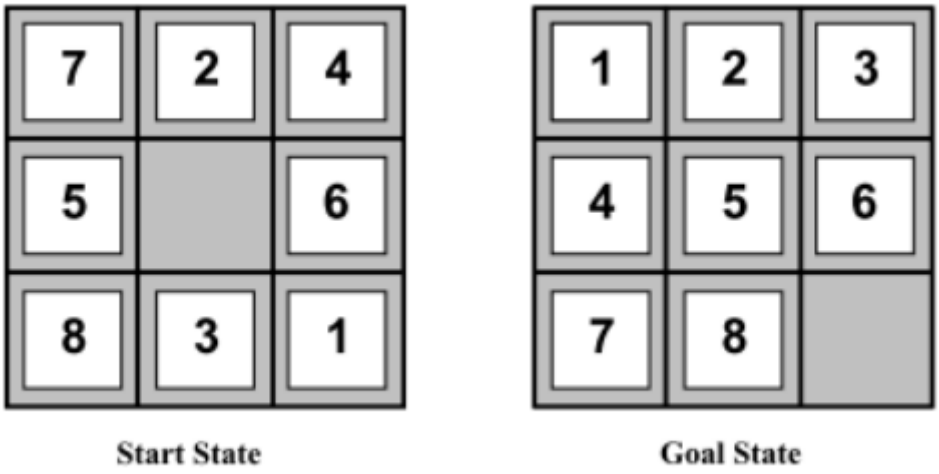


Figure 2

Let's now deconstruct the search problem by analysing its structure:

1. State: By using a 3x3 array composed of numbers from 1 to 8 and the letter "E" for the identification of the blank space, we could represent the initial state of the puzzle and also

follow its continuous transformation until the final state.

2. Action: So that we can implement more easily this solution, we could move only the blank tile down, up, to the left or right, instead of moving all the numbered tiles. In order to achieve this, we would also need to formulate conditions to validate which one of the available moves would be accepted for a certain state.
3. Goal test: We need to check if the current state matches the goal state, that is, whether we have actually obtained the correct final state. A suitable example can be seen in Figure 2.
4. Path cost: As 1 move corresponds to 1 cost unit, the total path cost can be measured in the number of moves made throughout the solution of the problem.

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:
```
!pip install simpleai
from simpleai.search import SearchProblem, astar, greedy, breadth_first, depth_first
import time
```

```
Requirement already satisfied: simpleai in /usr/local/lib/python3.10/dist-packages
(0.8.3)
```

In [ ]:
```
# Class containing methods to solve the puzzle
class PuzzleSolver(SearchProblem):
      # Action method to get the list of the possible numbers that can be moved in t
      def actions(self, cur_state):
          rows = string_to_list(cur_state)
          row_empty, col_empty = get_location(rows, 'E')

          actions = []
          if row_empty > 0: actions.append(rows[row_empty - 1][col_empty])
          if row_empty < 2: actions.append(rows[row_empty + 1][col_empty])
          if col_empty > 0: actions.append(rows[row_empty][col_empty - 1])
          if col_empty < 2: actions.append(rows[row_empty][col_empty + 1])
          return actions

      # Return the resulting state after moving a piece to the empty space
      def result(self, state, action):
          rows = string_to_list(state)
          row_empty, col_empty = get_location(rows, 'E')
          row_new, col_new = get_location(rows, action)
          rows[row_empty][col_empty], rows[row_new][col_new] = rows[row_new][col_new
          return list_to_string(rows)

      # Returns true if a state is the goal state
      def is_goal(self, state):
          return state == GOAL

      '''# # Returns an estimate of the distance from a state to the goal using the
      def heuristic(self, state):
          rows = string_to_list(state)
          distance = 0
          for number in '12345678E':
              row_new, col_new = get_location(rows, number)
              row_new_goal, col_new_goal = goal_positions[number]
              distance += abs(row_new - row_new_goal) + abs(col_new - col_new_goal)
          return distance '''

      # Returns the number of misplaced tiles -> H1
```

```
    def heuristic(self, state):
        rows = string_to_list(state)
        goal = string_to_list(GOAL)
        distance = sum([rows[i] != goal[i] for i in range(len(rows))]) - 1
        return distance
```

In [ ]:

```
# Convert list to string
def list_to_string(input_list):
    return '\n'.join(['-'.join(x) for x in input_list])

# Convert string to list
def string_to_list(input_string):
    return [x.split('-') for x in input_string.split('\n')]

# Find the 2D location of the input element
def get_location(rows, input_element):
    for i, row in enumerate(rows):
        for j, item in enumerate(row):
            if item == input_element:
                return i, j

# Final result that we want to achieve
GOAL = '''\
E-1-2
3-4-5
6-7-8'''

# Starting point - solution depth = 26
#INITIAL = '''\
#7-2-4
#5-E-6
#8-3-1'''

# Starting point - solution depth = 8
#INITIAL = '''\
#1-4-2
#5-E-8
#3-6-7'''

# Starting point - solution depth = 4
INITIAL = '''\
1-4-2
3-5-8
6-7-E'''


if __name__ == "__main__":

  # Create a cache for the goal position of each piece
  goal_positions = {}
  rows_goal = string_to_list(GOAL)
  for number in '12345678E':
      goal_positions[number] = get_location(rows_goal, number)

  init_time = time.time()

  # Create the A* solver object
  result_A = astar(PuzzleSolver(INITIAL))
  #result_G = greedy(PuzzleSolver(INITIAL))
  #result_BF = breadth_first(PuzzleSolver(INITIAL))
  #result_DF = depth_first(PuzzleSolver(INITIAL))

  print("--- %s seconds ---" % (time.time() - init_time))
```

```
# Print the results
for i, (action, state) in enumerate(result_A.path()):
    print()
    if action == None:
        print('Initial configuration')
    else:
        print('Step %s: After moving %s into the empty space' %(i, action))
    print(state)
print('Goal achieved!')
```

```
--- 0.0003066062927246094 seconds ---

Initial configuration
1-4-2
3-5-8
6-7-E

Step 1: After moving 8 into the empty space
1-4-2
3-5-E
6-7-8

Step 2: After moving 5 into the empty space
1-4-2
3-E-5
6-7-8

Step 3: After moving 4 into the empty space
1-E-2
3-4-5
6-7-8

Step 4: After moving 1 into the empty space
E-1-2
3-4-5
6-7-8
Goal achieved!
```

Discussions

To solve this problem, 4 different search algorithms were tested, such as:

1. A*
2. Greedy
3. Breadth-first
4. Depth-first

The code implemented in the previous section was adapted from an example package in the simpleai package.

For testing purposes, different test inputs were created. The first test case uses the 4 algorithms mentioned above, where the solution depth is equal to 4, this being a simple case. The second test is a Difficult case, with a solution depth of 26, that uses only the A* and Greedy search, although these algorithms were used together with a heuristic function (H1 and H2).

Simple case

In this case, the 4 search algorithms previously mentioned were used, all subjected to the same test input "1-4-2-3-5-8-6-7-E", this being the initial configuration. Let's assume that the H1 represents a heuristic function that returns the number of misplaced tiles and H2 a heuristic function that returns an estimate of the distance from a state to the goal using the manhattan distance.

The following table shows us the results presented by all the tested algorithms, when using H2:

Algorithm	Path cost	Runtime (seconds)
A*	4	0.00046
Greddy	4	0.00051
Breadth-first	4	0.00136
Depth-first	null	error

table 2

This next table displays the results, when using H1:

Algorithm	Path cost	Runtime (seconds)
A*	4	0.00042
Greddy	4	0.00038
Breadth-first	4	0.00629
Depth-first	null	error

table 3

Finally, this table shows us the data once again, but this time without using an additional heuristic function:

Algorithm	Path cost	Runtime (seconds)
A*	4	0.00606
Greddy	4	0.00055
Breadth-first	4	0.00139
Depth-first	null	error

table 4

After analysing all the tables above and studying their results, we can notice that the A*, Greedy and Breadth-first algorithms presented fairly similar results, with a solution that has a path cost of 4 and a very low runtime (which is also a key element for achieving an optimal solution to this problem). When using H2, the A* and Greedy algorithms showed a slightly better runtime than the others. At the same time, When using H1, we can also see that the A* and Greedy algorithms performed somewhat better. Furthermore, when using just the algorithms (no additional heuristic functions), the results were not so different, with the Greedy method performing slightly better than the rest (in terms of runtime).

On the other hand, the Depth-first algorithm didn't do as well as the others, as it wasn't able to run successfully, outputting a runtime error. Some of the reasons for this to happen might be

because of its exploration strategy, as it can possibly go too deep into one branch before starting to explore others, or the lack of other types of guidance.

Difficult case

For this case, only the A* and Greedy algorithms were used for solving the problem, being "7-2-4-5-E-6-8-3-1" the test input for both of them.

The following table shows us the results presented by both algorithms, when using H2:

Algorithm	Path cost	Runtime (seconds)
A*	26	19.517
Greddy	null	error

table 5

This next table displays the results, when using H1:

Algorithm	Path cost	Runtime (seconds)
A*	26	28.727
Greddy	null	error

table 6

By studying the previous tables, we can see that when using H2, the A* algorithm showed a better performance than when using H1, as the Runtime of its solution is almost 10 seconds faster, while still calculating the same path cost (of 26). However, the Greedy algorithm wasn't capable to run successfully, outputting a runtime error in both scenarios (with H1 and H2). Consequently, this leads us to accept that the A* outperforms the Greedy algorithm for this case scenario. This is probably due to the fact that the Greedy method has a lack of optimality, backtracking and its heuristic methods can lead to non-optimal outcomes when compared to A*.

Task 2.3 Maze Solver

Problem Description

This last task introduces a maze problem, where the solution is obtained by finding the optimal (shortest) path, from a starting point to a given destination. The initial position is labelled in the maze map with an "O" and the target position is labelled with an "X". At the same time, the maze is also composed of other symbols, such as the "#" to represent the walls', which means the player can't go through them and must only use the empty spaces to move. The image below (Figure 3) presents an example of a maze, as described previously.

```
MAP = """
##############################
#          #                 #   #
# ####     ########          #   #
#  o #     #                 #   #
#     ###      ####    ######    #
#          ####        #         #
#              #   #    #    #### #
#       ######     #        # x   #
#          #       #              #
##############################
"""
```

Figure 3

Let's now analyse the structure of the problem:

1. State: The player's current location on the map, which uses x and y coordinates, represents a new state.
2. Action: We can only move down, up, left or right to another adjacent blank space, in other words, we can't move to the wall ("#") position. Even though we have defined the actions in order to extend the search, we'll add four diagonal moves, up right, down right, down left and up left, which grants us four more actions.
3. Goal test: We must check if the current state position matches the target state (the coordinates are equal).
4. Path cost: Any of the basic moves (up, down, left or right) mentioned correspond to 1 cost unit. However, diagonal moves (up right, down right, down left or up left) correspond, according to the Pythagorean theorem, to a square root of cost 2, which is approximately 1.4. That being said, the total path cost can be determined by calculating all the moves made and their cost, during the completion of the maze.

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:
```
!pip install simpleai
from simpleai.search import SearchProblem, astar, greedy, breadth_first, depth_first
import math, time
```

```
Requirement already satisfied: simpleai in /usr/local/lib/python3.10/dist-packages
(0.8.3)
```

In [ ]:
```
MAP = """
##############################
#          #                 #   #
# ####     ########          #   #
#  o #     #                 #   #
#     ###      ####    ######    #
#          ####        #         #
#              #   #    #    #### #
```

```
#     ######    #       # x  #
#        #      #            #
##############################
"""
MAP = [list(x) for x in MAP.split("\n") if x]

COSTS = {
    "up": 1.0,
    "down": 1.0,
    "left": 1.0,
    "right": 1.0,
    #"up left": 1.4,
    #"up right": 1.4,
    #"down left": 1.4,
    #"down right": 1.4,
}
```

In [ ]:

```
class Maze(SearchProblem):

    def __init__(self, board):
        self.board = board
        self.goal = (0, 0)
        for y in range(len(self.board)):
            for x in range(len(self.board[y])):
                if self.board[y][x].lower() == "o":
                    self.initial = (x, y)
                elif self.board[y][x].lower() == "x":
                    self.goal = (x, y)

        super(Maze, self).__init__(initial_state=self.initial)

    def actions(self, state):
        actions = []
        for action in list(COSTS.keys()):
            newx, newy = self.result(state, action)
            if self.board[newy][newx] != "#":
                actions.append(action)
        return actions

    def result(self, state, action):
        x, y = state

        if action.count("up"):
            y -= 1
        if action.count("down"):
            y += 1
        if action.count("left"):
            x -= 1
        if action.count("right"):
            x += 1

        new_state = (x, y)
        return new_state

    def is_goal(self, state):
        return state == self.goal

    def cost(self, state, action, state2):
        return COSTS[action]

    #def heuristic(self, state):
    #    x, y = state
```

```
    #    gx, gy = self.goal
    #    return math.sqrt((x - gx) ** 2 + (y - gy) ** 2)
```

In [ ]:

```
problem = Maze(MAP)
init_time = time.time()
#result_A = astar(problem, graph_search=True)
#result_G = greedy(problem, graph_search=True)
#result_BF = breadth_first(problem, graph_search=True)
result_DF = depth_first(problem, graph_search=True)
#result_UC = uniform_cost(problem, graph_search=True)

print("--- %s seconds ---" % (time.time() - init_time))

path = [x[1] for x in result_DF.path()]
for y in range(len(MAP)):
    for x in range(len(MAP[y])):
        print('.' if (x,y) in path[1:-1] else MAP[y][x], end='')
    print()
print("Search: moves=%s, cost=%s." %(result_DF.depth, result_DF.cost))
```

```
--- 0.0030846595764160156 seconds ---
##############################
#         #              #   #
# ####    ########       #   #
#  o.#    #              #   #
#   .###     ####   ######   #
#   ......#### .....#        #
#         ....# .#  .#   #### #
#     ######....#  .....# x  #
#        #      #      ....  #
##############################
Search: moves=33, cost=33.0.
```

Discussions

To find a solution to this problem, we used some different search algorithms:

1. A*
2. Greedy
3. Breadth-first
4. Depth-first
5. Uniform-cost

The code implemented in the previous section was adapted from an example package in the simpleai package.

First of all, let's assume that H1 is a heuristic function used to try to find a better solution to the search problem. So H1 calculates the pythagoras theorem, in familiar algebraic notation, a^2 + b^2 = c^2, which allows the player to perform diagonal movements.

The following table shows us the results for each algorithm mentioned above, without the use of the H1 function (all tested with the same input test):

Algorithm	moves	Path cost	Runtime (seconds)
A*	33	33	0.00954
Greddy	33	33	0.00206

Algorithm	moves	Path cost	Runtime (seconds)
Breadth-first	33	33	0.00490
Depth-first	33	33	0.00411
Uniform-cost	33	33	0.00564

Table 7

This table below presents the results, when using H1:

Algorithm	moves	Path cost	Runtime (seconds)
A*	23	26.999	0.01206
Greddy	23	26.999	0.00119
Breadth-first	23	26.999	0.00386
Depth-first	38	49.599	0.00197
Uniform-cost	23	26.999	0.00472

Table 8

By studying the tables above, we can verify that, when not using H1, the results generated by the algorithms used are all very similar, as we get the solution with 33 moves, a path cost of 33 units and also very close runtime values. Since we didn't use a heuristic function at first, every move as the same cost value, which means that the algorithms are only going to take into consideration the structure of the maze and the basic moves, which means that no particular path will be prioritized. However, when using H1 we can notice an increase in performance quality, as the A*, Greedy, Breadth-first and uniform-cost algorithms are now able to complete the maze and come up with a solution with just 23 moves and a path cost of approximately 26.999, so less moves and consequently a lower path cost.

On the other hand, the Depth-first algorithm when submitted to these new conditions, showed negative results, as its solution was accomplished with 38 moves and a path cost of approximately 49.599. This is probably due to the fact that this algorithm is an uninformed search algorithm, which means that, it doesn't take into account the cost values to get to the solution, it's not actually designed to find the optimal solution (shortest path) and although H1 is a suitable heuristic for this problem, this algorithm does not benefit from it and in this case, it even causes suboptimal outcomes.

It's important to notice that the A* method performed better with the H1 function, because these heuristics actually support its conditions for optimality, specifically the admissibility (never overestimate the cost) and the consistency (the triangle inequality) which is probably why this algorithm would most likely perform better than the others in a more complex test case.

Conclusion

In conclusion, after completing this Lab work by solving its tasks and problems, we can say that, predominantly, the A* algorithm proved itself to perform better than the others, regarding the addressed problems. This advantage shown by this algorithm, can be explained by the structure

of its method and appropriate heuristics, as it always tries to find the best possible solution (optimal solution).

In further investigations, we could use other (more complex) search algorithms along with more challenging test cases, experiment with new and different heuristic functions and continue to explore new problem-solving strategies, hence boosting the performance and results of all these search algorithms.

Lab 3. Genetic Algorithms

This lab sheet aims to help us understand more about genetic algorithms, by studying 3 classic artificial intelligence problems. So in this lab report, we will analyse three different tasks:

1. The travelling salesman problem
2. The 8-queens problem
3. The map colouring problem

Each one of these will be solved using genetic algorithms, which will be tested in terms of its performance, fitness function, genetic operators and crossover characteristics.

Task 3.1 The Travelling Salesman Problem (TSP)

Problem Description

This first task presents the problem of a salesman who needs to visit a number of N cities, from 1 to N. For this problem, we should presume that all the distances between the cities are already known. With this in mind, the objective is to calculate the best (optimal) route (city order) that the salesman should take, in order to get the minimum total distance travelled.

Let's now see an example.

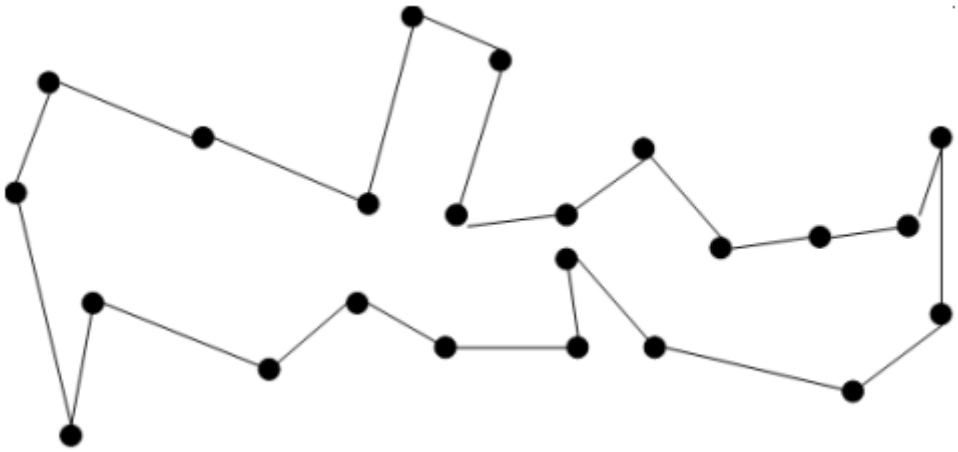


Figure 1

If we have 17 cities, the distances between each pair would be represented by the following array of values:

In [ ]:

```
TourSize = 17
distance_map = \
    [[0, 633, 257, 91, 412, 150, 80, 134, 259, 505, 353, 324, 70, 211, 268, 246, 121
    [633, 0, 390, 661, 227, 488, 572, 530, 555, 289, 282, 638, 567, 466, 420, 745, 5
    [257, 390, 0, 228, 169, 112, 196, 154, 372, 262, 110, 437, 191, 74, 53, 472, 142
    [91, 661, 228, 0, 383, 120, 77, 105, 175, 476, 324, 240, 27, 182, 239, 237, 84],
    [412, 227, 169, 383, 0, 267, 351, 309, 338, 196, 61, 421, 346, 243, 199, 528, 29
    [150, 488, 112, 120, 267, 0, 63, 34, 264, 360, 208, 329, 83, 105, 123, 364, 35],
    [80, 572, 196, 77, 351, 63, 0, 29, 232, 444, 292, 297, 47, 150, 207, 332, 29],
```

```
    [134, 530, 154, 105, 309, 34, 29, 0, 249, 402, 250, 314, 68, 108, 165, 349, 36],
    [259, 555, 372, 175, 338, 264, 232, 249, 0, 495, 352, 95, 189, 326, 383, 202, 23
    [505, 289, 262, 476, 196, 360, 444, 402, 495, 0, 154, 578, 439, 336, 240, 685, 3
    [353, 282, 110, 324, 61, 208, 292, 250, 352, 154, 0, 435, 287, 184, 140, 542, 23
    [324, 638, 437, 240, 421, 329, 297, 314, 95, 578, 435, 0, 254, 391, 448, 157, 30
    [70, 567, 191, 27, 346, 83, 47, 68, 189, 439, 287, 254, 0, 145, 202, 289, 55],
    [211, 466, 74, 182, 243, 105, 150, 108, 326, 336, 184, 391, 145, 0, 57, 426, 96]
    [268, 420, 53, 239, 199, 123, 207, 165, 383, 240, 140, 448, 202, 57, 0, 483, 153
    [246, 745, 472, 237, 528, 364, 332, 349, 202, 685, 542, 157, 289, 426, 483, 0, 3
    [121, 518, 142, 84, 297, 35, 29, 36, 236, 390, 238, 301, 55, 96, 153, 336, 0]]
```

For this example, the best path solution and respective total cost would be:

In [ ]:
```
OptTour = [0, 15, 11, 8, 4, 1, 9, 10, 2, 14, 13, 16, 5, 7, 6, 12, 3],
OptDistance = 2085
```

Before proceeding to the testing section, we need to define 3 main factors of the genetic algorithms:

1. Encoding scheme: For each city a number from 0 to 16 will be assigned, these cities will then be presented in a permutation (array).
2. Fitness function: We need to check the total cost of distances between all the adjacent cities (that constitute the path), while always looking for a optimal solution (shortest route) and taking into account that the salesman will eventually return to its initial position, so the cost between the final and first position should also be considered.
3. Genetic operators: There are 3 genetic operators we must acknowledge:
    A. Selection: tournament selection, as better solutions will then be used to create new ones. For example, the best solution out of 3.
    B. Crossover: We will choose a random place to swap the sub-strings between two parents and by doing that, recombine their genetic information to create new solutions while keeping the validity of the strings. One way of doing this is, for example, by copying the initial sub-string (solution) from one parent and then completing the second part by following that order in the second parent. We could take the path solution of two parent individuals in a generation and swap cities between them to create new individuals with different path solutions.
    C. Mutation: In order to maintain a certain genetic diversity in the population, we can choose two random places and swap them. We could change two random cities in the path solution.

Implementation and results

This section shows the implemented code and its respective output. We can also see the code responsible for the encoding scheme, fitness function and genetic operators.

In [ ]:
```
!pip install deap
import time
import array
import random
import numpy as np
from deap import algorithms, base, creator, tools
```

```
Collecting deap
  Downloading deap-1.4.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux
_2_17_x86_64.manylinux2014_x86_64.whl (135 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.4/135.4 kB 3.3 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
deap) (1.23.5)
Installing collected packages: deap
Successfully installed deap-1.4.1
```

In [ ]:

```
TourSize = 17
OptTour = [0, 15, 11, 8, 4, 1, 9, 10, 2, 14, 13, 16, 5, 7, 6, 12, 3],
OptDistance = 2085
distance_map = \
    [[0, 633, 257, 91, 412, 150, 80, 134, 259, 505, 353, 324, 70, 211, 268, 246, 121
    [633, 0, 390, 661, 227, 488, 572, 530, 555, 289, 282, 638, 567, 466, 420, 745, 5
    [257, 390, 0, 228, 169, 112, 196, 154, 372, 262, 110, 437, 191, 74, 53, 472, 142
    [91, 661, 228, 0, 383, 120, 77, 105, 175, 476, 324, 240, 27, 182, 239, 237, 84],
    [412, 227, 169, 383, 0, 267, 351, 309, 338, 196, 61, 421, 346, 243, 199, 528, 29
    [150, 488, 112, 120, 267, 0, 63, 34, 264, 360, 208, 329, 83, 105, 123, 364, 35],
    [80, 572, 196, 77, 351, 63, 0, 29, 232, 444, 292, 297, 47, 150, 207, 332, 29],
    [134, 530, 154, 105, 309, 34, 29, 0, 249, 402, 250, 314, 68, 108, 165, 349, 36],
    [259, 555, 372, 175, 338, 264, 232, 249, 0, 495, 352, 95, 189, 326, 383, 202, 23
    [505, 289, 262, 476, 196, 360, 444, 402, 495, 0, 154, 578, 439, 336, 240, 685, 3
    [353, 282, 110, 324, 61, 208, 292, 250, 352, 154, 0, 435, 287, 184, 140, 542, 23
    [324, 638, 437, 240, 421, 329, 297, 314, 95, 578, 435, 0, 254, 391, 448, 157, 30
    [70, 567, 191, 27, 346, 83, 47, 68, 189, 439, 287, 254, 0, 145, 202, 289, 55],
    [211, 466, 74, 182, 243, 105, 150, 108, 326, 336, 184, 391, 145, 0, 57, 426, 96]
    [268, 420, 53, 239, 199, 123, 207, 165, 383, 240, 140, 448, 202, 57, 0, 483, 153
    [246, 745, 472, 237, 528, 364, 332, 349, 202, 685, 542, 157, 289, 426, 483, 0, 3
    [121, 518, 142, 84, 297, 35, 29, 36, 236, 390, 238, 301, 55, 96, 153, 336, 0]]
```

In [ ]:

```
# fitness function as the sum of all the distances between each consecutive cities i
def evalTSP(individual):
    distance = distance_map[individual[-1]][individual[0]]
    for gene1, gene2 in zip(individual[0:-1], individual[1:]):
        distance += distance_map[gene1][gene2]
    return distance,

# create creator.FitnessMin and creator.Individual to be used in the toolbox
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", array.array, typecode='i', fitness=creator.FitnessMin)

# create the toolbox from the base
toolbox = base.Toolbox()

# add attribute generator to the toolbox
toolbox.register("indices", random.sample, range(TourSize), TourSize)

# initializers for individuals and population
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.indice
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# genetic operators and fitness function
toolbox.register("mate", tools.cxPartialyMatched)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.05)
# toolbox.register("select", tools.selRoulette)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evalTSP)
```

In [ ]:

```
random.seed(16)
pop = toolbox.population(n=500)
```

```
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("min", np.min)

init_time = time.time()

algorithms.eaSimple(pop, toolbox, 0.7, 0.2, ngen=50, stats=stats)
#algorithms.eaSimple(pop, toolbox, 0.7, 0.2, ngen=50, stats=stats, verbose=False)

print("--- %s seconds ---" % (time.time() - init_time))

best = tools.selBest(pop, 1)[0]
print("Best: %s\nTotal Distance: %s" %(best.tolist(), best.fitness.values))
```

```
gen     nevals  min
0       500     3369
1       395     3307
2       384     3261
3       386     3269
4       402     2938
5       371     2938
6       372     2697
7       361     2671
8       388     2671
9       377     2671
10      391     2487
11      348     2487
12      379     2487
13      386     2521
14      357     2521
15      366     2481
16      358     2481
17      390     2336
18      376     2457
19      402     2373
20      366     2373
21      360     2373
22      375     2278
23      388     2278
24      385     2199
25      396     2176
26      367     2176
27      382     2142
28      368     2175
29      367     2128
30      366     2115
31      376     2115
32      389     2088
33      374     2088
34      381     2088
35      380     2085
36      377     2085
37      383     2085
38      380     2085
39      418     2085
40      380     2085
41      367     2085
42      352     2085
43      364     2085
44      388     2085
45      373     2085
46      393     2085
47      388     2085
48      377     2085
49      375     2085
```

```
50      379     2085
--- 0.7595329284667969 seconds ---
Best: [1, 9, 10, 2, 14, 13, 16, 5, 7, 6, 12, 3, 0, 15, 11, 8, 4]
Total Distance: (2085.0,)
```

Discussions

To solve this problem, it was used the DEAP (Distributed Evolutionary Algorithms in Python), as it provides all the necessary genetic algorithms and related functions to solve this task.

We can see some more examples in this github page https://github.com/DEAP/deap and the online documentation of the DEAP can be found at https://deap.readthedocs.io/en/master/api/index.html.

This table presents us the results outputted by the genetic algorithm, when using the esSimple() function in our newly implemented solution:

Path cost	Runtime (seconds)	Number of generations
2085	0.75953	50

table 1

By analysing the results above, we can see that when using the esSimple() function (part of our genetic algorithm), we get an optimal path, 1-9-10-2-14-13-16-5-7-6-12-3-0-15-11-8-4, with an associated total cost of 2085 and a runtime of 0.75953 seconds. When looking at the optimal solution provided at the beginning of this task, we can notice that even though the path, 0-15-11-8-4-1-9-10-2-14-13-16-5-7-6-12-3 is different from the optimal one we got, the path cost value is 2085, which is equal. Therefore, we can verify that our solution is indeed an optimal one and so our results can be considered correct.

Task 3.2 The 8-Queens Problem

Problem Description

This next task is a very well-known one, that consists of having an 8x8 chessboard in which we need to put 8 queens, but in a way that none of these queens can attack each other. For example, we can't have two or more queens on the same column, row or diagonal.

In the following figure, we can see one example of a solution (out of the existing 92) to this problem.

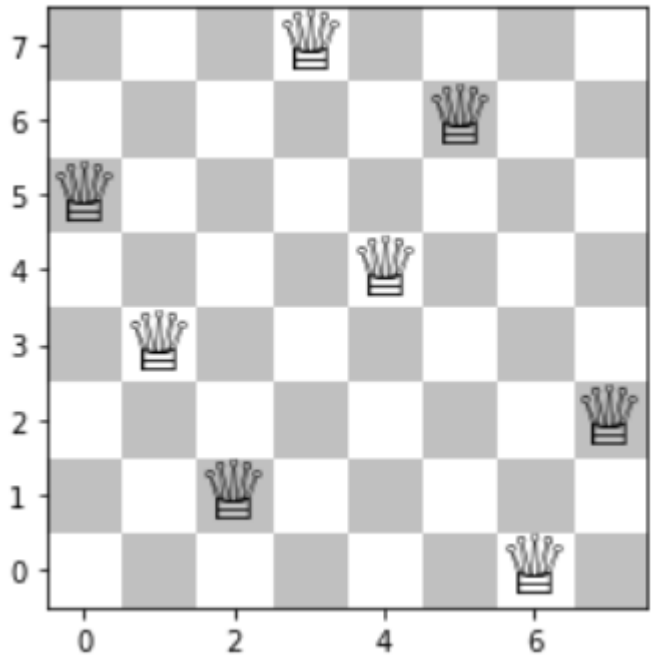


Figure 2

We now need to design this genetic algorithm:

1. Encoding scheme: For each queen, there is a respective position on the board, so we will use a permutation from 0 to 7, in other words, an array of 8 elements (8 queens), where each index represents the row and the actual value of the elements represents the column, in which the queen is placed on the board. This is an optimal choice, as the genetic algorithm is just going to work towards enforcing the diagonal constraint (as both the row and column constraints are already satisfied by the permutations).
2. Fitness function: We need to check how good a solution really is. To do so we can calculate the number of queens threatening each other, so the lower this number is, the better the individual is. Notice that when this number is equal to 0, we got ourselves a solution to this problem. As the other constraints are already taken care of, we will just calculate the number of queens threatening each other on the diagonals of the board, so for this specific 8x8 board, we have 15 raising diagonals and 15 falling diagonals, which are 30 diagonals in total. Therefore, we can utilize two arrays (of length 15) in order to store how many threats there are, so for a queen placed at position (x,y), the falling diagonal index can be calculated by doing x+y, because this formula stays constant as these positions are all in the same (falling) diagonal. On the other hand, the raising diagonal index can be calculated by doing x-y+7, as the same principle applies, but we need to add 7 so we can keep the calculated index value within boundaries (range of 0 to 14).
3. Genetic operators: We will take into account 3 different genetic operators:
    A. Selection: tournament selection, as better solutions will then be used to create other ones.
    B. Crossover: We will choose a random place to swap the sub-strings between two parents and by doing that, recombine their genetic information to create new solutions, while keeping the validity of the strings. One way of doing this is, for example, by copying the initial sub-string (solution) from one parent and then complete the second part by following that order in the second parent. We could take the solution of two parent

individuals in a generation and swap queen positions between them to create new individuals with different path solutions.

C. Mutation: In order to maintain a certain genetic diversity in the population, like choosing two random places and swap them. We could change two random queen positions in the solution.

Implementation and results

This section shows the implemented code and its respective output. We can also see the code responsable for the encoding scheme, fitness function and genetic operators.

In [ ]:
```
!pip install deap
import random, time
import numpy as np
from deap import algorithms, base, creator, tools
```

```
Requirement already satisfied: deap in /usr/local/lib/python3.10/dist-packages (1.4.
1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
deap) (1.23.5)
```

In [ ]:
```
NB_QUEENS = 12

def evalNQueens(individual):

    # create 2 list for the falling diagonal (fd) and the raising diagonal (rd)
    fd = np.zeros(2*NB_QUEENS-1)
    rd = np.zeros(2*NB_QUEENS-1)

    # count the number of queens placed on diagonals fd/rd
    for i in range(NB_QUEENS):
      fd[i+individual[i]] += 1
      rd[NB_QUEENS-1-i+individual[i]] += 1

    # sum the number of queens if more than 1 queen on a diagonal
    return np.sum(fd[fd>1]) + np.sum(rd[rd>1]),
```

In [ ]:
```
# enforce only 1 queen per column by using a list of NB_QUEENS
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

# enforce only 1 queen per row by using permutation
toolbox = base.Toolbox()
toolbox.register("permutation", random.sample, range(NB_QUEENS), NB_QUEENS)

# register all elements of the GA
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.permut
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("evaluate", evalNQueens)
toolbox.register("mate", tools.cxPartialyMatched)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=2.0/NB_QUEENS)
toolbox.register("select", tools.selTournament, tournsize=3)
```

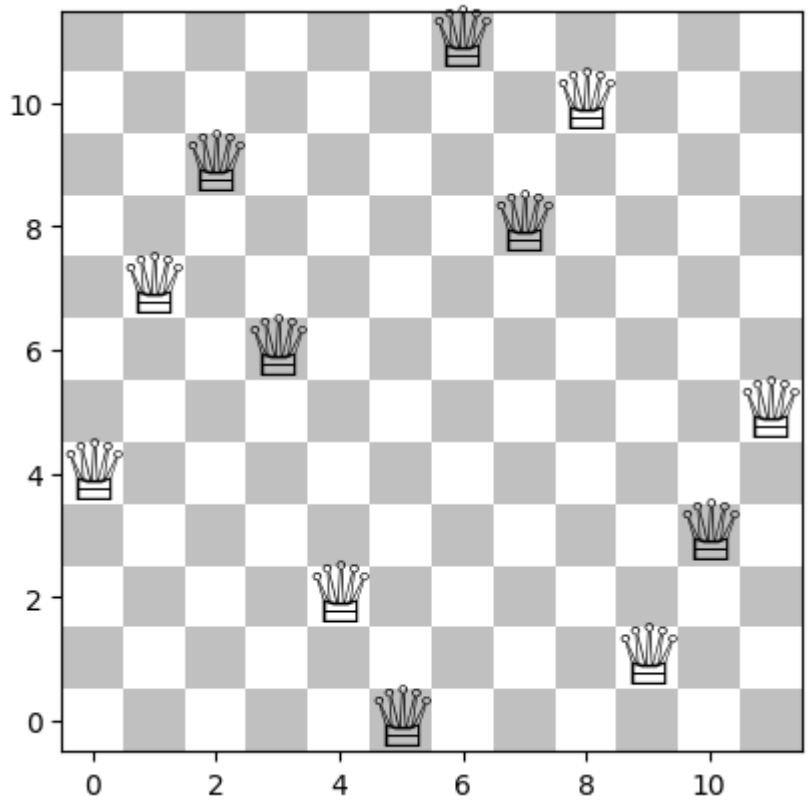
In [ ]:
```
# run the GA to get the result
random.seed(64)
pop = toolbox.population(n=300)
```

```
init_time = time.time()
algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=50, verbose=False)
print("--- %s seconds ---" % (time.time() - init_time))
best = tools.selBest(pop, 1)[0]
print("Best: %s. Fitness value: %s" %(best, best.fitness.values[0]))
```

```
--- 0.6540398597717285 seconds ---
Best: [4, 7, 9, 6, 2, 0, 11, 8, 10, 1, 3, 5]. Fitness value: 0.0
```

In [ ]:

```
# display the result
import matplotlib.pyplot as plt

chessboard = np.ones((NB_QUEENS, NB_QUEENS))
chessboard[1::2,0::2] = 0.75
chessboard[0::2,1::2] = 0.75
plt.imshow(chessboard, cmap='gray', origin='lower', vmin=0, vmax=1)
for x in range(NB_QUEENS):
    plt.text(x, best[x], '♕', fontsize=30, ha='center', va='center')
```



Discussions

To find a solution to this problem, again, it was used the DEAP (Distributed Evolutionary Algorithms in Python).

In the figure below, we can see an illustration of the solution generated when running the program.

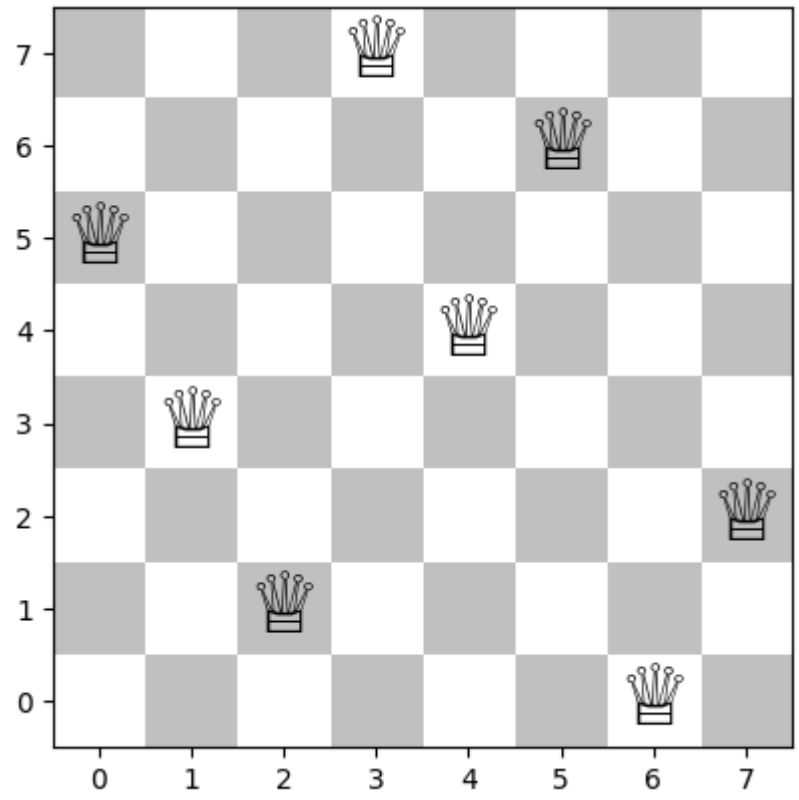


Figure 3

By looking at the results obtained, when once again using the esSimple() function, we notice that we successfully found a solution to the "8-queens problem", as none of the resulting queens are attacking one another, while the best position layout discovered for the queens is 5-3-1-7-4-6-0-2. The fitness value of this solution is 0 (as no queen has a target), which is also an indicator that we were successful in finding a solution.

For the sake of further investigation, we also tested this algorithm for a number of 12 queens. This new solution can be seen in the figure below.

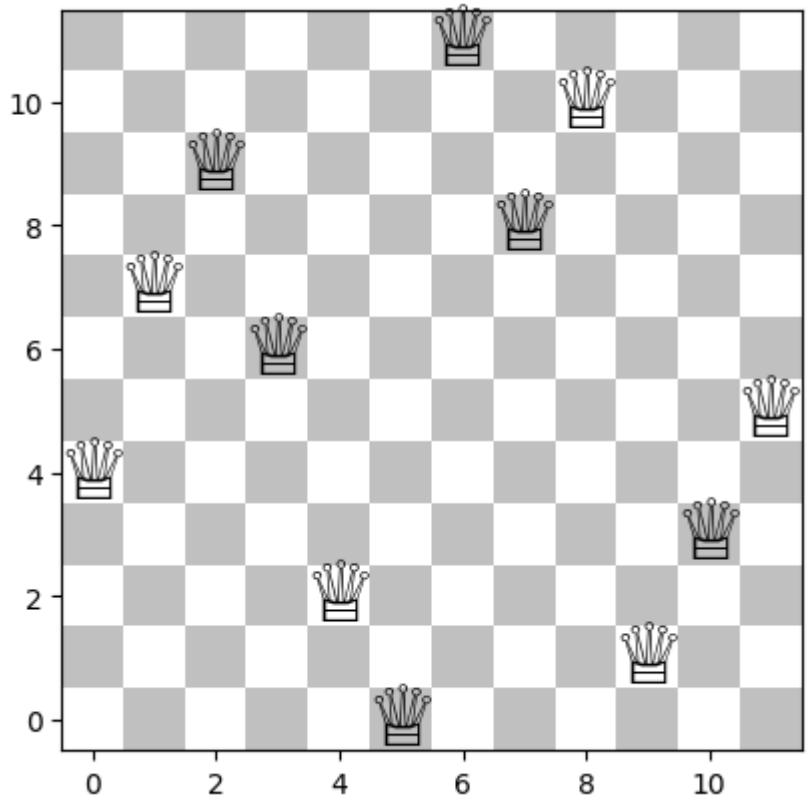


Figure 4

As we can see the best solution found to this new problem is also accepted, as once again, none of the resulting queens are attacking each other. The best position layout discovered for the queens is 4-7-9-6-2-0-11-8-10-1-3-5 and the fitness value of this solution is also equal to 0.

Task 3.3 The Map Colouring Problem

Problem Description

This last task presents the "map colouring problem", in which we have a map that needs to be coloured. So the objective is to colour this map using as many colours as possible, but no adjacent areas can have the same colour.

For this problem, we used a generic algorithm in order to create a solution that can colour an entire map with a maximum number of only 4 colours (as in the "four colours theorem").

In the figure below we can see a map example of this problem.

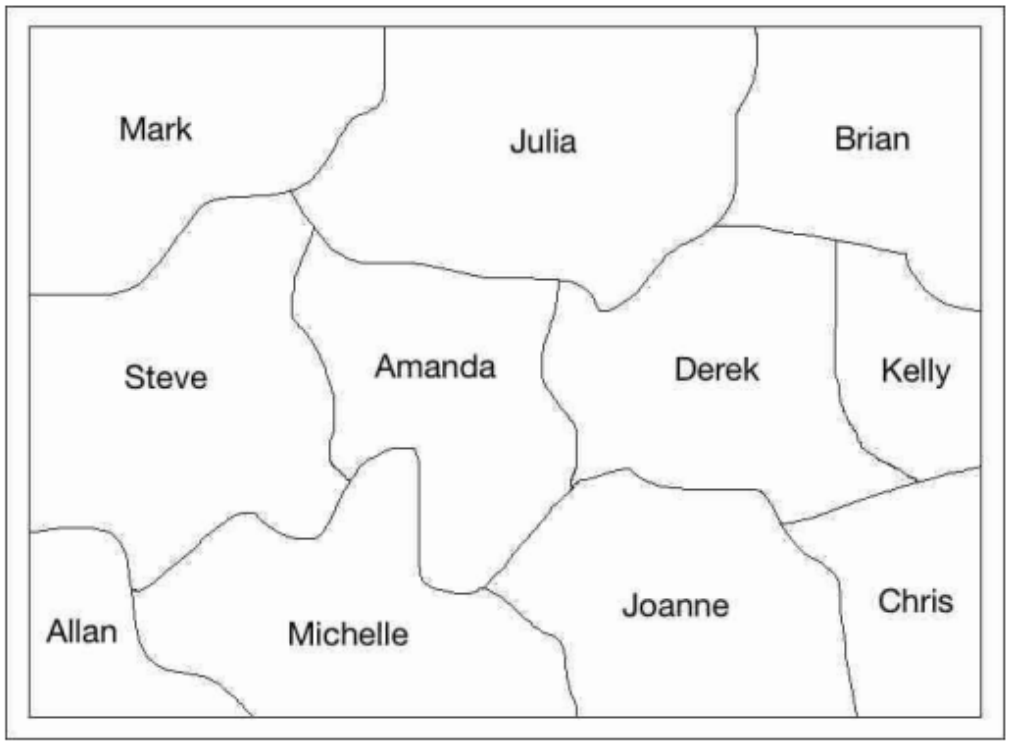


Figure 5

So in this example, we can see 11 different area names, that being, we can only use 4 colours to find a solution. In order to practically represent what is displayed in the map figure above, we can use an upper triangular matrix (array) that will contain the information about the adjency status for each area name concerning each other, so that value would be 0 if that certain area isn't adjacent and 1 otherwise. As an example, we could say that neighbours[0,1]=1 means Mark neighbouring Julia. We opted for this type of matrix instead of, for example, a full one, because the areas have reciprocal relationships when it comes to adjacency. We will also get a matrix in which its lower part is just zeros, this will also help us get rid of unnecessary data or redundancy and even save storage and other resources.

In [ ]:
```
neighbours =  [ [0,1,1,0,0,0,0,0,0,0,0],
                [0,0,1,1,1,0,1,0,0,0,0],
                [0,0,0,1,0,0,0,1,1,0,0],
                [0,0,0,0,0,1,1,0,1,0,0],
                [0,0,0,0,0,0,1,0,0,1,0],
                [0,0,0,0,0,0,1,0,1,1,1],
                [0,0,0,0,0,0,0,0,0,1,1],
                [0,0,0,0,0,0,0,0,1,0,0],
                [0,0,0,0,0,0,0,0,0,0,0],
                [0,0,0,0,0,0,0,0,0,0,1],
                [0,0,0,0,0,0,0,0,0,0,0]]
```

Let's now see the design of this genetic algorithm:

1. Encoding scheme: For representing the neighbourhood of each area, we will use an array with a size of 11 (as there are 11 names) that should represent the colour of each name, where the colours will be numbered from 0 to 3.

2. Fitness function: As we need to evaluate how good a solution is, we will calculate the number of adjacent areas that have the same colour, where the value 0 would indicate that we found a suitable solution for the problem, as there are 0 adjacent areas with the same colour.
3. Genetic operators: We will take into account 3 different genetic operators:
   A. Selection: tournament selection, as better solutions will then be used to create other ones.
   B. Crossover: We will choose a random place to swap the sub-strings between two parents and by doing that, recombine their genetic information to create new solutions while keeping the validity of the strings. One way of doing this is, for example, by copying the initial sub-string (solution) from one parent and then completing the second part by following that order in the second parent. We could take the (list) solution of two parent individuals in a generation and swap some of the colour positions between them to create new individuals with different colour solutions.
   C. Mutation: It would be beneficial to maintain a certain genetic diversity in the population, for that purpose, we could choose two random colour positions in the solution and swap them.

In [ ]:
```
!pip install deap
import array, time
import random
import numpy as np
from deap import creator, base, tools, algorithms
```

```
Collecting deap
  Downloading deap-1.4.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux
_2_17_x86_64.manylinux2014_x86_64.whl (135 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.4/135.4 kB 3.9 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
deap) (1.23.5)
Installing collected packages: deap
Successfully installed deap-1.4.1
```

In [ ]:
```
# Specify the variables
numColour = 4
numNames = 11
colours = ('red', 'green', 'blue', 'gray')
names = ('Mark', 'Julia', 'Steve', 'Amanda', 'Brian',
         'Joanne', 'Derek', 'Allan', 'Michelle', 'Kelly', 'Chris')

# Define the neighbours
neighbours =  [ [0,1,1,0,0,0,0,0,0,0,0],
                [0,0,1,1,1,0,1,0,0,0,0],
                [0,0,0,1,0,0,0,1,1,0,0],
                [0,0,0,0,0,1,1,0,1,0,0],
                [0,0,0,0,0,0,1,0,0,1,0],
                [0,0,0,0,0,0,1,0,1,1,1],
                [0,0,0,0,0,0,0,0,0,1,1],
                [0,0,0,0,0,0,0,0,1,0,0],
                [0,0,0,0,0,0,0,0,0,0,0],
                [0,0,0,0,0,0,0,0,0,0,1],
                [0,0,0,0,0,0,0,0,0,0,0]]
```

In [ ]:
```
def evalMapColouring(ind):
    val = 0
```

```
    for i in range(0, numNames):
      for j in range(0, numNames):
        if (neighbours[i][j] == 1) and (ind[i] == ind[j]):
            val += 1
    return val,

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", array.array, typecode='i', fitness=creator.FitnessMin)

toolbox = base.Toolbox()
toolbox.register("attr_colour", random.randint, 0, numColour-1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_co
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("evaluate", evalMapColouring)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutUniformInt, low=0, up=numColour-1, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)
```

In [ ]:

```
pop = toolbox.population(n=10)
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("min", np.min)
init_time = time.time()
pop, log = algorithms.eaSimple(pop, toolbox, cxpb=0.8, mutpb=0.4, ngen=100,
                               stats=stats, verbose=True)
print("--- %s seconds ---" % (time.time() - init_time))

best = tools.selBest(pop, 1)[0]
print("Best: %s. Fitness: %s." %(best.tolist(), evalMapColouring(best)[0]))
for i in range(numNames):
  print("%s ==> %s" %(names[i], colours[best[i]]))
```

```
gen     nevals  min
0       10      3
1       8       4
2       9       4
3       6       2
4       8       2
5       8       2
6       8       2
7       10      2
8       10      2
9       7       2
10      8       2
11      8       2
12      10      2
13      10      1
14      8       1
15      10      1
16      10      1
17      10      1
18      10      1
19      9       0
20      10      0
21      8       0
22      9       0
23      10      0
24      9       0
25      7       0
26      8       0
27      10      0
28      8       0
29      10      0
```

30	8	0
31	8	0
32	10	0
33	9	0
34	7	0
35	9	0
36	8	0
37	4	0
38	8	0
39	7	0
40	8	0
41	10	0
42	9	0
43	8	0
44	9	0
45	8	0
46	9	0
47	8	0
48	10	0
49	10	0
50	8	0
51	10	0
52	10	0
53	8	0
54	10	0
55	6	0
56	9	0
57	5	0
58	8	0
59	7	0
60	7	0
61	10	0
62	10	0
63	8	0
64	8	0
65	8	0
66	10	0
67	9	0
68	6	0
69	9	0
70	10	0
71	10	0
72	8	0
73	9	0
74	10	0
75	8	0
76	10	0
77	7	0
78	10	0
79	7	0
80	9	0
81	10	0
82	7	0
83	10	0
84	9	0
85	6	0
86	8	0
87	10	0
88	10	0
89	10	0
90	8	0
91	10	0
92	10	0
93	8	0

```
94      8       0
95      8       0
96      8       0
97      10      0
98      10      0
99      10      0
100     9       0
--- 0.06924057006835938 seconds ---
Best: [3, 1, 2, 3, 3, 2, 0, 3, 1, 1, 3]. Fitness: 0.
Mark ==> gray
Julia ==> green
Steve ==> blue
Amanda ==> gray
Brian ==> gray
Joanne ==> blue
Derek ==> red
Allan ==> gray
Michelle ==> green
Kelly ==> green
Chris ==> gray
```

Discussions

To solve this problem, once more it was used the DEAP (Distributed Evolutionary Algorithms in Python).

In the figure below, we can see an illustrated example of one possible solution generated by this algorithm. Notice that each run can give a different but also acceptable solution.

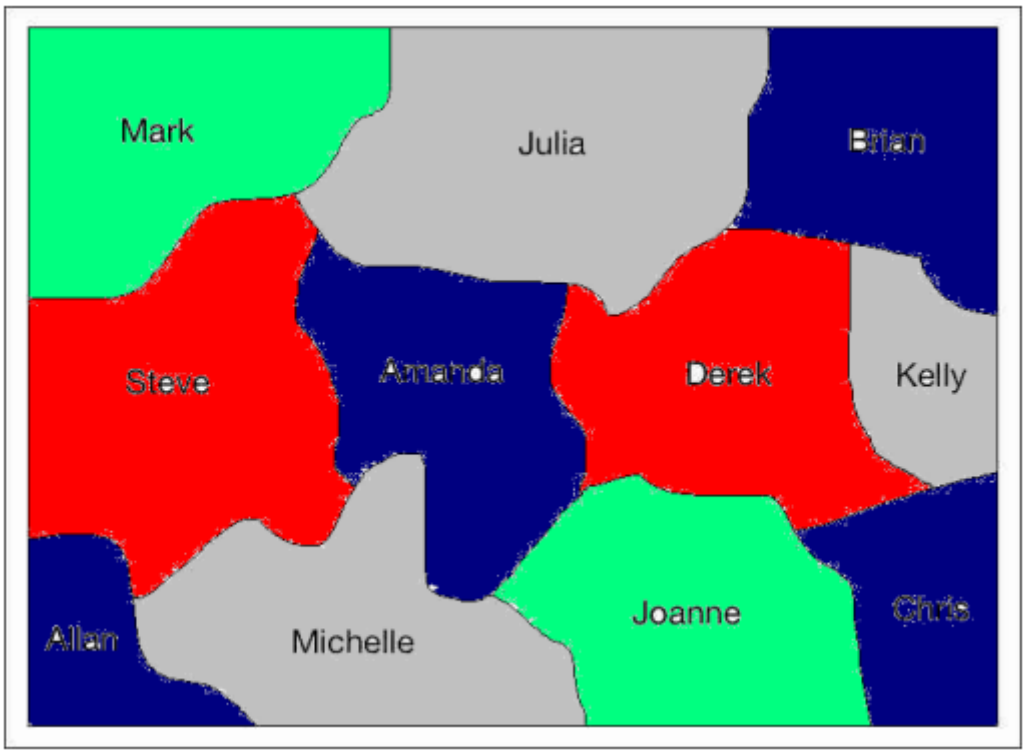


Figure 6

So some possible output examples could be, such as 1-3-0-2-2-1-0-2-3-3-2, 2-0-3-2-3-3-1-1-0-0-2, or for example, 3-1-2-3-3-2-0-3-1-1-3, all these are presented with a fitness value of 0.

By analysing the solutions previously mentioned, we can notice that none of the area names has the same colour as an adjacent one, also the fitness value is 0, which also means that we got an optimal solution. Therefore, we can assume that this algorithm revealed itself successful in finding a solution to this task.

Conclusion

In conclusion, after completing this Lab work by solving its tasks and problems, we can agree that by using the DEAP, we were able to find an optimal solution for these 3 tasks. By doing so, we also studied these 3 classic problems and therefore gained the skills to design genetic algorithms to solve these kinds of problem, as well to build the encoding scheme, fitness function and genetic operators.

In further investigations, we could use other and even more complex algorithms and data structures, as well as try to solve more advanced problems which would require better fitness functions and more evolved algorithmic genetic designs, which could also mean many more generations, individuals and solutions. Furthermore, we could also try to apply this knowledge to real-world applications or other areas of investigation.

Lab 4. Probabilistic Inference

This lab sheet aims to help us understand more about Probabilistic Inference, as we are going to study different probabilistic inference models and even solve 3 problems, which will be divided into the following tasks:

1. The iris classification problem
2. The diabetes diagnosis problem
3. The monty hall problem

Each one of these will be solved using the Naïve Bayes and Bayesian Networks.

Task 4.1 Iris Classification Using Naïve Bayes

Problem Description

This first task presents us with a flower classification problem, in which we will analyse the petal and sepal features of the flowers, in order to predict their species. For this, we will use a dataset that contains 5 attributes, such as four features of the flowers (sepal length, sepal width, petal length and petal width) and the iris species (three only). This dataset has a total number of 150 data samples of which 50 correspond for each species (we have 3 species).

We can see these species of flower in the figure below.

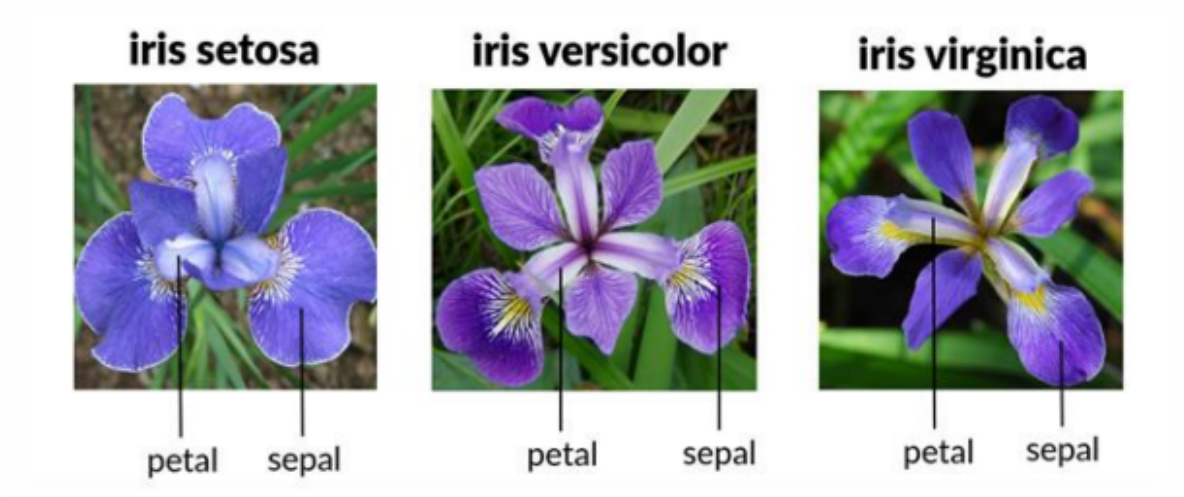


Figure 1

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:

```
!pip install scikit-learn
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
from sklearn.model_selection import train_test_split, cross_validate

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import patches
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-package
s (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packag
es (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-package
s (from scikit-learn) (1.11.3)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packag
es (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist
-packages (from scikit-learn) (3.2.0)
```

In [ ]:

```
# We only use two features (sepal and petaal length)
iris = datasets.load_iris()
X = iris.data
#X = iris.data[:, [0,2]]
Y = iris.target

# split the dataset into training (80%) and testing (20%), and fit the data into the
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
nb = GaussianNB()
nb.fit(X_train, Y_train)

# Use the naive Bayes model we just built to predict on the testing set, and display
Y_pred = nb.predict(X_test)
acc = accuracy_score(Y_test, Y_pred)
cm = confusion_matrix(Y_test, Y_pred)
cr = classification_report(Y_test, Y_pred)
print("Accuracy:", acc)
print("Confusion Matrix:\n", cm)
print("Prior:\n", nb.class_prior_)
print("Mean:\n", nb.theta_)
print("Variance:\n", nb.var_)
```

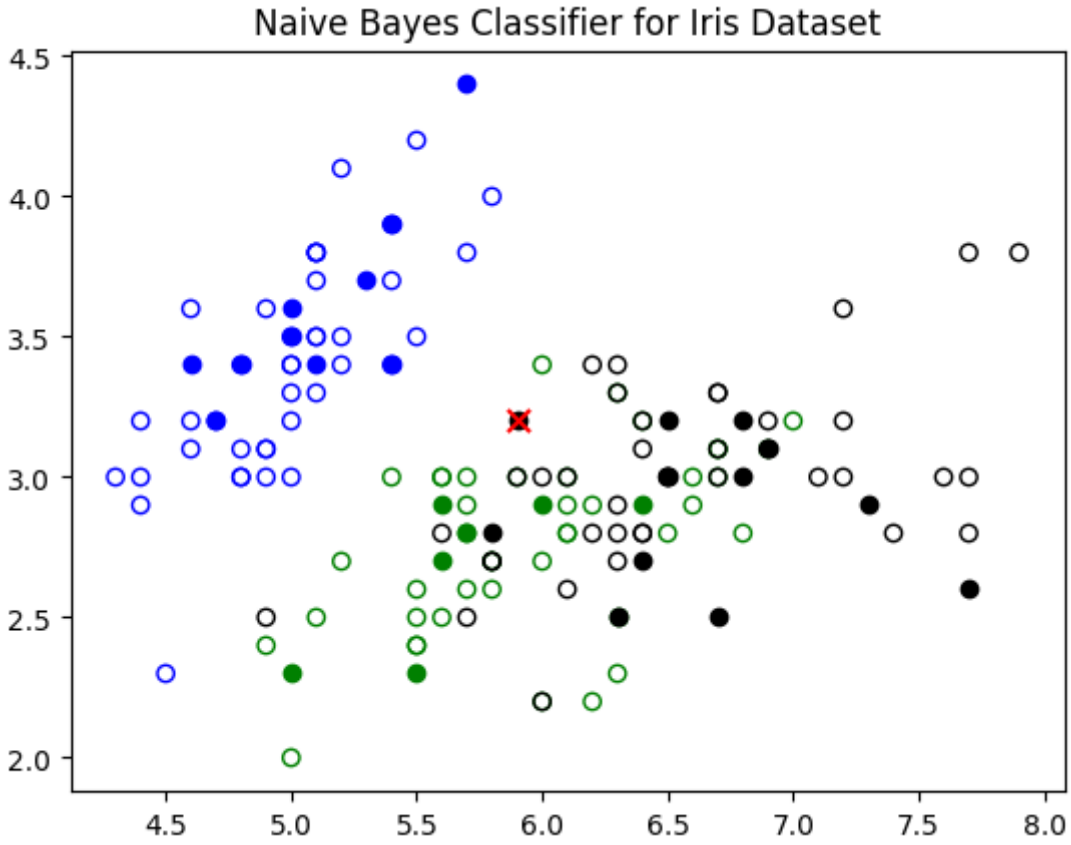
```
Accuracy: 0.9666666666666667
Confusion Matrix:
 [[10  0  0]
 [ 0  8  1]
 [ 0  0 11]]
Prior:
 [0.33333333 0.34166667 0.325     ]
Mean:
 [[4.9825     3.3875     1.46       0.2375    ]
 [5.97073171 2.7804878  4.27560976 1.31707317]
 [6.55641026 3.00512821 5.53333333 2.02307692]]
Variance:
 [[0.12194375 0.14109375 0.0334     0.01134375]
 [0.27572874 0.09913147 0.20964902 0.03702558]
 [0.43784353 0.10869165 0.29606838 0.08228797]]
```

In [ ]:

```
colours = 'bgk'
for i, colour in enumerate(colours):
  train_data = X_train[Y_train == i]
  plt.scatter(train_data[:,0], train_data[:,1], marker='o', facecolors='none', edgec
  #plt.scatter(train_data[:,2], train_data[:,3], marker='o', facecolors='none', edge
  test_data = X_test[Y_pred == i]
  plt.scatter(test_data[:,0], test_data[:,1], marker='o', facecolors=colour, edgecol
  #plt.scatter(test_data[:,2], test_data[:,3], marker='o', facecolors=colour, edgeco
```

```
error_data = X_test[Y_pred != Y_test]
plt.scatter(error_data[:,0], error_data[:,1], marker='x', facecolors='r', edgecolors
#plt.scatter(error_data[:,2], error_data[:,3], marker='x', facecolors='r', edgecolor

# ellipse = patches.Ellipse(xy=nb.theta_[0], width=nb.sigma_[0][0]*10, height=nb.sig
# # ellipse = patches.Ellipse(xy=(5,2), width=2, height=1)
# ax = plt.gca()
# ax.add_patch(ellipse)

plt.title("Naive Bayes Classifier for Iris Dataset")
plt.show()
```

```
<ipython-input-45-3b7d37fb0ffd>:11: UserWarning: You passed a edgecolor/edgecolors
('r') for an unfilled marker ('x').  Matplotlib is ignoring the edgecolor in favor of
the facecolor.  This behavior may change in the future.
  plt.scatter(error_data[:,0], error_data[:,1], marker='x', facecolors='r', edgecolor
s='r', s=64)
```



Discussions

To solve this problem, the sklearn or scikit-learn package was used, as it is a popular machine learning library and provides lots of important tools that were useful in this lab, such as the Bayesian inference methods and lots of machine learning algorithms necessary to this study. The online documentation can be found at https://scikit-learn.org/stable/user_guide.html.

For this first case, instead of using all four features, we only used two, sepal length and petal length, in order to simplify the problem and its results.

The following table presents us the results (an example) outputted by the program, when using the naive Bayes model:

accuracy	Confusion Matrix	Prior	Mean	Variance
0.8667	[[7 0 0] [0 9 2] [0 2 10]]	[0.3583 0.3250 0.3167]	[[5.0465 1.4790] [5.9435 4.2871] [6.6184 5.5657]]	[[0.1099 0.0295] [0.2050 0.1826] [0.4420 0.3306]]

table 1

Notice that each run can give different results as the dataset is randomly split into the training and testing sets.

By analysing the previous table, we can see that the ratio of correctly predicted instances to the total ones in the testing set, or in other words, how accurate was the model used in predicting the species based on the two features was approximately 0.8667 (which is a good result). Furthermore, The confusion matrix indicates the performance of the model, the prior shows us the probability of each species occurring in the sample before observing any features, the mean presents the average value of each feature for each species and we can also see the variance of each feature for each species.

In the figure below, we can see a plot relative to the example results displayed in the table above.

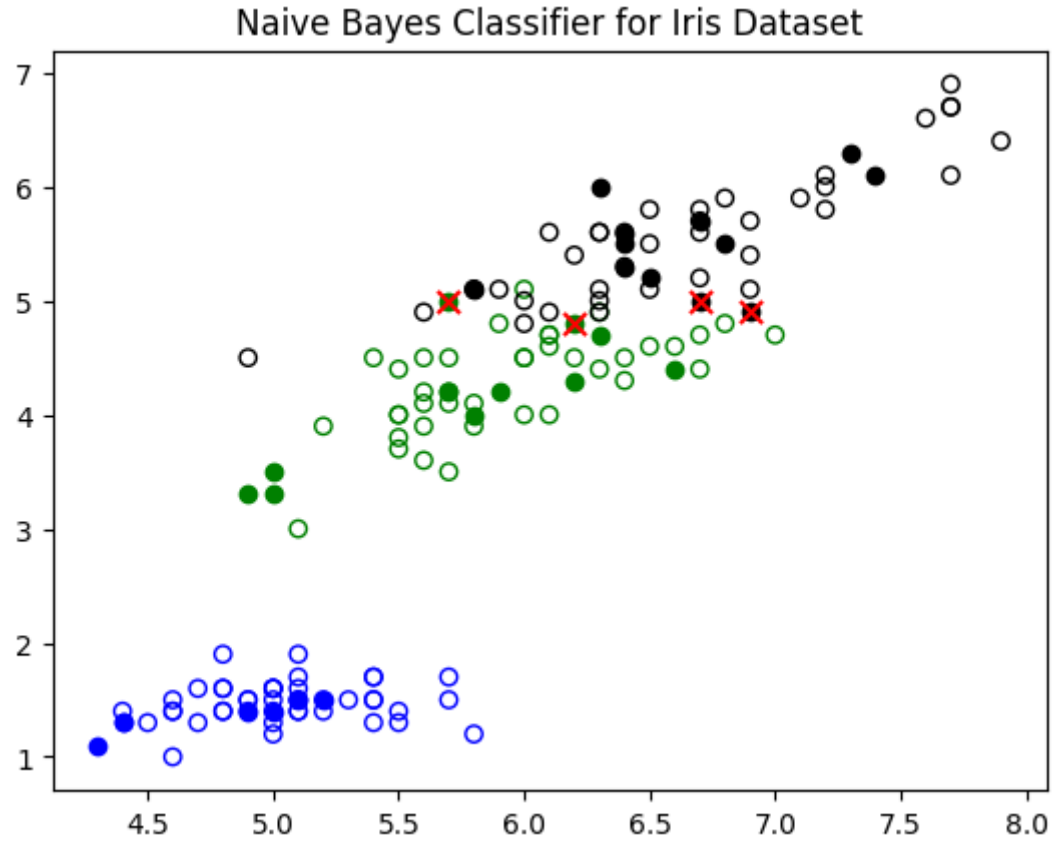


Figure 2

By looking at the plot, we can see the distributed data sample. Three different colours were used for representing the species, red crosses for the misclassified patterns, solid markers for the testing set, and empty ones for the training set. It's important to notice that, as two of the three existing species are not clearly separable in the dataset, we can see four values cross-marked, which occurred around the limit of these two classes.

For this second case, we will now use the four features available.

The table below shows us the results outputted for the full iris dataset:

accuracy	Confusion Matrix	Prior	Mean	Variance
0.9667	[[5 0 0] [0 11 0] [0 1 13]]	[0.3750 0.3250 0.3000]	[[5.0333 3.4466 1.4688 0.2533] [5.9589 2.7769 4.2820 1.3461] [6.5444 2.9527 5.5416 2.0527]]	[[0.1222 0.1496 0.0310 0.0113] [0.2813 0.1094 0.2301 0.0419] [0.4402 0.1052 0.3118 0.0813]]

table 2

By analysing the previous table, we can notice that the accuracy was higher this time, which means the results show a better level of prediction, some possible reasons for this to happen are, for example, the addition of features that may be more relevant for distinguishing the iris species, which can lead to a richer dataset and consequently to a more complex and overall better model.

In the figure below, we can see a plot relative to the example results displayed in the table above, where the first one (Figure 3) is relative to the first two features and the second one (Figure 4) is relative to the other existing two.

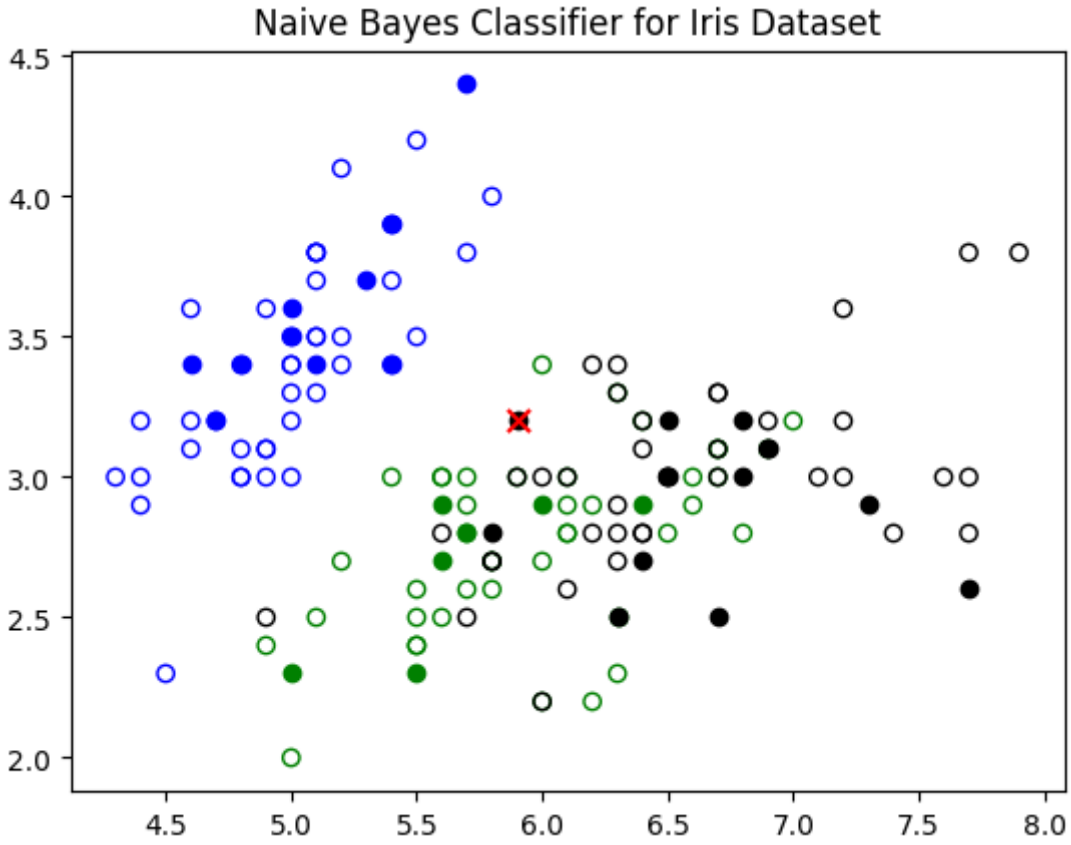


Figure 3

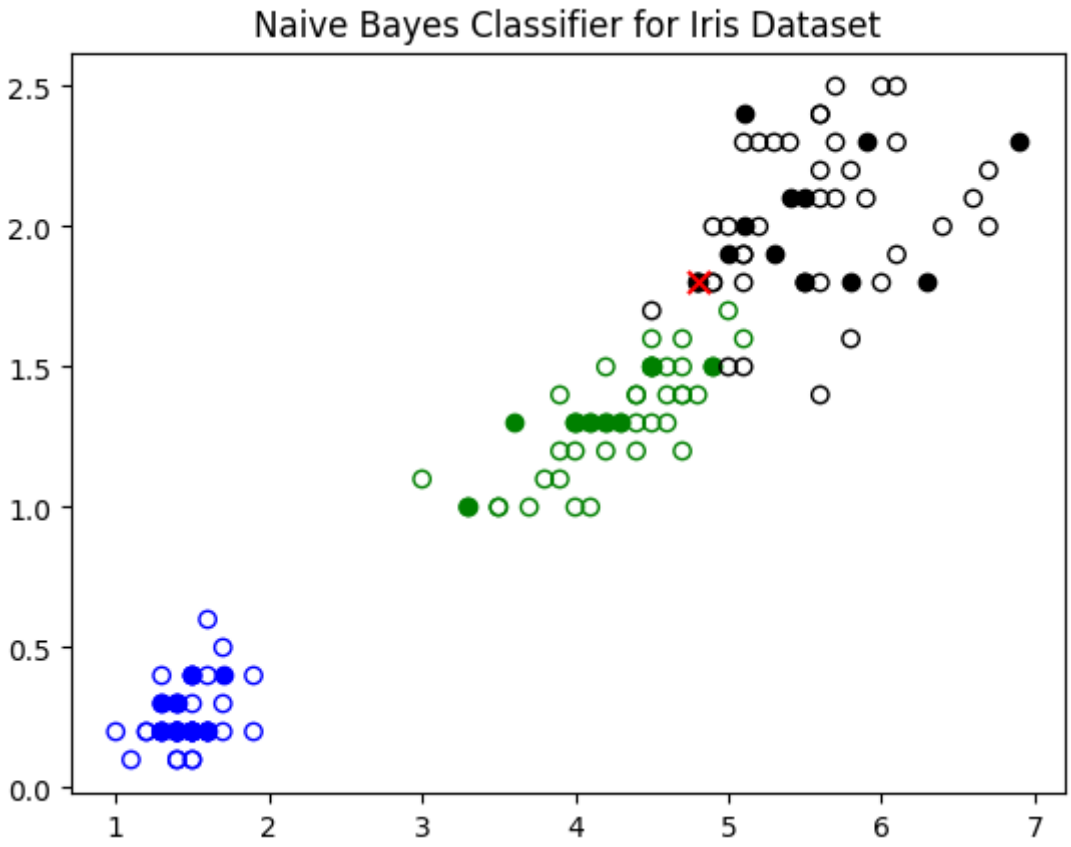


Figure 4

The plots above show us that, for the same reason as before, there is still (even though less) a value marked with a red cross (misclassified pattern).

Task 4.2 Diabetes Diagnosis

Problem Description

This next task presents us with a medical diagnosis problem, where we will predict if a certain patient is suffered from a specific disease, diabetes in this study case, or its levels of progression, when given numerous observations, such as basic personal details (age, gender, height, weight and others), body test data (blood sugar, heart rate, lipoproteins, and others) and symptoms (fever, headache, and others).

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:

```
!pip install scikit-learn
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split, cross_validate

# import numpy as np
import matplotlib.pyplot as plt
# from matplotlib import patches
import math
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-package
s (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packag
es (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-package
s (from scikit-learn) (1.11.3)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packag
es (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist
-packages (from scikit-learn) (3.2.0)
```

In [ ]:

```
diabetes = datasets.load_diabetes()
# X = diabetes.data[:,[2,3,9]]
X = diabetes.data
Y = [math.floor(x/150) for x in diabetes.target]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25)

nb = GaussianNB()
nb.fit(X_train, Y_train)
Y_pred = nb.predict(X_test)
acc = accuracy_score(Y_test, Y_pred)
cm = confusion_matrix(Y_test, Y_pred)
cr = classification_report(Y_test, Y_pred)

print("Accuracy:", acc)
print("Confusion Matrix:\n", cm)
print("Prior:\n", nb.class_prior_)
print("Mean:\n", nb.theta_)
print("Variance:\n", nb.var_)
```

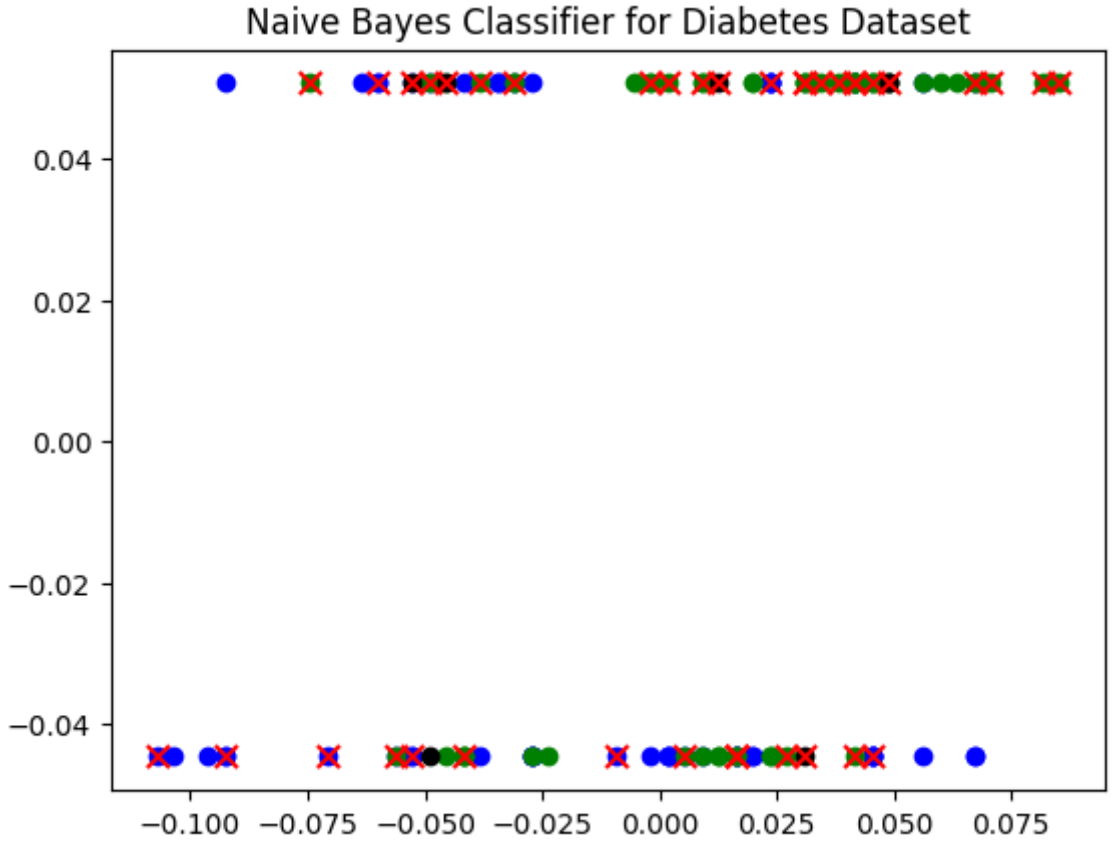
```
Accuracy: 0.6486486486486487
Confusion Matrix:
 [[39 19  0]
 [11 31  5]
 [ 1  3  2]]
Prior:
 [0.54380665 0.43202417 0.02416918]
Mean:
 [[-0.00914709 -0.00280598 -0.02001991 -0.01721048 -0.00633893 -0.00523432
   0.01729504 -0.01672129 -0.02255578 -0.01378016]
 [ 0.00998089 -0.00198015  0.01536536  0.01775517  0.00956375  0.00775648
  -0.01410222  0.01497061  0.02232395  0.01404244]
 [ 0.00583713  0.00301924  0.08109682  0.02832773  0.00617078  0.00584939
  -0.04017941  0.04168908  0.0457525   0.04603877]]
Variance:
 [[0.00213618 0.00223763 0.0013948  0.00177586 0.00217446 0.00227758
  0.00257269 0.00177764 0.00155205 0.00184507]
 [0.002322   0.00224657 0.00177613 0.00233222 0.00222325 0.00212438
  0.00137571 0.002031   0.00187924 0.00227871]
 [0.00198074 0.00227156 0.00089449 0.00308017 0.00153208 0.00202376
  0.00043861 0.00330891 0.00063405 0.00170257]]
```

In [ ]:

```
colours = 'bgk'
for i, colour in enumerate(colours):
  train_data = X_train[Y_train == i]
  plt.scatter(train_data[:,0], train_data[:,1], marker='o', facecolors='none', edgec
  test_data = X_test[Y_pred == i]
  plt.scatter(test_data[:,0], test_data[:,1], marker='o', facecolors=colour, edgecol

error_data = X_test[Y_pred != Y_test]
plt.scatter(error_data[:,0], error_data[:,1], marker='x', facecolors='r', edgecolors
```

```
# ellipse = patches.Ellipse(xy=nb.theta_[0], width=nb.sigma_[0][0]*10, height=nb.sig
# # ellipse = patches.Ellipse(xy=(5,2), width=2, height=1)
# ax = plt.gca()
# ax.add_patch(ellipse)

plt.title("Naive Bayes Classifier for Diabetes Dataset")
plt.show()
```

```
<ipython-input-69-5149f2f510ca>:9: UserWarning: You passed a edgecolor/edgecolors
('r') for an unfilled marker ('x').  Matplotlib is ignoring the edgecolor in favor of
the facecolor.  This behavior may change in the future.
  plt.scatter(error_data[:,0], error_data[:,1], marker='x', facecolors='r', edgecolor
s='r', s=64)
```



Discussions

To find a solution to this problem, again, it was used the sklearn or scikit-learn package.

For this problem, we used the diabetes dataset (provided in sklearn), which contains 10 features and one quantitative measure of diabetes progression. The progression measure was also turned into three different classes: 0-150, 150-300, and 300+.

The following table presents us the results (an example) outputted by the program when using the naive Bayes model:

accuracy	Confusion Matrix	Prior	Mean	Variance
0.6486	[[36 23 1] [9 35 1] [0 5 1]]	[0.5377 0.4380 0.0241]	[[-0.0065 -0.0001 -0.0227 -0.0169 -0.0095 -0.0080 0.0180 -0.0209 -0.0265 -0.0139] [0.0072 0.0020 0.0189 0.0153 0.0061 0.0040 -0.0154 0.0147 0.0233 0.0138] [	[[0.0023 0.0022 0.0013 0.0016 0.0021 0.0020 0.0022 0.0014 0.0015 0.0015] [0.0022 0.0022 0.0019 0.0022 0.0022 0.0022 0.0014 0.0020 0.0018 0.0023]

accuracy	Confusion Matrix	Prior	Mean	Variance
			0.0022 0.0149 0.0705 0.0231 0.0121 0.0129 -0.0369 0.0555 0.0359 0.0595]]]	[0.0018 0.0021 0.0015 0.0022 0.0021 0.0016 0.0019 0.0041 0.0017 0.0014]]

table 3

By analysing the previous table, we can notice that the accuracy rate is lower compared to the previous task, which indicates that the values obtained are not as precise in terms of the level of prediction. Therefore, through the information provided by the table above we can infer that there were some efficiency problems when it comes to correctly predicting data.

In the following figure, we can see a plot relative to the example results displayed in the table above, where, just for the sake of simplicity and behaviour study, we only considered the first two features.

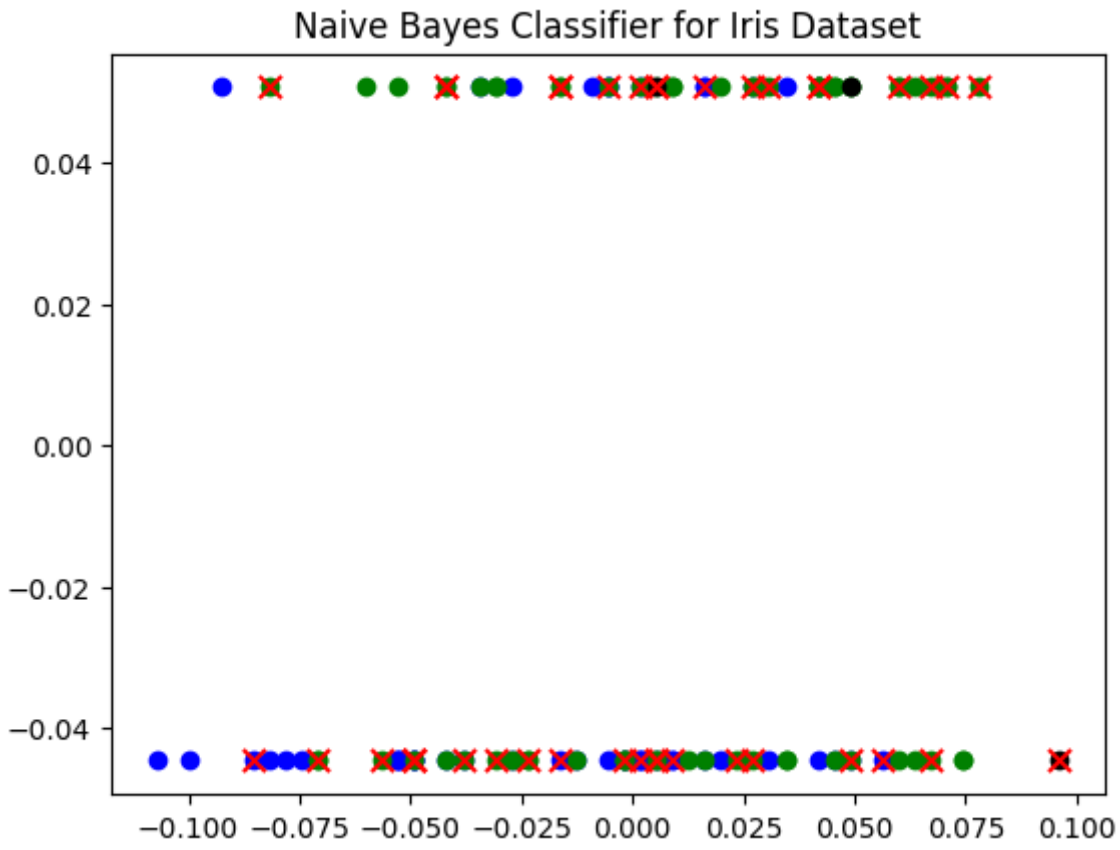


Figure 5

By looking at the plot, we can see that there are many values marked with a red cross (misclassified pattern), which adds to the conclusion that the results for this task were not as predicatively correct as they were in the previous one.

Task 4.3 Monty Hall Problem Using Bayesian Network

Problem Description

This last task presents the famous "Monty Hall" problem, which is a probability puzzle named after the host of an American TV show, called Monty Hall. So this problem consists of having 3

distinct doors, in which behind 1 of them is a car (that represents the prize) and behind the other 2 are goats (which are worthless). As a contestant, we pick one of these 3 doors and then the host, who knows what's behind each door, will open one of the doors that contains a goat. After doing this, the host will ask us (the contestant) if we would like to switch to the other closed door or, instead, stick with our original choice.

Now, the real challenge is to know what is the best choice to make. The answer to this question is, yes, we should switch our choice to the other unopened door, as it's to our advantage to do so, since now our odds of winning have just doubled.

The following figure shows a representation of the problem.

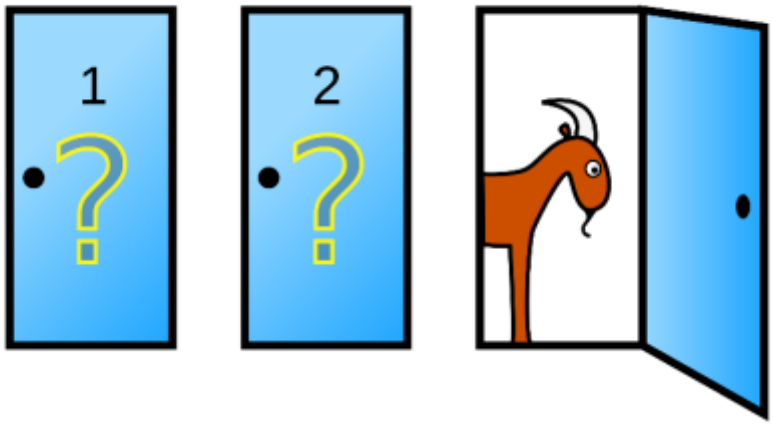


Figure 6

This is a very popular brain teaser problem, more info about it can be found at https://en.wikipedia.org/wiki/Monty_Hall_problem.

In order to solve this problem, we are going to use Bayesian Networks. So first of all, we have to define the variables, P is going to symbolize the prize, and since we have 3 doors available where it could be, then P={0, 1, 2}. As for the contestant's initial door chosen, C, we also assume that C={0, 1, 2}. For the host's choice, H, which will totally depend on where the car is and also what was the contestant's initial choice, we can say that H={0, 1, 2}. When talking about prior probabilities (initial probability about an event before any other has happened), we can infer that P(P=0) = P(P=1) = P(P=2) = 1/3, P(C=0) = P(C=1) = P(C=2) = 1/3 and, since the host's decision relies on P and C, we will have the following structure.

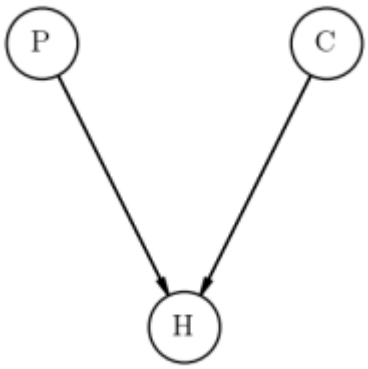


Figure 7

The figure above represents our Bayesian Network.

Let's now take a look at the following table, which shows us the conditional probability (probability of an event happening, given that another one has already occurred) of H given P and C is:

C	0			1			2		
P	0	1	2	0	1	2	0	1	2
H=0	0	0	0	0	0.5	1	0	1	0.5
H=1	0.5	0	1	0	0	0	1	0	0.5
H=2	0.5	1	0	1	0.5	0	0	0	0

Figure 8

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:

```
!pip install pgmpy
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
```

```
Collecting pgmpy
  Downloading pgmpy-0.1.24-py3-none-any.whl (2.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 33.6 MB/s eta 0:00:00
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (f
rom pgmpy) (3.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
pgmpy) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from
pgmpy) (1.11.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-package
s (from pgmpy) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (fro
m pgmpy) (1.5.3)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages
(from pgmpy) (3.1.1)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from
pgmpy) (2.0.1+cu118)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages
(from pgmpy) (0.14.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from
pgmpy) (4.66.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (fro
m pgmpy) (1.3.2)
Requirement already satisfied: opt-einsum in /usr/local/lib/python3.10/dist-packages
(from pgmpy) (3.3.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/di
st-packages (from pandas->pgmpy) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-package
s (from pandas->pgmpy) (2023.3.post1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist
-packages (from scikit-learn->pgmpy) (3.2.0)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-package
s (from statsmodels->pgmpy) (0.5.3)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-pack
```

```
ages (from statsmodels->pgmpy) (23.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (f
rom torch->pgmpy) (3.12.4)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-pa
ckages (from torch->pgmpy) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from
torch->pgmpy) (1.12)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (fro
m torch->pgmpy) (3.1.2)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packag
es (from torch->pgmpy) (2.0.0)
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from
triton==2.0.0->torch->pgmpy) (3.27.6)
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from t
riton==2.0.0->torch->pgmpy) (17.0.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from p
atsy>=0.5.2->statsmodels->pgmpy) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-pack
ages (from jinja2->torch->pgmpy) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-package
s (from sympy->torch->pgmpy) (1.3.0)
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.24
```

In [ ]:

```python
# Defining the network structure
model = BayesianModel([('Contestant', 'Host'), ('Prize', 'Host')])

# Defining the CPDs:
'''
cpd_c = TabularCPD('Contestant', 3, [[1/3], [1/3], [1/3]])
cpd_p = TabularCPD('Prize', 3, [[1/3], [1/3], [1/3]])
cpd_h = TabularCPD('Host', 3, [[0, 0, 0, 0, 0.5, 1, 0, 1, 0.5],
                               [0.5, 0, 1, 0, 0, 0, 1, 0, 0.5],
                               [0.5, 1, 0, 1, 0.5, 0, 0, 0, 0]],
                   evidence=['Contestant', 'Prize'], evidence_card=[3, 3])
'''

cpd_c = TabularCPD('Contestant', 4, [[0.25], [0.25], [0.25], [0.25]])
cpd_p = TabularCPD('Prize', 4, [[0.25], [0.25], [0.25], [0.25]])
cpd_h = TabularCPD('Host', 4, [[0,   0,    0,    0,    0,    1/3, 0.5,  0.5,  0,
                               [1/3, 0,    0.5,  0.5,  0,    0,   0,    0,    0.5,
                               [1/3, 0.5,  0,    0.5,  0.5,  1/3, 0,    0.5,  0,
                               [1/3, 0.5,  0.5,  0,    0.5,  1/3, 0.5,  0,    0.5,
                   evidence=['Contestant', 'Prize'], evidence_card=[4, 4])

# Associating the CPDs with the network structure.
model.add_cpds(cpd_c, cpd_p, cpd_h)
```

```
WARNING:pgmpy:BayesianModel has been renamed to BayesianNetwork. Please use BayesianN
etwork class, BayesianModel will be removed in future.
```

In [ ]:

```python
# Infering the posterior probability
from pgmpy.inference import VariableElimination

infer = VariableElimination(model)
posterior = infer.query(variables=['Prize'], evidence={'Contestant': 0, 'Host': 2},
print(posterior['Prize'])
```

```
WARNING:pgmpy:BayesianModel has been renamed to BayesianNetwork. Please use BayesianN
etwork class, BayesianModel will be removed in future.
WARNING:pgmpy:BayesianModel has been renamed to BayesianNetwork. Please use BayesianN
etwork class, BayesianModel will be removed in future.
```

```
+----------+--------------+
| Prize    |   phi(Prize) |
+==========+==============+
| Prize(0) |       0.2500 |
+----------+--------------+
| Prize(1) |       0.3750 |
+----------+--------------+
| Prize(2) |       0.0000 |
+----------+--------------+
| Prize(3) |       0.3750 |
+----------+--------------+
```

Discussions

To solve this problem, we used the pgmpy library, which provides graphical models including the Bayesian networks. The online documentation can be found at https://pgmpy.org/.

This table presents us the results outputted by the algorithm when using the Bayesian Network:

Prize	phi(Prize)
Prize(0)	0.3333
Prize(1)	0.6667
Prize(2)	0.0000

table 4

By analyzing the previous table and its results, we can notice that the decision to keep our original door choice has a prize probability of 1/3, on the other hand, switching to the other closed door gives us a prize probability of 3/2 (while obviously, the prize probability of the already revealed door is 0). Therefore, we can conclude that the best option is to switch our choice as our chances of winning are doubled.

Let's now study the results of a different scenario, where the rules of the game are changed and now we have 4 doors, while the other rules stay the same since we still choose one door and the host still opens one door that doesn't contain the prize.

This next table presents us the results outputted by this new algorithm submitted to the changes mentioned before when using the Bayesian Network:

Prize	phi(Prize)
Prize(0)	0.2500
Prize(1)	0.3750
Prize(2)	0.0000
Prize(3)	0.3750

table 5

By analysing the table above, we can see that the probabilities have changed a lot. Initially, the probability of choosing the prize is 1/4, as the host reveals one of the doors with a goat, the prize probability of that door will obviously, instantly become 0. Furthermore, as the prize probability is 1/4 and one goat was revealed, this leaves us with 3/4 of the prize probability that is distributed for the two remaining doors, which gives us a 3/8 of the prize probability for these

two other unopened doors. Therefore, we can conclude that in this scenario, it's also in our interest to switch our choice to one of these two doors, since we have a higher chance of winning the prize.

Conclusion

In conclusion, after completing this Lab work by solving its tasks and problems, we can agree that by using the Naïve Bayes and Bayesian Networks, we were able to find a solution for these 3 tasks. By doing so, we also studied these 3 classic problems and therefore gained additional knowledge on how these tools and methods can help us to predict data and work on probabilistic inference.

In further investigations, we could use and explore nonlinear methods, create more complex Bayesian Networks, compare the performance of Naïve Bayes with other machine learning algorithms, change the training and testing set to get different results, and explore other scenarios. Furthermore, we could also try to apply this knowledge to real-world applications or other areas of investigation.

Lab 5. Fuzzy Systems

This lab sheet aims to help us understand more about fuzzy systems, as we are going to study this new concept and use it to solve 3 problems, which will be divided into the following tasks:

1. The tipping problem
2. The project risk assessment problem
3. The washing machine fuzzy controller problem

Task 5.1 The Tipping Problem

Problem Description

This first task presents us with a tipping problem, which is meant to recommend how much to tip for a food service, in this case, in a restaurant as well as its service quality.

This problem is actually an example very much based on the scikit-fuzzy package (which we will mention in the discussions of this task).

First of all, we need to define the problem in terms of its food and service quality. We are going to have 3 different linguistic variables:

1. The food, which will have "poor", "average" and "good" as linguistic values.
2. The service, which will have "poor", "average" and "good" as linguistic values.
3. The tip, which will have "low", "medium" and "high" as linguistic values.

Note that the food and the service will serve as inputs, while the tip will be an output.

We can see the memberships (in which all present the same behaviour) of the linguistic values represented in the figure, or in other words, the fuzzy sets:

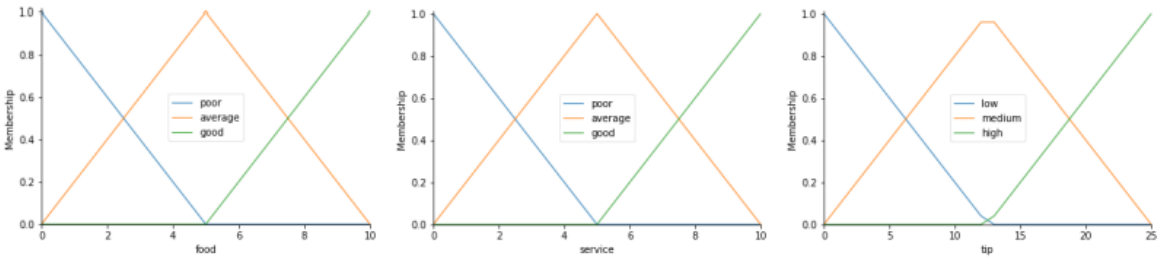


Figure 1

Let's now see the rules that we will use throughout this task:

1. IF the service was poor OR the food quality was poor THEN the tip will be low
2. IF the service was average THEN the tip will be medium
3. IF the service was good OR the food quality was good THEN the tip will be high

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:

```
!pip install scikit-fuzzy
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

```
Collecting scikit-fuzzy
  Downloading scikit-fuzzy-0.4.2.tar.gz (993 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 994.0/994.0 kB 6.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.10/dist-package
s (from scikit-fuzzy) (1.23.5)
Requirement already satisfied: scipy>=0.9.0 in /usr/local/lib/python3.10/dist-package
s (from scikit-fuzzy) (1.11.3)
Requirement already satisfied: networkx>=1.9.0 in /usr/local/lib/python3.10/dist-pack
ages (from scikit-fuzzy) (3.1)
Building wheels for collected packages: scikit-fuzzy
  Building wheel for scikit-fuzzy (setup.py) ... done
  Created wheel for scikit-fuzzy: filename=scikit_fuzzy-0.4.2-py3-none-any.whl size=8
94079 sha256=9b8d6afd09a7965caffa84a74b8fbec1e5fe37997147d8564f0b68c372a573dc
  Stored in directory: /root/.cache/pip/wheels/4f/86/1b/dfd97134a2c8313e519bcebd95d3f
edc7be7944db022094bc8
Successfully built scikit-fuzzy
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.4.2
```

In [ ]:

```
# New Antecedent/Consequent objects hold universe variables and membership functions
food = ctrl.Antecedent(np.arange(0, 11, 1), 'food')
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# Auto-membership function population is possible with .automf(3, 5, or 7)
food.automf(3)
service.automf(3)
tip.automf(3, names=['low', 'medium', 'high'])

# food.view()
# service.view()
# tip.view()
```

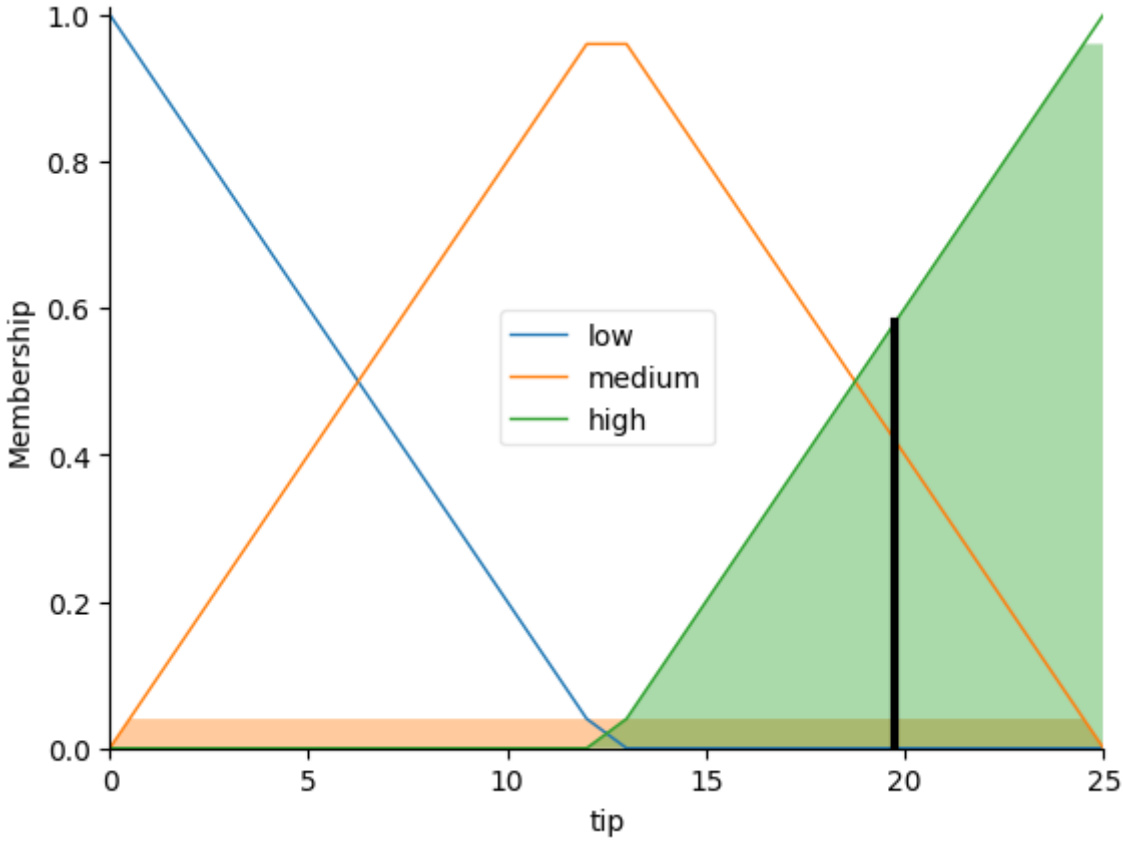
In [ ]:

```
# Define the rules
rule1 = ctrl.Rule(food['poor'] | service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(service['good'] | food['good'], tip['high'])

# Create the control system and its simulation
tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
```

In [ ]:

```
# Pass inputs to the ControlSystem
tipping.inputs({'food': 6.5, 'service': 9.8})

# Crunch the numbers
tipping.compute()
print("Suggested tip:", tipping.output['tip'], "%")
tip.view(sim=tipping)
```

```
Suggested tip: 19.76409495548962 %
```



Discussions

To solve this problem, the scikit-fuzzy package was used, as it is a popular fuzzy logic library and provides lots of important tools that were useful in this lab, such as lots of common fuzzy logic methods. The online documentation can be found at https://pythonhosted.org/scikit-fuzzy/.

The project github page can also be found at https://github.com/scikit-fuzzy/scikit-fuzzy.

For this problem, in order to compute and calculate the tip, as inputs we used the food with a rating of 6.5 and the service quality with a rating of 9.8. Therefore, we can say that, in this case, we used a considerable "average" range of food as well as a mostly "good" range of service. The program outputted a value result of approximately 19.764%, given the previous inputted values. Note that the suggested tip can only go up to 25%.

In the following figure, we can see a plot regarding the tip, and relative to the fuzzy system computed when running the program.

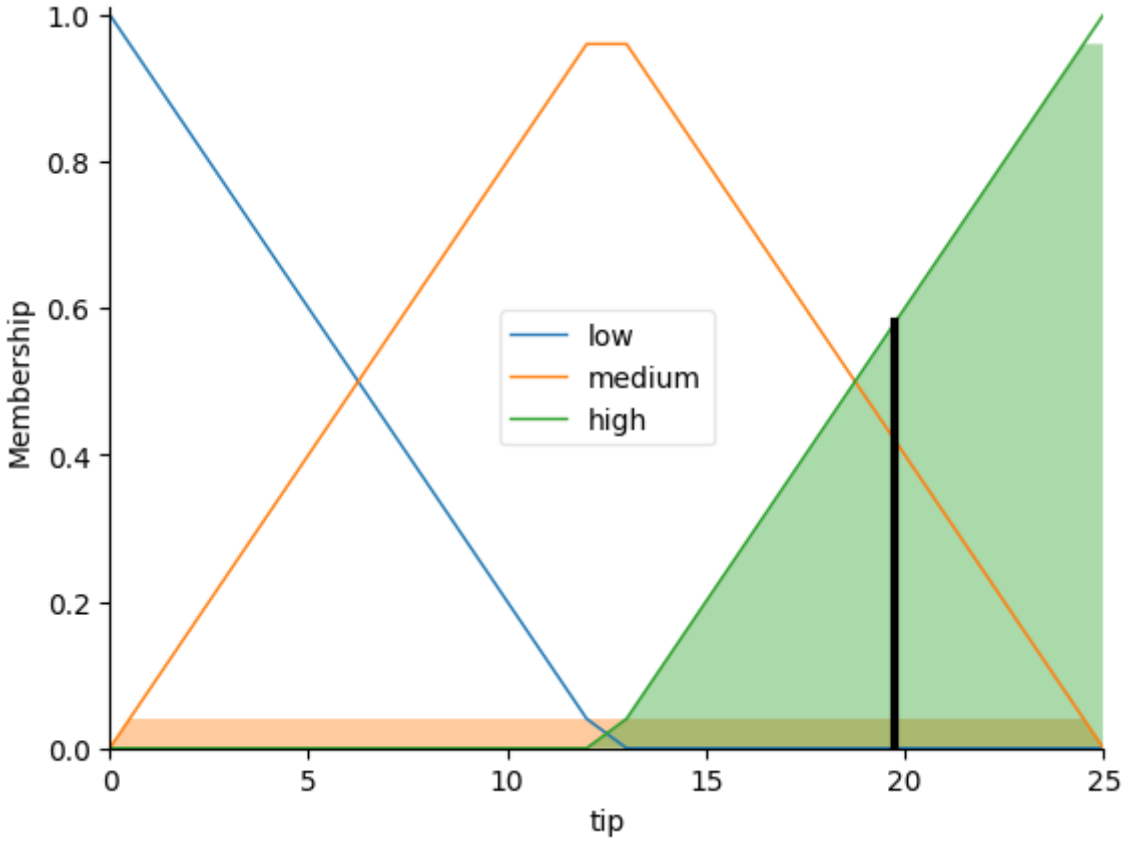


Figure 2

By analysing the results, we can conclude that the suggested tip is approximately 19.764% this being a "medium" to "high" range tip, which demonstrates how we can use fuzzy logic and its applications when it comes to complex decision-making, instead of just using a simple binary decision, such as a "yes" or "no" decision label.

Task 5.2 Project Risk Assessment

Problem Description

This next task presents a project risk assessment problem, in which we will assess and evaluate the risk related to the making of a certain project.

We now need to define the problem in terms of its linguistic variables. We will have 3 different linguistic variables:

1. The funding, which will have "inadequate", "marginal" and "adequate" as linguistic values.
2. The staffing, which will have "small" and "large" as linguistic values.
3. The risk, which will have "low", "normal" and "high" as linguistic values.

Note that the funding and the staffing will serve as inputs, while the risk will be an output.

The following figure shows us the memberships (and their behaviours) of the linguistic values previously mentioned, or in other words, the fuzzy sets:

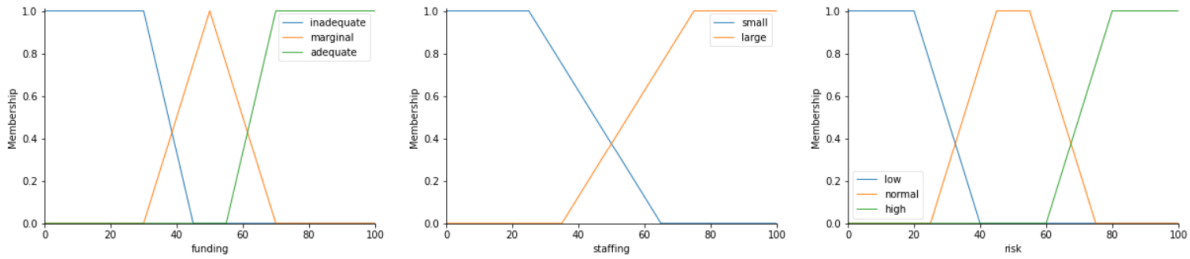


Figure 3

Note that these membership functions (taken from the lecture) are not defined exactly the same as the ones we are going to use in our solution.

Let's now look at the rules that we will use throughout this task:

1. IF project_funding is adequate
   OR project_staffing is small
   THEN risk is low
2. IF project_funding is marginal
   AND project_staffing is large
   THEN risk is normal
3. IF project_funding is inadequate
   THEN risk is high

So overall, for this problem we now have that:

Fuzzy system:
Inputs: project_funding (x), project_staffing (y)
Output: risk (z)
Three rules:
Rule: 1

> IF x is A3
> OR y is B1
> THEN z is C1

Rule: 2

> IF x is A2
> AND y is B2
> THEN z is C2

Rule: 3

> IF x is A1
> THEN z is C3

Implementation and results

This section shows the implemented code and its respective output.

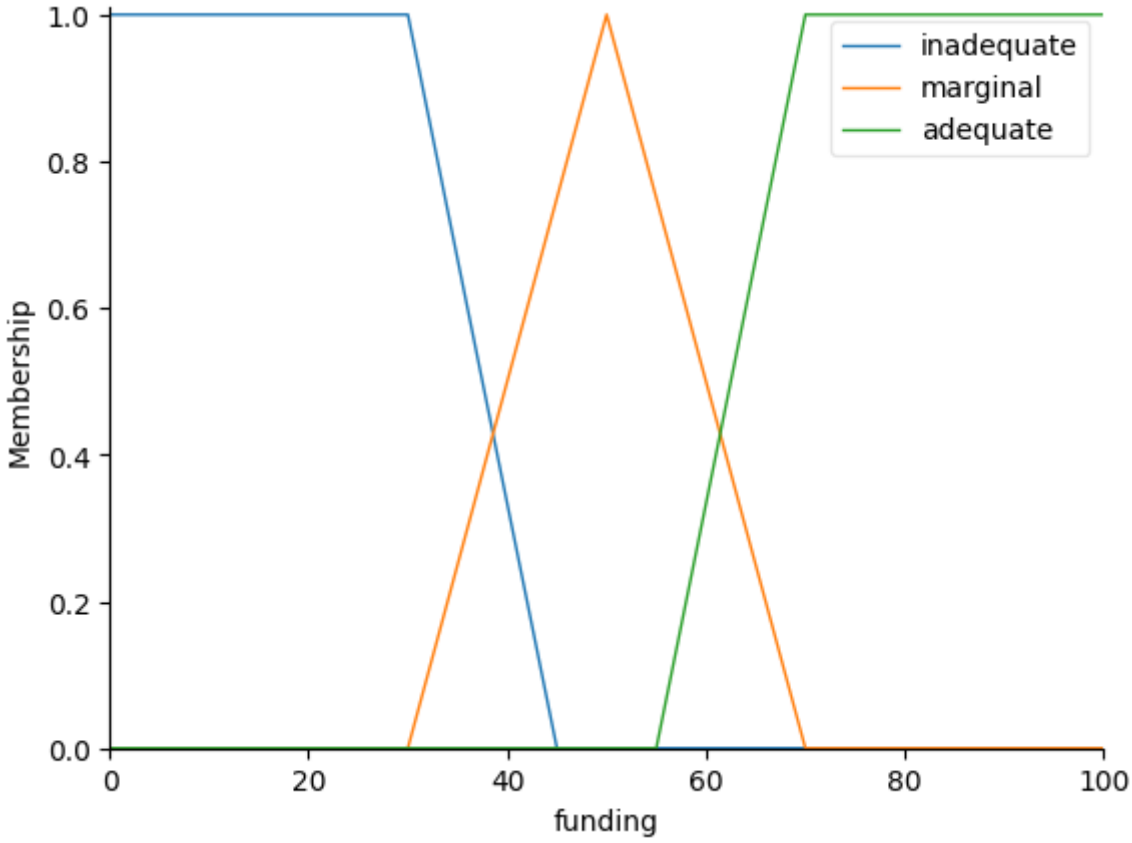
In [ ]:

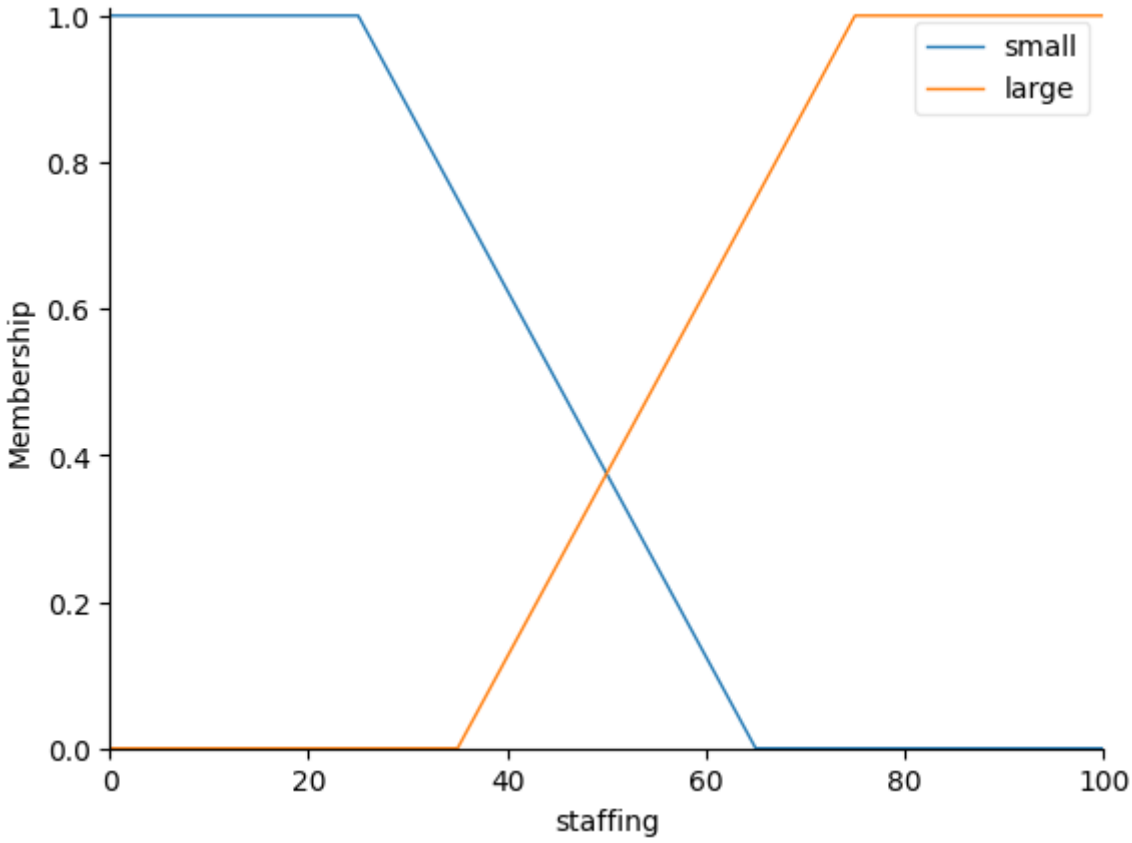
```
!pip install scikit-fuzzy
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

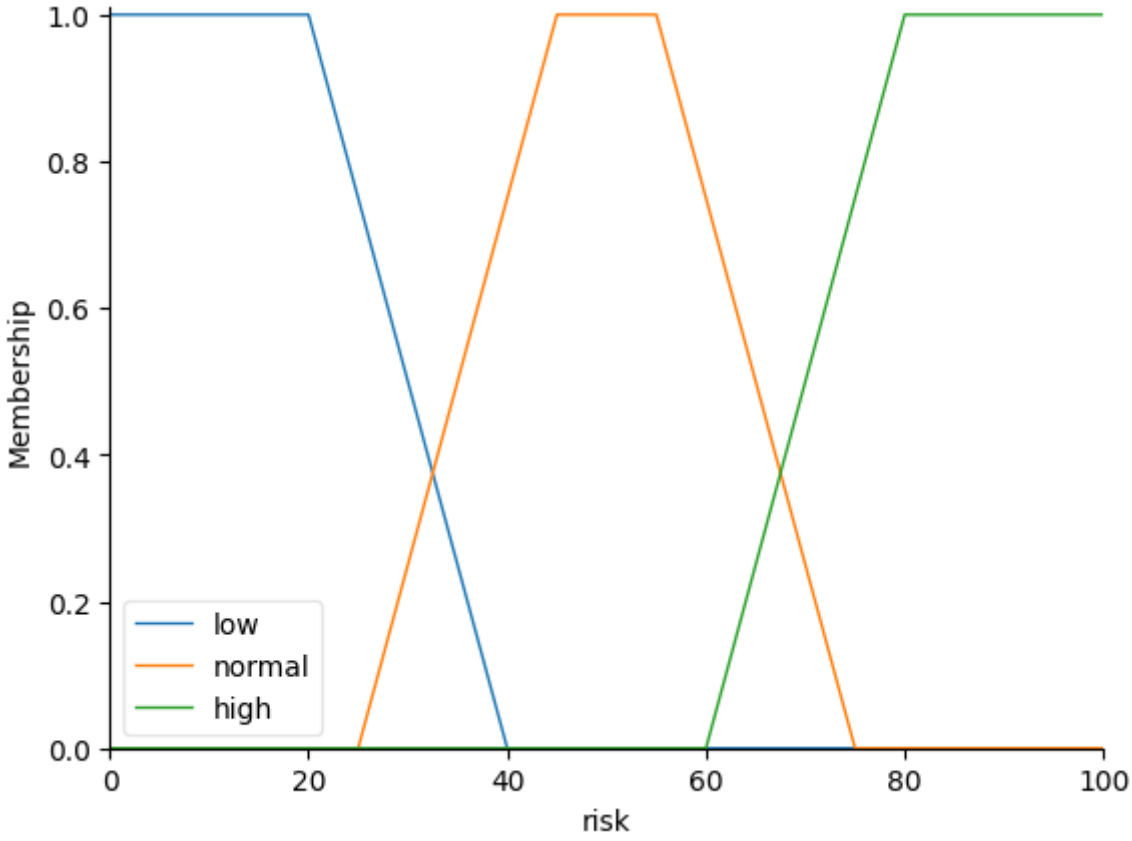
```
Collecting scikit-fuzzy
  Downloading scikit-fuzzy-0.4.2.tar.gz (993 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 994.0/994.0 kB 11.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.10/dist-package
s (from scikit-fuzzy) (1.23.5)
Requirement already satisfied: scipy>=0.9.0 in /usr/local/lib/python3.10/dist-package
s (from scikit-fuzzy) (1.11.3)
Requirement already satisfied: networkx>=1.9.0 in /usr/local/lib/python3.10/dist-pack
ages (from scikit-fuzzy) (3.2)
Building wheels for collected packages: scikit-fuzzy
  Building wheel for scikit-fuzzy (setup.py) ... done
  Created wheel for scikit-fuzzy: filename=scikit_fuzzy-0.4.2-py3-none-any.whl size=8
94079 sha256=9894b7437e7a542121d4207abf32f1b7fa2dfde0fc698619aa3df975c9634e4d
  Stored in directory: /root/.cache/pip/wheels/4f/86/1b/dfd97134a2c8313e519bcebd95d3f
edc7be7944db022094bc8
Successfully built scikit-fuzzy
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.4.2
```

In [ ]:

```
# Linguistic variables for antecedents/consequent
funding = ctrl.Antecedent(np.arange(0, 101, 1), 'funding')
staffing = ctrl.Antecedent(np.arange(0, 101, 1), 'staffing')
risk = ctrl.Consequent(np.arange(0, 101, 1), 'risk')

# membership functions for each linguistic values
funding['inadequate'] = fuzz.trapmf(funding.universe, [0, 0, 30, 45])
funding['marginal'] = fuzz.trimf(funding.universe, [30, 50, 70])
funding['adequate'] = fuzz.trapmf(funding.universe, [55, 70, 100, 100])
staffing['small'] = fuzz.trapmf(staffing.universe, [0, 0, 25, 65])
staffing['large'] = fuzz.trapmf(staffing.universe, [35, 75, 100, 100])
risk['low'] = fuzz.trapmf(risk.universe, [0, 0, 20, 40])
risk['normal'] = fuzz.trapmf(risk.universe, [25, 45, 55, 75])
risk['high'] = fuzz.trapmf(risk.universe, [60, 80, 100, 100])

funding.view()
staffing.view()
risk.view()
```


1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
20
40
60
80
100
funding
inadequate
marginal
adequate


1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
20
40
60
80
100
staffing
small
large



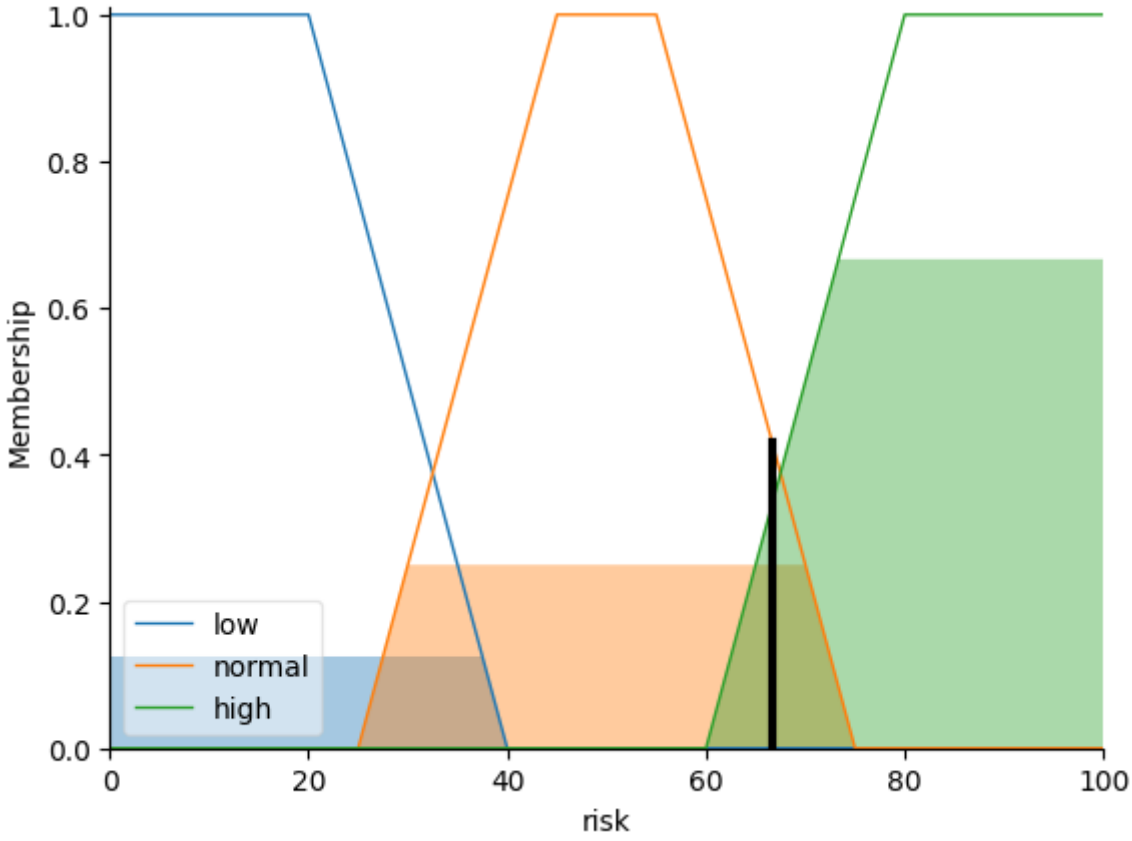
In [ ]:

```
# Define the rules
rule1 = ctrl.Rule(funding['adequate'] | staffing['small'], risk['low'])
rule2 = ctrl.Rule(funding['marginal'] & staffing['large'], risk['normal'])
rule3 = ctrl.Rule(funding['inadequate'], risk['high'])
rule4 = ctrl.Rule(funding['inadequate'] & staffing['large'], risk['high'])

# Create the control system and its simulation
#ctrl_sys = ctrl.ControlSystem([rule1, rule2, rule3])
ctrl_sys = ctrl.ControlSystem([rule1, rule2, rule3, rule4])
ctrl_sim = ctrl.ControlSystemSimulation(ctrl_sys)
```

In [ ]:

```
# Pass inputs to the ControlSystem
ctrl_sim.inputs({'funding': 35, 'staffing': 60})

# Crunch the numbers
ctrl_sim.compute()
print("Project risk:", ctrl_sim.output['risk'], "%")
risk.view(sim=ctrl_sim)
```

```
Project risk: 66.66529281135837 %
```



Discussions

To find a solution to this problem, again, the scikit-fuzzy package was used.

For this problem, when talking about the inputs, we used a funding rating of 35 and a staffing rating of 60. Therefore, we can say that we used a considerable "inadequate" range funding as well as a considerable "large" range staffing. The program outputted a value result of approximately 66.665%, given the previous inputted values.

In the next 3 figures above, we can see the membership functions for each linguistic value.

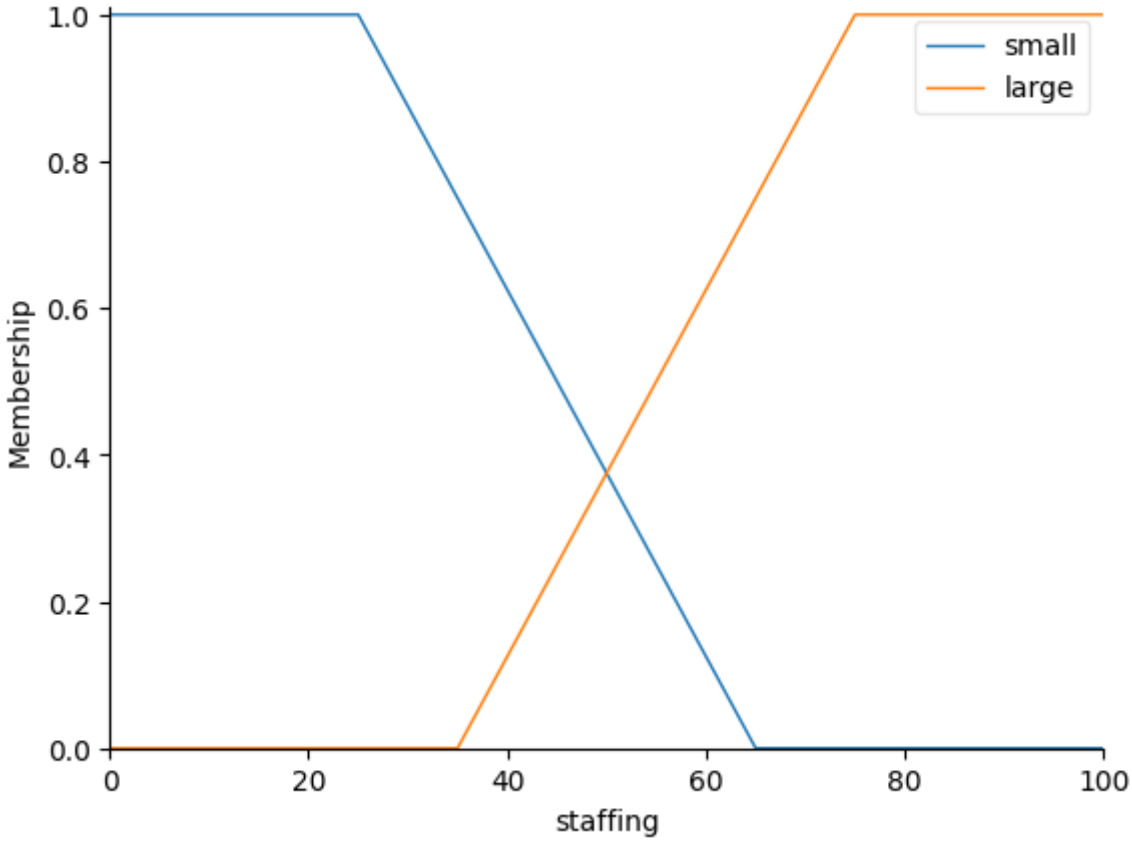


Figure 4

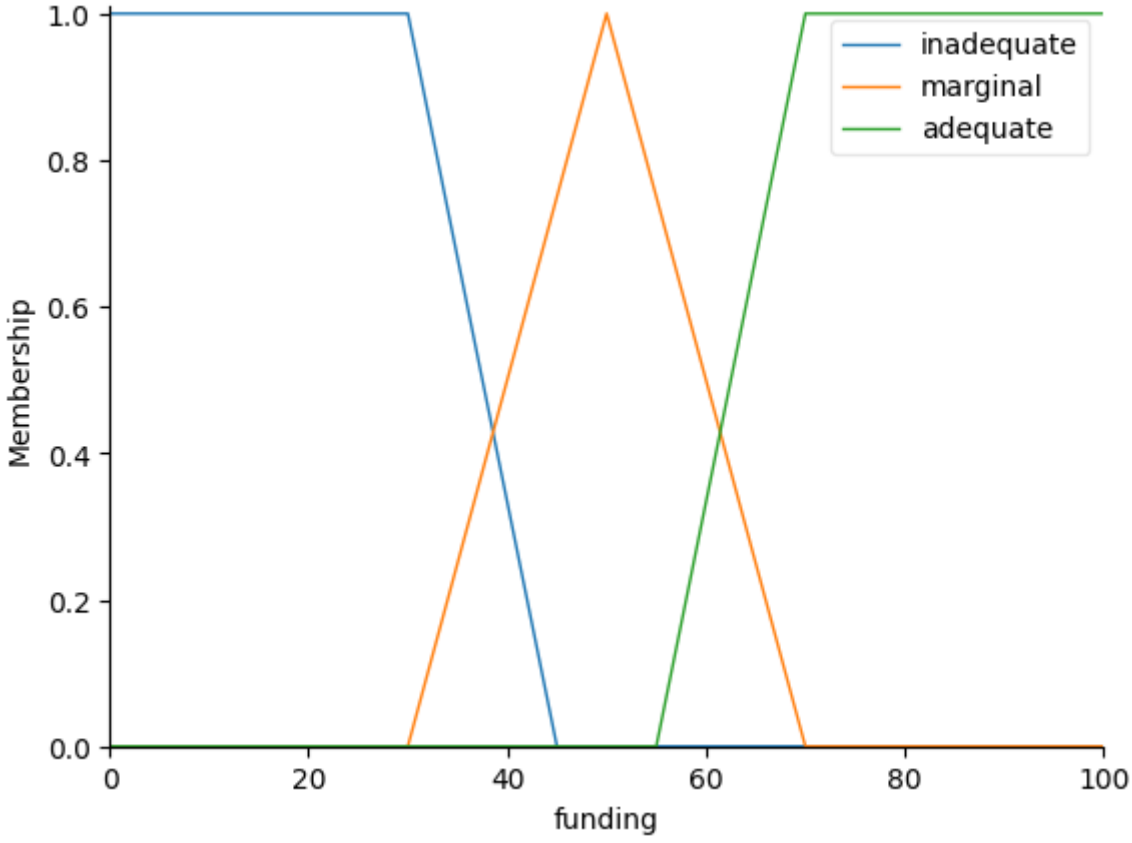


Figure 5

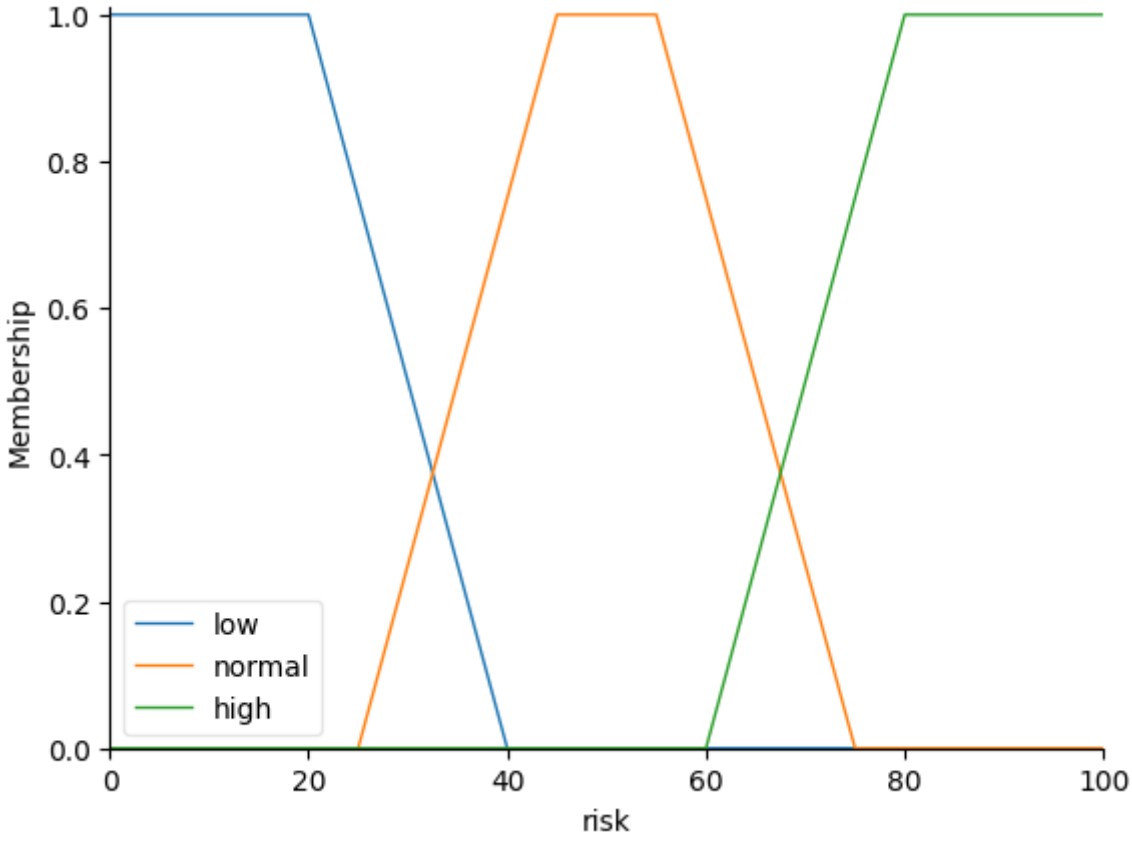


Figure 6

We can notice that the plots obtained are fairly similar to the ones we saw in the problem description or the ones we discussed in the lecture (Negnevitsky 2011), although they are not literally the same, as we can infer by our outputted result (of approximately 66.665%) which is different than the one we got in the lecture, which was approximately 67.4%, so it is indeed slightly different.

In the following figure, we can see a plot regarding the risk, and relative to the fuzzy system computed when running the program.

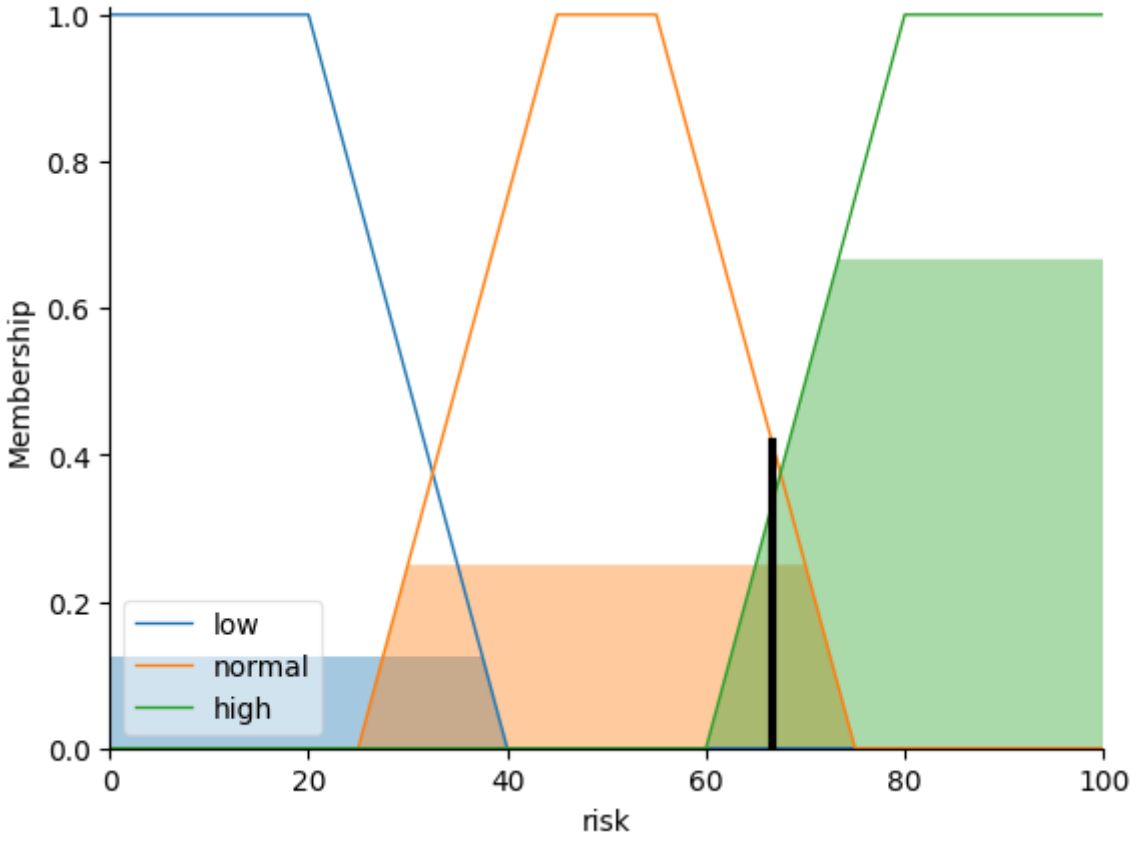


Figure 7

By analysing the results, we can conclude that the risk for this project is approximately 66.665%, this being a mostly "normal" to slightly "high" range risk.

Let's now study the results of a different scenario, where the rules of the problem are now different as we add a new rule to the existing ones, while everything else, such as linguistic variables, values, inputs, and outputs continues the same.

Let's see the final rule structure:

1. IF project_funding is adequate
   OR project_staffing is small
   THEN risk is low
2. IF project_funding is marginal
   AND project_staffing is large
   THEN risk is normal
3. IF project_funding is inadequate
   THEN risk is high
4. IF project funding is inadequate
   AND project staffing is large
   THEN risk is high

In the following figure, we can see a plot regarding the risk, and relative to the new fuzzy system computed when running the altered program.

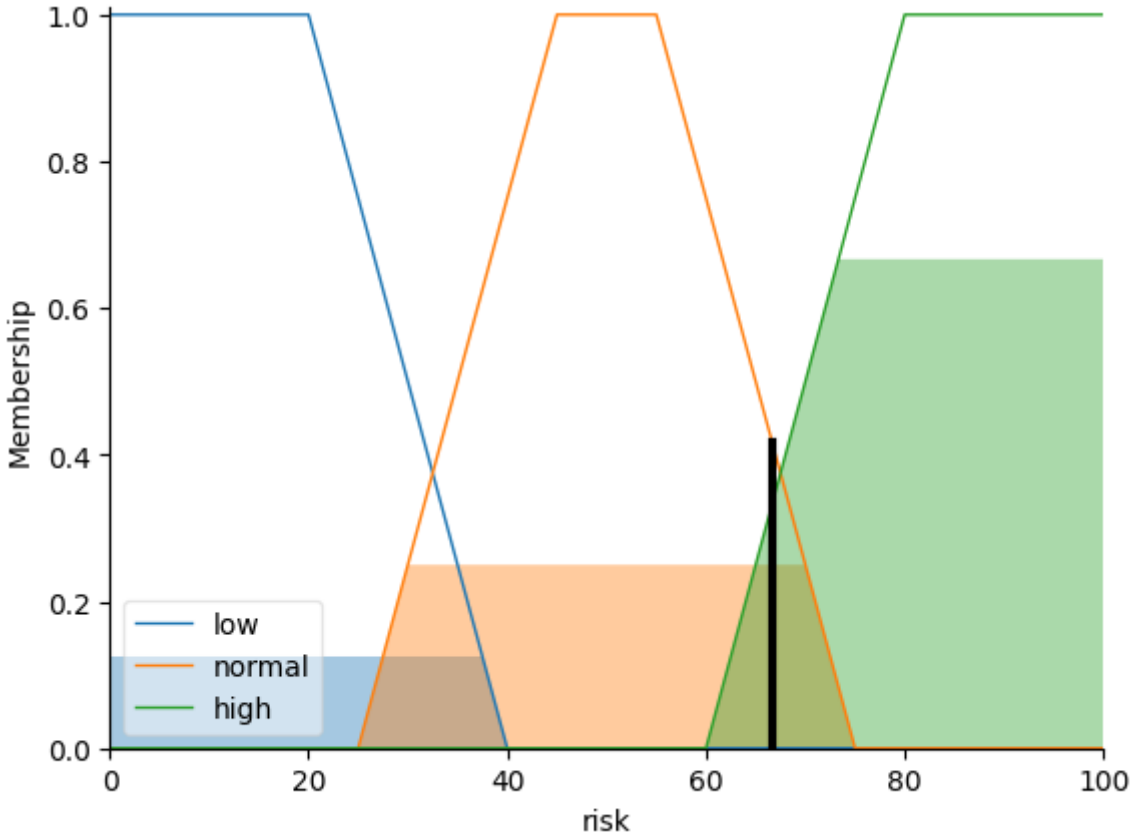


Figure 8

By analising the results, we can notice that these results are equal to the ones obtained before adding rule 4, as the risk calculated is still approximately 66.665% and the fuzzy system plot is also the same as before.

Although in the Mamdani fuzzy inference process when using certain input values each rule will contribute to the final output, in this case, the results revealed themselves as similar, this can be explained by looking at how the Mamdani fuzzy inference process works.

Firstly, in the "fuzzification" step, we take the inputs given and calculate their degree of membership to which they belong regarding the appropriate fuzzy sets.

Secondly, we will check the existing fuzzy Rules, for this particular case, we are going to focus on rules 3 and 4 (these are the same except for one more AND statement) as they both deal with the funding "inadequate" value. At the same time, in this "rule evaluation" for the given inputs, when processing the funding and the staffing values, the funding degree of membership for "Inadequate" is probably lower than the staffing degree of membership for "large", so in rule 4, as we have the AND statement, we will see that the resulting membership function is clipped, in other words, the correlation minimum method will happen and the top of the membership function is sliced, therefore, we will choose the minimum degree level out of these fuzzified inputs (the funding degree of membership for "Inadequate" in this case).

After this, we have the "aggregation", in which we will take all the resulting outputs and combine them into a single fuzzy set. That will be used to calculate the overall risk level.

Finally, in the "defuzzification" we will turn the final result into a crisp number, in other words, a single number.

So as rules 3 and 4 will give us the same result, both scenarios have the same results.

Task 5.3 Washing Machine Fuzzy Controller

Problem Description

This last task presents the washing machine fuzzy controller problem, which takes us to a more day-to-day life application, such as a fuzzy controller for washing machines. So for this problem, we will create a fuzzy controller for washing machines.

First, it is necessary to define the problem in terms of its linguistic variables. We will have 3 different linguistic variables:

1. The degree of dirtiness, which will have "SD(small)", "MD(medium)", "LD(large)" as linguistic values.
2. The size of load of clothes, which will have "SL(small)", "ML(medium)", "LL(large)" as linguistic values.
3. The washing time, which will have "VS(very short)", "S(short)", "M(medium)", "L(long)", "VL(very long)" as linguistic values.

Note that the degree of dirtiness and the size of a load of clothes will be the inputs, while the washing time will be the output.

The figure below shows us the memberships (and their behaviour) of the linguistic values previously mentioned, or in other words, the fuzzy sets:

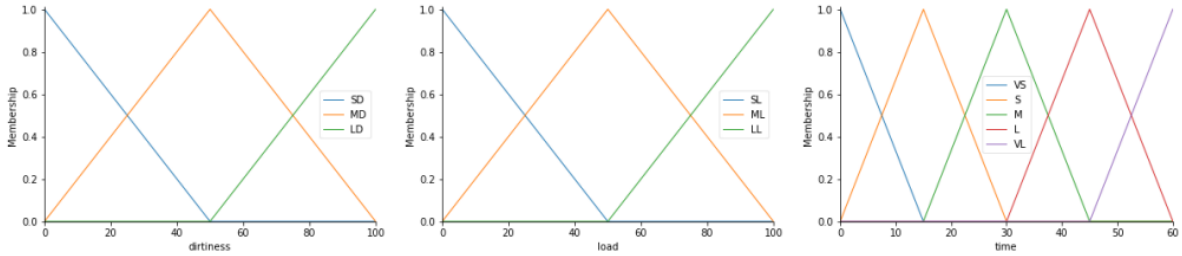


Figure 9

Let's now look at the following table and see the rules that we will use throughout this task:

Degree of dirtiness	Size of load of clothes: SL	ML	LL
SD	**VS**	**M**	**L**
MD	**S**	**M**	**L**
LD	**M**	**L**	**VL**

Fuzzy Associative Memory

table 1

The table above displays the rules in a Fuzzy Associative Memory (FAM). As an example of how this translates, the first rule of this FAM is defined as the following:

Rule: 1

> IF dirtiness is SD
> AND load is SL
> THEN washing time is VS

Implementation and results

This section shows the implemented code and its respective output.

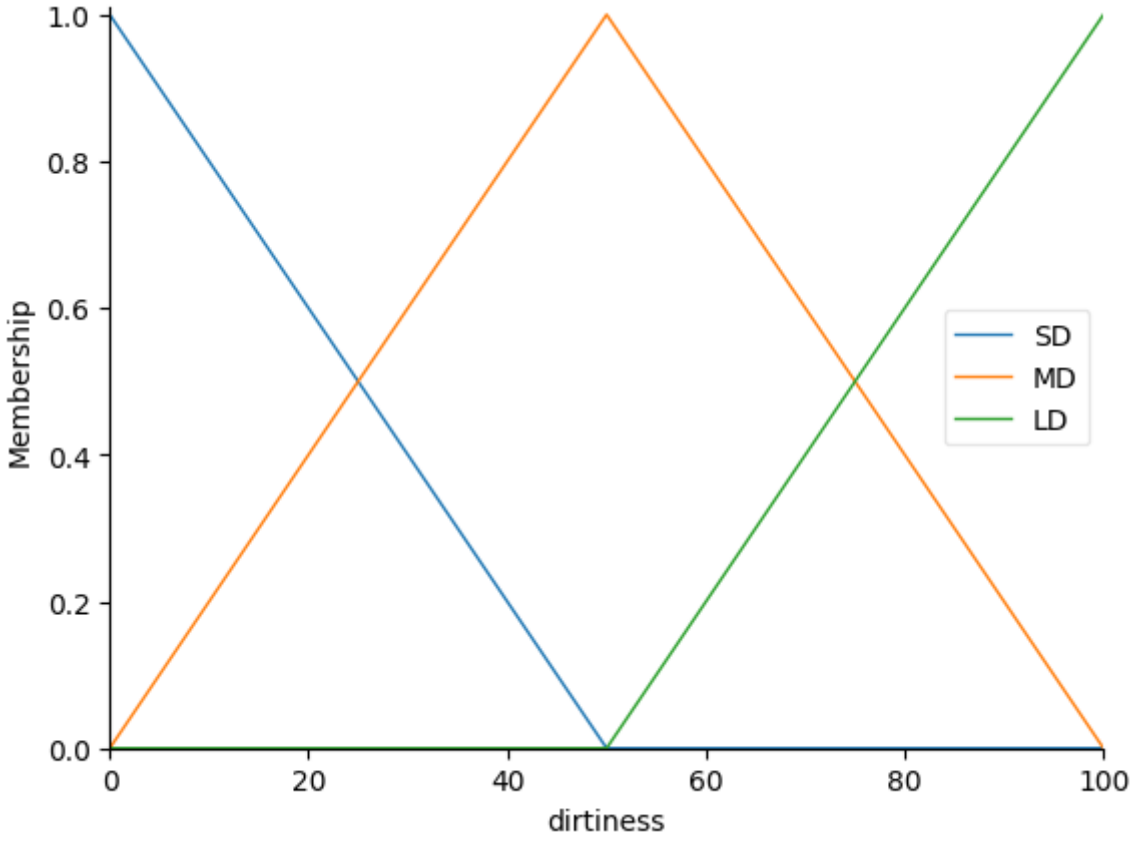
In [ ]:

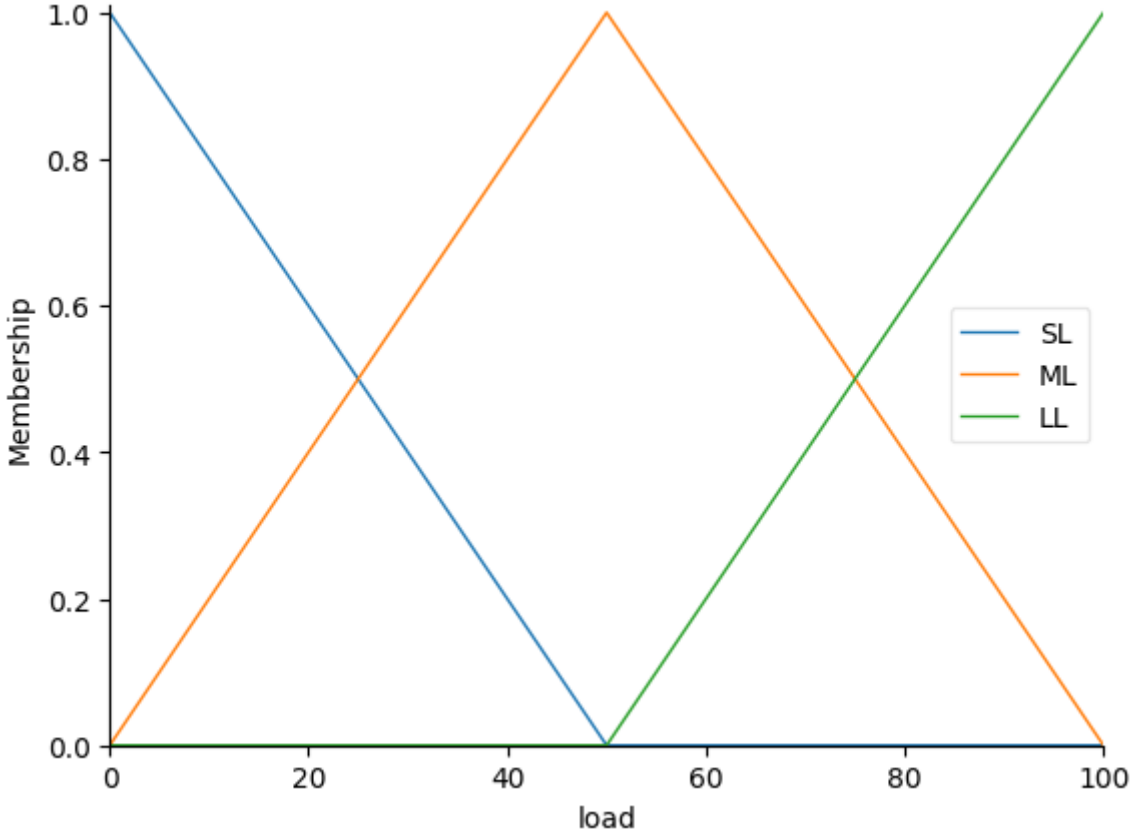
```
!pip install scikit-fuzzy
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

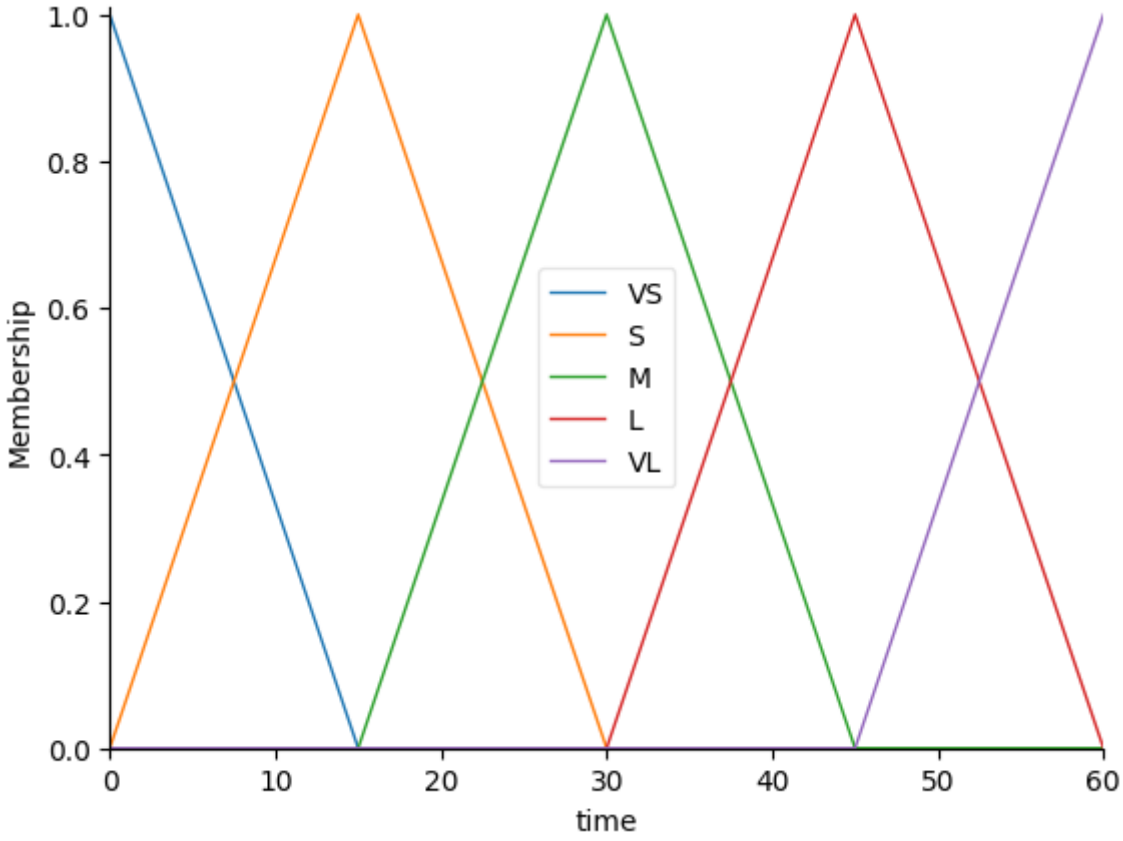
```
Collecting scikit-fuzzy
  Downloading scikit-fuzzy-0.4.2.tar.gz (993 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 994.0/994.0 kB 8.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.10/dist-package
s (from scikit-fuzzy) (1.23.5)
Requirement already satisfied: scipy>=0.9.0 in /usr/local/lib/python3.10/dist-package
s (from scikit-fuzzy) (1.11.3)
Requirement already satisfied: networkx>=1.9.0 in /usr/local/lib/python3.10/dist-pack
ages (from scikit-fuzzy) (3.2)
Building wheels for collected packages: scikit-fuzzy
  Building wheel for scikit-fuzzy (setup.py) ... done
  Created wheel for scikit-fuzzy: filename=scikit_fuzzy-0.4.2-py3-none-any.whl size=8
94079 sha256=5fa81c0d23ecef9bd88a503f1a473d465a9680949fc859a2f36b2e4e68d0f96b
  Stored in directory: /root/.cache/pip/wheels/4f/86/1b/dfd97134a2c8313e519bcebd95d3f
edc7be7944db022094bc8
Successfully built scikit-fuzzy
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.4.2
```

In [ ]:

```
# Linguistic variables for antecedents/consequent
dirtiness = ctrl.Antecedent(np.arange(0, 101, 1), 'dirtiness')
load = ctrl.Antecedent(np.arange(0, 101, 1), 'load')
time = ctrl.Consequent(np.arange(0, 61, 1), 'time')

# membership functions for each linguistic values
dirtiness.automf(3, names=['SD', 'MD', 'LD'])
load.automf(3, names=['SL', 'ML', 'LL'])
time.automf(5, names=['VS', 'S', 'M', 'L', 'VL'])

dirtiness.view()
load.view()
time.view()
```


1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
20
40
60
80
100
dirtiness
SD
MD
LD


1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
20
40
60
80
100
load
SL
ML
LL



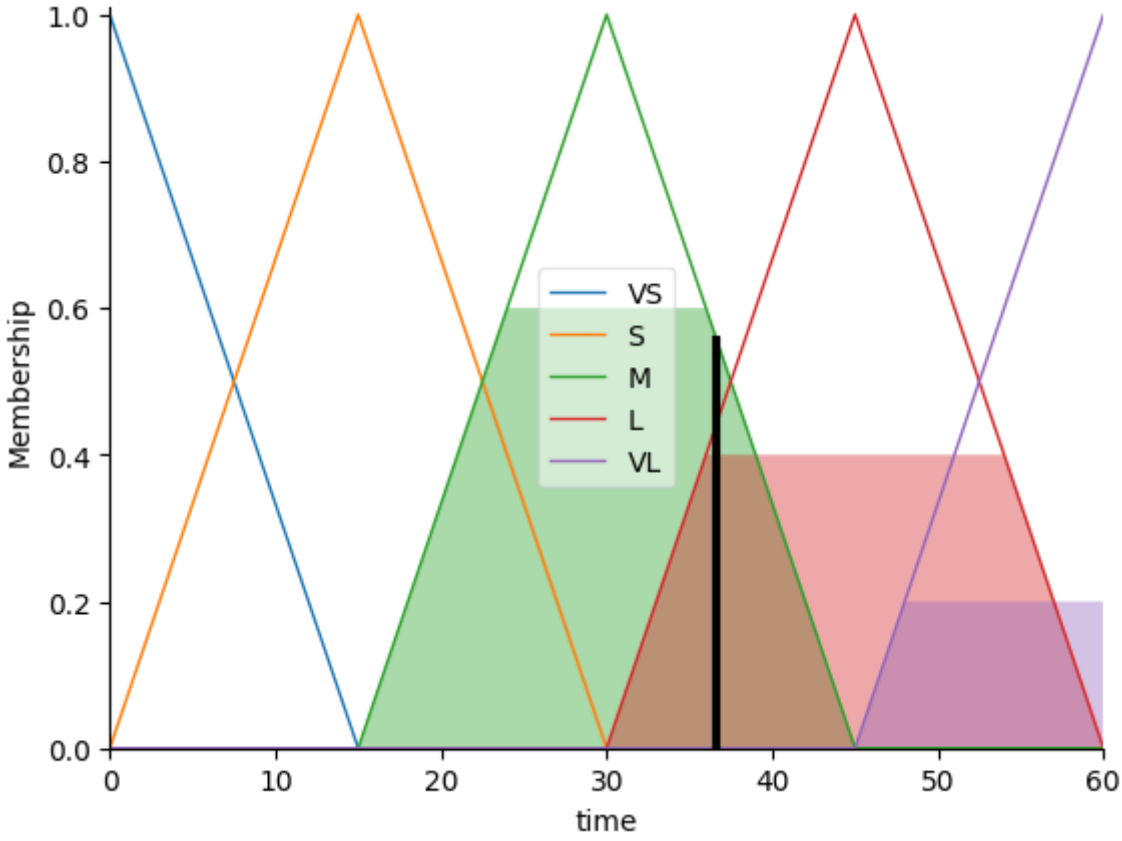
In [ ]:

```
# Define the rules
rule1 = ctrl.Rule(dirtiness['SD'] & load['SL'], time['VS'])
rule2 = ctrl.Rule(dirtiness['SD'] & load['ML'], time['M'])
rule3 = ctrl.Rule(dirtiness['SD'] & load['LL'], time['L'])
rule4 = ctrl.Rule(dirtiness['MD'] & load['SL'], time['S'])
rule5 = ctrl.Rule(dirtiness['MD'] & load['ML'], time['M'])
rule6 = ctrl.Rule(dirtiness['MD'] & load['LL'], time['L'])
rule7 = ctrl.Rule(dirtiness['LD'] & load['SL'], time['M'])
rule8 = ctrl.Rule(dirtiness['LD'] & load['ML'], time['L'])
rule9 = ctrl.Rule(dirtiness['LD'] & load['LL'], time['VL'])

# Create the control system and its simulation
ctrl_sys = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule
ctrl_sim = ctrl.ControlSystemSimulation(ctrl_sys)
```

In [ ]:

```
# Pass inputs to the ControlSystem
ctrl_sim.inputs({'dirtiness': 60, 'load': 70})

# Crunch the numbers
ctrl_sim.compute()
print("Washing time:", ctrl_sim.output['time'])
time.view(sim=ctrl_sim)
```

```
Washing time: 36.650793650793666
```



Discussions

To solve this problem, once again, the scikit-fuzzy package was used.

For this problem, regarding the inputs, we used a degree of dirtiness of 60% and a size of load of 70%. So we can say that we used a considerable (mostly) "MD(medium)" range degree of dirtiness as well as both "ML(medium)" and "LL(large)" range size of a load of clothes. Given these inputs, the program outputted a washing time value result of approximately 36.650.

In the next 3 figures above, we can see the membership functions for each linguistic value.

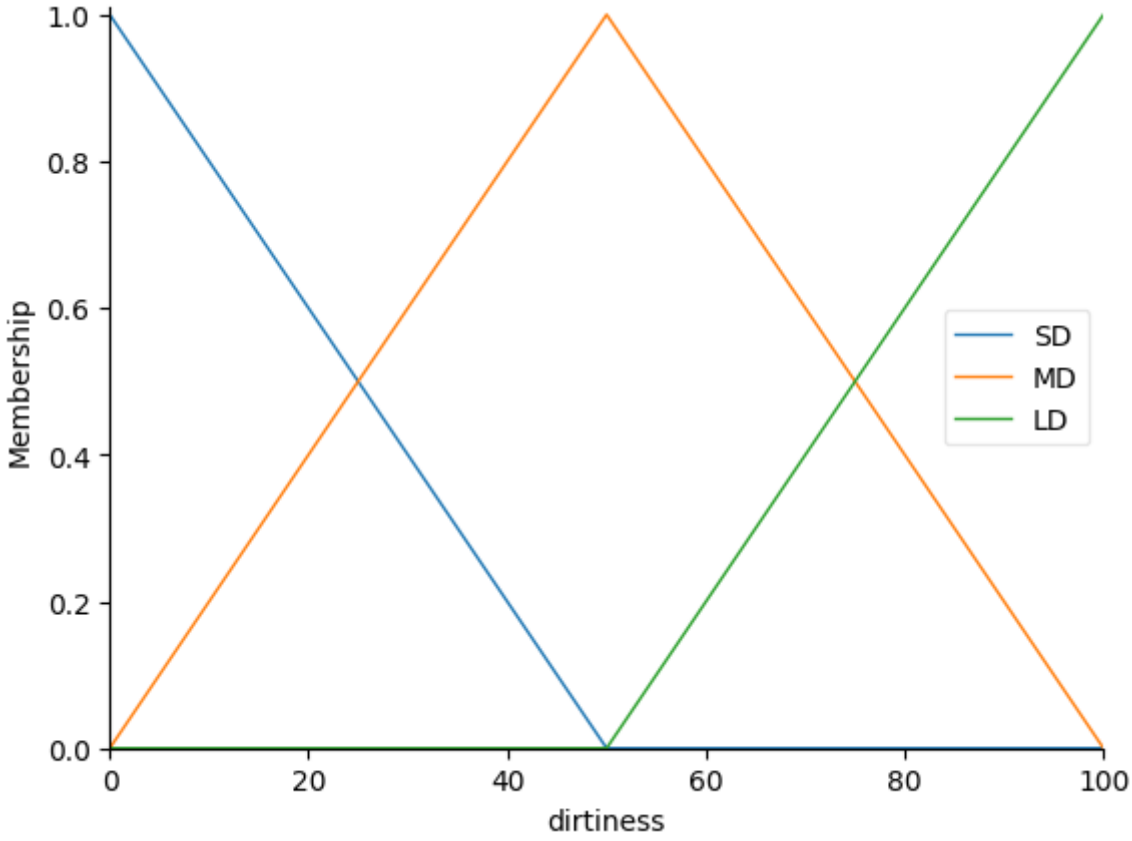


Figure 11

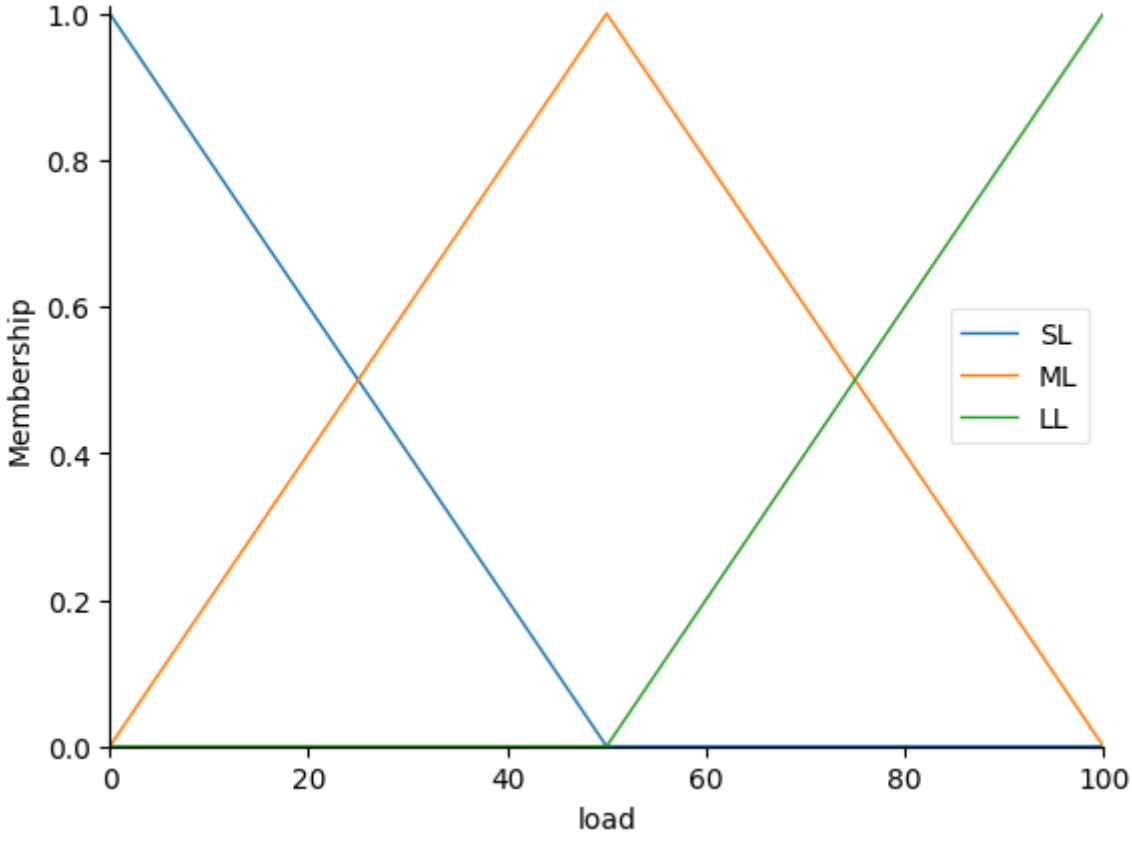


Figure 12

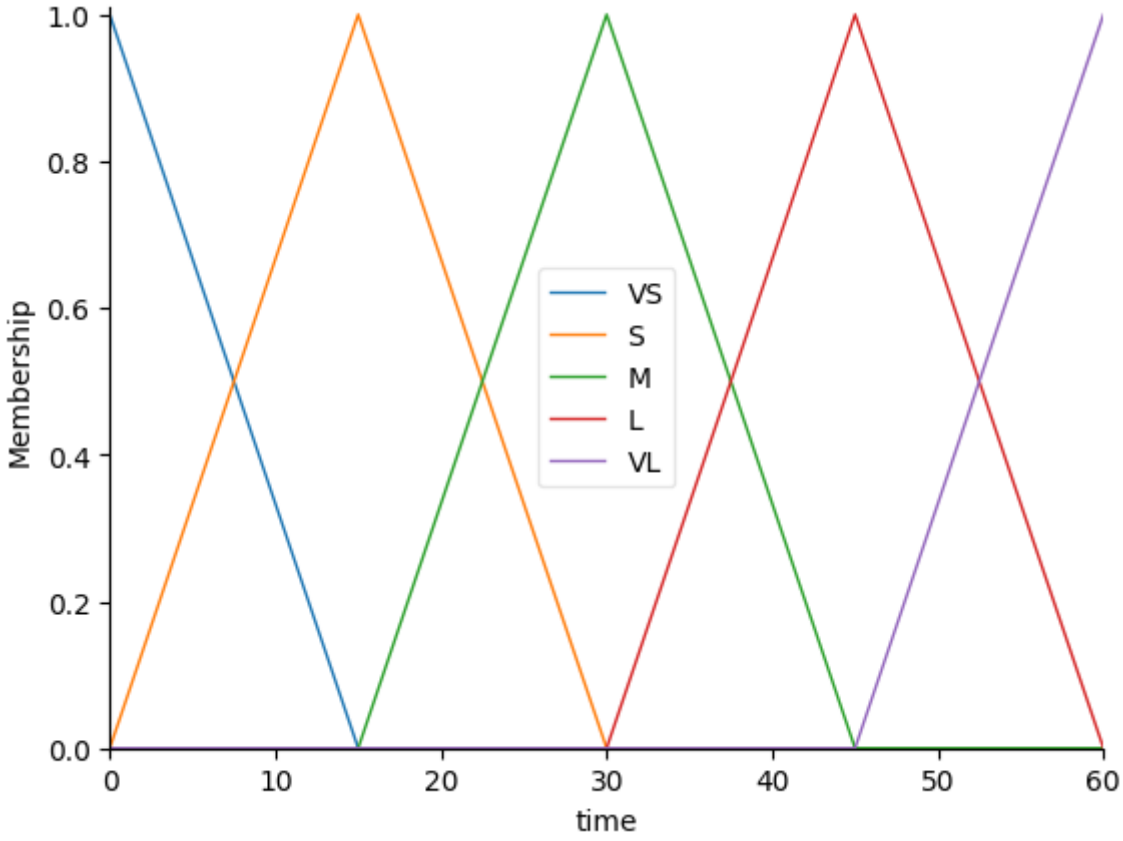


Figure 13

We can notice that the membership functions obtained are defined as expected (in a simple triangular shape).

In the following figure, we can see a plot regarding the washing time, and relative to the fuzzy system computed when running the program.

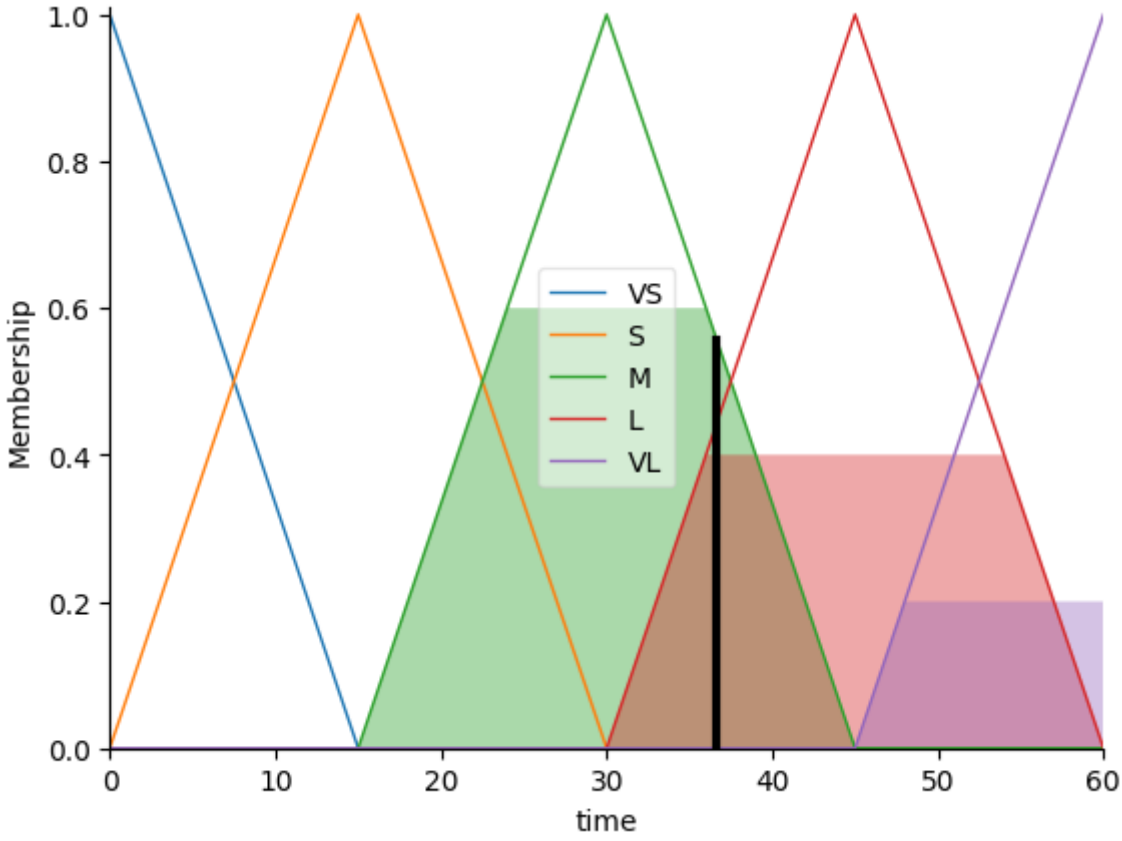


Figure 14

By analysing the results, we can conclude that the washing time for this project is approximately 36.650 when the degree of dirtiness is 60% and the size of load is 70%, this being a mostly "L(long)" and "VL(very long)" range washing time.

Conclusion

In conclusion, after completing this Lab work by solving its tasks and problems, we can agree that by using fuzzy systems, we were able to find a solution for these 3 tasks. By doing so, we also studied these 3 classic problems and therefore gained additional knowledge on linguistic variables, linguistic values, fuzzy membership functions, rules and how these are implemented in a program.

In further investigations, we could study improved and more complex decision-making models, for example, while also using some machine learning and deep learning approaches. On top of that, we could study more complex and nuanced scenarios, in order to generate and explore more precise and adaptable fuzzy systems. Furthermore, we could also try to apply this knowledge to real-world applications or other areas of investigation.

Lab 6. Natural Language Processing

This lab sheet aims to help us understand more about natural language processing, as we are going to study this new topic, and use it to solve 3 problems, which will be divided into the following tasks:

1. The sentiment analysis problem
2. The text classification problem
3. The topic modelling problem

Task 6.1 Sentiment Analysis

Problem Description

This first task presents us with a sentiment analysis problem, in which we will take movie reviews, and perform a sentiment analysis of these reviews. This task is also known as the “Hello World” problem for NLP (natural language processing). Despite its popularity, it is still difficult to perform this analysis, while getting almost flawless results and performances, as there are many "human attributes" to writting, such as sarcasm, slang, and other writing techniques or styles that make it hard for the computer to fully understand and process.

In order to solve this task, we will use a dataset called "movie_reviews" (from the nltk corpus), which contains 1000 positive reviews, as well as 1000 negative ones.

The following figure shows us an excerpt of a review, as an example of the type of language and writing used in the reviews presented in this dataset.

```
( note : there are spoilers regarding the film's climax ; the election , of
course )
we see matthew broderick , a man torn to a primal state ; he's been
unfaithful to his wife , lied to and manipulated his students , and by the
same token they've demeaned his masculinity , his self-respect , his
desperate attempt at changing the world .
and yet , he equates the cause of his pain , his torment , with tracy flick
( reese witherspoon ) .
no matter how many students have come and gone , and disappointed him as an
educator , she's the real threat .
about to give in , and divulge that she's won by only a lone vote ,
broderick's mccalister turns in defeat , sees tracy's euphoric celebration
in the outside corridor and says , 'no' .
the fact that he simultaneously lusts after her ideologically further
illustrates that freudian foundation of entitlement which all men , no
matter how obscure , have in their relationships ; a traditional
expectation of success , to usurp and surpass women as a proverbial
industry .
she can't go higher than him .
```

Figure 1

In our implemented solution for this problem, we will use the simple word count features, as we will count the occurrence of each word in the document, such as "tf" and "tf-idf", in which "tf" stands for term frequency, and "idf" for inverse document frequency. We can see how to calculate these features in the figure below taken from the lecture.

$$tf(t,d) = \frac{count\ of\ t\ in\ d}{total\ number\ of\ words\ in\ d}$$

$$idf(t,D) = log\frac{total\ number\ of\ documents\ in\ D}{number\ of\ documents\ containing\ t}$$

$$tfidf(t,d,D) = tf(t,d) * idf(t,D)$$

Figure 2

By using this technique, we are going to use one-hot vectors, which contain a binary representation of the words processed (for each word the vector representation is all 0s except for a 1 value), and then accumulate these one-hot vectors to create a new vector that has got all the frequencies of existing words in that document.

We will also use the Naïve Bayes method, as we used in a previous lab task, except in this task we will use the multinomial model, which uses a different model for probability distribution than the one previously used. This method will be necessary for text classification, in this case, to classify the reviews as positive or negative.

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:
```
!pip install nltk
import nltk
from nltk.corpus import movie_reviews

nltk.download('movie_reviews')
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.
1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from
nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (fro
m nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-pack
ages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from
nltk) (4.66.1)
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
```

Out[ ]:
```
True
```

In [ ]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn import metrics
from sklearn.model_selection import train_test_split
```

In [ ]:

```
# Get the positive and negative review IDs
fileids_pos = movie_reviews.fileids('pos')
fileids_neg = movie_reviews.fileids('neg')

# Load the reviews
raw_data = []
for i in range(len(fileids_pos)):
  raw_data.append(movie_reviews.raw(fileids_pos[i]))
for i in range(len(fileids_neg)):
  raw_data.append(movie_reviews.raw(fileids_neg[i]))

# The corresponding labels for the reviews, 0 for postive, 1 for negative
labels = [0] * len(fileids_pos) + [1] * len(fileids_neg)

# Split the training and testing set by 80-20%
X_train, X_test, Y_train, Y_test = train_test_split(raw_data, labels, test_size=0.2)
```

In [ ]:

```
# Calculate the tf-idf features from the training set
tfidf = TfidfVectorizer(use_idf=False)
tfidf_data = tfidf.fit_transform(X_train)
print(tfidf_data.shape)

# Train the naive Bayes model for prediction
classifier = MultinomialNB().fit(tfidf_data, Y_train)
```

```
(1600, 36359)
```

In [ ]:

```
# Performance on the testing set
testing_tfidf = tfidf.transform(X_test)
predictions = classifier.predict(testing_tfidf)
print(metrics.classification_report(Y_test, predictions, target_names=['pos', 'neg']
```

```
              precision    recall  f1-score   support

         pos       0.95      0.52      0.67       199
         neg       0.67      0.98      0.80       201

    accuracy                           0.75       400
   macro avg       0.81      0.75      0.73       400
weighted avg       0.81      0.75      0.73       400
```

In [ ]:

```
print(X_train[0])
print(testing_tfidf[0])
```

```
jerry springer has got nothing on " wild things . "
john mcnaughton's new thriller tackles more tawdry themes in less than two hours than
springer's notoriously sleazy talk show broadcasts in two weeks -- bisexuality , thre
esomes , poolside catfights , slutty rich bimbos , even redneck gator-wrestling , the
y're all part of the movie's raucous , complex storyline .
but even trash tv topicality can't drag " wild things " down -- this crazy campfest p
lays like something you'd find late-night on the usa network , only infinitely more p
alatable and with a solid ensemble cast .
despite a smattering of needless scenes ( most of them sexual in nature ) , there's w
icked fun to be had here .
 " wild things " would be a guilty pleasure , only there's no guilty feeling involved
in having a good time with it .
```

high school guidance counselor sam lombardo (matt dillon) is well-liked in the town of blue bay , especially by pretty , popular kelly van ryan (denise richards) , whose family name is among the florida yachting enclave's most financially prominent .
hoping to take her crush to a physical level , kelly seductively slinks into lombardo's house after washing his jeep for a fundraiser , but , the very next day , tearfully admits to her trollop mother (theresa russell) that she was raped .
before long , blue bay detectives ray duquette (kevin bacon) and gloria perez (daphne rubin-vega) are listening to similar allegations from kelly's rebel classmate suzie toller (neve campbell) .
lombardo , who maintains his innocence , hires neck brace-sporting , opportunistic lawyer ken bowden (bill murray) to defend him in court .
the previews give away the following revelations , so if you haven't seen any of the movie's spots on the television or in the theater , you might want to skip to the next paragraph .
while cross-examining suzie on the witness stand , bowden gets her to break down and admit that the alleged rapes never took place -- that kelly had concocted this entire scheme because she was angry that lombardo was sleeping with her mother and not with her .
to pay lombardo for the damages , kelly's mother breaks her daughter's trust fund and gives him $8 . 5 million .
but lombardo , kelly and suzie are actually all working together , and plan to take the money and run as fast as they can .
duquette and perez , however , begin to suspect that there's more afoot to the case than just false accusations .
if there's a major drawback to " wild things , " it's that it's oversexed to a fault .
the much-talked-about hotel room menage-a-trois between dillon , campbell and richards is a turn-off .
it's also cut short (sorry , guys) , and should have been cut shorter -- the movie grinds to a halt for pure titillation once too often .
what we don't see is far more effective than what we do .
another example of this is kevin bacon needlessly going the full monty in a shower scene .
er , no thanks .
also , bacon's duquette feels simultaneously underdeveloped and overwritten .
daphne rubin-vega , from broadway's " rent , " tries to compensate for a superfluous character .
theresa russell is just plain wooden .
and when , in the end , all is out in the open , ask yourself if certain scenes involving these three were really necessary .
but what keeps the movie from being throwaway junk is an engaging chain of surprises (some predictable , some not) that never seems to end .
" wild things " has more twists than a crate full of corkscrews , and most are so gleefully , over-the-top nasty that you can't help but be charmed by their absurd showmanship .
a great deal of amusement also comes from watching bill murray in a supporting part that appears to have been written for his sly comedic talent ; murray's a stitch , especially when pulling up beside the van ryan limo after winning lombardo's case and flipping them off .
and don't leave when the closing credits hit the screen , or you'll miss the film's best part -- four bonus flashbacks that smooth over plot holes while offering a few more tiny turns , plus a final scene that caps everything off with a great stunner of a bombshell .
speaking of bombshells , denise richards , who plays almost every scene in a blue bikini top , does the teen tease thing with a malicious allure that she was never allowed to flaunt in " starship troopers . "
matt dillon flexes his sleepy-voiced sex appeal , and pulls off personality changes with chameleonic precision .
neve campbell , lovely as ever except when sporting a blond wig , gives suzie a vengeful vulnerability that makes her the most interesting member of the conspiring trio .
re-edited and toned down a bit , the dynamics between these three actors could have carried the film to greater lengths .
but what we're given works well enough .
" wild things " is highly entertaining and , indeed , very wild .

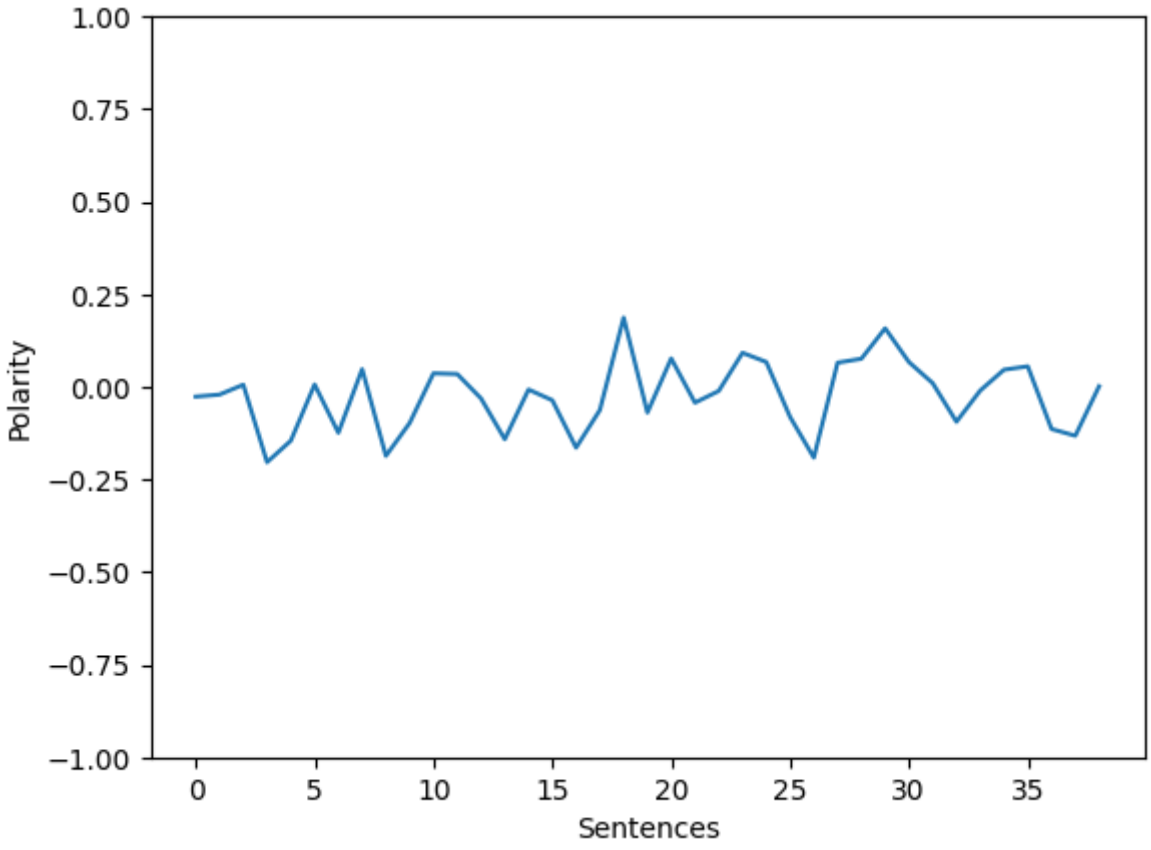
```
 (0, 198)      0.012845874888467699
 (0, 405)      0.012845874888467699
 (0, 650)      0.0899211242192739
 (0, 868)      0.025691749776935398
 (0, 869)      0.012845874888467699
 (0, 878)      0.012845874888467699
 (0, 904)      0.012845874888467699
 (0, 925)      0.012845874888467699
 (0, 975)      0.012845874888467699
 (0, 1117)     0.06422937444233849
 (0, 1129)     0.012845874888467699
 (0, 1334)     0.0770752493308062
 (0, 1386)     0.012845874888467699
 (0, 1389)     0.012845874888467699
 (0, 1404)     0.0385376246654031
 (0, 1421)     0.012845874888467699
 (0, 1531)     0.025691749776935398
 (0, 1562)     0.025691749776935398
 (0, 1606)     0.23122574799241857
 (0, 1666)     0.012845874888467699
 (0, 1792)     0.012845874888467699
 (0, 1981)     0.051383499553870796
 (0, 2062)     0.012845874888467699
 (0, 2143)     0.1413046237731447
 (0, 2211)     0.012845874888467699
 :     :
 (0, 35205)    0.025691749776935398
 (0, 35235)    0.012845874888467699
 (0, 35270)    0.012845874888467699
 (0, 35419)    0.025691749776935398
 (0, 35462)    0.025691749776935398
 (0, 35480)    0.0385376246654031
 (0, 35494)    0.012845874888467699
 (0, 35498)    0.012845874888467699
 (0, 35558)    0.051383499553870796
 (0, 35645)    0.0385376246654031
 (0, 35716)    0.012845874888467699
 (0, 35757)    0.012845874888467699
 (0, 35774)    0.051383499553870796
 (0, 35825)    0.012845874888467699
 (0, 35836)    0.025691749776935398
 (0, 35951)    0.012845874888467699
 (0, 35953)    0.012845874888467699
 (0, 36013)    0.012845874888467699
 (0, 36033)    0.012845874888467699
 (0, 36124)    0.025691749776935398
 (0, 36131)    0.012845874888467699
 (0, 36156)    0.012845874888467699
 (0, 36201)    0.1413046237731447
 (0, 36208)    0.025691749776935398
 (0, 36211)    0.012845874888467699
```

In [ ]:

```
# Evaluate the sentiment for each sentence in a review, and plot the variation of se
import matplotlib.pyplot as plt

sentences = X_test[0].split('.')

testing_tfidf = tfidf.transform(sentences)
predictions = classifier.predict_proba(testing_tfidf)
polarity = [x[0] - x[1] for x in predictions]
# polarity = [x[0] if (x[0] > x[1]) else -x[1] for x in predictions]

plt.xlabel('Sentences')
```

```
plt.ylabel('Polarity')
plt.plot(polarity)
plt.ylim(-1, 1)
```

Out[ ]: (-1.0, 1.0)



Discussions

To solve this problem, the sklearn or scikit-learn package was used, as it is a popular machine learning library and provides lots of important tools that were useful in this lab, such as the "tf-idf" feature and other probabilistic inference models, such as the Naïve Bayes model. Notice that this package was already used in a previous lab. The online documentation can be found at https://scikit-learn.org/stable/user_guide.html.

At the same time, the nltk package was used, which is widely used for natural language processing purposes. The online documentation can be found at https://www.nltk.org/, https://www.nltk.org/book/.

For this first case, we used the "tf-idf" feature, with a training set 80% of the data and a testing set of 20%.

The figure below presents (an example) the results outputted by the program, when using the naive Bayes model:

	precision	recall	f1-score	support
pos	0.91	0.61	0.73	212
neg	0.68	0.93	0.79	188
accuracy			0.76	400
macro avg	0.80	0.77	0.76	400
weighted avg	0.80	0.76	0.76	400

Figure 3

Notice that each run can give different results.

By analysing the previous output, we can see, once more, the metrics provided by the program for the two classes, "pos" and "neg", in which we can notice that 91% of the positive instances predicted were correct (as the precision value for this instance is 0.91), while 68% of the negative instances predicted were correct. Meanwhile, 61% of the positive instances were successfully identified (as the recall value for this instance is 0.61), while 93% of the negative instances were successfully identified. Moreover, we can see that the overall accuracy was 0.76, which means that 76% of the elements in the testing set were correctly classified (as positive or negative).

In the following figure, we can see a plot relative to the sentiment analysis of the first review in our testing set.

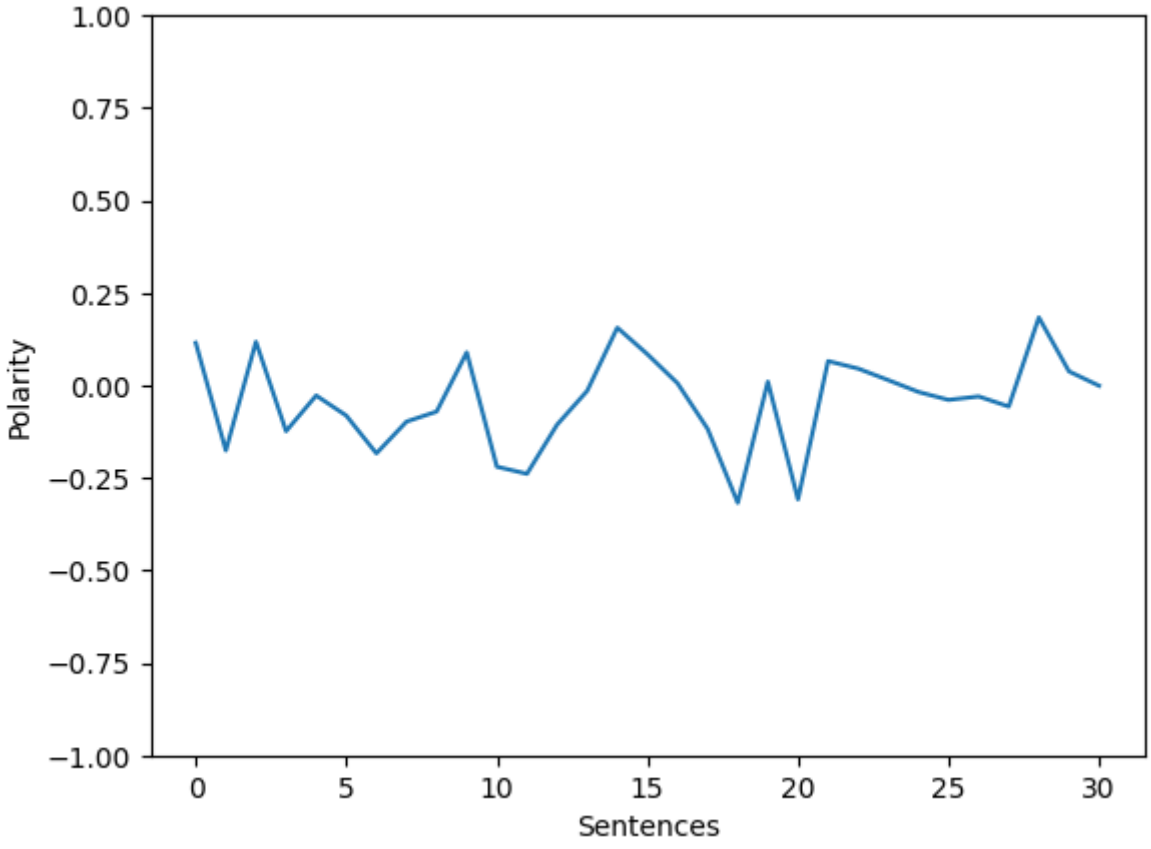


Figure 4

By looking at the plot, we can see the distributed data sample. Therefore this plot shows us how the sentiment value changed throughout the sentences, from the beginning to the end of the

review.

For this second case, we just used the "tf" feature, with a training set 80% of the data and a testing set of 20%.

The figure below presents us the results outputted by the program for this second scenario:

	precision	recall	f1-score	support
pos	0.95	0.52	0.67	199
neg	0.67	0.98	0.80	201
accuracy			0.75	400
macro avg	0.81	0.75	0.73	400
weighted avg	0.81	0.75	0.73	400

Figure 5

By analysing the previous output, we can see the metrics provided by the program for the two classes, "pos" and "neg", in which we can notice that 95% of the positive instances predicted were correct (as the precision value for this instance is 0.95), while 67% of the negative instances predicted were correct. Meanwhile, 52% of the positive instances were successfully identified (as the recall value for this instance is 0.52), while 98% of the negative instances were successfully identified. Moreover, we can see that the overall accuracy was 0.75, which means that 75% of the elements in the testing set were correctly classified (as positive or negative).

In the following figure, we can see a plot relative to the sentiment analysis of the first review in our testing set, while considering the change of the feature.

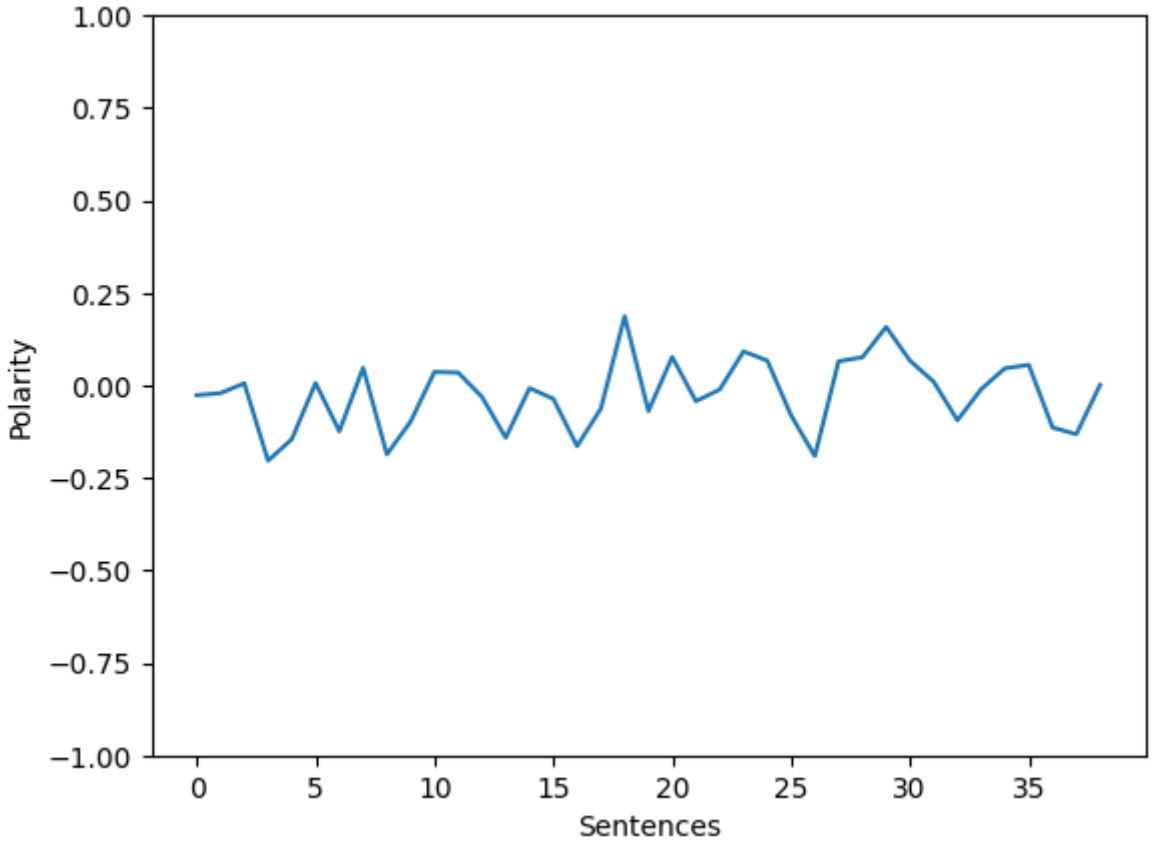


Figure 6

By analysing these results and the different scenarios studied, we can notice that even though the results were acceptable, there were almost no differences in these outputs when opting for the "tf" or "tf-idf" feature. Some possible reasons for this similarity in results are, for example, the small and common vocabulary in these reviews, or even the possible high similarity between most reviews in this dataset.

Task 6.2 Text Classification

Problem Description

This next task presents a text classification problem, in which we will perform a multi-class classification, instead of only using two classes as in the previous task.

To solve this problem, we will use a dataset called "newsgroups" (from sklearn), which is a collection of messages from 20 new groups, such as comp.graphics, rec.sport.hockey, sci.space, talk.religion.misc, and others.

The following figure shows us an excerpt of a message, as an example of the type of language and writing used in the messages presented in this dataset from the comp.graphics group.

From: rboykin@cscsparc.larc.nasa.gov (Rick Boykin)

Subject: Lookin Form 3-D model of Loom

Organization: NASA Langley Research Center, Hampton, VA USA

Lines: 25 Distribution: world

NNTP-Posting-Host: cscsparc.larc.nasa.gov

Keywords: 3-D Loom Model

Hi folks,

I'm doing an animated film on new methodes in loom research (You know, the thing they make cloth with.) and need a model of a loom. The format should be in ascii faceted geometry and fairly straight forward to figure out. Any help or pointers would be greatly appreciated.

-Thanks

Rick Boykin

Figure 7

In our implemented solution for this task, once again, we will use the simple word count features, such as "tf" and "tf-idf". We are also going to use the same classification method as the one used in the previous task, the Naïve Bayes model method.

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [ ]:
```
# Get the training dataset for the specified categoires
categories = ['rec.sport.hockey', 'talk.religion.misc',
              'comp.graphics', 'sci.space']
training_data = fetch_20newsgroups(subset='train', categories=categories)
```

In [ ]:
```
# Create the tf-idf transformer
#tfidf = TfidfVectorizer(use_idf=True)
tfidf = TfidfVectorizer(use_idf=False)
training_tfidf = tfidf.fit_transform(training_data.data)
print(training_tfidf.shape)

# Train a Multinomial Naive Bayes classifier
classifier = MultinomialNB().fit(training_tfidf, training_data.target)
```

```
(2154, 35956)
```

In [ ]:
```
from sklearn import metrics

testing_data = fetch_20newsgroups(subset='test', categories=categories)
testing_tfidf = tfidf.transform(testing_data.data)
predictions = classifier.predict(testing_tfidf)
print(metrics.classification_report(testing_data.target, predictions, target_names=c
```

```
                    precision    recall  f1-score   support

  rec.sport.hockey       0.96      0.88      0.92       389
talk.religion.misc       0.88      0.99      0.93       399
     comp.graphics       0.65      0.97      0.78       394
         sci.space       1.00      0.12      0.22       251

          accuracy                           0.81      1433
         macro avg       0.87      0.74      0.71      1433
      weighted avg       0.86      0.81      0.76      1433
```

In [ ]:
```
errors = [i for i in range(len(predictions)) if predictions[i] != testing_data.targe

for i, post_id in enumerate(errors[:5]):
  print("----------------------------------------------------------------------")
  print("%s --> %s\n" %(testing_data.target_names[testing_data.target[post_id]],
                        testing_data.target_names[predictions[post_id]]))
  print(testing_data.data[post_id])
```

```
----------------------------------------------------------------------
comp.graphics --> sci.space

From: robert@slipknot.rain.com (Robert Reed)
Subject: Re: ACM SIGGRAPH (and ACM in general)
Reply-To: Robert Reed <robert@slipknot.rain.com>
Organization: Home Animation Ltd.
Lines: 50

In article <1993Apr29.023508.11556@koko.csustan.edu> rsc@altair.csustan.edu (Steve Cu
nningham) writes:
|
```

|And no, SIGGRAPH 93 has not skipped town -- we're preparing the best
|SIGGRAPH conference yet!

Speaking of SIGGRAPH, I just went through the ordeal of my annual registration for SIGGRAPH and re-upping of membership in the ACM last night, and was I ever grossed out! The new prices for membership are almost highway robbery!

For example:

> SIGGRAPH basic fee went from $26 last year to $59 this year for the same thing, a 127% increase. Those facile enough to arrange a trip to the annual conference could reduce this to $27 by selecting SIGGRAPH Lite, which means SIGGRAPH is charging an additional $32 (or so) for the proceedings and the art show catalog, essentially.
>
> TOPLAS went up 40% in cost, way outstripping the current inflation rate.
>
> Basic SIGCHI fees remainded the same, but whereas before SIGCHI membership included UIST and Human Factors conferences proceedings, these are now an extra cost option. Bundling that back into the basic rate, equivalent services have gone up 100% in cost.
>
> SIGOIS membership cost has up 33%, but they've also split out the Computer Supported Cooperative Work conference proceedings that used to be included with membership. Adding that cost back in means this SIG also has doubled its membership fee.

What really galls me is that the ACM sent out brochures a couple months ago touting their new approach to providing member services, and tried to make it sound like they were offering NEW services. But with the exception of a couple, like SIGGRAPH, all the "plus" services appear to be just splitting the costs into smaller piles so that they don't look so big. But their recommended changes to my membership would have me paying 90% more than last year for a 31% increase in services (measured by cost, not by value), and, curiously, a 31% inflation rate on the publications I got last year.

Is anyone out there as galled by this extortion as I am?

Robert Reed Home Animation Ltd. 503-656-8414
robert@slipknot.rain.com 5686 First Court, West Linn, OR 97068

SHOOTING YOURSELF IN THE FOOT IN VARIOUS LANGUAGES AND SYSTEMS

Motif: You spend days writing a UIL description of your foot, the
  trajectory, the bullet, and the intricate scrollwork on the ivory handles
  of the gun. When you finally get around to pulling the trigger, the gun
  jams.

---

talk.religion.misc --> sci.space

From: markbr%radian@natinst.com (mark)
Subject: Re: RFD: misc.taoism
Nntp-Posting-Host: zippy.radian.com
Organization: n.o.y.b
Lines: 16

In article <1993Apr22.004331.22548@coe.montana.edu> uphrrmk@gemini.oscs.montana.edu (Jack Coyote) writes:
>Sunlight shining off of the ocean.
>
The universe, mirrored in a puddle.
>

>Aleph null bottles of beer on the wall, Aleph null bottles of beer!
>Take one down, pass it around ... Aleph null bottles of beer on the wall!
>
Isn't it amazing how there *always* seems to be *another* bottle of bheer there?

Aleph *one* bottles of beer on the wall, Aleph *one* null bottles of beer!

        you, too, are a puddle.
        As above, so below.

        mark

---
rec.sport.hockey --> sci.space

From: catcher@netcom.com (Paul W. Francisco)
Subject: Re: NHL LETTER (***QUITE LONG***)
Organization: Night People
Lines: 56

In article <1993Apr26.002033.22710@ramsey.cs.laurentian.ca> maynard@ramsey.cs.laurent
ian.ca (Roger Maynard) writes:
>In <1993Apr25.224654.23092@sol.ctr.columbia.edu> phoenix@startide.ctr.columbia.edu
(Ali Lemer) writes:
>>I'm not even going to bother to reply to your ranting post; the letter is
>>addressed to Gary Bettman, not you. He can reply as he wishes.
>
>Not so. Your post is addressed to all. I quote:

As she said, the letter is addressed to Bettman. The post (which, though
having pretty much the same content, is an entirely different entity) was
addressed to all. When she puts the letter in the mail I doubt it will
say "To: All". I figure she wanted to let people here see what was in it
since it is a topic that interests a lot of folks here. That's an entirely
different purpose than sending the letter to Bettman.

Unfortunately...

>And I guess I let you know didn't I?

>You do NOT represent a *large* body of hockey fans by any standard you care
>to use to define "large". 65 people constitutes nothing more than a tiny
>group of fanatics and is in no way representative of "a large body of hockey
>fans on the Internet" let alone of "one of the largest computer networks in
>the world."

The "largest computer networks in the world" phrase is a definition of
the Internet, not a group as a whole that she claims to represent. As
for the business of whether or not it is large, it is large compared
to say, the number of folks on r.s.h. who are sending a letter to thank
him for changing the names, at least to this point. And just for my
own curiosity I thought I'd look up the "official" definition of large
in the dictionary. It reads:

large - 1. having more than usual power, capacity, or scope.
        2. exceeding most other things of like kind in quantity or size.

Now I have no idea how many letters Bettman may have gotten on the issue
or how many people may have signed them. 65 people may be up there,
thereby validating definition 2. I would also wager that the geographical
range of signatures is quite large, which would give it a large scope.

>He might be impressed by the size of the list of names.

Why would he be impressed with this unless it were "large"?

I am of course assuming that you actually agree with what you are writing
and are not simply trying to be a pain about the whole issue to hear
yourself speak.

I personally don't know whether or not I agree with the letter. I have
very mixed emotions about it. I like the names as they are, and don't
think they make it that difficult to learn the game, but there might be
a shred of validity to the change.

--
Paul W. Francisco   In the shadow an angel cries...
catcher@netcom.com   - Front Line Assembly

----
talk.religion.misc --> sci.space

From: pboxrud@magnus.acs.ohio-state.edu (Paul D Boxrud)
Subject: Religion and marriage
Nntp-Posting-Host: top.magnus.acs.ohio-state.edu
Organization: The Ohio State University
Distribution: na
Lines: 20

     I wasn't sure if this was the right newsgroup to post this to, but I guess
the misc is there for a reason. Here goes... I am getting married in June to
a devout (Wisconsin Synod) Lutheran. I would classify myself as a strong
agnostic/weak athiest. This has been a a subject of many discussions between
us and is really our only real obstacle. We don't have any real difficulties
with the religious differences yet, but I expect they will pop up when we have
children. I have agreed to raise the
children "nominally" Lutheran. That is, Lutheran traditions, but trying to
keep an open mind. I am not sure if this is even possible though. I feel that
that the worst quality of being devoutly religous is the lack of an open mind.

     Anyway, I guess I'll get on with my question. Is anyone in the same
situation and can give some suggestions as to how to deal with this? We've
taken the attitude so far of just talking about it a lot and not letting
anything get bottled up inside. Sometimes I get the feeling we're making this
much bigger than it actually is. Any comments would be greatly appreciated.
Also, please e-mail responses since I don't get a chance to read this group
often. :-(

Paul

----
talk.religion.misc --> sci.space

From: b645zaw@utarlg.uta.edu (stephen)
Subject: Re: Biblical Backing of Koresh's 3-02 Tape (Cites enclosed)
News-Software: VAX/VMS VNEWS 1.41
Nntp-Posting-Host: utarlg.uta.edu
Organization: The University of Texas at Arlington
Lines: 33

In article <1993Apr21.154750.24341@maths.tcd.ie>,
pmoloney@maths.tcd.ie (Paul Moloney) writes...

>cotera@woods.ulowell.edu (Ray Cote) writes:
>> David Thibedeau (sp?), one of the cult members, said that the fire
>>was started when one of the tanks spraying the tear gas into the
>>facilities knocked over a lantern.
>

```
>In two places at once? Bit of a coincidence, that.

Never lived out in the country I see. 4 years ago I had a place
where I had to carry in propane every month, hook the bottle up
to copper line, to supply both the stove, and a type of water-
heater called a flash-heater. A flash heater has a pilot lamp.

Here's the point. If the Davidians had their propane tanks hooked
up to copper (or some such) lines, run through the ceiling spaces
-- when the FBI started wrecking the place, they could easily have
ruptured the lines. Which then would start spreading out through
the overhead. And since it was a country home, it wasn't necessarily
built with non-flamable insulation.

It's probably more plausible than anything else, that the fire started
mainly as a result of accident -- or willful negligence on the part of
the FBI, which should have known better (ie. manslaughter).

It's certain that if the tanks hadn't been used that day -- the fire
wouldn't have started.

>Whatever the faults the FBI had, the fact is that responsibility
>for those deaths lies with Koresh.

Paul, what "fact?"
```

Discussions

To find a solution to this problem, again, the sklearn or scikit-learn package was used.

For this first case, we used the "tf-idf" feature, and we used 4 different categories, such as "rec.sport.hockey", "talk.religion.misc", "comp.graphics", and "sci.space".

The figure below presents us the results outputted by the program, when using the naive Bayes model:

	precision	recall	f1-score	support
rec.sport.hockey	0.97	0.90	0.93	389
talk.religion.misc	0.93	0.99	0.96	399
comp.graphics	0.83	0.98	0.90	394
sci.space	1.00	0.72	0.84	251
accuracy			0.92	1433
macro avg	0.93	0.90	0.91	1433
weighted avg	0.93	0.92	0.92	1433

Figure 8

By analysing the previous results, we can see the metrics provided by the program for the four classes, in which we can notice that 97% of the "rec.sport.hockey" instances predicted were correct, 93% of the "talk.religion.misc" instances predicted were correct, 83% of the "comp.graphics" instances predicted were correct, and 100% of the "sci.space" instances predicted were correct. At the same time, 90% of the "rec.sport.hockey" instances were

successfully identified, 99% of the "talk.religion.misc" instances were successfully identified, 98% of the "comp.graphics" instances were successfully identified, and 72% of the "sci.space" instances were successfully identified. Furthermore, we notice that the overall accuracy was 0.92, which means that 92% of the elements in the testing set were correctly classified.

For this second case, we just used the "tf" feature, while considering the same 4 categories, such as "rec.sport.hockey", "talk.religion.misc", "comp.graphics", and "sci.space".

The figure below presents us the results outputted by the program, when using the naive Bayes model, while considering the change of the feature:

	precision	recall	f1-score	support
rec.sport.hockey	0.96	0.88	0.92	389
talk.religion.misc	0.88	0.99	0.93	399
comp.graphics	0.65	0.97	0.78	394
sci.space	1.00	0.12	0.22	251
accuracy			0.81	1433
macro avg	0.87	0.74	0.71	1433
weighted avg	0.86	0.81	0.76	1433

Figure 9

By analysing the results above, we can see, once more, the metrics provided by the program for the four classes, in which we can notice that 96% of the "rec.sport.hockey" instances predicted were correct, 88% of the "talk.religion.misc" instances predicted were correct, 65% of the "comp.graphics" instances predicted were correct, and 100% of the "sci.space" instances predicted were correct. At the same time, 88% of the "rec.sport.hockey" instances were successfully identified, 99% of the "talk.religion.misc" instances were successfully identified, 97% of the "comp.graphics" instances were successfully identified, and 12% of the "sci.space" instances were successfully identified. Furthermore, we notice that the overall accuracy was 0.81, which means that 81% of the elements in the testing set were correctly classified.

Note that we can also check some of the incorrect results (so we can understand if we study more about what went wrong and why), which are presented at the end of the output when executing the program.

By analysing these results and the different scenarios studied, we can notice that when changing features, the results are approximately the same, except in the "sci.space" category, where we can see a difference in the recall value. Some possible reasons for this difference in this particular category may be, for example, the "tf-idf" method considers both the frequency of words within the documents and its frequency across the whole dataset. On the other hand, the type of language and terms used in this category can also affect the "IDF" (as there is a possibility that many terms, such as scientific ones, can only be seen in these type of documents), which ultimately can lead to better performance when exploring the "sci.space" category.

Task 6.3 Topic Modelling

Problem Description

This last task presents topic modelling, as we will use The LDA (latent Dirichlet allocation), which is a very popular method used for topic modelling, to distinguish and analyse the hidden semantic structures or topics of a large text collection, which can be used for various purposes, such as automatic categorisation of documents, text mining, text information retrieval, and others.

The LDA method lies in the premise that every document in a given collection contains at least one hidden topic, which means that every topic or theme is composed of a number of words. So in this problem, we will use this method to discover these underlying topics, as well as their word distribution (supporting words), which can be done by maximizing the likelihood of observing the given data, given the topics and words. It is also important to note that this method does not need to have the knowledge about any topics in the documents.

In our implemented solution for this problem, we will use as data two documents about "artificial intelligence" and "football", which will be the documents that are present in the corpus (collection), and can be seen in the implementation section shown next.

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:
```
import nltk
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from gensim import models, corpora
```

In [ ]:
```
documents = [
  """
  Artificial intelligence (AI), sometimes called machine
  intelligence, is intelligence demonstrated by machines, unlike
  the natural intelligence displayed by humans and animals. Leading
  AI textbooks define the field as the study of "intelligent
  agents": any device that perceives its environment and takes
  actions that maximize its chance of successfully achieving its
  goals. Colloquially, the term "artificial intelligence" is often
  used to describe machines (or computers) that mimic "cognitive"
  functions that humans associate with the human mind, such
  as "learning" and "problem solving".
  """,
  """
  Association football, more commonly known as football or
  soccer, is a team sport played with a spherical ball between
  two teams of 11 players. It is played by approximately 250
  million players in over 200 countries and dependencies, making it
  the world's most popular sport. The game is played on a
  rectangular field called a pitch with a goal at each end. The
  object of the game is to outscore the opposition by moving the
  ball beyond the goal line into the opposing goal. The team with
  the higher number of goals wins the game.
  """
]
```

In [ ]:
```
# Clean the data by using stemming and stopwords removal
nltk.download('stopwords')
stemmer = SnowballStemmer('english')
stop_words = stopwords.words('english')
texts = [
  [stemmer.stem(word) for word in document.lower().split() if word not in stop_words
  for document in documents
  ]
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [ ]:
```
# Create a dictionary from the words
dictionary = corpora.Dictionary(texts)

# Create a document-term matrix
doc_term_mat = [dictionary.doc2bow(text) for text in texts]

# Generate the LDA model
num_topics = 2
ldamodel = models.ldamodel.LdaModel(doc_term_mat,
        num_topics=num_topics, id2word=dictionary, passes=25)
```

In [ ]:
```
num_words = 5
for i in range(num_topics):
  print(ldamodel.print_topic(i, topn=num_words))

print('\nTop ' + str(num_words) + ' contributing words to each topic:')
for item in ldamodel.print_topics(num_topics=num_topics, num_words=num_words):
    print('\nTopic', item[0])
    list_of_strings = item[1].split(' + ')
    for text in list_of_strings:
        details = text.split('*')
        print("%-12s:%0.2f%%" %(details[1], 100*float(details[0])))
```

```
0.035*"intellig" + 0.035*"human" + 0.025*"machin" + 0.015*"associ" + 0.015*"call"
0.036*"team" + 0.036*"goal" + 0.036*"play" + 0.026*"game" + 0.026*"ball"

Top 5 contributing words to each topic:

Topic 0
"intellig"  :3.50%
"human"     :3.50%
"machin"    :2.50%
"associ"    :1.50%
"call"      :1.50%

Topic 1
"team"      :3.60%
"goal"      :3.60%
"play"      :3.60%
"game"      :2.60%
"ball"      :2.60%
```

In [ ]:
```
new_docs = [
  """
  Jager thinks this is just the start of AI eating the beautiful
  game. “We have a dedicated team that focuses only on artificial
  intelligence and machine learning for sports teams,” he
  says. “That is not only for soccer, but for Formula One and
```

```
  American football. We have a baseball team, and we're talking
  right now with cricket teams.”
  """
]

new_texts = [
  [stemmer.stem(word) for word in document.lower().split() if word not in stop_words
  for document in new_docs
  ]
new_doc_term_mat = [dictionary.doc2bow(text) for text in new_texts]

vector = ldamodel[new_doc_term_mat]
print(vector[0])
```

```
[(0, 0.50678587), (1, 0.4932142)]
```

Discussions

To solve this problem, the gensim package was used, which is used in natural language processing and topic modelling. More specifically, we will use the Latent Dirichlet Allocation (LDA).The online documentation can be found at https://radimrehurek.com/gensim/.

The sklearn or scikit-learn package was also used for completing this task.

Firstly, we used the 2 documents (2 topics) previously mentioned and applied the LDA method, The figure below shows us the results (an example) outputted by the program:

```
Top 5 contributing words to each topic:

Topic 0
"intellig"   :3.50%
"human"      :3.50%
"machin"     :2.50%
"associ"     :1.50%
"call"       :1.50%

Topic 1
"team"       :3.60%
"goal"       :3.60%
"play"       :3.60%
"game"       :2.60%
"ball"       :2.60%
```

Figure 10

Notice that each run can give different results.

By analysing the previous results, we can see that the LDA model identified 2 different topics, where each of the words related, as shown above, have a significant weight to contribute to identifying the correspondent topic, so in this case, the words or terms related to artificial intelligence can be seen in "topic 0" and ones related to football can be seen in "topic 1". So we

can say that higher word percentages mean that these words are more central to the related topic.

Let's now see the results when we add a new document, which can be seen in the code implementation section (along with the first ones), so we can apply the trained LDA model to this new text document, and then analyse these new results.

The new results outputted are displayed in the form of a vector structure, such as [(0, 0.50678587), (1, 0.4932142)], which indicates the computation of its projection vector, from the LDA model.

By analysing these results, we can see that regarding "topic 0", this new document presents a weight association value of 0.50678587, and regarding "topic 1", it presents a weight association value of 0.4932142. So we can agree that these results seem acceptable, as the new document talks about both topics, and therefore combine elements from both.

Conclusion

In conclusion, after completing this Lab work by solving its tasks and problems, we can agree that by using all these methods, such as the simple word count features, "tf" and "tf-idf", the Naïve Bayes, and the LDA, we were able to find a solution for these 3 tasks. By doing so, we gained additional knowledge on how these tools and methods can be applied to solve natural language processing problems, and how these can be useful for different scenarios.

In further investigations, we could study more about other natural language processing methods, while also exploring different problems. Thus, we could also expand our corpus and add more documents and content, so we can study different kinds of results and see how these methods perform under different conditions.

Lab 7. Image Processing

This lab sheet aims to help us understand more about image processing, as we are going to study this new topic, and use it to solve 3 problems, which will be divided into the following tasks:

1. The face detection problem
2. The image segmentation problem
3. The object recognition problem

Task 7.1 Face Detection

Problem Description

This first task presents us with a face detection problem, which is the basis for any type of image processing that includes faces. For this problem, we will use the Haar cascade face detector method, so we can find and analyse faces when given a certain input image to explore.

There are many other face recognition methods developed, but even though the Haar cascade face detector is probably not the most accurate one, is currently the fastest one.

In order to successfully perform face recognition, the Haar cascade face detector uses rectangle features, which are patterns that will help to find out the edges of the lines in an input image, as well as some possible abrupt changes when it comes to the pixels involved, so in other words, these features will work to represent local image characteristics. At the same time, these features will analyse a face, and identify the contrast between the eyes and the surrounding face (it should be expected that the eye region is darker than the cheek area), by using these rectangles positioned over the area of the eyes and also over the cheek area (while calculating the difference between the sum of pixel values in between these two regions). The same technique is applied to the eyes and nose areas, in which, again, these adjacent rectangles will be placed in these areas, where the pixel comparisons will be made when it comes to darker and lighter regions (two eyes darker than nose bridge). We can see these features in the figure below taken from the lecture.

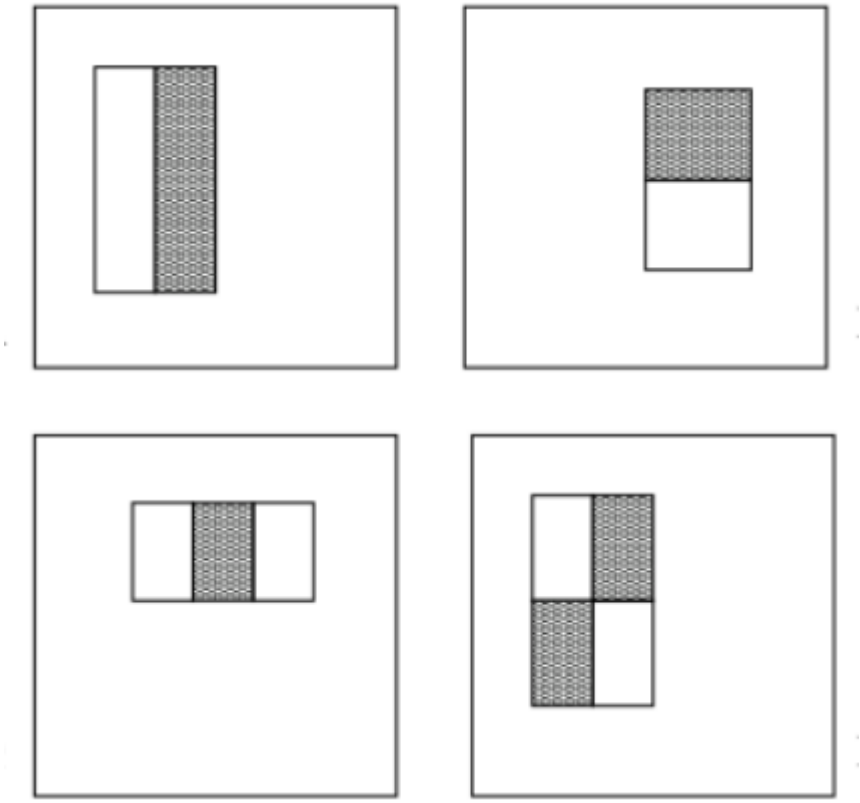


Figure 1

This model also uses a cascade of classifiers technique, in which sub-windows will be accepted or rejected. So this mechanism will quickly and effectively eliminate regions of the image that do not contain any faces, which will be very useful since most of the input image will be rejected. This will help to save computation and resources when analysing the image.

Therefore, these techniques will be the key to getting quick and effective results.

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:
```
!pip install opencv-python
import cv2
from google.colab.patches import cv2_imshow
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packag
es (4.8.0.76)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packag
es (from opencv-python) (1.23.5)
```

In [ ]:
```
# Load the face detection model from the model file
!wget 'https://github.com/yongminli/data/raw/main/haarcascade_frontalface_default.xm
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

```
--2023-11-07 01:40:52--  https://github.com/yongminli/data/raw/main/haarcascade_front
alface_default.xml
Resolving github.com (github.com)... 192.30.255.112
Connecting to github.com (github.com)|192.30.255.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/yongminli/data/main/haarcascade_frontalfa
```

```
ce_default.xml [following]
--2023-11-07 01:40:53--  https://raw.githubusercontent.com/yongminli/data/main/haarca
scade_frontalface_default.xml
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 1
85.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:
443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 930127 (908K) [text/plain]
Saving to: ‘haarcascade_frontalface_default.xml.1’

haarcascade_frontal 100%[===================>] 908.33K  --.-KB/s    in 0.04s

2023-11-07 01:40:53 (23.4 MB/s) - ‘haarcascade_frontalface_default.xml.1’ saved [9301
27/930127]
```

In [ ]:

```python
# Load the model file and image files to Google Colab
from google.colab import files
file = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving w8_7.1.6.jpg to w8_7.1.6.jpg
```

In [ ]:

```python
# Read an input image
img = cv2.imread('w8_7.1.6.jpg')

# Convert it into a grayscale image
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Detect faces
# faces = face_cascade.detectMultiScale(image=gray, scaleFactor=1.2, minNeighbors=5,
faces = face_cascade.detectMultiScale(image=gray, minNeighbors=5)

# Display the detecition results
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
cv2_imshow(img)
```

Discussions

To solve this problem, the OpenCV library was used, as it is a popular library for image processing and computer vision, and also provides lots of important tools and functions that were useful in this lab. The online documentation can be found at https://pypi.org/project/opencv-python/.

The files used in this lab can be found at https://drive.google.com/drive/folders/1vf9INW8CBx7LH5R-YD2RS7uwqeAtBpjv?usp=sharing.

For this first case, we will test the trained model of the Haar cascade face detector, by using a "clear" image as input when it comes to face detection, in which the intention here is to see the algorithm perform quickly and successfully, as the faces in the image are frontal, the lighting is bright, and the people presented are easily identified. The detection will then occur with the detectMultiScale() function, implemented in the solution shown in the previous section.

In the following figure, we can see a photo of students, which will be used as input for this case.

Figure 2

The figure below presents an image of the result outputted by the program.

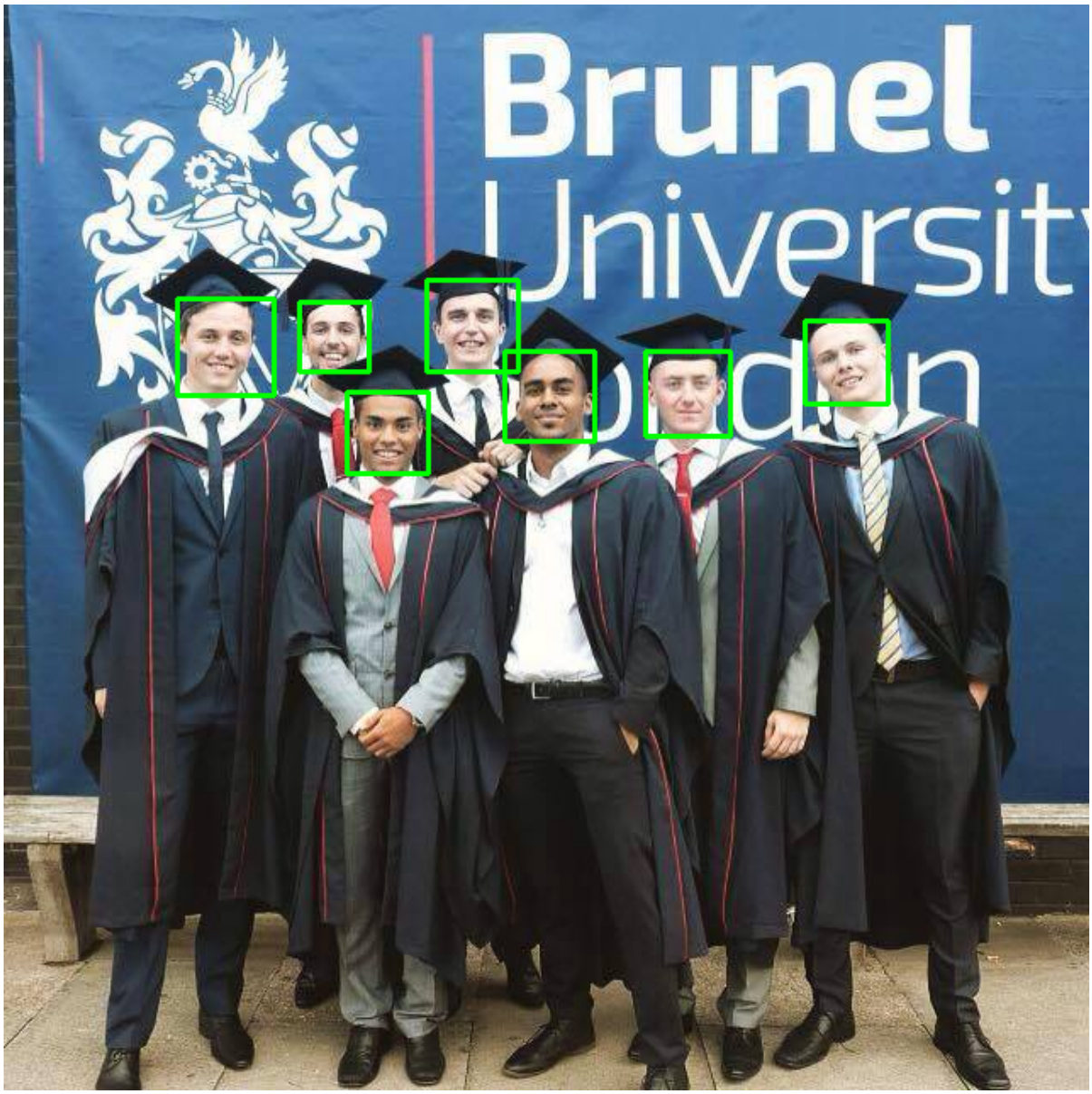


Figure 3

By analysing the result, we can see that the method performed as intended, and therefore the result can be considered accurate, as this resulting image has got all the faces correctly identified.

For this second case, we will test the trained model of the Haar cascade face detector, by using a photo of a cat as input, which obviously is not human, but still presents a face. The face displayed in the image is still frontal, and the lighting is also still bright.

In the following figure, we can see an image of both the input that will be used for this case, as well as the result outputted by the program.

Figure 4

By analysing the figure above, we can see that the input and the output images are equal, which means that the method was not able to identify the face of the animal, as this method is used mostly for identifying human traits, and not animal or objects.

For this final case, we will test the trained model of the Haar cascade face detector, by using an image of a street as input, in which there are lots of different elements, such as many objects and people. The faces displayed in the image are not frontal, and the lighting is darker.

The figure below presents an image of both the input that will be used for this case, as well as the result outputted by the program.

Figure 5

By looking at the result, we can see that, once again, the input and the output images are equal, which means that the method was not able to identify the faces of the people presented.

Task 7.2 Image Segmentation

Problem Description

This next task presents an image segmentation problem, in which we will partition an image into meaningful regions. This is a well-known topic in the world of image processing and computer vision.

To perform image segmentation, we will use the R-CNN model, which is a high-level segmentation method. This method works toward outlining the boundaries of objects within an image while combining object recognition and segmentation. So this model, after successfully identifying the objects and their boundaries, will present these boundaries, along with creating bounding boxes for each one of these objects or regions identified. It is also important to notice that the model chosen, along with its training data, are key elements for its performance when dealing with these images.

For this problem, we will use a pre-built R-CNN model (mask_rcnn_coco.h5).

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:

```
#Mount Google drive to read the files from
from google.colab import drive
drive.mount('/content/drive')
path = "/content/drive/My Drive/data/"
```

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

In [ ]:

```
!pip install pixellib
import pixellib
from pixellib.instance import instance_segmentation
import cv2
from google.colab.patches import cv2_imshow
```

```
Collecting pixellib
  Downloading pixellib-0.7.1-py3-none-any.whl (430 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 430.3/430.3 kB 7.2 MB/s eta 0:00:00
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (fro
m pixellib) (9.4.0)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-package
s (from pixellib) (0.19.3)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packag
es (from pixellib) (4.8.0.76)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages
(from pixellib) (3.7.1)
Requirement already satisfied: imgaug in /usr/local/lib/python3.10/dist-packages (fro
m pixellib) (0.4.0)
Collecting labelme2coco (from pixellib)
  Downloading labelme2coco-0.2.4-py3-none-any.whl (19 kB)
Collecting imantics (from pixellib)
  Downloading imantics-0.1.12.tar.gz (13 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: Cython in /usr/local/lib/python3.10/dist-packages (fro
m pixellib) (3.0.5)
Collecting pyQt5 (from pixellib)
  Downloading PyQt5-5.15.10-cp37-abi3-manylinux_2_17_x86_64.whl (8.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.2/8.2 MB 90.3 MB/s eta 0:00:00
Collecting fvcore (from pixellib)
  Downloading fvcore-0.1.5.post20221221.tar.gz (50 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.2/50.2 kB 7.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting iopath (from pixellib)
  Downloading iopath-0.1.10.tar.gz (42 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.2/42.2 kB 5.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting omegaconf (from pixellib)
  Downloading omegaconf-2.3.0-py3-none-any.whl (79 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.5/79.5 kB 10.7 MB/s eta 0:00:00
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages
(from pixellib) (2.2.1)
Requirement already satisfied: termcolor in /usr/local/lib/python3.10/dist-packages
(from pixellib) (2.3.0)
Collecting yacs (from pixellib)
  Downloading yacs-0.1.8-py3-none-any.whl (14 kB)
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (f
rom pixellib) (0.9.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from
pixellib) (4.66.1)
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (fro
m pixellib) (0.18.3)
Requirement already satisfied: pydot in /usr/local/lib/python3.10/dist-packages (from
pixellib) (1.4.2)
Collecting dataclasses (from pixellib)
```

```
  Downloading dataclasses-0.6-py3-none-any.whl (14 kB)
Collecting hydra-core (from pixellib)
  Downloading hydra_core-1.3.2-py3-none-any.whl (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.5/154.5 kB 17.3 MB/s eta 0:00:00
Collecting black (from pixellib)
  Downloading black-23.11.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.wh
l (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 74.1 MB/s eta 0:00:00
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.10/dist-package
s (from black->pixellib) (8.1.7)
Collecting mypy-extensions>=0.4.3 (from black->pixellib)
  Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)
Requirement already satisfied: packaging>=22.0 in /usr/local/lib/python3.10/dist-pack
ages (from black->pixellib) (23.2)
Collecting pathspec>=0.9.0 (from black->pixellib)
  Downloading pathspec-0.11.2-py3-none-any.whl (29 kB)
Requirement already satisfied: platformdirs>=2 in /usr/local/lib/python3.10/dist-pack
ages (from black->pixellib) (3.11.0)
Requirement already satisfied: tomli>=1.1.0 in /usr/local/lib/python3.10/dist-package
s (from black->pixellib) (2.0.1)
Requirement already satisfied: typing-extensions>=4.0.1 in /usr/local/lib/python3.10/
dist-packages (from black->pixellib) (4.5.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
fvcore->pixellib) (1.23.5)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages
(from fvcore->pixellib) (6.0.1)
Collecting portalocker (from iopath->pixellib)
  Downloading portalocker-2.8.2-py3-none-any.whl (17 kB)
Collecting antlr4-python3-runtime==4.9.* (from hydra-core->pixellib)
  Downloading antlr4-python3-runtime-4.9.3.tar.gz (117 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 117.0/117.0 kB 14.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from
imantics->pixellib) (4.9.3)
Collecting xmljson (from imantics->pixellib)
  Downloading xmljson-0.2.1-py2.py3-none-any.whl (10 kB)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from i
mgaug->pixellib) (1.16.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from
imgaug->pixellib) (1.11.3)
Requirement already satisfied: imageio in /usr/local/lib/python3.10/dist-packages (fr
om imgaug->pixellib) (2.31.6)
Requirement already satisfied: Shapely in /usr/local/lib/python3.10/dist-packages (fr
om imgaug->pixellib) (2.0.2)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packag
es (from scikit-image->pixellib) (3.2.1)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-
packages (from scikit-image->pixellib) (2023.9.26)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-pa
ckages (from scikit-image->pixellib) (1.4.1)
Collecting sahi>=0.8.19 (from labelme2coco->pixellib)
  Downloading sahi-0.11.15-py3-none-any.whl (105 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 105.4/105.4 kB 12.3 MB/s eta 0:00:00
Requirement already satisfied: jsonschema>=2.6.0 in /usr/local/lib/python3.10/dist-pa
ckages (from labelme2coco->pixellib) (4.19.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-pac
kages (from matplotlib->pixellib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-package
s (from matplotlib->pixellib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-pa
ckages (from matplotlib->pixellib) (4.44.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-pa
ckages (from matplotlib->pixellib) (1.4.5)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-pac
```

```
kages (from matplotlib->pixellib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist
-packages (from matplotlib->pixellib) (2.8.2)
Collecting PyQt5-sip<13,>=12.13 (from pyQt5->pixellib)
  Downloading PyQt5_sip-12.13.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.wh
l (338 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 338.1/338.1 kB 37.4 MB/s eta 0:00:00
Collecting PyQt5-Qt5>=5.15.2 (from pyQt5->pixellib)
  Downloading PyQt5_Qt5-5.15.2-py3-none-manylinux2014_x86_64.whl (59.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.9/59.9 MB 15.7 MB/s eta 0:00:00
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packag
es (from jsonschema>=2.6.0->labelme2coco->pixellib) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/li
b/python3.10/dist-packages (from jsonschema>=2.6.0->labelme2coco->pixellib) (2023.7.
1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6.0->labelme2coco->pixellib) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packa
ges (from jsonschema>=2.6.0->labelme2coco->pixellib) (0.12.0)
Collecting opencv-python (from pixellib)
  Downloading opencv_python-4.7.0.72-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x8
6_64.whl (61.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.8/61.8 MB 11.9 MB/s eta 0:00:00
Collecting pybboxes==0.1.6 (from sahi>=0.8.19->labelme2coco->pixellib)
  Downloading pybboxes-0.1.6-py3-none-any.whl (24 kB)
Collecting fire (from sahi>=0.8.19->labelme2coco->pixellib)
  Downloading fire-0.5.0.tar.gz (88 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 88.3/88.3 kB 12.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting terminaltables (from sahi>=0.8.19->labelme2coco->pixellib)
  Downloading terminaltables-3.1.10-py2.py3-none-any.whl (15 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (f
rom sahi>=0.8.19->labelme2coco->pixellib) (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
dist-packages (from requests->sahi>=0.8.19->labelme2coco->pixellib) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
s (from requests->sahi>=0.8.19->labelme2coco->pixellib) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
ackages (from requests->sahi>=0.8.19->labelme2coco->pixellib) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
ackages (from requests->sahi>=0.8.19->labelme2coco->pixellib) (2023.7.22)
Building wheels for collected packages: fvcore, iopath, antlr4-python3-runtime, imant
ics, fire
  Building wheel for fvcore (setup.py) ... done
  Created wheel for fvcore: filename=fvcore-0.1.5.post20221221-py3-none-any.whl size=
61400 sha256=df6b55a4b3aadfa0e9834cdaddef68435f05d42234957fff5a132664fe105e40
  Stored in directory: /root/.cache/pip/wheels/01/c0/af/77c1cf53a1be9e42a52b48e5af216
9d40ec2e89f7362489dd0
  Building wheel for iopath (setup.py) ... done
  Created wheel for iopath: filename=iopath-0.1.10-py3-none-any.whl size=31532 sha256
=66ca7be3070e38b51992137e58f7048c41c9ca549867b4d46c160547621de402
  Stored in directory: /root/.cache/pip/wheels/9a/a3/b6/ac0fcd1b4ed5cfeb3db92e6a0e476
cfd48ed0df92b91080c1d
  Building wheel for antlr4-python3-runtime (setup.py) ... done
  Created wheel for antlr4-python3-runtime: filename=antlr4_python3_runtime-4.9.3-py3
-none-any.whl size=144555 sha256=3b9a3c6da5da093fc0f7f9e8411123a0598ca4a67988cd70a4ba
bfecc4149ea1
  Stored in directory: /root/.cache/pip/wheels/12/93/dd/1f6a127edc45659556564c5730f6d
4e300888f4bca2d4c5a88
  Building wheel for imantics (setup.py) ... done
  Created wheel for imantics: filename=imantics-0.1.12-py3-none-any.whl size=16011 sh
a256=1bbc4d4fb09863a728072dd5f7729e1cc6fd789db228097d7c69bb2f0747aab5
  Stored in directory: /root/.cache/pip/wheels/56/6a/be/4c60e88b14abec4e93234a1f7f91c
e8abe1ae88a2b3eaad3ac
```

```
  Building wheel for fire (setup.py) ... done
  Created wheel for fire: filename=fire-0.5.0-py2.py3-none-any.whl size=116933 sha256
=08769e641eabb41be626f8ca48f3a52e6b873b31a096dc71e7e62653e5a942f6
  Stored in directory: /root/.cache/pip/wheels/90/d4/f7/9404e5db0116bd4d43e5666eaa3e7
0ab53723e1e3ea40c9a95
Successfully built fvcore iopath antlr4-python3-runtime imantics fire
Installing collected packages: xmljson, PyQt5-Qt5, dataclasses, antlr4-python3-runtim
e, yacs, terminaltables, PyQt5-sip, pybboxes, portalocker, pathspec, opencv-python, o
megaconf, mypy-extensions, fire, sahi, pyQt5, iopath, imantics, hydra-core, black, fv
core, labelme2coco, pixellib
  Attempting uninstall: opencv-python
    Found existing installation: opencv-python 4.8.0.76
    Uninstalling opencv-python-4.8.0.76:
      Successfully uninstalled opencv-python-4.8.0.76
Successfully installed PyQt5-Qt5-5.15.2 PyQt5-sip-12.13.0 antlr4-python3-runtime-4.9.
3 black-23.11.0 dataclasses-0.6 fire-0.5.0 fvcore-0.1.5.post20221221 hydra-core-1.3.2
imantics-0.1.12 iopath-0.1.10 labelme2coco-0.2.4 mypy-extensions-1.0.0 omegaconf-2.3.
0 opencv-python-4.7.0.72 pathspec-0.11.2 pixellib-0.7.1 portalocker-2.8.2 pyQt5-5.15.
10 pybboxes-0.1.6 sahi-0.11.15 terminaltables-3.1.10 xmljson-0.2.1 yacs-0.1.8
```

In [ ]:

```python
import torch
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

model = instanceSegmentation()
model.load_model(path + "pointrend_resnet50.pkl")
# model.segmentImage(path + "segment.jpg", show_bboxes=False, output_image_name="/co
segmask, output = model.segmentImage(path + "objects.jpg", show_bboxes=False)
cv2_imshow(output)
```

```
WARNING:fvcore.common.checkpoint:The checkpoint state_dict contains keys that are not
used by the model:
  proposal_generator.anchor_generator.cell_anchors.{0, 1, 2, 3, 4}
```

In [ ]:

```
# img = cv2.imread("/content/gdrive/MyDrive/Datasets/CS5707/data/image_path_out.jpg"
# cv2_imshow(img)
```

Discussions

To solve this problem, the PixelLib package was used, which is used for image and video segmentation. The online documentation can be found at https://pixellib.readthedocs.io/en/latest/.

For this first case, we will test the trained R-CNN model, by using an image of a street as input, the same image used in the previous task (Figure 5), which can be considered a quite "complex" image. The image segmentation will then occur with the model.segmentImage() function, implemented in the solution shown in the previous section.

The figure below presents an image of the result outputted by the program.

Figure 6

By looking at the result, we can see that the model performed as intended, as the model was able to correctly identify most objects, and then clearly represent their boundaries in the resulting outputted image.

For this second case, we will test the trained R-CNN model, by using a photo of a person and some animals in a field scenario, which will give us lots of different elements to test and analyse.

In the following figure, we can see the image that will be used as input for this case.

Figure 7

The figure below presents an image of the result outputted by the program.

Figure 8

By analysing the result, we can see that the model performed as expected, as, once again, the model was able to correctly identify the objects, as well as the living entities presented, and then clearly represent their boundaries in the resulting outputted image.

So we can say that this model was overall successful when it comes to image segmentation and identifying most object boundaries while also being able to identify distinct elements in the images provided.

Task 7.3 Object Recognition

Problem Description

This last task presents the topic of object recognition, which is also a very important topic when it comes to image processing and computer vision. So this method will allow us to identify different object categories when analysing an image (or video), while only scanning the input image once.

To perform object recognition, we will use the YOLO model (considered one of the best for this purpose), which is different than any type of R-CNN model. So this model, in order to perform object recognition, uses an image as input, which is then applied to a neural network. At the same time, various regions will be identified in the image, in a grid format, which will then predict bounding boxes and class probabilities. These bounding boxes will correspond to a certain class, and will display a label.

Implementation and results

This section shows the implemented code and its respective output.

In [ ]:
```
!pip install cvlib
import cvlib
from cvlib.object_detection import draw_bbox, YOLO
```

```
Collecting cvlib
  Downloading cvlib-0.2.7.tar.gz (13.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.1/13.1 MB 66.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
cvlib) (1.23.5)
Collecting progressbar (from cvlib)
  Downloading progressbar-2.5.tar.gz (10 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (f
rom cvlib) (2.31.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (fro
m cvlib) (9.4.0)
Requirement already satisfied: imageio in /usr/local/lib/python3.10/dist-packages (fr
om cvlib) (2.31.6)
Requirement already satisfied: imutils in /usr/local/lib/python3.10/dist-packages (fr
om cvlib) (0.5.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
dist-packages (from requests->cvlib) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
s (from requests->cvlib) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
ackages (from requests->cvlib) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
ackages (from requests->cvlib) (2023.7.22)
Building wheels for collected packages: cvlib, progressbar
  Building wheel for cvlib (setup.py) ... done
```

```
  Created wheel for cvlib: filename=cvlib-0.2.7-py3-none-any.whl size=10046370 sha256
=d9c953f608d68f22890710f2bf459088840abcb6546f79f01535f8a08c2d2b0d
  Stored in directory: /root/.cache/pip/wheels/9e/a5/d4/fe37b48fe4f4b621ba5e574a99123
0070f3cc4f02322a01489
  Building wheel for progressbar (setup.py) ... done
  Created wheel for progressbar: filename=progressbar-2.5-py3-none-any.whl size=12067
sha256=4f0bac7e408845c3da59261d775451e1c49d0687d84a0b0575f1081324c1ddb4
  Stored in directory: /root/.cache/pip/wheels/cd/17/e5/765d1a3112ff3978f70223502f604
7e06c43a24d7c5f8ff95b
Successfully built cvlib progressbar
Installing collected packages: progressbar, cvlib
Successfully installed cvlib-0.2.7 progressbar-2.5
```

In [ ]:

```
# Download the YOLOv3 model configuration, weights and labels files
# !wget https://github.com/yongminli/data/raw/main/yolov3-tiny.cfg -O yolov3-tiny.cf
# !wget https://github.com/yongminli/data/raw/main/yolov3-tiny.weights -O yolov3-tin
# !wget https://github.com/yongminli/data/raw/main/yolov3.txt -O yolov3.txt

# !wget http://www.brunel.ac.uk/~csstyyl/tmp/yolov3.cfg -O yolov3.cfg
# !wget http://www.brunel.ac.uk/~csstyyl/tmp/yolov3.weights -O yolov3.weights
# !wget http://www.brunel.ac.uk/~csstyyl/tmp/yolov3.txt -O yolov3.txt
# path = ""

# Alternatively you can save the data to your Google Drive, and load the data from t
from google.colab import drive
drive.mount('/content/drive')
path = "/content/drive/My Drive/data/"
```

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

In [ ]:

```
# The default YOLO model files
#config = path + 'yolov3.cfg'
#weights = path + 'yolov3.weights'
#labels = path + 'yolov3.txt'

# # Alternative the "tiny" version of YOLO, which is faster but less accurate
config = path + 'yolov3-tiny.cfg'
weights = path + 'yolov3-tiny.weights'
labels = path + 'yolov3.txt'

# Construct the YOLOv3 Model
yolo = YOLO(weights, config, labels)
```

```
[INFO] Initializing YOLO ..
```

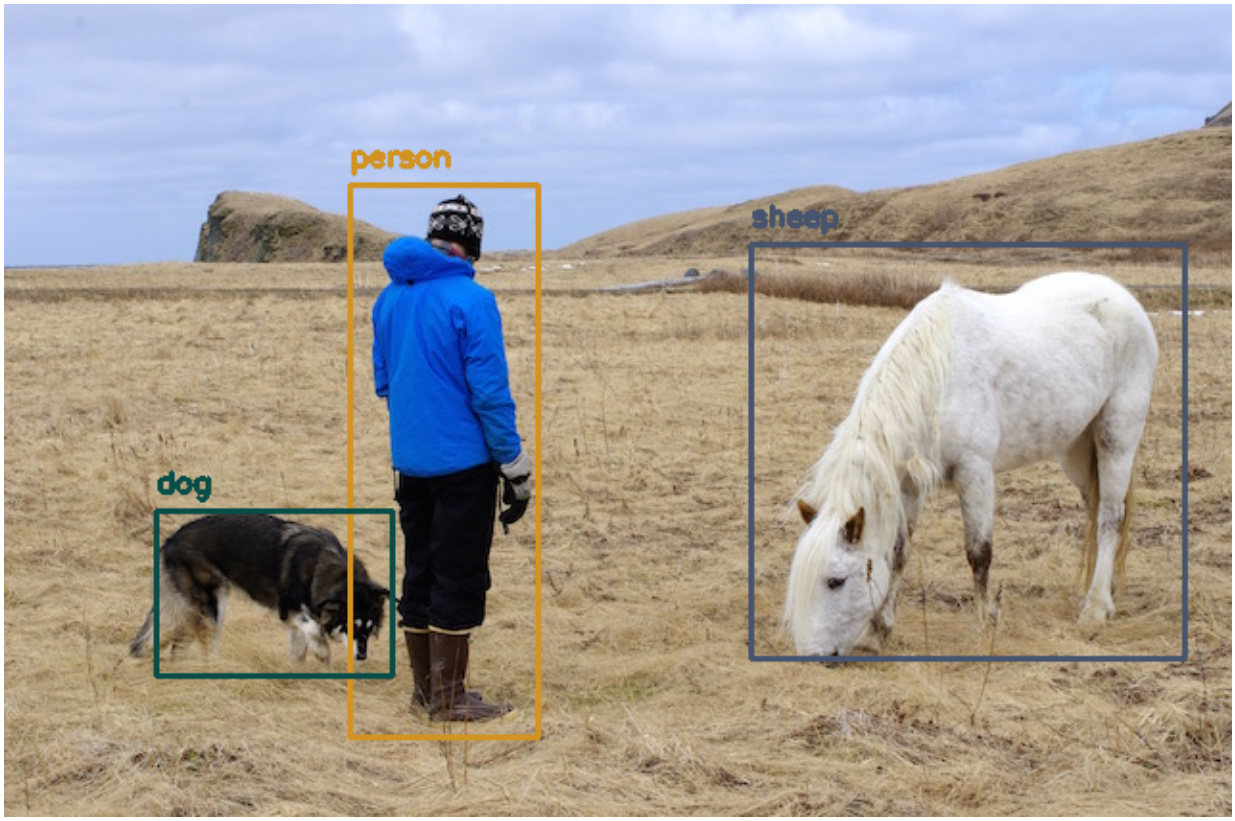
In [ ]:

```
# # Load the image files to Google Colab
# from google.colab import files
# file = files.upload()
```

In [ ]:

```
import cv2
from google.colab.patches import cv2_imshow

# Read an image file
img = cv2.imread(path+'objects2.jpg')
# img = cv2.imread(path+'objects2.jpg')
cv2_imshow(img)
print()

# Detect objects from the image, and display the results
bbox, label, conf = yolo.detect_objects(img)
# bbox, label, conf = yolo.detect_objects(img, confidence=0.25, nms_thresh=0.2)
```

```
yolo.draw_bbox(img, bbox, label, conf)
cv2_imshow(img)
```



Discussions

To solve this problem, the cvlib package was used, which is used for object detection. The online documentation can be found at https://www.cvlib.net/.

For this first case, we will test the default YOLO version model, which is more accurate but slower. For this case we will test two different images as inputs, the first will be an image of a

person and some animals in a green field scenario, the same image used in the previous task (Figure 7). The second image will be, once again, an image of a person, this time facing backward, and some animals in a mountain field scenario. The object detection will occur with the yolo.detect_objects() function, implemented in the solution shown in the previous section.

The figure below presents the image of the result outputted by the program for this first case when inputting the first image mentioned above.

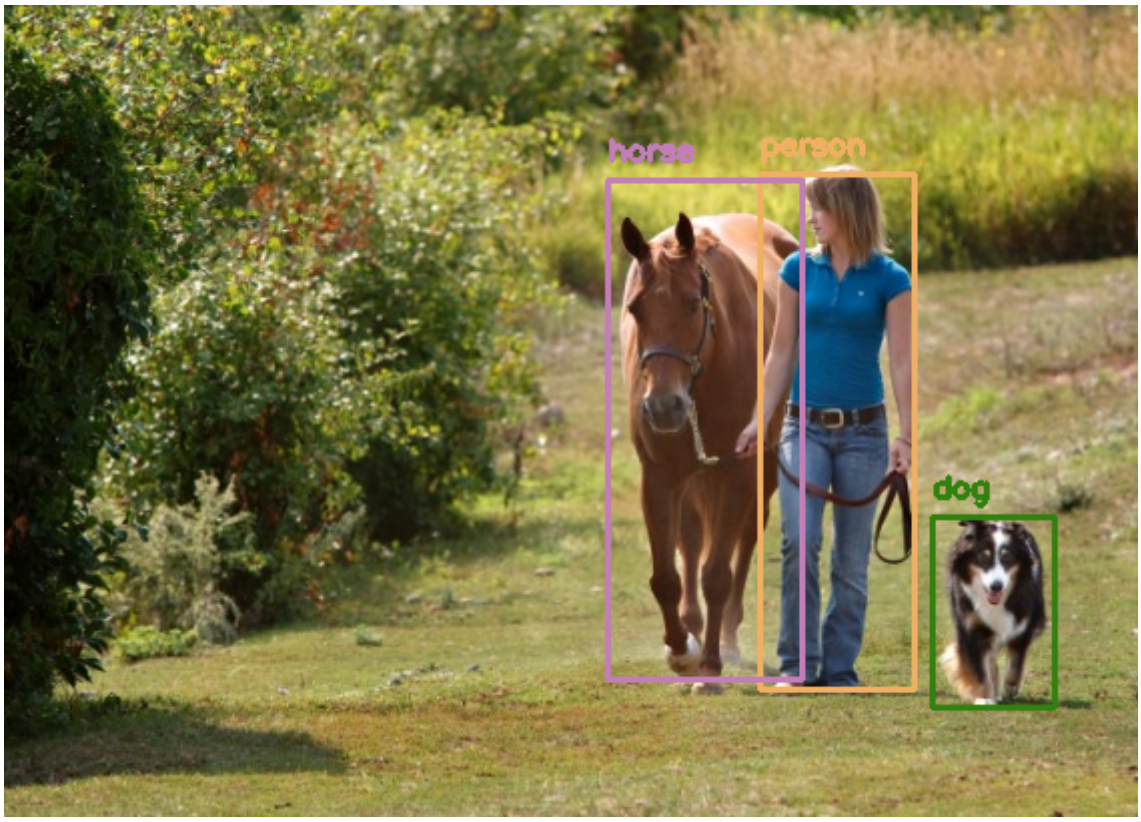


Figure 9

By looking at the resulting figure above, we can notice that the model performed as intended, as the model was able to correctly identify the objects (the living entities) and label them adequately while displaying the bounding boxes with the correct class identifier.

In the following figure, we can see the image that will be used as the second input for this first case.

Figure 10

The figure below presents an image of the corresponding result outputted by the program.

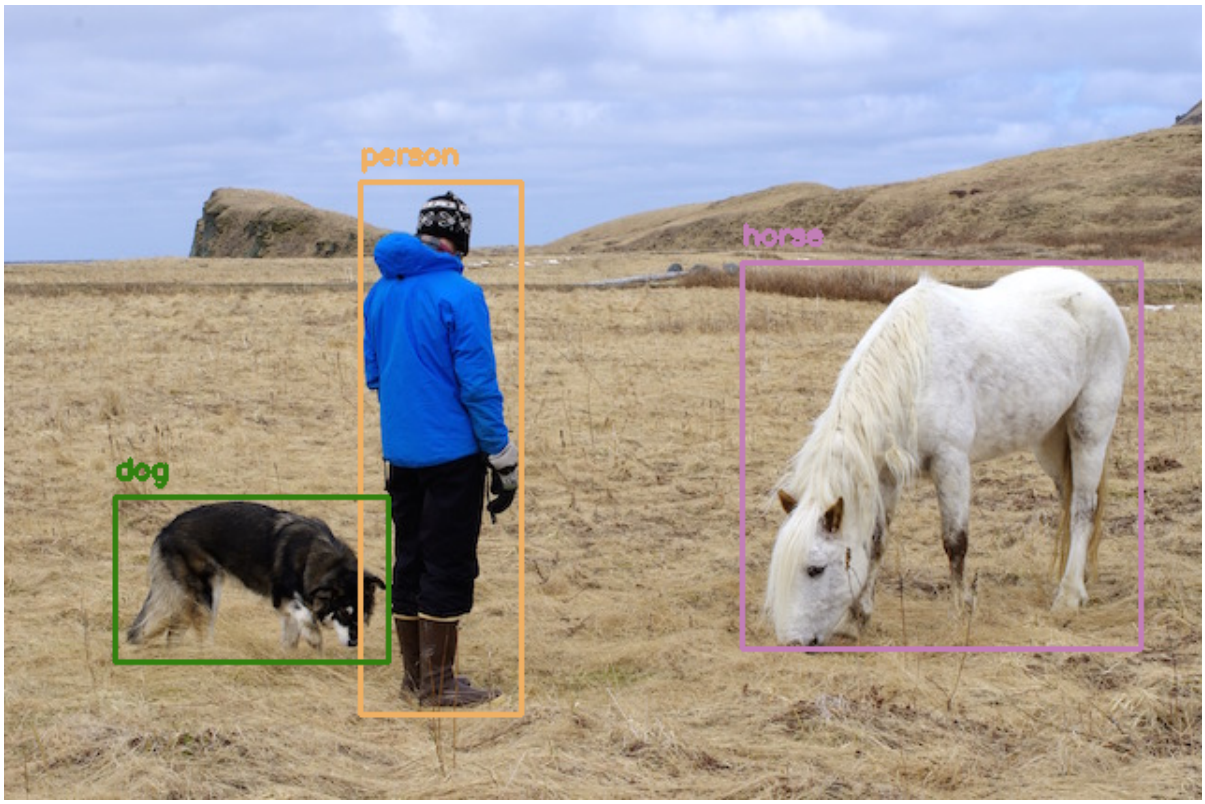


Figure 11

By analysing the result, we can see that the model performed as expected, as, once again, the model was able to correctly identify the objects (the living entities), and label them adequately.

For this second case, we will test the tiny YOLO version model, which is less accurate but faster. For this case, we will test as input the same images used in the first case.

The figure below presents an image of the corresponding result outputted by the program, for the first inputted image (Figure 7), when using this new model.

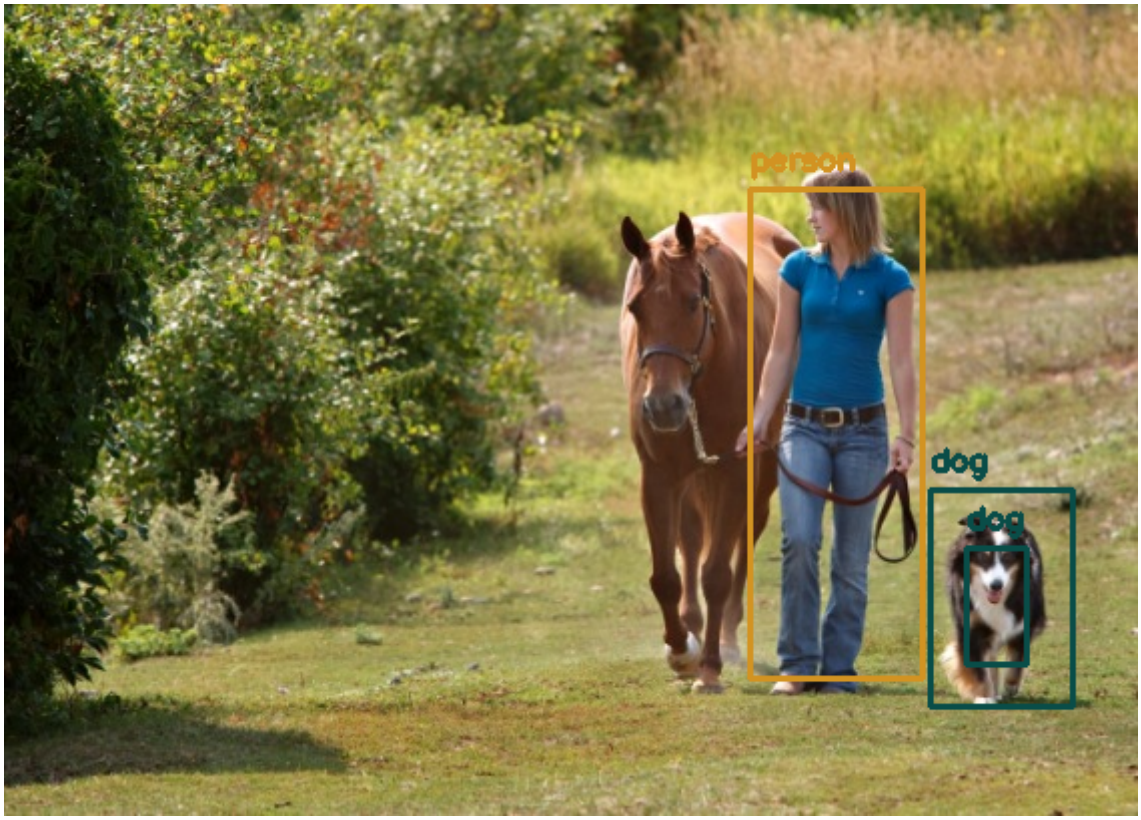


Figure 12

By looking at the image above, we can notice that this algorithm was not able to correctly identify nor label all the elements present in the picture, as the horse presented in the image was not even identified or classified, on the other hand, the dog was both identified and classified two times.

The figure below presents an image of the corresponding result outputted by the program, for the second inputted image (Figure 10), when using this new model.

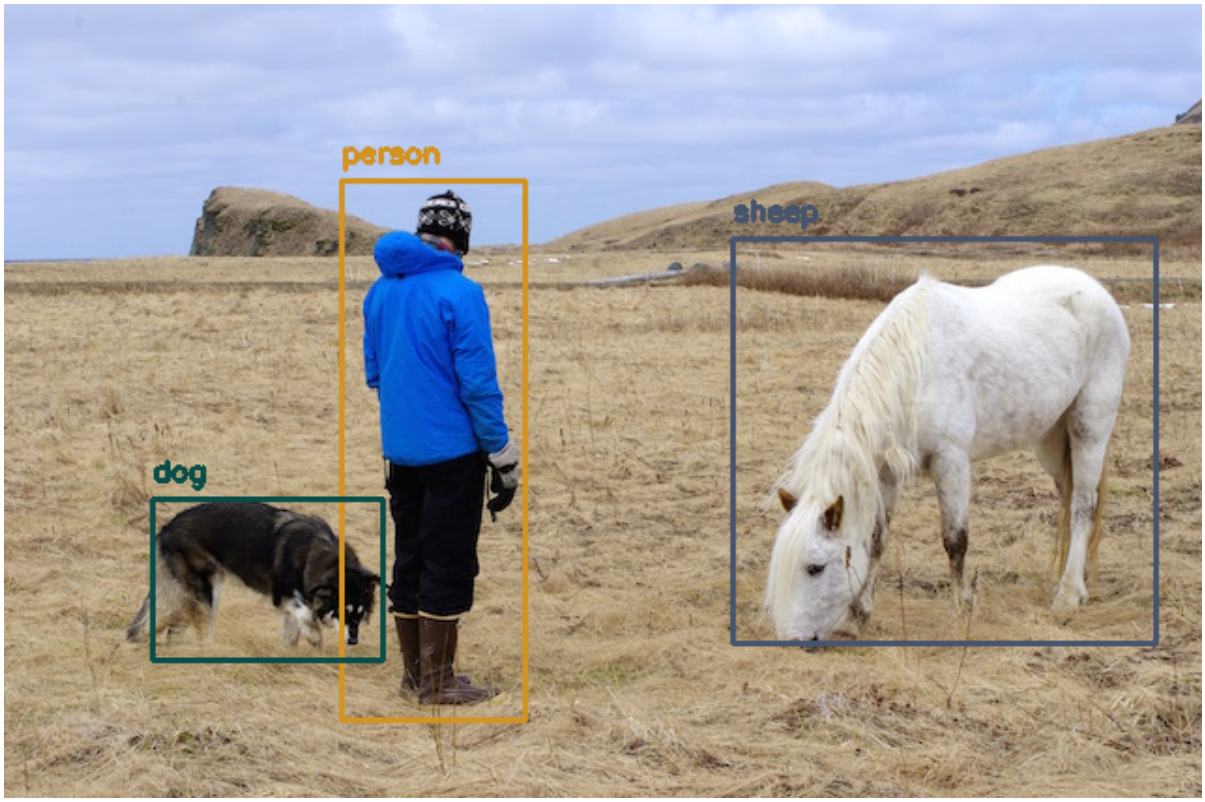


Figure 13

By analysing the resulting image, we can notice that, once more, this algorithm wrongly identified and labelled certain elements in the image, the horse in particular, which was identified but classified as a sheep, which is a wrong label.

This difference in results for these two cases studied (when using the default and tiny YOLO version models) were predictable, because, as we mentioned before, these two models present different performing mechanisms, as the first one (default YOLO) is more accurate but slower. The size of these models is also worth mentioning, as the default version is approximately 250MB, and the tiny one is 35MB, these are aspects that can be reflected in the results produced. We can also notice that the tiny version is probably more adequate for real-time applications, while the default version is more suitable if we are looking for higher accuracy results and time is not as important.

Conclusion

In conclusion, after completing this Lab work by solving its tasks and problems, we can agree that by using all these methods, such as the Haar cascade detector, the R-CNN model, and the YOLO models, we were able to find a solution for these 3 tasks. By doing so, we gained additional knowledge on how these tools and methods can be applied to solve image processing problems, and how these can be useful for different scenarios.

In further investigations, we could study more about other image processing methods, while also exploring how these are applied in real-life scenarios, as image processing is such an intriguing and popular topic in today's life. Moreover, we could also expand our studies to video analysis problems, with the use of, for example, R-CNN and YOLO models, and even extend our investigations to machine learning and deep learning areas.