B(E)3M33UI — Exercise ML 1: Bayesian and non-Bayesian decision task formulations.

Petr Pošík

February 26, 2018

1 Problem description

We will show the basics of Python programming and at the same time demonstrate the principles of Bayesian and non-Bayesian decision making on a simple example: we want to help our physicians diagnose our common diseases.

A person that comes to the general practitioner (GP) may be *healthy*, or may have a *cold* or a *flu*. GP has only one type of observation, the measurement of body temperature divided intro discrete intervals: it can be *below 37*, *37-38*, *38-39*, or *over 39* degrees Celsius. GP must decide whether the person needs *no cure* at all, or just needs *tea and sweat*, or some kind of medicine *drugs*, or GP can send the person to a *specialist*. Can we help the physician design optimal strategy?

Task 1: Define the sets X, K, and D, and represent them as lists `X`, `K`, and `D` in Python.

2 Bayesian formulation

Task 2: Formulate the task in the Bayesian framework. What other information do we need?

Task 3: Make sure you understand the given joint probability, p_{XK}.

Based on enough historical data, let's assume that the joint probability distribution, p_{XK}, of X and K for people coming to the doctor's office is as follows:

	state K		
temperature X	healthy	cold	flu
below 37	0.05	0.03	0.01
37-38	0.05	0.15	0.03
38-39	0.02	0.25	0.10
over 39	0.01	0.10	0.20

The above probability distribution is stored in a bit different way (as a DB table) in the file `ML1_pXK.csv`.

Task 4: Implement a helper function `load_data(filename)` to load the data (the joint distribution and later the penalty matrix).

The function shall return a dictionary, where the first 2 columns in the CSV file represent the keys, and the third column represents the values.

Task 5: Find in the documentation what the `zip()` function does and how it can be used when constructing a dictionary. Try out the following code. It may come handy in the next task.

```
>>> keys = ['k1','k2','k3']
>>> values = ['v1','v2','v3']
>>> list(zip(keys,values))

>>> d = dict(zip(keys, values))
>>> d
```

Task 6: Implement a set of functions allowing you to easily compute the marginal and conditional distributions based on p_{XK}.

Implement functions

- `get_pX()` to compute p_X,
- `get_pK()` to compute p_K,
- `get_pXgK()` to compute $p_{X|K}$, and
- `get_pKgX()` to compute $p_{K|X}$.

Each function shall return a dictionary with properly formed keys and adequately computed values.

In the following, we will have to iteratively add several values to individual dictionary items. In such situation an error often occurs, that when adding the first value, the dictionary item does not exist yet. Python offers several solutions to this:

- you can pre-initialize the dictionary with correct starting values (`0.0` in our case),
- you can use the `dict.get(key,default_value)` method, which returns the value for the key, if the key exists in the dictionary, otherwise it returns the `default_value`, or
- you can use the `collections.DefaultDict` class which allows you to specify how it should initialize the nonexisting items.

Task 7: Design your own penalty matrix $W\colon K \times D \to R$ and store it in `ML1_W.csv`. Load the data into variable `W` using the function `load_data()`.

Fill in the table below. Keep the following in mind:

- Use penalties between 0 and 1000. If the decision for a particular state does not imply any costs, assign 0; if the decision is maximally unsuitable for a particular state, assign 1000.
- Try to incorporate various kinds of costs:
  - the cost of insufficient cure, when a strong cure is needed,
  - the cost of too strong cure, when not needed,
  - the loss of the physician's reputation if the patient is sent to the specialist with a trivial illness, etc.

There is no correct penalty assignment; the goal is to show what strategy will be optimal for your penalty matrix.

	decision D			
state K	no	tea	drugs	spec
healthy				
cold				
flu				

Fill in the values into the prepared file skeleton in `ML1_W.csv` and load the contents using function `load_data()`.

Task 8: Make sure you understand what a *strategy* is in the Bayesian formulation. Create function `make_strategy()` which takes a list of possible observations and a list of corresponding decisions, and returns a strategy, i.e. a dictionary.

For us a strategy shall be represented by a dictionary, so that we can ask `q[x]` (What is the decision for observation x?).

Task 9: How many different strategies $q\colon X \to D$ are there?

Task 10: Given all the strategies will have the same keys, we can represent them (at first) just as a tuple of decisions. All the possible strategies can be generated using the `itertools.product()` function. Generate a list of all possible 4-tuples of decisions.

Task 11: Create function `make_list_of_strategies()`, which takes the list of possible observations `X`, and the list of possible decisions `D`, and produces a list of all possible strategies, i.e. list of dictionaries.

Task 12: Create function `print_strategy()` which takes a strategy (dictionary) `q` and a list of observations (keys) `X`, and pretty-prints the strategy so that the order of keys is such as specified in `X`.

2.1 Bayesian strategy via complete search

Task 13: Create function `risk()` which returns the risk for a particular strategy q. What inputs does the function need?

$$R(q) = \sum_{x \in X} \sum_{k \in K} p_{XK}(x,k) \cdot W(k, q(x))$$

Task 14: Create function `find_bayesian_strategy()`. What inputs does the function need?

Go through all possible strategies, compute their risks, find the strategy with the minimal risk. Return a 2-tuple: the Bayesian strategy and its risk.

You may find `numpy.argmin()` and `numpy.argmax()` useful.

2.2 Bayesian strategy via partial risks

Task 15: Create function `partial_risk()`, which returns the partial risk for a particular decision d and observation x. What other inputs are needed?

$$R(d,x) = \sum_{k \in K} p_{K|X}(k|x) \cdot W(k,d)$$

Task 16: Create function `find_bayesian_strategy_via_partial_risks()`.

For each observation, compute the partial risk of all decisions, and assign the decision with the minimal partial risk. Return a 2-tuple: the optimal strategy and its risk.

3 Estimating the hidden state

Let's move to a different task - estimating the hidden state K, i.e. $D = K$.

Task 17: Can the physician say anything about the hidden state of a patient **before she actually sees the patient?**

Task 18: If the physician learns a new information about the patient – the body temperature X, she should update her beliefs and maybe change her estimate. Make sure you understand what a *strategy* is in this case. How many different strategies $q\colon X \to K$ are there? Can you create their list?

3.1 MAP estimation

Task 19: Implement function `find_MAP_strategy()` which returns the hidden state k^* with the minimal probability of error.

3.2 Minimax formulation

We still want to estimate the object state K based on the observation X. The strategy should assign a state to each observation with the aim to minimize the maximal probabilities of wrong decision across all true states. We will need only the conditional probabilities $p_{X|K}$; priors p_K and penalties W are not required.

Task 20: Implement function `find_minimax_strategy()` which returns such a strategy that minimizes the maximal probability of wrong decisions across all possible states.

A Fallback penalty matrix

If your Bayesian strategy does not show anything interesting, you can try the following cost matrix:

	decision D			
state K	no	tea	drugs	spec
healthy	0	10	300	300
cold	30	0	200	200
flu	1000	800	10	100

This penalty matrix is saved in the file `fallback_W.txt`. You can just copy it to `ML1_W.csv`.