Assignment 2

COMP9418 – Advanced Topics in Statistical Machine Learning

Lecturer: Gustavo Batista

Last revision: Thursday 28th October, 2021 at 11:35

Assignment designed by Jeremy Gillen

Instructions

Submission deadline: Sunday, 21st November 2021, at 18:00:00.

Late Submission Policy: The penalty is set at 20% per late day. This is a ceiling penalty, so if a group is marked 60/100 and they submitted two days late, they still get 60/100.

Form of Submission: This is a group assignment. Each group can have up to **two** students, please register the groups on WebCMS on the Groups page. Write the names and zIDs of each student at the top of `solution.py` and in your report. **Only one member of the group should submit the assignment**.

There is a maximum file size cap of 5MB, so make sure your submission files do not in total exceed this size.

You are allowed to use any Python library used in the tutorial notebooks or given in the example code. No other library will be accepted, particularly libraries for graph and probabilistic graphical models representation and operation. Also, you can reuse any piece of source code developed in the tutorials.

Submit your files using give. On a CSE Linux machine, type the following on the command-line:

```
$ give cs9418 ass2 solution.py report.pdf *.csv *.py *.pickle *.json
```

Zero or more csv/pickle/json files can be submitted to store the parameters of your model, to be loaded by solution.py during testing. You may submit `data.csv` as one of your files, if your program needs it. Zero or more python helper files may be included in the submission, if you want to organise your code using multiple files.

Alternatively, you can submit your solution via WebCMS.

Recall the guidance regarding plagiarism in the course introduction: this applies to this assignment, and if evidence of plagiarism is detected, it will result in penalties ranging from loss of marks to suspension.

Description

In this assignment, you will write a program that plays the part of a "smart building". This program will be given a real-time stream of sensor data from throughout the building, and use this data to decide whether to turn on the lights in each room. Your goal is to minimise the cost of lighting in the building, while also trying to make sure that the lights stay on if there are people in a room. Every 15 seconds, you will receive a new data point and have to decide whether each light should be turned on or off. There are several types of sensors in the building, each with different reliability and data output. You will be given a file called `data.csv` containing one day of complete data with all sensor values and the number of people in each room.

This assignment can be approached in many different ways. We will not be giving any guidance on what algorithms are most appropriate.

Your solution must include a Probabilistic Graphical Model as the core component. Other than that you are free to use any algorithm **as part of your approach**, including any algorithm available in Python's sklearn library.

It is recommended you start this assignment by discussing several different possible approaches with your partner. Make sure you discuss what information you have available, what information is uncertain, and what assumptions it may be reasonable to make.

Every area on the floor plan is named with a string of the form 'r', 'c', 'o', or 'outside'. 'r','c' and 'o' stand for room, corridor, and open area respectively.

Data

The file `data.csv` contains complete data that is representative of a typical weekday in the office building. This data includes the output of each sensor as well as the true number of people in each room. This data was generated using a simulation of the building, and your program will be tested against many days of data generated by the same simulation. Because this data would be expensive to collect, you are only given 2400 complete data points, from a single workday. The simulation attempts to be a realistic approximation to reality, so it includes many different types of noise and bias. You should treat this project as if the data came from a real office building, and is to be tested on real data from that building. You can make any assumptions that you think would be reasonable in the real world, and you should describe all assumptions in the report. Part of your mark will be determined by the feasibility of your assumptions, if applied to the real world.

The number of people who come to the office each day varies according to this distribution: num_people = round(Normal(mean=20, stddev=1)). This information was obtained from records of the number of workers present each day, and the empirical distribution of num_people was found to be identical to round(Normal(mean=20, stddev=1)).

Data format specification

Sensor data

Your submission file must contain a function called `get_action(sensor_data)`, which receives sensor data in the following format:

```
sensor_data = {'reliable_sensor1': 'motion', 'reliable_sensor2': 'motion',
'reliable_sensor3': 'motion', 'reliable_sensor4': 'motion',
'unreliable_sensor1': 'motion', 'unreliable_sensor2': 'motion',
'unreliable_sensor3': 'motion', 'unreliable_sensor4': 'motion',
'door_sensor1': 0, 'door_sensor2': 0, 'door_sensor3': 0, 'door_sensor4':0,
'robot1': ('r1', 0), 'robot2': ('r16', 0), 'time': datetime.time(8, 0), 'electricity_price': 0.81}
```

The motion and door sensors report on motion (caused by the presence of people in the room) from the entire previous 15 seconds, but the robot reports an instantaneous count of the number of people in the room it is in.

The possible values of each field in sensor_data are:

- reliable_sensors and unreliable_sensors can have the values ['motion', 'no motion']. All reliable_sensors are of the same brand and are usually quite accurate. unreliable_sensors are a different type of motion sensor, which you have heard tends to be a little less accurate.
- door_sensors count how many people passed through a door (in either direction), so it can be any integer.

- The `robot` sensors are robots that wander around the building and count the number of people in each room. The value is a 2-tuple of the current room, and the number of people counted. I.e. if the robot goes into r4 and counts 8 people, it would have the value ('r4',8). If it goes into corridor 'c2' and no one is present, it would have value ('c2',0).
- Any of the sensors may fail at any time, in which case they will have the value `None`. They may start working again.

The value of `time` is a datetime.time object representing the current time. Datapoints will be provided in 15 second resolution, i.e., your function will be fed data points from 15 second intervals from 8 am - 6 pm.

Training data

The file `data.csv` contains a column for each of the above sensors, as well as columns for each room, which tell you the current number of people in that room. The columns of `data.csv` are the following and can be divided into two groups:

1. Columns that represent readings from sensors, as described in the previous section: reliable_sensor1, reliable_sensor2, reliable_sensor3, reliable_sensor4, unreliable_sensor1, unreliable_sensor2, unreliable_sensor3, unreliable_sensor4, robot1, robot2, door_sensor1, door_sensor2, door_sensor3, door_sensor4, time, electricity_price.

2. Columns that are present **only** in the training data and provide the ground truth with the number of people in each room, corridor, open area, and outside the building: r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31, r32, r33, r34, r35, c1, c2, c3, c4, o1, outside. This ground truth data provides the instantaneous count of people per room (i.e. every 15 seconds, a snapshot of each room is magically taken at exactly the same time, and the number of people in each room is counted. If someone passes through multiple rooms within 15 seconds, they will not increment the count in multiple rooms, only in one room).

Note that the first column of `data.csv` is the index, and has no name.

You should use this data to learn the parameters of your model. Also, you can save the parameters to csv files that can be loaded during testing.

Action data

`get_action()` must return a dictionary with the following format. Note that every numbered room named "r" in the building has lights that you can turn on or off. All other rooms/corridors have lights that are permanently on, which you have no control over, and which do not affect the cost.

```
actions_dict = {'lights1': 'off', 'lights2': 'off', 'lights3': 'off',
'lights4': 'off', 'lights5': 'off', 'lights6': 'off', 'lights7': 'off',
'lights8': 'off', 'lights9': 'off', 'lights10': 'off', 'lights11': 'off',
'lights12': 'off', 'lights13': 'off', 'lights14': 'off', 'lights15': 'off',
'lights16': 'off', 'lights17': 'off','lights18': 'off', 'lights19': 'off',
'lights20': 'off', 'lights21': 'off', 'lights22': 'off', 'lights23': 'off',
'lights24': 'off', 'lights25': 'off', 'lights26': 'off', 'lights27': 'off',
'lights28': 'off', 'lights29': 'off','lights30': 'off', 'lights31': 'off',
'lights32': 'off', 'lights33': 'off', 'lights34': 'off','lights35': 'off'}
```

The outcome space of all actions is ('on','off').

In the provided `example_solution.py`, there is an example code stub that shows an example of how to set up your code.

Figure 1 shows the floor plan specification.

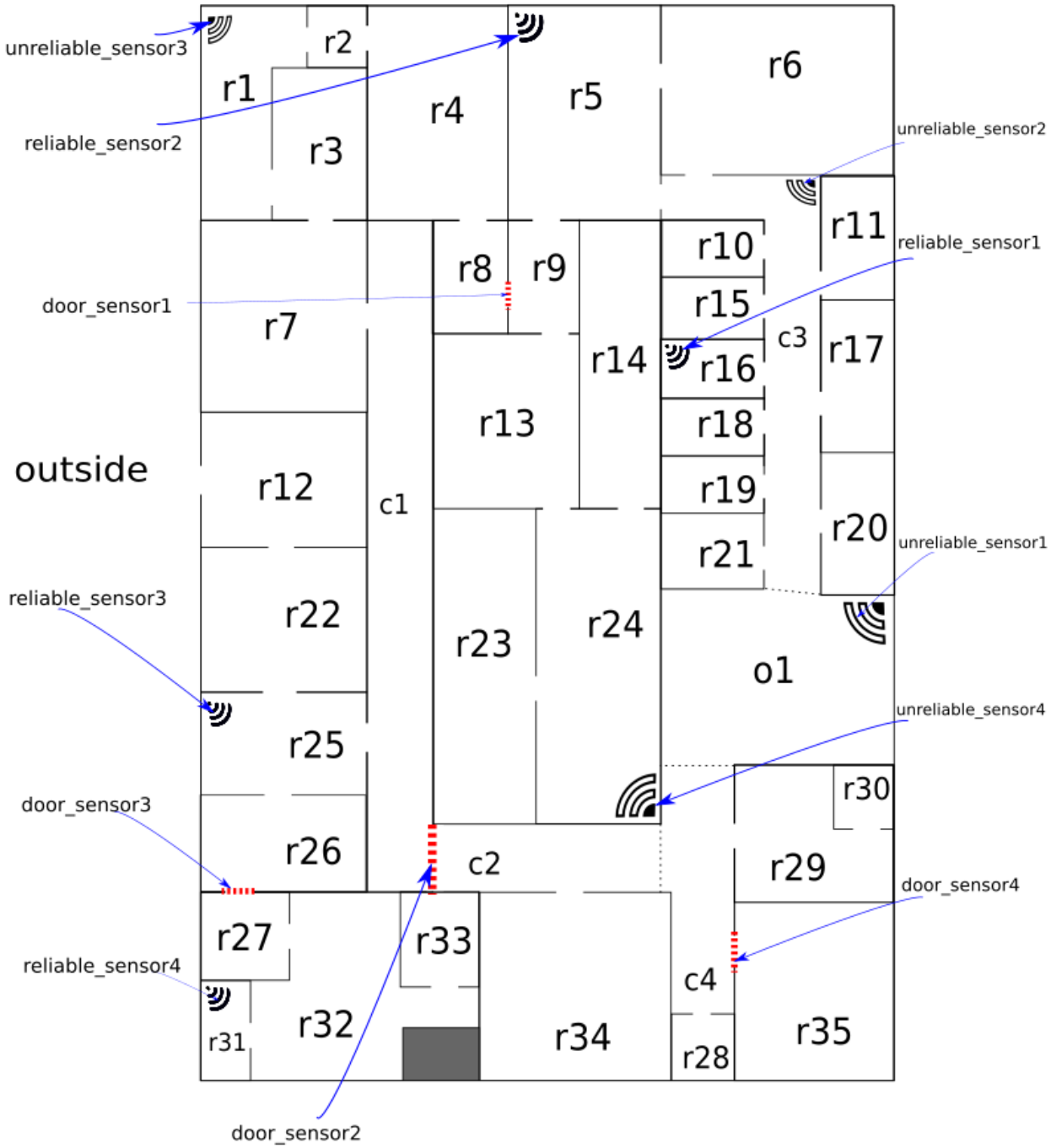

Figure 1: Floor plan (note that dotted grey lines denote the boundaries of areas when the boundary is unclear).

Cost specification

If a light is on in a room for 15 seconds, it usually costs you about 1 cent. The exact price of electricity goes up and down (randomly), but luckily, the electricity provider lists the current price online, so this price is included in the sensor_data. If there are people in a room and there is no light on, it costs you 4 cents per person every 15 seconds, because of lost productivity. The cost can be calculated exactly using the complete training data, so it is also based on an instantaneous count of the number of people in each room.

Your goal is to minimise the total cost of lighting plus lost productivity, added up over the whole day. You do not need to calculate this cost, the testing code will calculate it using the actions returned by your function, and the true locations of people (unavailable to you). The file `example_test.py` shows exactly how the cost is calculated.

Testing specification

Your program must be submitted as a python file called `solution.py`. During testing, `solution.py` will be placed in a folder with `test.py`. A simpler version of `test.py` has been provided (called `example_test.py`), so you can confirm that testing will work. A more elaborate version of `test.py` will be used to grade your solution.

There is a strict upper time limit for the final submission. If your submission runs for longer than 1800 seconds for 10 days (180s/day), it will be cut off, and you will receive 0 marks for the programming section of the assignment. You should aim to be much faster than this, to receive efficiency marks.

`solution.py` will be reloaded for each new day (using importlib.reload), so you may do any daily setup in that file outside of the get_action function, and it will still work.

We will run a test evaluation one week before the deadline, and release the cost and time of each students model. This is to help you confirm that your model doesn't have any major errors, and works with the evaluation system. To participate, make a submission for the assignment before 1 week before the final deadline. We will make an announcement to remind you. To make sure you have something to submit, try to make sure you have at least a minimum working model by this time.

Report

Your report should cover the following points:

- What algorithms you used, a brief description of how they work, and their time complexity.
- A short justification of the methods you used (if you tried different variations, describe them).
- Any assumptions you made when creating your model.

The report must be less than 2000 words (around 4 pages of text). The only accepted format is PDF.

Marking Criteria

This assignment will be marked according to the following criteria:

1. 50% of the mark will be determined by the cost incurred by your code after several days of (hidden) simulated data. An average cost of 130000 or more will receive 0 marks, and 58000 or below will receive full marks (50). Costs in between these values will receive marks linearly interpolated between 0 and 50.
2. 20% of the mark will be determined by the description of the algorithms used, and a short justification of the methods used.
3. 10% of the mark will be determined by a description of the assumptions and/or simplifications you made in your model, and whether those assumptions would be effective in the real-world.
4. 10% of the mark will be determined by the quality and readability of the code.

5. 10% of the mark will be determined by the efficiency of the code. The efficiency marks will be assigned like so:

Mark	Time
10%	<200s
8%	<300s
6%	<400s
4%	<500s
2%	<600s
0%	$\geq$ 600s

So if the 10 day test runs in 250 seconds, you will receive 8 out of 10 of the efficiency marks. All tests will run on the CSE servers.

Items 2 and 3 will be assessed using the report. Items 1, 4 and 5 will be assessed using python files.

Please include the code you used for learning your parameters, even if that code is never called during test time and parameters are simply loaded from files.

Bonus Marks

Bonus marks will be given to the top 10 performing programs (10 percentage points for 1st place, 1 percentage point for 10th place). E.g. if you score 98% in the assignment, and come fifth in the final ranking, then you would receive 103% for the assignment.