Adverse reactions of COVID-19 vaccines analyzed using Bayesian Networks

Emmanuele Bollino

July 2021

Abstract

This project was developed during the COVID-19 outbreak. Governments from all around the world are doing massive vaccination campaigns. There are several manufacturers of vaccines, and some people are facing with side effects following the injections. Most of the side effects are self solving but in this period there is an open debate about safety.

The aim is to study the correlation between some factors and the gravity of the adverse events of COVID-19 vaccines.

Contents

1 Introduction

This project was developed during the COVID-19 outbreak. In this period the governments from all around the world are doing massive vaccination campaigns. There are several manufacturers of vaccines, and some people are facing with side effects by these injections. Most of the side effects are self solving but in this period there is an open debate about the safety of these vaccines.

Objective This project deals with vaccine adverse events reported in the U.S.A. on the VAERS database that contains the adverse events of vaccines in the U.S.A. from 1990.

The aim of this project is to study the correlation between some factors and the gravity of the adverse events of COVID-19 vaccines. Given some risk-factors and other information about the injection, we want to determine the probability of having a serious adverse reaction. Because of the lack of non-adverse events, all the considerations are done considering that the patient has for sure an adverse reaction.

The study is done using Bayesian Networks, a probabilistic graphical model that allows to easily manage a set of stochastic variables linked with some correlation.

For the sake of simplicity, data contained in the database have hardly been simplified, loosing important information about the domain. Because of this, the results are considered over simplistic and not reliable. This project only shows that using probabilistic graphical models for this domain is possible.

Tools The project realization is contained in the attached Python Jupyter Notebook. The libraries used are: PGMpy, for dealing with probabilistic graphical models; Pandas, for managing data; Networkx, for showing graphs.

2 Data

Data are taken from VAERS but only from 2021. So, from the beginning of 2021 to June 2021. This dataset is maintained by CDC, all the information about it is available in the VAERS User Guide. To sum up this document, VAERS stands for Vaccine Adverse Event Reporting System and, as the name suggests, it is a collection of all the reported adverse events supposedly related to a vaccine injection in the U.S.A. Not all the adverse events are reported. Vaccine injections without any adverse reaction are not recorded.

Records represent adverse events and for each of them information about the patient, the injection and the event is recorded.

2.1 Preparation

In order to make the data fit our aim and for the sake of simplification, data need to go though a process involving several steps. For doing this, Pandas

library is used.

Import Original data are split in multiple csv files, so they have firstly to be read in a dataframe and the to be joined in a unique table.

Filter The purpose is to consider only adverse events related to COVID-19 vaccines, so, rows involving other diseases are removed. Moreover, several columns are dropped and only the most relevant are kept.

Bucketization Some columns that contain continue values or a big domain, are bucketized. One of these is the age in which the order relation is kept even if not used in BN. This kind of discretization is preferred w.r.t. using something more accurate like parameterized distributions, because of the domain.

Cleaning Data need some cleaning because of missing values and values that represent empty fields. Boolean columns containing string values are parsed to boolean.

Approximation Some columns, like allergies and illnesses, contain strings that represent a list of values. The issue is that these values are not standardized and each person who filled the cell inserted a non-structured text. Because of the amount of rows, it is not possible to do a precise cleaning of these fields. So, the solution adopted here is an over approximation of those columns: we consider only if a value is empty or not, reducing the strings to booleans. In this way, every illness is the same as the other. From now on we will only know if a patient has some previous illness or not. Moreover, if the value is not known we naively assume that it is empty. Due to this, if a patient illness is not recorded we assume that he/she is fully healthy. This process is the main reason of non-reliability of the results of this project.

Aggregation So far data have gone through several manipulations, but they still have too many columns for an easy-to-manage bayesian network. Indeed, if we try to make pgmpy learn the structure of the network, we will see that it is a complete mess and computational heavy. It would be too hard for a non-physician to deal with it. That's why some aggregation between columns is needed. The aggregation involves the final damage to the patient. Death, disability and hospitalization are combined into a single node. This leads to a degradation of the knowledge because there could have been the case that more than one of these fields were true. In this way we're taking only the most serious damage. We also assume that if none of these fields were active, it means that the adverse reaction self-solved.

Renaming This is the concluding step. For the sake of readability, some columns are renamed into a more human-readable way.

Each row contains data about a reported adverse reaction. It holds information about the injection, the reaction and the patient. The meaning of the columns is in table 1.

Column	Outcomes	Meaning
AGE	'Baby', 'Minor', 'Young', 'Adult', 'Upper adult', 'Old', 'Over 80'	Patient's age
SEX	'M', 'F'	Patient's sex
VISIT	True, False	Whether an ER visit was requested or not
ALLERGIES	True, False	Whether the patient has some allergies or not
MANU	'MODERNA', 'PFIZER\BIONTECH', 'JANSSEN'	Manufacturer of the injection
DOSE	1, 2, 3, 4	Dose number
DAMAGE	'Died or almost', 'Disability', 'Medium' 'Self-solving'	Patient's final damage
ILLNESS	True, False	Whether the patient has some previous illnesses or not

Table 1: Legend of the dataframe used

3 Network Structure

Now that data is ready, it's time to construct the bayesian network associated to them. The library used is pgmpy. The structure of a bayesian network is a DAG.

3.1 Learning

Because we don't have enough knowledge about the domain, we prefer make pgmpy learn directly the structure rather than designing it from scratch.

Scoring

In order to learn the optimal network structure, a scoring function is needed (score based approach). For this purpose, the BIC score is used. BIC stands for Bayesian Information Criterion, it is a criterion for model selection based on the likelihood function. It considers the principle that the more the parameters, the more the overfitting, introducing a penalty for too complex structures. The lower the BIC score, the better the model.

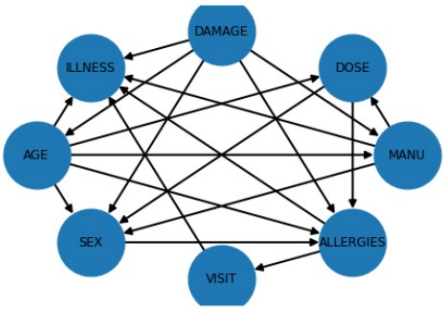

Figure 1: BN directly learnt

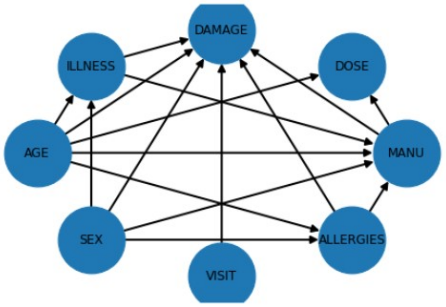

Figure 2: BN learnt with fixed edges and black-list

Search

Given a criterion, a search strategy is needed in order to construct the best possible network or, at least, a good one. An exhaustive search is infeasible because in this case it requires a search among 72057594037927936 graphs (as pgmpy suggests). Instead, a classical Hill Climb search is performed. In this way a model is found in about 30 attempts even if it represents only a local optima.

Attempts

The first attempt is the one in figure 3.1. In this case the model is directly learnt without any relevant parameter. It can be easily seen that it's a complete mess. There are too many relations and sometimes they are in the wrong direction. Also, some relevant associations are not identified.

A complete automatic approach is not so great. For this reason, a background knowledge is used. It consists of a list of fixed edges [Table 2] for which we know for sure that a relation exists, and a black-list [Table 3] in which there are edges that can't be included in the final solution. A new train has been done considering these lists and the final outcome is in figure 3.1.

Edge	Reason
AGE, DAMAGE	adverse reaction is different in young and elderly people
AGE, ILLNESS	aging can cause illnesses
AGE, MANU	the vaccine manufacturer is chosen because of patient's age by physicians
ALLERGIES, DAMAGE	a presence of allergies can cause different types of adverse reactions
ALLERGIES, MANU	the vaccine manufacturer is chosen because of patient's age allergies by physicians
ILLNESS, MANU	the vaccine manufacturer is chosen because of patient's age illnesses by physicians
ILLNESS, DAMAGE	a presence of previous illnesses can cause different types of adverse reactions
MANU, DAMAGE	different vaccines can cause different adverse reactions
SEX, DAMAGE	patient's sex can influence the adverse reaction
VISIT, DAMAGE	a timely ER visit can influence the final damage or it can be a signal of the severity of the adverse reaction

Table 2: Fixed edges

4 Parameters

Now that the network graph is ready, we need to fulfill it with Conditional Probability Tables. Each node has to hold a CPD with respect to its parents. This is essential for making probabilistic inferences.

4.1 Learning

CPDs are learnt from the collected data through a bayesian estimation. Firstly, state counts are computed: how often each state of a random variable occurs given its parents. For each parent configuration, a counting is done. Then, prior CPDs are determined considering BDeu (Bayesian Dirichlet equivalent uniform), that represents a prior belief of the distribution. Finally, the real CPDs are updated considering the state counts. A final check on the resulting model is done: whether the probabilities are coherent.

CPDs are not reported here because of space reasons. They're printed inside the attached notebook.

An idea was to use canonical distributions like Noisy-OR for the sake of compactness but in pgmpy, even if there exists a class for a Noisy-OR model, can't be used as an alternative of CPDs. There's an open issue for this. Moreover, for some nodes, the independent failure probability of causes could have been an oversimplified assumption and this context is not fully boolean.

Edge	Reason
AGE, SEX	sex is not influenced by any other factor in this context
SEX, AGE	age is not influenced by any other factor in this context
VISIT, ALLERGIES	allergies are not influenced by a visit
VISIT, ILLNESS	illnesses are not influenced by a visit
VISIT, DOSE	the dose is not influenced by a visit (which happens after)
ALLERGIES, SEX	sex is not influenced by any other factor in this context
ALLERGIES, VISIT	we assume that the visit is not influenced by allergies but a deeper analysis should be done
DAMAGE, ILLNESS	the damage from the adverse reaction can't cause illnesses
DAMAGE, MANU	the damage from the adverse reaction can't cause the manufacturer of the injection received
ALLERGIES, ILLNESS	we assume that allergies and illnesses are two completely separated entities but a deeper analysis should be done
ILLNESS, ALLERGIES	we assume that allergies and illnesses are two completely separated entities but a deeper analysis should be done
ILLNESS, SEX	sex is not influenced by any other factor in this context

Table 3: Black list

5 Analysis

Now that the bayesian network is completed, several analyses can be done. It's time to use this tool.

5.1 Structural

The first considerations deal with the topology of the network, the DAG. Because this graph represents a bayesian network, we can draw some conclusions just by looking at it without considering CPDs. It's pretty straightforward doing this with pgmpy.

Active trails

An active trail can be seen as a group of influence. Starting from a node and eventual evidences it's possible to retrieve its active trails. An active trail represents a flow of influence. For instance,

```
bayesian_model.active_trail_nodes('ILLNESS', observed=['AGE', 'SEX'])
```

returns the following trail

```
{'ILLNESS': {'DAMAGE', 'DOSE', 'ILLNESS', 'MANU'}}
```

This means that given the age and the sex of a patient, its illness state influences the damage suffered and the other reported nodes.

Because of the high number of connections, active trails are always pretty big.

Markov blankets

A Markov blanket of a node is a subset of nodes that fully determines the desired node. The Markov blanket of a node is composed by its parents, children, and children's parents. It represents the set of information needed to close the set of influence of a node. For instance,

```
bayesian_model.get_markov_blanket('DAMAGE')
```

returns the following trail

```
['ALLERGIES', 'SEX', 'VISIT', 'AGE', 'MANU', 'ILLNESS']
```

Independence

Independence assertions can be interesting. There are 342 possible assertions for this network even if some of them are redundant. Two nodes are independent given some evidences if and only if there isn't an active trail between them.

5.2 Probabilistic inferences

Making probabilistic inferences is the most interesting part in the usage of bayesian networks. The probability of an event given prior knowledge can be easily computed.

Exact inference

Exact inference refers to the process of making a probabilistic inference using the Bayes theorem and the chain rule. In this way, the precise posterior probability is computed, fully exploiting the available CPDs. Naive methods like inference by enumeration are computationally expensive and inefficient. Alternative solutions do exist and one of them is the so called Variable Elimination method. It consists of a dynamic programming approach, avoiding doing repetitive calculations. Summing-out operations are done for previously computed factors. It's possible to use it because in this network are present only discrete values.

For instance, asking the probabilities of the damage given that a patient doesn't have any illnesses nor allergies, is done in the following way:

```
inference = VariableElimination(bayesian_model)
res = inference.query(variables=['DAMAGE'],
                      evidence={
                          'ILLNESS': False,
                          'ALLERGIES': False,
                      })
```

The outcome of this query is in table 4.

DAMAGE	phi(DAMAGE)
Died or almost	0.0326
Disability	0.0117
Medium	0.0405
Self-solving	0.9152

Table 4: Simple query

Approximate inference

Exact inference in large bayesian networks could be computationally too heavy even with such techniques as variable elimination. For this reason, sometimes approximate inference techniques are used. Most of them are based on stochastic processes. Long story short, random samples are generated, CPDs are examined and after enough samples have been generated, the probability of the query is calculated. The more the samples, the higher the accuracy, but also the higher the computational cost. So, a trade off has to be found.

One of the simplest approximate inference algorithm is Rejection Sampling. However, because it discards a lot of samples (wasting some computation), here a smarter approach is used: Likelihood Weighting. All the samples generated are used, and a weight is associated to each sample.

I haven't found any method in pgmpy that computes a probability using likelihood weighting. So, I implemented a custom method for that:

```
def lw_sampling_infer(model, query, value, evidence, size=500):
    evidences = []
    for key, v in evidence.items():
        evidences.append(State(key, v))

    sampler = BayesianModelSampling(model)
    samples = sampler.likelihood_weighted_sample(size=size, evidence=evidences)

    containing = (samples[samples[query] == value])['_weight'].sum()
    total = samples['_weight'].sum()

    return containing / total, samples
```

Weighted samples are generated through pgmpy but the probability computation is done by hand.

In the notebook is shown a brief performance and accuracy comparison between variable elimination and likelihood weighting. The result is that the probability computed differs from 0.04 and the time of approximate inference is faster by 282% (it's not a reliable benchmark but only an empirical valuation).

6 Conclusion

We have seen how bayesian networks are useful is such context as this. The inference made are not reliable, but it was pretty straight forward to build the whole network. Algorithms for structure and CPD inference are crucial in a real context. Moreover, the quality and the amount of data collected is relevant in order to guarantee a good quality of the results. Finally, an expert of the context should supervise the work. It could do that just by looking at the structure of the network, and because of its simplicity, it will not be too time-consuming. Indeed, bayesian networks are a simple and power tool.