IN-STK5000 Project 1 - Project 5

Anders Bredesen Hatlelid
Even Tronstad
Jacob Nicolai Arthur Sjødin
Torgeir Ladstein Waagbø

November 6, 2021

1 Our Utility

We define utility $U(X, Y, A) \to \mathbb{R}$ (i.e. a function of pre vaccine/treatment features, post treatment features and the action we took) as the sum of rewards for each individual. The rewards are a weighted sum of the outcomes:

$$U(X, Y, A) = \sum_{i=1}^{N} r_i(x_i, y_i, a_i)$$

And reward is the weighted sum of symptoms:

$$r_i^{\text{no action}} = w^T y$$

$$r_i^{a_i} = w^T y \cdot 2 - \text{Cost}(a_i)$$

(Currently we ignore the features in our utility. This implies for instance that any-one dying is equally bad, regardless of their prior condition. Which is not necceastily a good property of our utility)

We can let 'Cost' be the monetary cost of the treatment/vaccine. I.e. 100$.

Further, we set the rewards to be all lower than zero, making it a loss function of sort, which implies that a utility of zero is the absoulte best. This makes the 'action' reward worse than the 'no action', refelcting that interfering is worse than not. If at a later stage we have more vaccines, then we must tune the coefficient more carefully.

2 Expected utility

For each action we can calculate the expected utility given this formula:

$$\mathrm{E}_{a_i} U = \sum_{y \in Y} U(x, y, a_i) \cdot p(y|x, a_i)$$

We can then choose the action a_{best} which maximizes this expectation. However, since we do not know $p(y|x, a_i)$, we must estimate it with a model. Since we assume the symptom to be independent of each other, we can simplify the expected reward $Er_i = Ew^T y$ as the weighted sum of expected symptoms:

$$\mathrm{E}\, r_i = \sum_{i \in |S|} w_i \,\mathrm{E}(s_i) \tag{1}$$

3 Our model of choice

We will estimate $p(s_i|a_t)$ using a beta - Bernoulli model (like in the last project), where we model the probability of a symptom as independently with different θ_i. To update this model, we need a summary statistic, i.e. the number of sick given treatment, and the number of sick given no treatment. Since this is a simple count, it has a sensitivity of 1, making the implementation of the Laplace mechanism easy.

For the decision of giving treatment, using a privacy method adapted for database squeries makes little sense, because a single decision would be equivalent with a database with one row. Therefore, we choose the randomized response mechanism, with some parameter p to be decided.

4 Differential Privacy

The definition of differential privacy is from the book: Definition 3.4.2 (ϵ-Differential Privacy). A stochastic algorithm $\pi : X \to A$, where X is endowed with a neighbourhood relation N, is said to be ϵ-differentially private if:

$$|\ln \frac{\pi(a|x)}{\pi(a|x')}| \leq \epsilon$$

In particular: If π is a counting algorithm, i.e.

$$\pi(a|x) = \sum_i 1(x \text{ satisiying some condition})$$

Then the centralized Laplace algorithm is ϵ-differentially given $\lambda = 1/\epsilon$. This is due to the fact that a counting function has sensitivity 1, because given a database x and a neighboring database x' (i.e. xNx' is satisfied), then the diverging row could either satisfy the same condition, making the count equal, or not satisfy the condition, making the count one less. We therefore use the Laplace mechanism in the beta - Bernoulli model explained below:

$$\begin{aligned} \theta_i &\sim \text{Beta}(a, b) \\ y_i &\sim \text{Bernoulli}(\theta_i) \\ \theta_i|y_i &\sim \text{Beta}(a', b') \end{aligned} \tag{2}$$

Where a' and b' are given by the Laplace mechanism instead of the usual counting:

$$\begin{aligned} c &= \sum_{j=1}^{n} y_{i,j} + Laplace(1/\epsilon) \\ a' &= a + c \\ b' &= b + n - c \end{aligned} \tag{3}$$

5 Additively updating our model

When updating our model, we have the possibility of either using the former posterior as a new prior, adding the noisy count, and proceeding as normal, however this would also add an increasing amount of Laplace noise. Another possibility is to retrain using our original prior and accumulated data. In this case, we must consider the privacy of repeated queries to our Data base. Let's say we in total apply Q queries, giving the total DP of:

$$\sum_{i=1}^{Q} \epsilon = Q\epsilon$$

To still achieve the same ϵ-differentially privacy, we must then use $\epsilon_Q = \epsilon/Q$. This could decrease the total noise at the end of our application of the policy, however it would apply greater noise in the beginning. Since our model is the least accurate in the beginning, this in undesirable. Therefore, we choose the accumulation approach.

6 Randomized response

For any given individual, the decision to vaccinate is given but the maximizer of the reward, given a randomized response. The response has the following structure:

* Flip a coin with probability θ_1. If heads, answer truthfully. * If tails, flip another coin with probability θ_2 and answer yes or no given heads or tails.

The goal of this mechanism is to satisfy the following equation:

$$p(y|y') = \frac{p(y'|y)p(y)}{p(y')} = p(y)$$

Where y is the truth and y' is the answer. In other words, knowing the response of the individual gives no additional information useful for determining the probability y being true. It follows that $p(y'|y) = p(y')$ to achieve this.

$$
\begin{aligned}
p(y'|y) &= p(\text{answer 'yes' when it is the truth}) \\
&= p_{f1} + (1 - p_{f1})p_{f2} = \theta_1 + (1 - \theta_1)\theta_2 \\
p(y'|\overline{y}) &= (1 - \theta_1)\theta_2 \\
p(y') &= p(y)p(y'|y) + (1 - p(y))p(y'|\overline{y}) \\
&= p(y)(\theta_1 + (1 - \theta_1)\theta_2) + (1 - p(y))((1 - \theta_1)\theta_2)
\end{aligned} \tag{4}
$$

Where p_{fi} is the probability of flip number i giving heads. (to be continued...)

7 Further objectives and questions

We did not manage in time to test the effects of privacy in this run, however we have made our methods flexible enough to easily add it later. We are however wondering if our results make sense, like for instance 100% of the population having fever, and no other symptoms.

We are also unsure if the randomized response is are favorable as opposed to the exponential method. Since we do have a utility function, using it seems plausible, however the utility in that particular use case is regarding the usefulness of the statistic, not the health of a population.

8 Source code

In [8]:
```
from covid.simulator import Population
from covid.auxilliary import symptom_names
import numpy as np
import pandas as pd

from covid.policy import Policy
```

In [65]:
```
symptom_names = ['covid_recovered','covid_positive', 'no_taste_smell',
'fever','headache', 'pneumonia','stomach','myocarditis', 'blood_clots','death']
```

In [9]:
```
w = np.array([0.2, 0.1, 0.1, 0.1, 0.5, 0.2, 0.5, 1.0, 100])
assert w.shape[0] == 9, 'Shape of weights does not fit number of symptoms'
```

In [10]:
```
class Model:
    def __init__(self, nsymptom, nvacc):
        # Priors for the beta-bernoulli model
        self.a = np.ones(shape=[nvacc, nsymptom])/2 # using jeffreys prior
        self.b = np.ones(shape=[nvacc, nsymptom])/2 # using jeffreys prior
        self.nvacc = nvacc
        self.nsymptom = nsymptom

    def update(self, features, actions, outcomes):
        """
        for index in range(self.nvacc):
            print(outcomes[np.where(actions == (index-1))])
            print(actions)
            print(outcomes)
            if (np.sum(outcomes[np.where(actions == index - 1)], axis=1).size != 0):
                self.a[index] += np.sum(outcomes[np.where(actions == index - 1)], axis=1)
                self.b[index] += np.sum(outcomes[np.where(actions == index - 1)]==0, axis=1)\
                    - np.sum(outcomes[np.where(actions == index - 1)], axis=1)
            else:
                self.b[index] += np.sum((outcomes==0)[np.where(actions == index - 1)], axis=1)
        """
        for index, outcome in enumerate(outcomes):
            self.a[int(actions[index])] += outcome[1:]
            self.b[int(actions[index])] += 1 - outcome[1:]

    def get_params(self):
        return self.a, self.b


    def get_prob(self, features, action):
        return self.a[action] / (self.a[action] + self.b[action])


    def retrain(self, features, actions, outcomes):
        # Use aggregated database of outcomes etc...

        self.a = np.ones(shape=[self.nvacc, self.nsymptom])/2 # using jeffreys prior
        self.b = np.ones(shape=[self.nvacc, self.nsymptom])/2 # using jeffreys prior

        self.update(features, actions, outcomes)
```

In [41]:
```
class Naive(Policy):
    def get_utility(self, features, action, outcome):
        utility = 0
        for t, o in enumerate(outcome):
            utility -= np.dot(w, o[1:])*(1+int(action[t] != -1))

        return utility

    def set_model(self, model):
        self.model = model

    def get_action(self, features):
        """Get a completely random set of actions, but only one for each individual.

        If there is more than one individual, feature has dimensions t*x matrix, otherwise it is an x-s
ize array.

        It assumes a finite set of actions.

        Returns:
        A t*|A| array of actions
        """
        n_obs = features.shape[0]
        actions = np.zeros(n_obs)
        #u_list = np.zeros((n_obs, self.n_actions))
        for index, t in enumerate(features):
            u_list = []
            for a in self.action_set:
                u_list.append(self.get_expected_utility(a, t))
            actions[index] = np.argmax(np.array(u_list))
        return actions

    def get_expected_utility(self, action, features):
        p = self.model.get_prob(features, action)
        return -np.dot(p, w)*(1+int(action != -1))

    def observe(self, features, action, outcomes):
        self.model.update(features, action, outcomes)
```

In [52]:
```
## Baseline simulator parameters
n_genes = 128
n_vaccines = 3
n_treatments = 4
n_population = 100000
n_symptoms = 9

# symptom names for easy reference
from covid.auxilliary import symptom_names
```

In [53]:
```
population = Population(n_genes, n_vaccines, n_treatments)
X = population.generate(n_population)
n_features = X.shape[1]
```

```
<__array_function__ internals>:5: VisibleDeprecationWarning: Creating an ndarray from ragged nested s
equences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) i
s deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
```

In [54]:
```
print("With a for loop")
vaccine_policy = Naive(2, np.array([-1, 0]))
vaccine_policy.set_model(Model(n_symptoms, 2))
# The simplest way to work is to go through every individual in the population
Y = np.zeros((n_population, n_symptoms+1))
A = np.zeros(n_population)
for t in range(n_population):
    #print("Person nr: ", t)
    a_t = vaccine_policy.get_action(X[t].reshape((1, n_features)))
    A[t] = a_t
    # Then you can obtain results for everybody
    y_t = population.vaccinate([t], a_t.reshape((1, 1)))
    Y[t] = y_t
    # Feed the results back in your policy. This allows you to fit the
    # statistical model you have.
    vaccine_policy.observe(X[t], a_t, y_t)
```

```
With a for loop
Initialising policy with  2 actions
A = { [-1  0] }
```

In [61]:
```
print(len(symptom_names))
```

```
10
```

In [68]:
```
def print_pre_statistics(X):
    print(f'Statistic (N={X.shape[0]})')
    for i in range(len(symptom_names)-1):
        print(f'{symptom_names[i].ljust(15)} {X[:, i].sum()}')
```

In [75]:
```
print_pre_statistics(Y)
print(A)
```

```
Statistic (N=100000)
covid_recovered 0.0
covid_positive  0.0
no_taste_smell  0.0
fever           100000.0
headache        0.0
pneumonia       0.0
stomach         0.0
myocarditis     0.0
blood_clots     0.0
[0. 0. 1. ... 0. 0. 0.]
```

In [76]:
```
a, b = vaccine_policy.model.get_params()
print(a[0]/(a[0] + b[0]))
print(a[1]/(a[1] + b[1]))
```

```
[5.02542867e-06 5.02542867e-06 9.99994975e-01 5.02542867e-06
 5.02542867e-06 5.02542867e-06 5.02542867e-06 5.02542867e-06
 5.02542867e-06]
[9.84251969e-04 9.84251969e-04 9.99015748e-01 9.84251969e-04
 9.84251969e-04 9.84251969e-04 9.84251969e-04 9.84251969e-04
 9.84251969e-04]
```