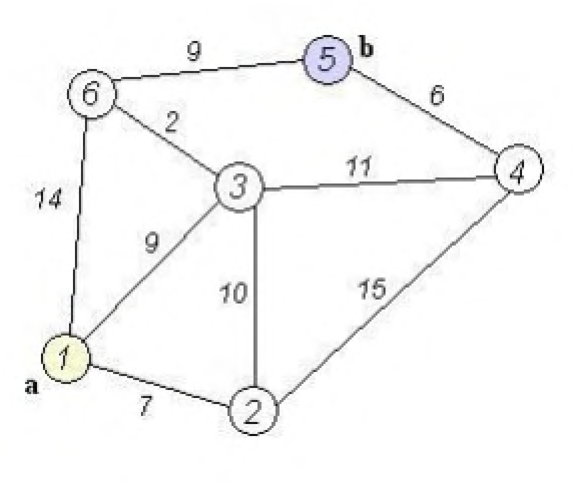
Lab 2. Search

Task 2.1 Route Planning

Problem Descriptions

The image depicts a map where there are 6 cities, numbered from 1 to 6, and each is a node on the route. The distances between them, if connected directly, are show on the edges in the graph. This task consists of finding the shortest route to city number five from city one.



To be able to solve the problem, we firse need to define the required variables.

STATE: each city corresponds to a number between one and six as illustrated in the image.

ACTION: the action is the destination city to which we are moving, starting from the current state, which is the city where we are. For example, if we are currently in city 3 and the next city we are moving to is 6, the action will be 6.

GOAL TEST: it consists of verifying whether the current state is indeed the final destination, to understand if the goal has been reached. In this task the goal is city 5.

PATH COST: the cost of the route is indicated in the illustration. For implementation we create a 6x6 array, where each element is the distance between a pair of cities (0 for self-position and '#' for any unconnected cities).

Implementation and Results

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

```
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 from 1.
    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]

```

```
problem = Route(1, 5)
# result = breadth_first(problem, graph_search=True)
# result = depth_first(problem, graph_search=True)
result = uniform_cost(problem)

# Print the results
path = [x[1]+1 for x in result.path()]
print("The route is %s, and total cost is %s" %(path, result.cost))
```

```
The route is [1, 3, 6, 5], and total cost is 20
```

Discussions

In this task, three different search methods were compared, namely breadth-first serch, depth-first search and uniform-cost search. Comparing the results obtained, it emerges that the

Breadth-first search and Depth-first search methods both give the solution 23, while Uniform-cost search has the solution 20. This means that Uniform-cost search gave us the optimal solution, which is the shortest solution.

Lab 2. Search

Task 2.2 8-Puzzy Problem

Problem Descriptions

Implement a programme that solves the 8-puzzle problem, which can be described as follows: there are 8 numbered tiles and one empty tile in a 3x3 grid. The boxes are randomly shuffled, so the goal is to reorder them by ending with the empty one at the first position. The goal is also to reach the final result with the least number of moves.

In order to solve the problem, the required variables must first be defined:

STATE: the state of a numbered tile is the position of the tile in a 3x3 array in which there are the numbers 1 to 8 and the empty tile is indicated by 'E'. This is represented with a string containing dashes so that we can know the conformation of the puzzle at each given state.

ACTION: To make the implementation easier, we use the movement of the empty tile as the action, instead of the numbered ones. Since there are four possible movements (up, down, left, right), we need to check the boundary conditions to see which one is valid in the state we are in.

GOAL TEST: check if the current state is the goal state, i.e. "E-1-2-3-4-5-6-7-8".

PATH COST: Since cost is the number of moves used to solve the problem, the cost of one move will correspond to each step.

Implementation and Results

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

```

```
# 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 to the empty space
    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], rows[row_empty][col_empty]
        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 manhattan distance
    # 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
    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
```

```
# 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'''

# 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)

# Create the A* solver object
result = astar(PuzzleSolver(INITIAL))
# result = greedy(PuzzleSolver(INITIAL))
# result = breadth_first(PuzzleSolver(INITIAL))
# result = depth_first(PuzzleSolver(INITIAL))

# Print the results
for i, (action, state) in enumerate(result.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!')
```

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

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

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

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

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

Step 5: After moving 8 into the empty space
1-4-2
```

```
3-5-E
6-7-8

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

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

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

Discussions

Simple case

In this task, we considered the simple case '1-4-2-3-5-8-6-7-E'. Using the A*, greedy and breadth-first search methods, we obtained a solution with a cost of 4 moves, instead the depth-first search method gives us a runtime error.

Using heuristics with the number of misplaced tiles, the A* search gave us a run time of 0.000246s, the greedy 0.000192s and breath-first search 0.000724s. So we can draw the conclusion that the greedy method is the fastest, and the breath-first the slowest.

Using instead a heuristic value for the sum of the distances of the tiles from their correct positions, we obtain a runtime of 0.000290s for A* search, 0.000314s for greedy search and 0.000714s for breath-first search. In this case we can conclude that A* gave the fastest solution and again breadth-first is the one with the highest runtime.

Difficult case

This task consists of implementing the A* method and greedy search starting with the case '7-2-4-5-E-6-8-3-1'.

Following the same approach as in the previous case and using heuristics for the numbers of misplaced tiles, both search methods give us a run time error.

Using the heuristic for the sum of the distances of the tiles from their target position, the A* method gives a run time of 22.5s, with 26 moves taken, while the greedy search still gives us an error.

Run time errors occurred because the nodes generated in the search were too excessive to be controlled and the calculation became too heavy for the memory and thus the RAM. Or the wrong heuristic values were used, which made it too difficult to solve the problem.

Lab 2. Search

Task 2.3 Maze Solver

Problem Descriptions

What the algorithm has to solve is the delineation of a route that can take a person from a starting position to a given destination. Within the map, the starting position is represented by 'O' and each movement is indicated by 'x'. All obstacles that must then be avoided will be illustrated with "#". To start with, we define the state, action, goal test and path cost.

STATE: is the position on the map indicated by the x and y co-ordinates.

ACTION: is the movement that takes the person towards the destination. Possible movements are UP, DOWN, RIGHT, LEFT always considering obstacles close to the current state. To define diagonal movements for extended searches we can also use the commands LEFT UP. LEFT DOWN; RIGHT UP or RIGHT DOWN.

GOAL TEST: checks whether the co-ordinates of the destination point correspond with those of the current state.

PATH COST: The cost assigned to each movement is 1, but for diagonal movements in extended search, it is 1.4.

Implementation and Results

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

```
    Requirement already satisfied: simpleai in /usr/local/lib/python3.6/dist-packages (0.8.2)
```

```
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,
}

```

```

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)
```

```
problem = Maze(MAP)
result = astar(problem, graph_search=True)
# result = greedy(problem, graph_search=True)
# result = breadth_first(problem, graph_search=True)
# result = depth_first(problem, graph_search=True)
# result = uniform_cost(problem, graph_search=True)

path = [x[1] for x in result.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("A* Search: moves=%s, cost=%s." %(result.depth, result.cost))
```

```
##############################
#       #           #  #
# ####   ########     #  #
#  o.#   # ........    #  #
#   .### ....####  .######   #
#   ......####     .#       #
#          #  #  .#   #### #
#    ######   #  .....#.x  #
#      #    #     ...   #
##############################
A* Search: moves=33, cost=33.0.
```

Discussions

In this task, we could consider the direct distance between the current position and the target as a heuristic function. This distance can be calculated using the Pythagorean theorem.

The first method used for solving the task was A*search, which gave a cost result of 33, also taking 33 moves. It is interesting to see how using the other methods all give us the exact same result, both in terms of cost and number of moves, but the calculated paths to the destination are different. However, if more obstacles were added in different positions, the A* serach would be the method that would still give us an optimal result.

If we consider the case where diagonal moves are added, in contrast, the values obtained earlier are reduced from 33 moves to 23, and from 33 to 26.93 in cost. Furthermore, if we use the breadth-first and uniform-cost methods, we obtain the same results. With the greedy search, on the other hand, the cost is 26.96, with the same number of moves, while with the depth-first search it gives us a higher number of moves, i.e. 38, and also a higher cost, of 49.597. The increase in cost and moves in the depth-first search is due to the fact that the algortmo had to go back after exploring all the nodes all the way through. Concluding, we can select the A* search as the optimal method over the others.

Lab 3. Genetic Algorithms

Task 3.1 The Travelling Salesman Problem

Problem Descriptions

The problem is based on a salesman who has to visit a number of cities, 17 in this case, and has to find the best route in order to minimise the total distance of the travel and passing through each of the cities. To solve the problem, we know the distance between the various cities. The optimal solution for this problem would be:

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

OptDistance = 2085

For the implementation of a genetic algorithm, it is necessary to define the encoding scheme, fitness function and genetic operators.

ENCODING SCHEME: We use symbolic encoding, i.e. the permutation of the 17 numbers representing the cities.

FITNESS FUNCTION: it is the sum of the distances between the cities that are neighbouring, also considering the distance between the last city and the first as the seller has to complete the round.

GENETIC OPERATORS: as a selection method we can use tournament selection, e.g. selecting the best three chromosomes. For crossover we can randomly choose a city within a string so that the next bits are swapped with another parent. However, we must consider maintaining string validity, i.e. all cities can only be present once. For mutation, it is sufficient to exchange two cities within the same string.

Implementation and Results

```
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, 518],
    [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, 297],
    [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, 236],
    [505, 289, 262, 476, 196, 360, 444, 402, 495, 0, 154, 578, 439, 336, 240, 685, 390],
    [353, 282, 110, 324, 61, 208, 292, 250, 352, 154, 0, 435, 287, 184, 140, 542, 238],
    [324, 638, 437, 240, 421, 329, 297, 314, 95, 578, 435, 0, 254, 391, 448, 157, 301],
    [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, 336],
    [121, 518, 142, 84, 297, 35, 29, 36, 236, 390, 238, 301, 55, 96, 153, 336, 0]]
```

```
!pip install deap
import array
import random
import numpy as np
from deap import algorithms, base, creator, tools
```

```
# fitness function as the sum of all the distances between each consecutive cities in individual
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.indices)
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)

```

```
random.seed(16)
pop = toolbox.population(n=500)
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("min", np.min)
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)

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

Discussions

Looking at the final output, we can see that the solution was found after 50 generations, with a total distance of 2085. This is equivalent to the optimal distance of the solution described at the beginning, although the sequence of cities is not the same, probably because there is more than one combination that leads to the same result.

Lab 3. Genetic Algorithms

Task 3.2 The 8-Queens Problem

Problem Descriptions

The 8-queens problem is based on placing 8 queens on the chessboard so that they cannot attack each other; to do this, there cannot be more than one piece on the same row, column or diagonal. First we define the encoding scheme, fitness value and genetic operators.

ENCODING SCHEME: We can consider three encoding schemes in this problem. In the first we do not set any of the three constraints (row, column or diagonal), so the algorithm will do all the work.

In the second encoding scheme we consider the 8 queens placed in different columns, in order from 0 to 7, and in this way we already satisfy one constraint. Since we may end up with numbers of equal values in the list as they are randomly generated, it will leave the genetic algorithm to enforce the row and diagonal constraints.

Scheme 3 applies both the column constraint (as scheme 2) and the row constraint, so the genetic algorithm only needs to satisfy the diagonal constraint.

FITNESS FUNCTION: as a fitness function we can consider the number of queens attacking each other, and so the algorithm will have to minimise this outcome by choosing the individuals with the smallest number, up to 0. Using encoding scheme 3, we need only worry about the queens on the diagonals, and since in an 8x8 chessboard there are 15 falling diagonals and 15 raising diagonals, we can use two lists of size 15 to store the number of attacking queens. In the 8x8 chessboard, we have 15 (8x2-1=15) falling diagonals and 15 raising diagonals. So we can use two lists of size 15 to store the number of attacking queens. For a queen located at (x, y), the index to the falling diagonal is x+y, and that to the raising diagonal is x-y+7.

GENETIC OPERATORS: Since it is a minimisation problem as the solution is found when no queen attacks another, we can use tournament selection as the selection method. We can apply crossover by swapping two parts of two chromosomes, which means swapping the positions of some queens in two different settings of the chessboard. We must always respect the constraints imposed earlier, i.e. we must apply the crossover without having two queens on the same row or column. For the mutation we will instead have to switch two queens within the same individual.

Implementation and Results

```
!pip install deap
import random
import numpy as np
from deap import algorithms, base, creator, tools

```

```
NB_QUEENS = 8
#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]),

```

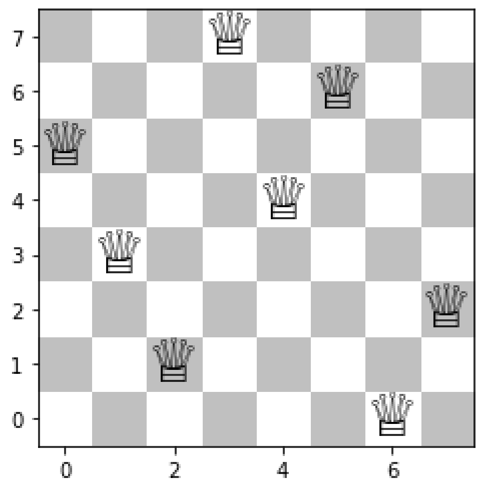
```
# enforce only 1 queen per column by using a list of NB_QUEENS
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

# 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.permutation)
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)
```

```
# run the GA to get the result
random.seed(64)
pop = toolbox.population(n=300)
algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=50, verbose=False)
best = tools.selBest(pop, 1)[0]
print("Best: %s. Fitness value: %s" %(best, best.fitness.values[0]))
```

```
Best: [5, 3, 1, 7, 4, 6, 0, 2]. Fitness value: 0.0
```

```
# 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
```

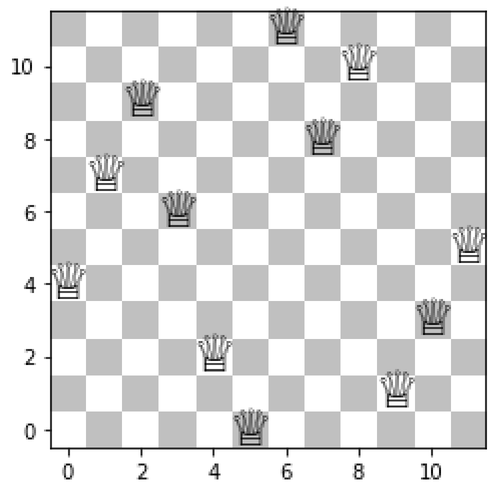
```
plt.imshow(chessboard, cmap='gray', origin='lower', vmin=0, vmax=1)
for x in range(NB_QUEENS):
    plt.text(x, best[x], '♕', fontsize=30, ha='center', va='center')
```



```
# run the GA to get the result
random.seed(64)
pop = toolbox.population(n=300)
algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=50, verbose=False)
best = tools.selBest(pop, 1)[0]
print("Best: %s. Fitness value: %s" %(best, best.fitness.values[0]))
```

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

```
# 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=25, ha='center', va='center')
```



Discussions

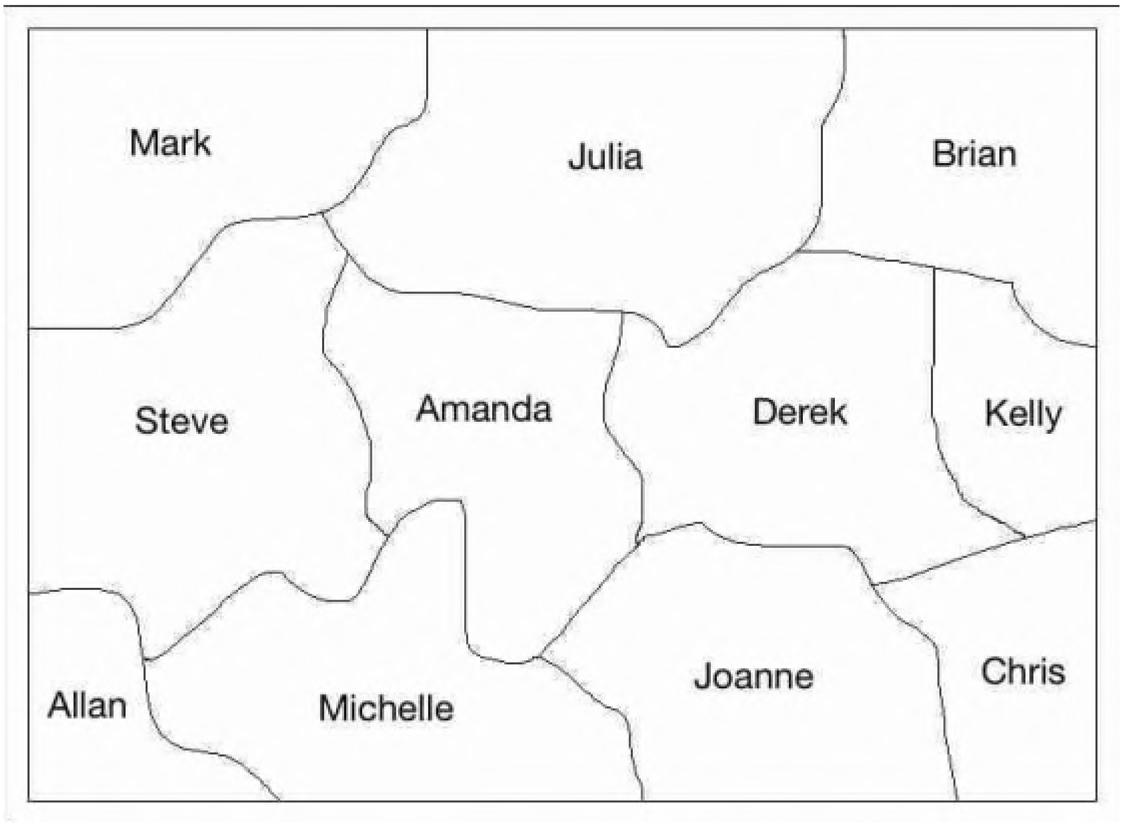
In this task, a genetic algorithm has been implemented which is able to place a given number of queens on a chessboard so that they do not attack each other. In particular, the case of an 8x8 chessboard with 8 queens was first considered and run, after which the number of queens was expanded to 12, and the chessboard then became 12x12.

Lab 3. Genetic Algorithms

Task 3.3 The Map Colouring Problem

Problem Descriptions

The map colouring problem consists of using a given number of colours to colour a map, while taking into account that there must be no bordering regions with the same colour. In this task, there are 11 regions to be coloured and the colours that can be used are 4.



To define the bordering regions, we can use an upper triangular matrix where 1 stands for neighbouring and 0 does not. Choosing an upper triangular matrix over a full matrix is due to neighborhood mutuality, so we set the bottom of the matrix to zero. The encoding scheme, fitness function and genetic operators can now be defined.

ENCODING SCHEME: you can for instance use a list of 11 regions and assign each one a number from 0 to 3, representing the colour.

FITNESS FUNCTION: the number of neighbouring regions with the same colour has to be evaluated, in order to always have a lower number until zero is reached.

GENETIC OPERATORS: in this problem one could use tournament selection as a selection method, because the aim is to minimise the result. Cross over is then applied between two individuals thus exchanging the colours assigned to the regions and mutation, on the other hand, is applied by changing a number, that is a colour, within the same list of regions.

Implementation and Results

```
!pip install deap
import array
import random
import numpy as np
from deap import creator, base, tools, algorithms
```

```
# 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]]

```

```
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_colour, numNames)
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)

```

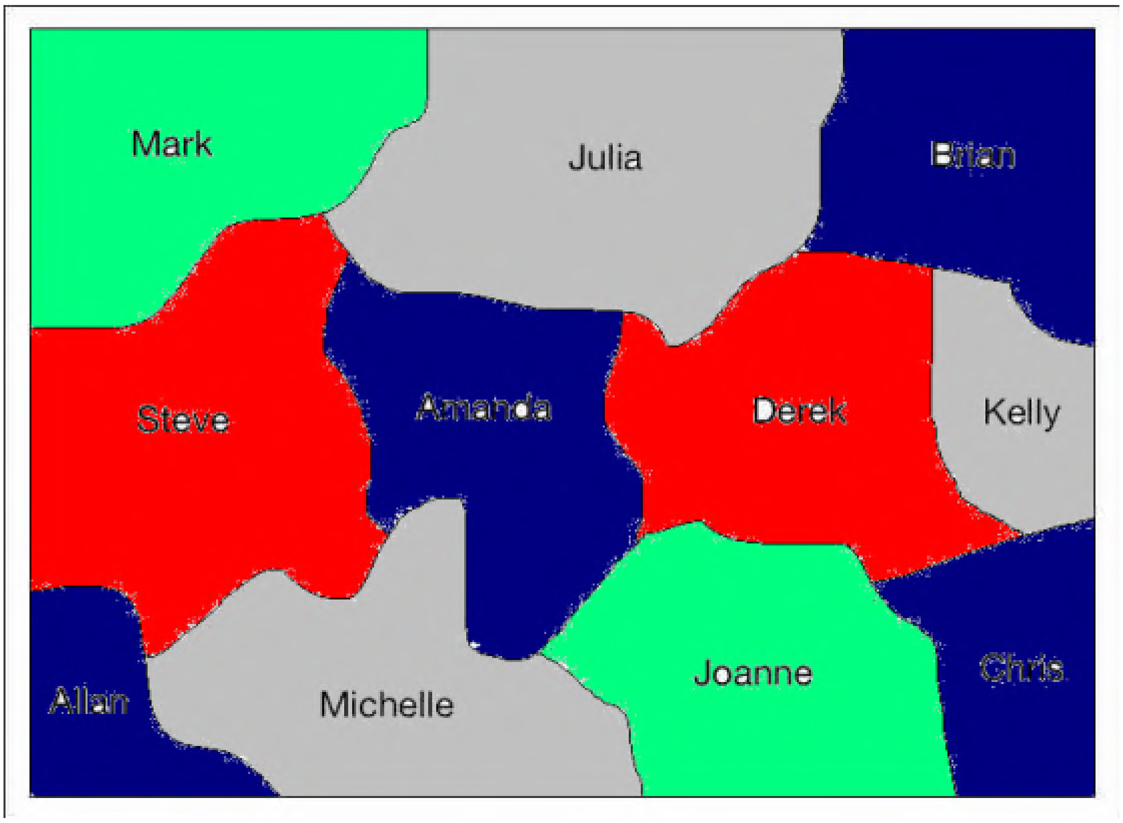
```
pop = toolbox.population(n=10)
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("min", np.min)
pop, log = algorithms.eaSimple(pop, toolbox, cxpb=0.8, mutpb=0.4, ngen=100,
                               stats=stats, verbose=True)

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	2
1	10	0
2	10	1
3	9	1
4	9	1
5	8	0
6	8	0
7	9	0
8	8	0
9	10	0
10	7	0
11	10	0
12	8	0
13	6	0
14	8	0
15	8	0
16	10	0
17	10	0
18	10	0
19	9	0
20	7	0
21	9	0
22	10	0
23	7	0
24	10	0
25	8	0
26	8	0
27	10	0
28	10	0
29	8	0
30	9	0
31	8	0
32	6	0
33	10	0
34	8	0
35	8	0
36	10	0
37	10	0
38	8	0
39	8	0
40	9	0
41	10	0
42	10	0
43	9	0
44	9	0
45	10	0
46	8	0
```

```
47   9     0
48   8     0
49   10         0
50   8     0
51   7     0
52   8     0
53   10         0
54   10         0
55   10         0
56   10         0
```

Discussions

As a final output we obtain the hundred generations it took to arrive at the solution, in which the fitness function is therefore zero. The solution corresponds to a string of 11 numbers from 0 to 3, which then indicates the 11 regions to which the 4 colours have been assigned in such a way that there are no equal neighbouring colours.



Lab 4. Probabilistic Inference

Task 4.1 Iris Classification Using Naïve Bayes

Problem Descriptions

In this task a classification problem using the Naive Bayes terorem has to be solved. The Iris flow dataset, made famous by Fisher, will be used. It contains five variables, four are flower features, i.e. sepal length, petal length, sepal width and petal width, the fifth is the species, three in total. The task is therefore based on predicting the species of a particular flower based on the characteristics contained in the dataset.

The algorithm will first be run using two of the four flower characteristics contained in the dataset (sepal length and petal length), for simplicity and clarity's sake, then with all four.

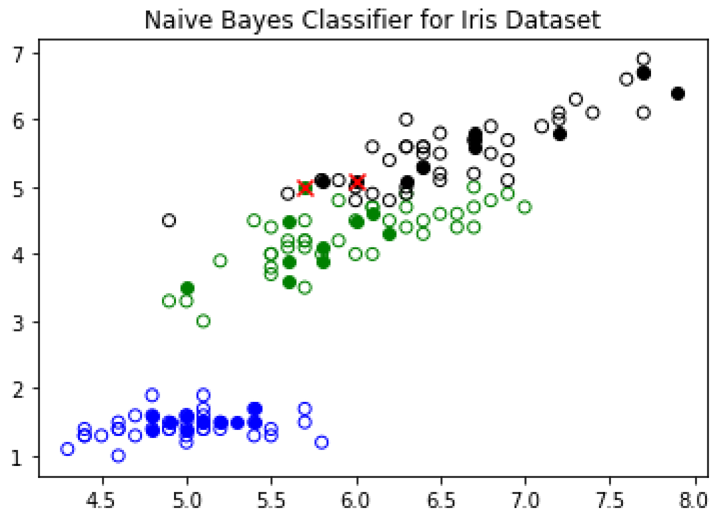
Implementation and Results

```
!pip install sklearn
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

```

```
# 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 naive Bayes model
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 the results.
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.sigma_)

```

```
Accuracy: 0.9333333333333333
Confusion Matrix:
 [[10  0  0]
 [ 0  9  1]
 [ 0  1  9]]
Prior:
 [0.33333333 0.33333333 0.33333333]
Mean:
 [[4.985  1.4475]
 [5.9775 4.275 ]
 [6.5575 5.5275]]
Variance:
 [[0.138275   0.03399375]
 [0.29074375 0.214375  ]
 [0.37194375 0.29849375]]
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:103: FutureWarning: Attribute `sigma
  warnings.warn(msg, category=FutureWarning)
```

```
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', edgecolors=colour)
  test_data = X_test[Y_pred == i]
  plt.scatter(test_data[:,0], test_data[:,1], marker='o', facecolors=colour, edgecolors=colour)

error_data = X_test[Y_pred != Y_test]
plt.scatter(error_data[:,0], error_data[:,1], marker='x', facecolors='r', edgecolors='r', s=64)

#ellipse = patches.Ellipse(xy=nb.theta_[0], width=nb.sigma_[0][0]*10, height=nb.sigma_[0][1]*10, edgecolor='b
#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()
```

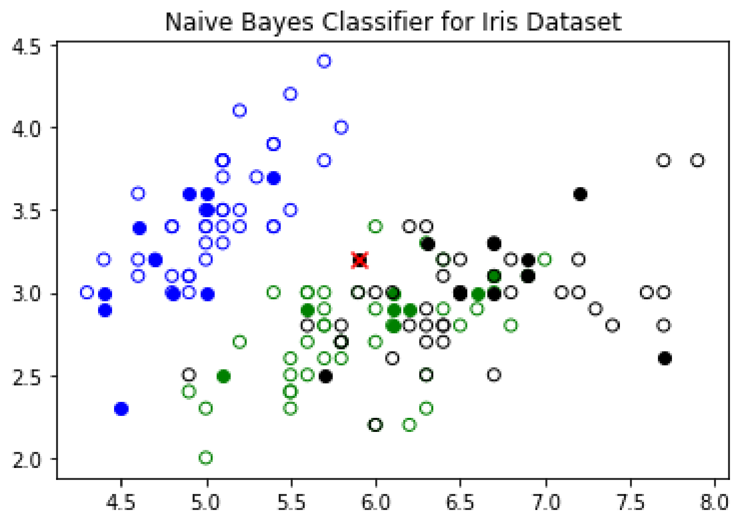


```
# Now we use all four features
iris = datasets.load_iris()
X = iris.data
#X = iris.data[:, [0,2]]
Y = iris.target

# split the dataset into training (80%) and testing (20%), and fit the data into the naive Bayes model
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 the results.
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.sigma_)
```

```
Accuracy: 0.9666666666666667
Confusion Matrix:
 [[11  0  0]
 [ 0  9  1]
 [ 0  0  9]]
Prior:
 [0.325      0.33333333 0.34166667]
Mean:
 [[5.06666667 3.49230769 1.46666667 0.24358974]
 [5.8925     2.7275     4.245      1.3125    ]
 [6.55609756 2.95365854 5.52926829 1.97560976]]
Variance:
 [[0.11504274 0.11763314 0.03452992 0.0096384 ]
 [0.26019375 0.10299375 0.193975   0.03909375]
 [0.41612136 0.09858418 0.28743605 0.06916122]]
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:103: FutureWarning: Attribute `sigma
  warnings.warn(msg, category=FutureWarning)
```

```
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', edgecolors=colour)
  test_data = X_test[Y_pred == i]
  plt.scatter(test_data[:,0], test_data[:,1], marker='o', facecolors=colour, edgecolors=colour)

error_data = X_test[Y_pred != Y_test]
plt.scatter(error_data[:,0], error_data[:,1], marker='x', facecolors='r', edgecolors='r', s=64)

# ellipse = patches.Ellipse(xy=nb.theta_[0], width=nb.sigma_[0][0]*10, height=nb.sigma_[0][1]*10, edgecolor='
# # 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()
```



Discussions

Naive Bayes has been used to formulate the algorithm, estimating the posterior probability of the species given the features, from the prior probability of each iris class and the conditional probability of the features taken separately, given the species.

To illustrate and make the result clearer, the data can be plotted in a scatter plot that differentiates the species with different colours, and uses solid markers for the testing set, empty markers for the training set, and red crosses for the misclassified patterns. The output showed that there was no clear distinction between two of the three species analysed, and therefore we obtained errors in their intersection boundary.

The first graph shows how, considering only the length of the sepals and petals, the species differ because they have different values, i.e. one has generally shorter petals and sepals than the others, one has intermediate values and several individuals of the last species have higher values. Considering the second graph, in which the classification was made according to all four characteristics, a more random distribution of the individuals is observed, meaning that the species that had a shorter length of petals and sepals compensates with a greater width.

Lab 4. Probabilistic Inference

Task 4.2 Diabetes Diagnosis Using Naïve Bayes

Problem Descriptions

In this task, a medical diagnosis problem will be tackled, where the goal is to predict whether a patient has a disease or the level of disease progression. A diabetes dataset will be used, which contains 10 features and a quantitative measurement of diabetes progression. Patient observations correspond to personal data such as age, gender, height, weight, tests like blood sugar, heart rate, lipoproteins and symptoms e.g. fever or headache.

Implementation and Results

```
!pip install sklearn
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
```

```
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.sigma_)

```

```
Accuracy: 0.6756756756756757
Confusion Matrix:
 [[47 19  0]
 [ 9 27  3]
 [ 1  4  1]]
Prior:
 [0.51963746 0.45619335 0.02416918]
Mean:
 [[-0.00526112 -0.00086013 -0.02101334 -0.01711349 -0.01028066 -0.00942024
   0.01451003 -0.01695366 -0.02041805 -0.01636978]
 [ 0.00819768  0.0008098   0.01752732  0.01635464  0.00827229  0.00545096
  -0.01389898  0.01414276  0.02383294  0.01368027]
 [ 0.01673474  0.02684968  0.06627691  0.0360742   0.00582679  0.00072159
  -0.03649777  0.03707644  0.04692384  0.06985588]]
Variance:
 [[0.00221154 0.00225651 0.00146332 0.00166069 0.00212625 0.00201469
  0.00261449 0.00155957 0.00165128 0.00191088]
 [0.00218137 0.00226668 0.00203968 0.00217859 0.00239541 0.00236454
  0.0016172  0.00212146 0.00188261 0.00219692]
 [0.00168715 0.00170367 0.00112269 0.002369   0.00088885 0.00094093
  0.00037423 0.00182382 0.00121177 0.0020693 ]]
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:103: FutureWarning: Attribute `sigma_` was deprecated in 1.0 and will be removed in1.2. Use `
  warnings.warn(msg, category=FutureWarning)
```

Discussions

To simplify the resolution of the task, to make the progression measures classifiable, the original data were converted into three categories: 0 to 150, 150 to 300 and 300 and upwards. They were then divided into training set and testing set with a 3-1 split.From the output obtained, it can be seen that in a realistic problem like this, the accuracy of the result hardly exceeds 0.70, due to the complexity of the dataset.

Problem Description:

Describe the problem here...

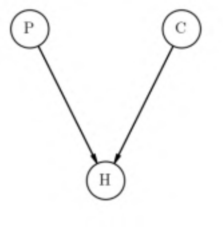
Lab 4. Probabilistic Inference

Task 4.3 Monty Hall Problem Using Bayesian Network

Problem Descriptions

The monty hall problem is taken from a famous American TV show and is based on the contestant's probability of getting the prize by choosing between three doors, two of which hide a goat and one a car (the prize). The contestant is initially asked to choose a door, consequently the host will open one of the two remaining doors, where he knows there is no prize. Now the contestant is offered to swap the door chosen at the beginning for the remaining one, and it is on this choice that the game is based. This is because by swapping the door the contestant actually has twice the chance of winning, even though it may seem that the probability is 50%.

In order to solve this task, we can use the Bayesian Network, and therefore need to define three variables. Furthermore, the three doors will be identified as 0,1 and 2. The first variable is the prize (P), which can hide behind each door, and so the prior probability is 1/3 for all three. The other variable is the contestant (C), i.e. his initial choice, and again the prior probability is 1/3. The last variable is the action performed by the host H, which depends on the position of the prize and the choice of the contestant. The Bayesian Network will thus be composed as in the following picture.



Later on, a variant of the game will also be implemented, with the same rules, but considering four doors instead of three.

Implementation and Results

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

```
# 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,   0.5, 1/3, 0.5, 0,   0.5, 0.5, 1/3],
#                                [1/3, 0,   0.5, 0.5, 0,   0,   0,   0,   0.5, 0,   1/3, 0.5, 0.5, 0,   0.5, 1/3],
#                                [1/3, 0.5, 0,   0.5, 0.5, 1/3, 0,   0.5, 0,   0,   0,   0,   0.5, 0.5, 0,   1/3],
#                                [1/3, 0.5, 0.5, 0,   0.5, 1/3, 0.5, 0,   0.5, 0.5, 1/3, 0,   0,   0,   0,   0]],
#                    evidence=['Contestant', 'Prize'], evidence_card=[4, 4])

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

```

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

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

```
+----------+--------------+
| Prize    |   phi(Prize) |
+==========+==============+
| Prize(0) |       0.3333 |
+----------+--------------+
| Prize(1) |       0.6667 |
+----------+--------------+
| Prize(2) |       0.0000 |
+----------+--------------+
```

```
#We now implement the case with 4 doors

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

# Defining the CPDs:

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,    0.5,  1/3, 0.5,  0,    0.5,  0.5,  1/3],
                               [1/3, 0,    0.5,  0.5,  0,    0,   0,    0,    0.5,  0,    1/3, 0.5,  0.5,  0,    0.5,  1/3],
                               [1/3, 0.5,  0,    0.5,  0.5,  1/3, 0,    0.5,  0,    0,    0,   0,    0.5,  0.5,  0,    1/3],
                               [1/3, 0.5,  0.5,  0,    0.5,  1/3, 0.5,  0,    0.5,  0.5,  1/3, 0,    0,    0,    0,    0]],
                  evidence=['Contestant', 'Prize'], evidence_card=[4, 4])

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

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

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

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

Discussions

The output obtained depicts the probability of each door after one has been opened by the host. In the first case where there are three doors in total, we see how the result shows that the probability of the prize being behind the second door increases by 33%, and is therefore double the probability of the prize being hidden behind the door chosen by the contestant. This means that it is always better to change the initial choice in this game situation.

In the variant with 4 doors, on the other hand, the odds of finding the prize are distributed differently. However, we can see that the contestant still has less chance of winning if he stays with his initial choice, with which he has a 25% chance of winning. This is because, in this case, there being two remaining doors, the probability of finding the prize by changing doors is 37% (for both doors). Therefore, although the probability of winning in general is reduced, even in this situation it is better for the contestant to change his choice, and it is indifferent with which of the two remaining doors he decides to change.

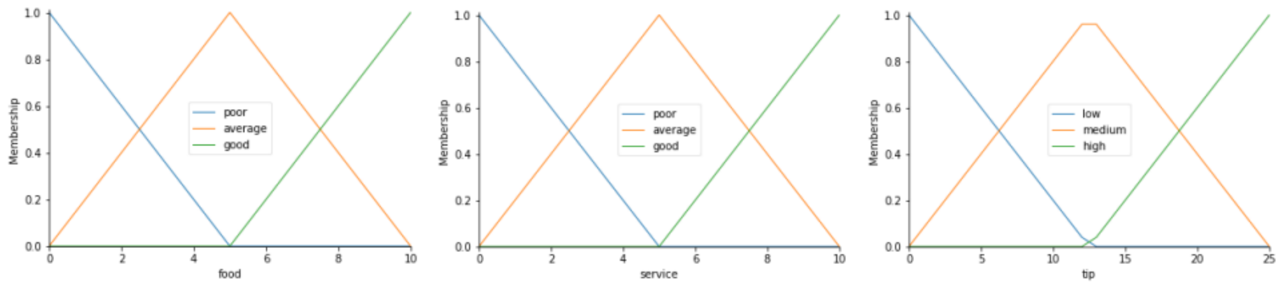
Lab 5. Fuzzy Systems

Task 5.1 The Tipping Problem

Problem Descriptions

The tipping problem consists of working out, using a fuzzy system, what tip to give a restaurant, based on the quality of service and food. So in total we have two linguistic variables as input, namely service and food, and one variable as final output which corresponds to the tip. For food and service we have the values poor, average, and good. For the tip they are low, medium, and high. Moreover, “food” and “service” are defined in the scale of 0-10, while “tip” in the range of 0-25%.

The membership of the linguistic variables are defined in the same way for simplicity and are depicted in the following image:



There are three rules:

1. IF the service was good

   OR the food quality was good

   THEN the tip will be high.

2. IF the service was average

   THEN the tip will be medium.

3. IF the service was poor

   AND the food quality was poor

   THEN the tip will be low.

Implementation and Results

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

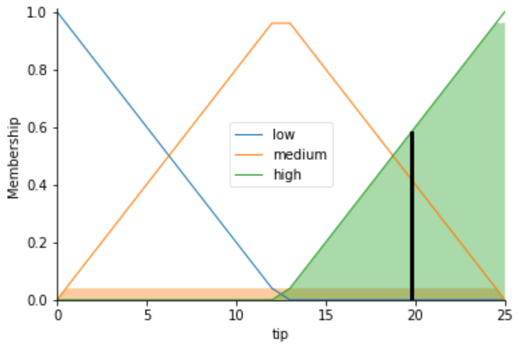
```
# New Antecedent/Consequent objects hold universe variables and membership functions
food = ctrl.Antecedent(np.arange(0, 11, 1), 'food')
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# 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()
```

```
# 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)
```

```
# 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 test the functioning of the algorithm, two judgement values were chosen to be assigned to the input variables, namely 6.5 for food and 9.8 for service.

The solution obtained is a tip of 19.8% and as the graph depicts, the value belongs to the "high" range of the output variable, which is tip.

Lab 5. Fuzzy Systems

Task 5.2 Project Risk Assessment

Problem Descriptions

The problem addressed in this task regards the project risk assessment, i.e. it consists of calculating the risk of a project, which in this case can be low, normal or high by having project funding (inadequate, marginal, adequate) and staffing (small, large) as liguistic variables. In addition, the following rules will be applied:

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

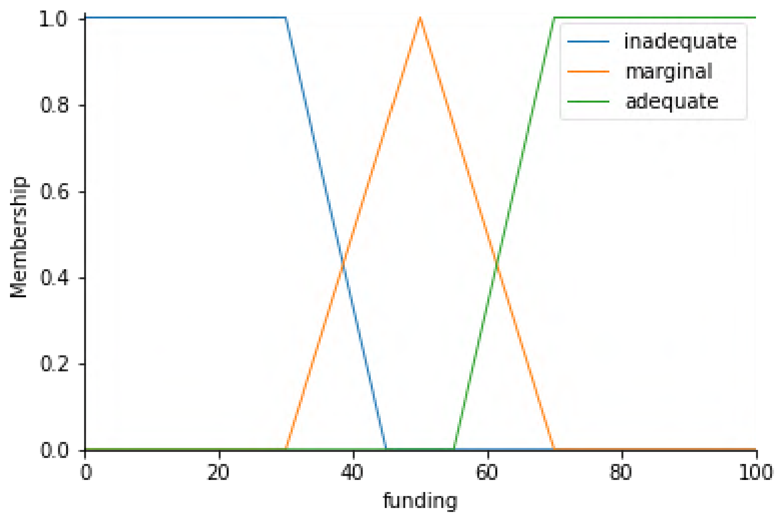
   THEN risk is high

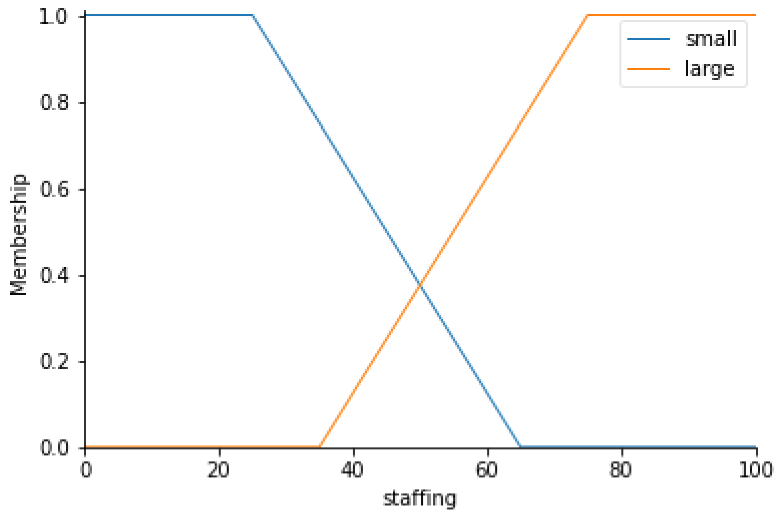
Implementation and Results

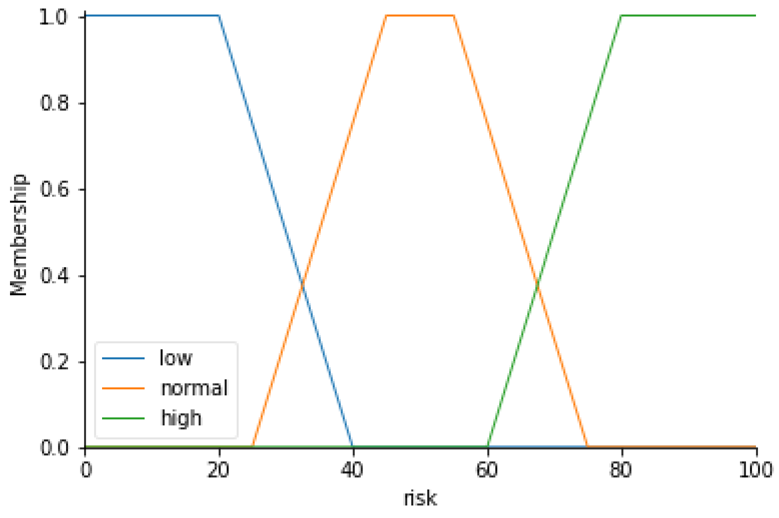
```
!pip install scikit-fuzzy
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

```
# 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()
```





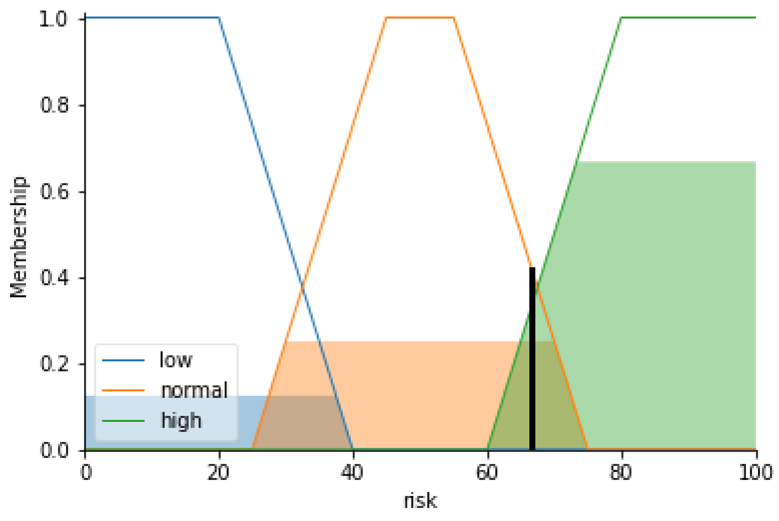


```
# Define the rules
rule1 = ctrl.Rule(funding['adequate'] | staffing['small'], risk['low'])
rule2 = ctrl.Rule(funding['marginal'] & staffing['large'], risk['normal'])
rule3 = ctrl.Rule(funding['inadequate'], risk['high'])
# rule4 = ctrl.Rule(funding['inadequate'] & staffing['large'], risk['high'])

# 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)
```

```
# 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)

```

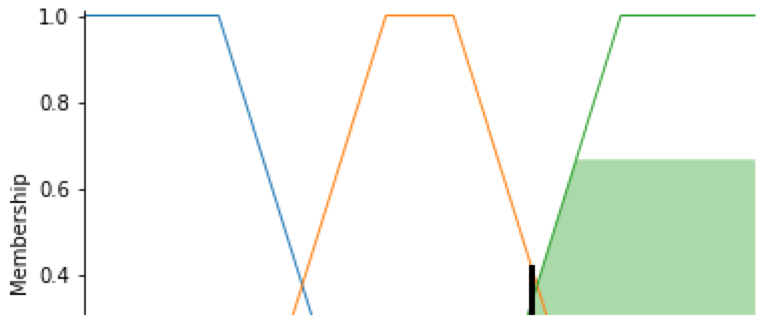
```
Project risk: 66.66529281135837 %
```



```
# We now add another rule
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, rule4])
ctrl_sim = ctrl.ControlSystemSimulation(ctrl_sys)
```

```
# 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

In this task, the risk of a project was calculated using funding and staffing as variables and implementing an algorithm with fuzzy logic. Two cases were evaluated, the first by applying three rules as described above, in the second a fourth rule was then added, the following: IF project funding is inadequate AND project staffing is large THEN risk is high In both cases, the project risk (output variable) was calculated considering funding=35% and staffing=60%. The solution obtained is the same for both cases implemented, i.e. the risk is 66.7%.

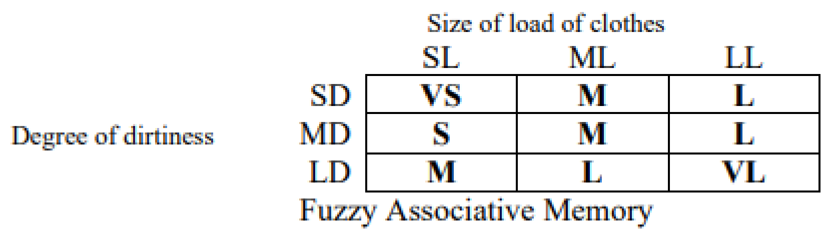
Lab 5. Fuzzy Systems

Task 5.3 Washing Machine Fuzzy Controller

Problem Descriptions

The objective of this task is to programme a washing machine controller following fuzzy logic. We will again use two inpute variables, namely degree of dirtiness and size of load of clothes, and one output variable, the washing time. The linguistic values are the following: Degree of Dirtiness: SD(small), MD(medium), LD(large) Size of Load of Clothes: SL(small), ML(medium), LL(large) Washing Time: VS(very short), S(short), M(medium), L(long), VL(very long)

In this case, the rules are contained in a Fuzzy Associative Memory (FAM):



Size of load of clothes

Degree of dirtiness	SL	ML	LL
SD	**VS**	**M**	**L**
MD	**S**	**M**	**L**
LD	**M**	**L**	**VL**

Fuzzy Associative Memory

The rules in the table can be interpreted as in the following example:

IF dirtiness is SD

AND load is SL

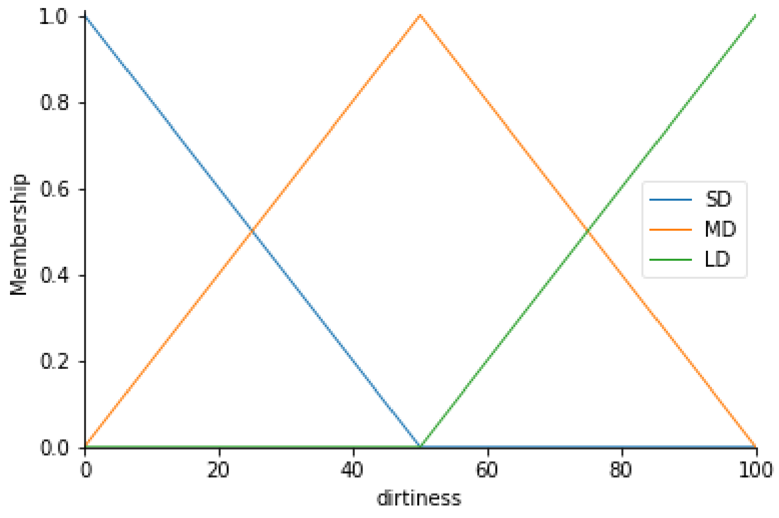
THEN washing time is VS

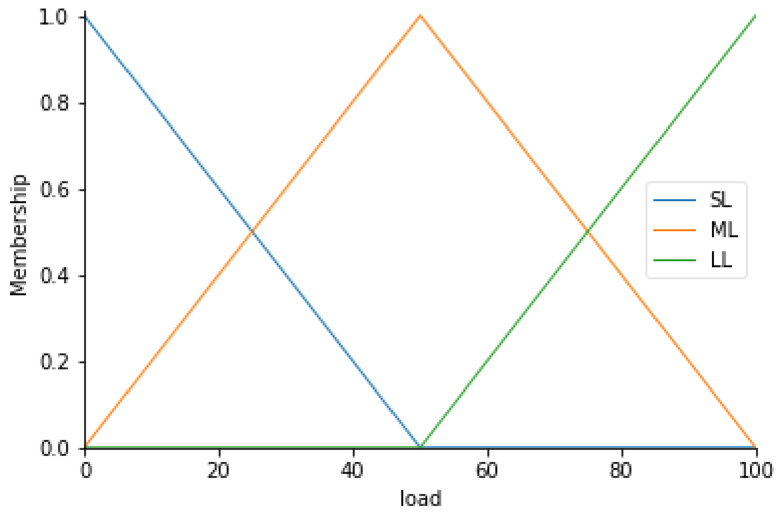
Implementation and Results

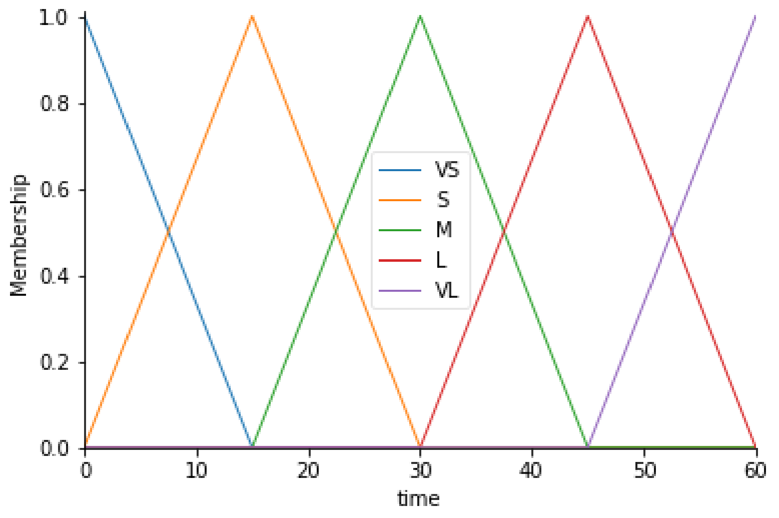
```
!pip install scikit-fuzzy
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

```
# 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()
```





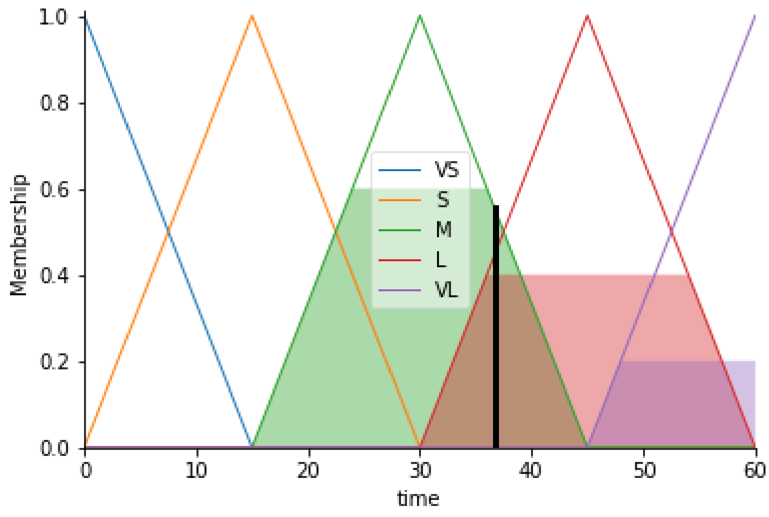


```
# Define the rules
rule1 = ctrl.Rule(dirtiness['SD'] & load['SL'], time['VS'])
rule2 = ctrl.Rule(dirtiness['SD'] & load['ML'], time['M'])
rule3 = ctrl.Rule(dirtiness['SD'] & load['LL'], time['L'])
rule4 = ctrl.Rule(dirtiness['MD'] & load['SL'], time['S'])
rule5 = ctrl.Rule(dirtiness['MD'] & load['ML'],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

```
# 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

In this task, a washing machine controller was implemented, which is based on two variables: the degree of dirtiness and the size of load of clothes. To test the operation of the control system, two values were chosen for the two variables, namely 60% for the degree of dirtiness and 70% for the size of load. The result obtained refers to the output variable, the washing time, and is 36.7.

Lab 6. Natural Language Processing

Task 6.1 Sentiment Analysis

Problem Descriptions

Thi task consists of implementing one of the most widespread problems in the field of natural language processing, sentimen analysis, which, however, does not always prove easy to tackle, due to the different writing styles, the various slangs and the different meanings that certain expressions may have. The text to be analysed here is taken from the "movie_review" dataset, which is a dataset with 1000 positive and 1000 negative reviews.

Implementation and Results

```
!pip install nltk
import nltk
from nltk.corpus import movie_reviews

nltk.download('movie_reviews')
```

```
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
```

```
# 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)
```

```
# Calculate the tf-idf features from the training set
tfidf = TfidfVectorizer(use_idf=True)
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, 36323)
```

```
# 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       0.85      0.72      0.78       202
         neg       0.75      0.87      0.81       198

    accuracy                           0.80       400
   macro avg       0.80      0.80      0.79       400
weighted avg       0.80      0.80      0.79       400
```

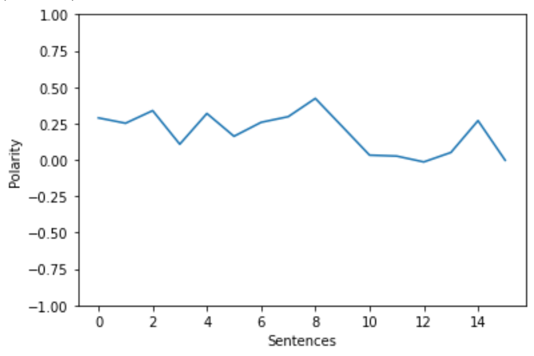
```
print(X_train[0])
print(testing_tfidf[0])
```

```
this summer , one of the most racially charged novels in john grisham's series , a time to kill , was made into a major motion picture .
on january 3 of this year , director rob reiner basically re-released the film under the title of ghosts of mississippi .
based on the true story of 1963 civil rights leader medgar evars' assassination , ghosts of mississippi revolves around the 25-year legal battle faced by myrlie eva
so she turns to assistant district attorney and prosecutor bobby delaughter ( alec baldwin , heaven's prisoners ) to imprison the former kkk member .
```

```
ghosts sets its tone with an opening montage of images from african-american history , from slave-ship miseries to life in the racist south of the 1960's .
but all too soon , the white folks take over , intoning lines like " what's america got to do with anything ?
this is mississippi ! "
as beckwith , james woods , with his head larded with latex most of the time as an old man , teeters between portraying evil and its character .
meanwhile , goldberg turns in a very serious and weepy performance as the wife who wouldn't let her husband's death rest until she got the conviction .
both deserve serious oscar-consideration .
this brings us to the dull performance of baldwin .
let's face it , trying to match matthew mcconaughey's wonderful acting in a time to kill is basically impossible .
and baldwin is living proof of this , as no emotions could be felt .
it seemed as if he actually had to struggle to shed a single tear .
either poor acting or poor directing , but something definitely went wrong .
another strange mishap was the fact that goldberg's facial features didn't change , as she looked the same in the courtroom as she did holding her husband's de
yet woods' was plastered with enough make up to make him look like goldberg's father .
at least the make-up was realistic .
with some emotional moments in the poorly written script , ghosts of mississippi lacked in heart , when its predecessor , a time to kill , brought tears to everyone
don't get me wrong , the movie wasn't all that bad , but if you've seen grisham's masterpiece , then don't expect this one to be an excellent film .
 ,

  (0, 36159)	0.024837241123616738
  (0, 36118)	0.02496890661256656
  (0, 35916)	0.01597939248903686
  (0, 35880)	0.04634574818967332
  (0, 35807)	0.04439093334039668
  (0, 35798)	0.03403515751271626
  (0, 35738)	0.02127478922613567
  (0, 35645)	0.0478429542949432
  (0, 35639)	0.041540331049150586
  (0, 35613)	0.01703286568659344
  (0, 35461)	0.05432866927690065
  (0, 35457)	0.058892525187043396
  (0, 35381)	0.020847313598640227
  (0, 35359)	0.033339353193436234
  (0, 35217)	0.22746568301190154
  (0, 35188)	0.050970145942871914
  (0, 35168)	0.013186572883856447
  (0, 34651)	0.03602860975817351
  (0, 34292)	0.055269121495952644
  (0, 34205)	0.0701393734376125
  (0, 33509)	0.034090387972316186
  (0, 33382)	0.056306179744235935
  (0, 32913)	0.05058568122083257
  (0, 32740)	0.07897824865407392
  (0, 32714)	0.11405249622690902
  :	:
  (0, 4372)	0.03496569959466136
  (0, 4138)	0.15926783608170172
  (0, 4114)	0.06665287013989037
  (0, 4097)	0.04628013738109607
  (0, 4068)	0.06726270397373887
  (0, 3980)	0.032723278318862015
  (0, 3065)	0.011568650440543398
```

```python
# Evaluate the sentiment for each sentence in a review, and plot the variation of sentiment
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)
```

```
(-1.0, 1.0)
```



Discussions

The first output we get is performance data, namely precision, recall, f1-score and support. Specifically, precision is the ability of a classification model to identify all data points in a relevant class. Precision is the ability to return only the data points in a class. The f1 score provides the harmonic mean of accuracy and recall. Finally, support is the number of samples of the true response that fall into that class. The polarity graph at the end refers to the overall sentiment transmitted by a particular text, phrase or word, and can be expressed as a numerical score known as a 'sentiment score'. As we see from the graph, this score can be a number between -1.00 and 1.00, with 0 representing neutral sentiment.

Lab 6. Natural Language Processing

Task 6.2 Text Classification

Problem Descriptions

In this task, a classification method will again be implemented, but in this case more than two classes will be considered. The dataset to be analysed is a set of messages belonging to 20 news groups, such as comp.graphics, sci.space, tal.religion.misc. etc. For the representation of the text, we will still use word count features such as th or tf-idf, and the Naïve Bayes method.

Implementation and Results

```python
from sklearn.datasets import fetch_20newsgroups
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
```

+ Codice + Testo

```python
# 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)
```

```python
# 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)
```

```python
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=categories))

```

```
                    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
```

```python
errors = [i for i in range(len(predictions)) if predictions[i] != testing_data.target[i]]

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 Cunningham) 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
```

Discussions

During the implementation of the algorithm, problem was simplified by choosing only four categories: 'rec.sport.hockey', 'talk.religion.misc', 'comp.graphics', 'sci.space'. Subsequently, the texts were converted into tf-idf or tf vectors and then trained on the Naïve Bayes classifier. For the testing set, the one provided by the dataset was used, then simply converted and the classification done.

Lab 6. Natural Language Processing

Task 6.3 Topic Modelling

Problem Descriptions

In this topic modelling task, the objective is to discover the semantic structure of a vast text corpus. One method of topic modelling is the latent Dirichlet, which is based on considering a different hidden topic for each document in a corpus. This is done by maximising the posterior probability of the corpus given the topic and words.

Implementation and Results

```
import nltk
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from gensim import models, corpora

```

```
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.
  """
]
```

```
# 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
  ]
```

```
# 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)

```

```
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*"field" + 0.015*"call"
0.036*"play" + 0.036*"team" + 0.036*"goal" + 0.026*"ball" + 0.026*"game"

Top 5 contributing words to each topic:

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

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

```
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, 0.5067322), (1, 0.49326774)]
```

Discussions

To apply the topic modelling algorithm, initially two texts with different content have been analysed, one on artificial intelligence and one on football. The texts have then been cleaned by removing the stop words and applying stemming. After this, a dictionary has been created and a document-term matrix constructed using the bag-of-words (BoW) model. The result we obtain is two groups of words, five for each text, from which we can extrapolate the two topics. Once we have trained the LDA implemented above, we can enter a new text to see if it contains the topics obtained previously. In fact, in the last part of the code, a new document was analysed and the output obtained shows that both topics are present, "AI" and "football".almost with the same percentage.

Lab 7. Image Processing

Task 7.1 Face Detection

Problem Descriptions

The objective of this task is to implement an algorithm capable of performing a face detection operation from any image, exploiting the Haar cascade method. This is still one of the most widely used, although it is not always the most accurate, as it requires a frontal position of the face. The method to be used works by exploiting features, i.e. a synthetic representation calculated from pixel values. These are called Haar features and are calculated from the difference of the sums of pixels of two or more adjacent rectangular areas. The idea that gives the Haar cascade method its name, however, is the attempt to reduce the computational cost by organising the detection algorithm in a cascade of classifiers. These are applied in sequence to a certain window or portion of the image and if the features do not match a face, that window is immediately discarded and the algorithm moves on to the next one. The idea is that since the vast majority of tested windows are negative (i.e. do not contain the object), it is beneficial to reject them with as little computation as possible.

Implementation and Results

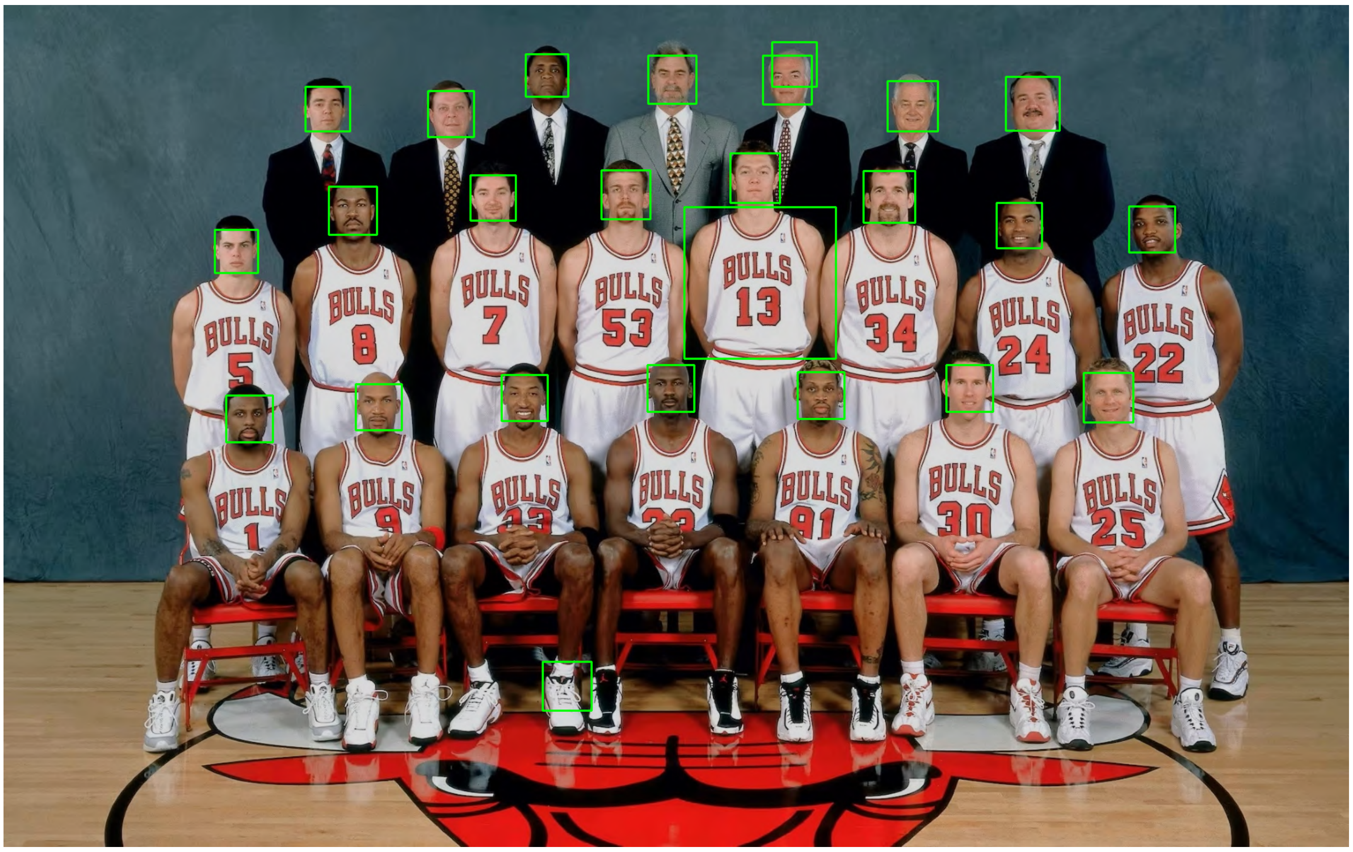
```
!pip install opencv-python
import cv2
from google.colab.patches import cv2_imshow
```

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

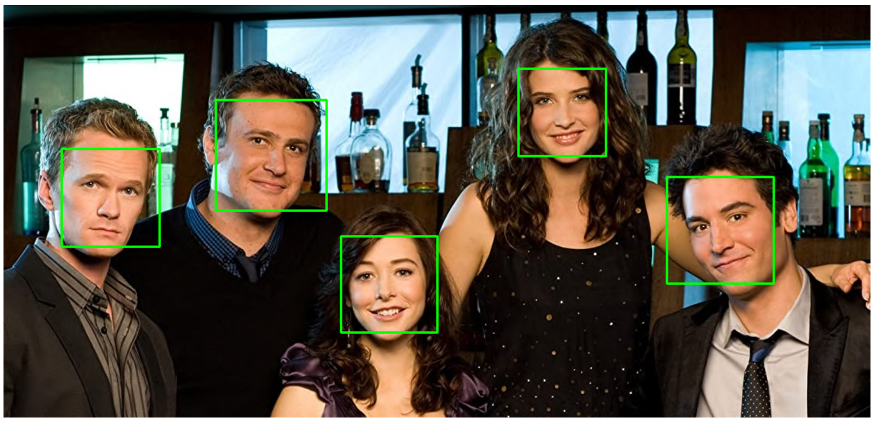
```
# Load the model file and image files to Google Colab
from google.colab import files
file = files.upload()
file = files.upload()
file = files.upload()
```

Testo del titolo predefinito

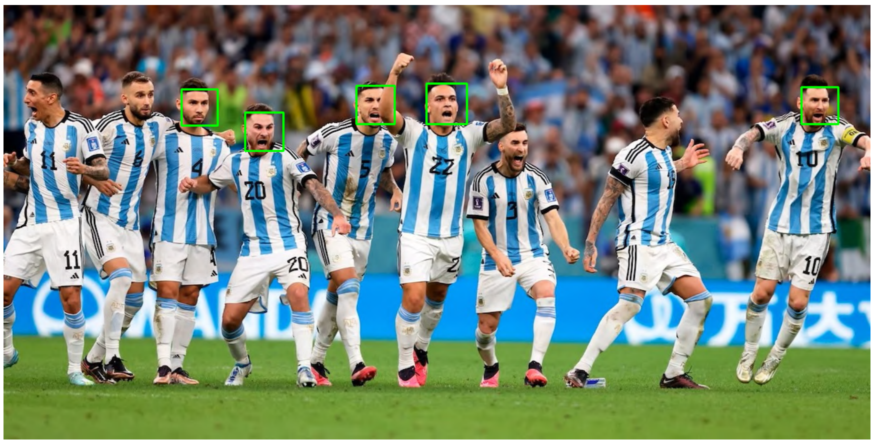
```
#@title Testo del titolo predefinito
# Read an input image
img = cv2.imread('Picture 1.jpg')
#img = cv2.imread('Picture 2.jpg')
#img = cv2.imread('Picture 3.jpg')

# 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, minSize=(50,50))
#faces = face_cascade.detectMultiScale(image=gray, minNeighbors=9)

# 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)



```



```
# Read an input image
#img = cv2.imread('Picture 1.jpg')
img = cv2.imread('Picture 2.jpg')
#img = cv2.imread('Picture 3.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, minSize=(50,50))
#faces = face_cascade.detectMultiScale(image=gray, minNeighbors=9)

# 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)
```



```
# Read an input image
#img = cv2.imread('Picture 1.jpg')
#img = cv2.imread('Picture 2.jpg')
img = cv2.imread('Picture 3.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, minSize=(50,50))
#faces = face_cascade.detectMultiScale(image=gray, minNeighbors=9)

# 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

First of all, looking at the results, one can see that after processing the image, we obtain an output photo in which the detected faces are highlighted by a green rectangle. To test the implemented method, three images with different characteristics were chosen. In the first one, the faces present are all positioned frontally, which makes Haar's features particularly efficient. However, although all faces are detected correctly, there are some additional incorrect detections, such as the shoe of a player. This probably means that the positioning of light and shadow can sometimes deceive the classifier. In the second one there are fewer faces and they are more or less all frontal, which is why the algorithm detects them efficiently. Finally, the last image represents a series of players in motion, where therefore the faces are turned in different directions, and in fact in the output image only some of the faces present have been detected, precisely because this method has difficulty in recognising faces that are not in a frontal position.

In addition, all photos were processed with the same value for the 'minNeighbors' parameter, but this can actually affect the result obtained. It specifies how many neighbours each candidate rectangle must have in order to maintain it. In other words, this parameter influences the quality of the detection. A higher value results in fewer detections but higher quality, with the advantage of eliminating false positives, but with the risk of ignoring faces.

Lab 7. Image Processing

Task 7.2 Image Segmentation

Problem Descriptions

The topic addressed in this task is image segmentation, in other words, being able to detect and distinguish objects within an image. Partitioning is done on the basis of the characteristics of pixels that are grouped into different regions. For the implementation, we will use a pre-built R-CNN model of approximately 246MB, to detect and contour objects in the picture.

Implementation and Results

```
!pip install pixellib
import pixellib
from pixellib.instance import instance_segmentation
import cv2
from google.colab.patches import cv2_imshow
!pip install tensorflow==2.4
```

```
# Download the mask rcnn model
!wget https://github.com/ayoolaolafenwa/PixelLib/releases/download/1.2/mask_rcnn_coco.h5 -O mask_rcnn_coco.h5
path = ""

# Alternatively you can save the data to your Google Drive, and load the data from there
#from google.colab import drive
#drive.mount('/content/drive')
#path = "/content/drive/My Drive/data/"

```

```
# Create the model object and load the trained model
model = instance_segmentation()
model.load_model(path + "mask_rcnn_coco.h5")
```

```
WARNING:tensorflow:From /usr/local/lib/python3.8/dist-packages/tensorflow/python/util/deprecation.py:605: calling map_fn_v2 (from tensorflow.python.ops.ma
Instructions for updating:
Use fn_output_signature instead
```

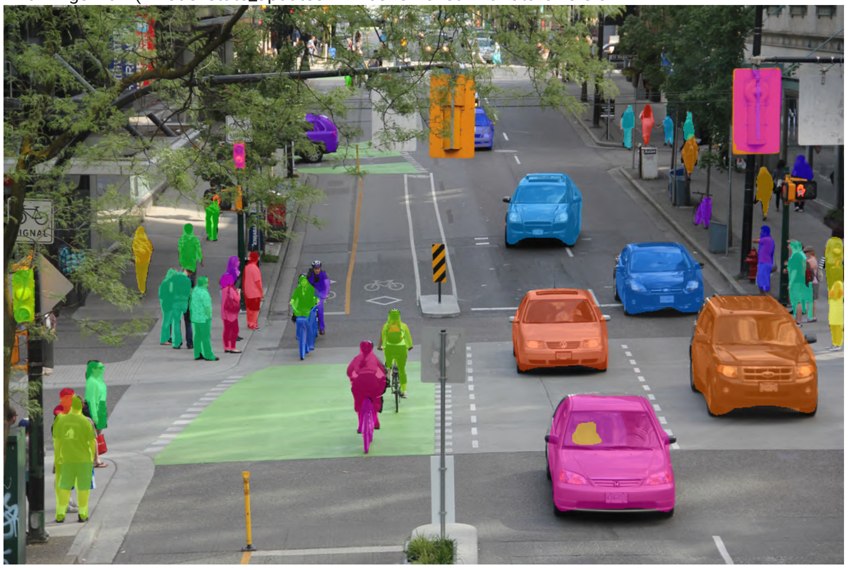
```
# # Load the image files to Google Colab
from google.colab import files
#file = files.upload()
#file = files.upload()
file = files.upload()
```

```
Scegli file Nessun file selezionato    Upload widget is only available when the cell has been
executed in the current browser session. Please rerun this cell to enable.
Saving pic6.jpg to pic6.jpg
```

```
input = path + "pic1.jpg"
img = cv2.imread(input)
cv2_imshow(img)

```

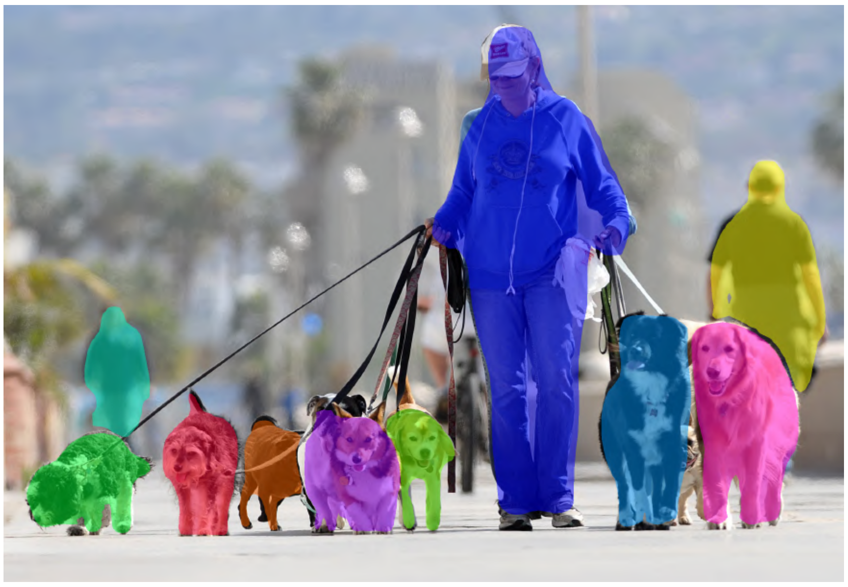
```
segmask, output = model.segmentImage(input, show_bboxes= False)
#print(segmask)

cv2_imshow(output)
```

```
/usr/local/lib/python3.8/dist-packages/tensorflow/python/keras/engine/training.py:2325: UserWarnir
  warnings.warn('`Model.state_updates` will be removed in a future version. '
```



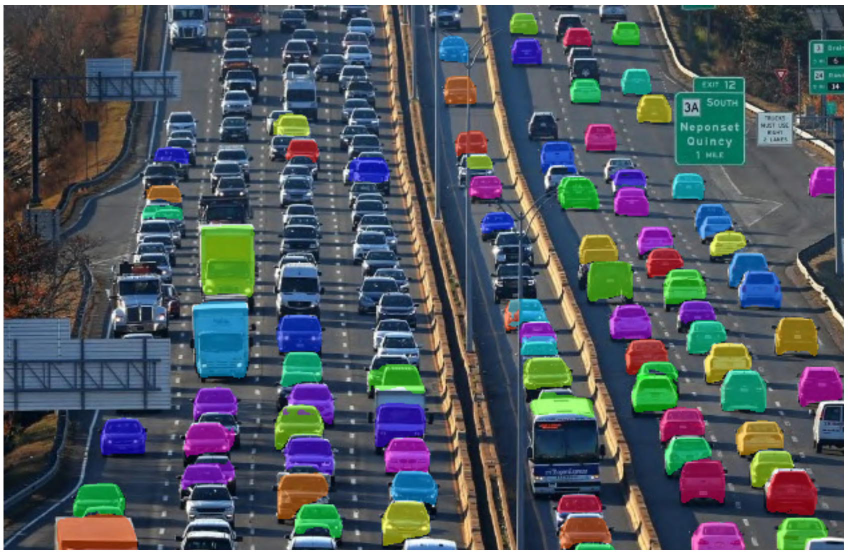
```
input = path + "pic2.jpg"
img = cv2.imread(input)
cv2_imshow(img)
```

```
segmask, output = model.segmentImage(input, show_bboxes= False)
#print(segmask)

cv2_imshow(output)
```



```
input = path + "pic6.jpg"
img = cv2.imread(input)
cv2_imshow(img)
```

```
segmask, output = model.segmentImage(input, show_bboxes= False)
#print(segmask)

cv2_imshow(output)
```



Discussions

In order to test the implemented algorithm, three images with different characteristics were inserted to understand just how the model behaves. The first depicts a road junction, with therefore several elements to be detected. The algorithm in this case was quite efficient, as most of the people, cars and traffic lights in the picture were highlighted. In the second picture, it was tested with animals, although with fewer elements than in the first. Again, almost all dogs and people present were detected, although it is evident how it struggles to portion out two figures that are overlapping, for example two dogs in the picture. This is especially evident in the last image, in which a busy road is shown, and it is clear that in the right-hand roadway, where the cars are further apart, the model is more efficient, compared to the left where there is a lot of overlap.

Lab 7. Image Processing

Task 7.3 Object Recognition

Problem Descriptions

The last task concerns object recognition, a computer vision technique to identify objects in images or videos. The model that will be used is YOLO (You Only Look Once), currently one of the most advanced as it only scans the image once, as the name suggests, unlike the R-CNN, Fast R-CNN and Faster R-CNN methods. There will be two models within the algorithm, a version called 'yolov3' which is more precise but slower, and a version called 'yolov3-tiny' which is the opposite, faster and less accurate.

Implementation and Results

```
!pip install cvlib
import cvlib
from cvlib.object_detection import draw_bbox, YOLO
```

```
# Download the YOLOv3 model configuration, weights and labels files
#!wget https://github.com/yongminli/data/raw/main/yolov3-tiny.cfg -O yolov3-tiny.cfg
#!wget https://github.com/yongminli/data/raw/main/yolov3-tiny.weights -O yolov3-tiny.weights
#!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 there
from google.colab import drive
drive.mount('drive')
path = "drive/My Drive/data/"

```

```
Mounted at drive
```

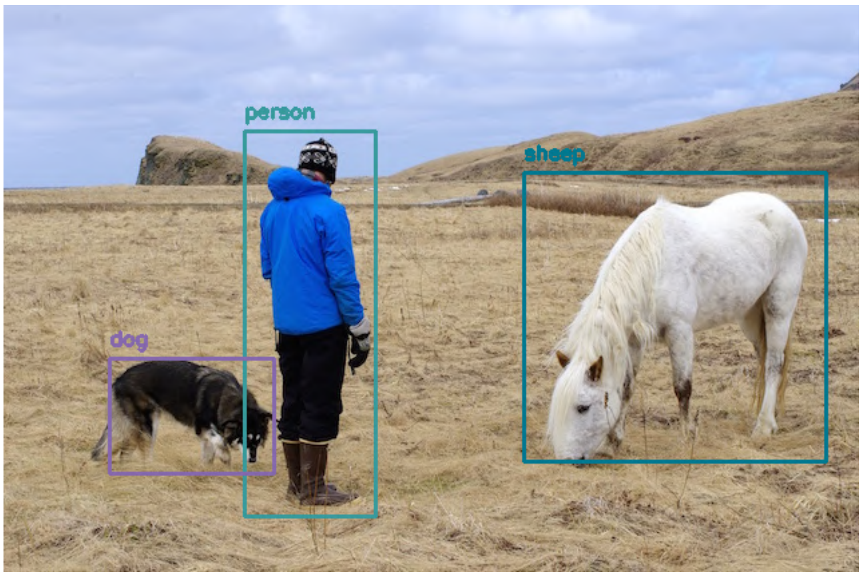
```
# The default YOLO model files
#config = path + 'yolov3.cfg'
#weights = path + 'yolov3.weights'
#labels = path + 'yolov3.txt'

# # 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 ..
```

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

```
Scegli file  Nessun file selezionato      Upload widget is only available when the cell has been
executed in the current browser session. Please rerun this cell to enable.
```

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

# Read an image file
img = cv2.imread(path+'objects1.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 test the algorithm, it was used an image in which there is a person and two different animals. The purpose of the yolo method is then to detect all the elements in the image, frame them, and specify what they are. As mentioned above, for this task we tried using two versions of Yolo ("yolov3" and "yolov3-tiny"), which perform differently. In the first case, using the 'yolov3' version, the result obtained is efficient, as all the elements in the picture were detected and each was given the right name. In the second case, on the contrary, by applying the 'tiny' version, the result was obtained faster, but the efficiency of the detection decreased, because the algorithm was able to detect all the elements, but was not precise in the definition of the horse, which was mistaken for another animal.