Partially Observed Data

Samuel Murray, Ruibo Tu

This tutorial will focus on the missing data issue of “learning". You will implement the EM-algorithm (code skeleton given) for a small Bayesian network, and investigate how missing data affects the “learning". But before we get to that, let’s briefly recap some building blocks of Probabilistic Graphical Models (PGMs) and what we mean by “learning" in PGMs.

1 Part I

1.1 Bayesian Networks

Simply put, a probabilistic graphical model consists of a structure, or graph, that describes how variables influence each other, and weights that describes how strong the influence is. Here, we limit our attention to Bayesian networks, which can be represented as a directed graph and a table-CPD (conditional probability distribution) for each node. Each node represents a variable, and a directed edge from node A to node B means that the value of B depends directly on the value of A. In this case, A would be called a parent of B. Each variable can assume a finite set of discrete values (e.g. 0, 1, a_0, a_1, a_2), and to fully specify the Bayesian network we need to know the probability of each value given the values of parent nodes. These are stated in a table, an example of which is given below, where A is the only parent of B.

Table 1: Table for the Bayesian network where A is the parent of B.

	B=0	B=1
A=0	0.3	0.7
A=1	0.8	0.2

From this table we can read that $p(B = 0|A = 0) = 0.3$, $p(B = 1|A = 0) = 0.7$ etc. It can be observed that each value should be between 0 and 1, and each row needs to sum to 1, since the row sum is $\sum_{b\in val(B)} p(B = b|Pa(B))$, where $val(B)$ denotes all possible values of B, and $Pa(B)$ the parents of B.

1.2 Learning

Learning generally means to infer some knowledge from given data. As described above, a Bayesian network consists of a graph (structure) and a table-CPD for each node (probabilities). Thus, we could imagine learning either the structure, the probabilities or both. Structure learning can be a significantly more difficult task, especially as the number of variables grows, since there is an exponential number of possible graphs that can represent the variables. Further, to decide which structure is best, we need to compare all structures (possibly removing some cases that deviate from prior knowledge too much) which in itself requires us to estimate the parameters for that given structure. That is, parameter estimation is a subtask of structure learning, and to keep this tutorial focused we will only cover parameter estimation. For small toy examples the "true" structure can often be known by reasoning about causality (we dodge the philosophical discussion on whether causality actually exists or not).

As an example, imagine we roll two dice and record the individual values along with the sum. It is clear that the two dice do not influence each other, but that both influence the sum. And even though the sum can tell us something about the values of the dice (if the sum is low, each dice rolled low), it does not affect the rolls. The true graph is thus “(Dice A) –> (Sum) <– (Dice B)". In this case, the sum is deterministic given the dice rolls, but the same reasoning would go for a random variable. Imagine that we don't know the probabilities for the dice. We could then make a number of experiments, and use the collected data to infer all the probabilities. Typically, we want to find the parameters $\widehat{\theta}$ (probabilities) that maximise the (log) likelihood of the observed data. Formally, if D is the dataset and θ are the values of the CPDs that we want to determine, the maximum likelihood solution is $\widehat{\theta} = \arg\max_\theta p(D|\theta)$. Due to the independences in Bayesian networks, the likelihood function can be decomposed so that we can treat parts of the graph independently. In essence, what we end up with is simply counting occurrences and using the frequencies as estimates.

1.3 Missing Data

We say that we have missing, incomplete or partially observed data when we have data points for which some variables are not recorded. This could be due to human error (the variable was not recorded by mistake), the variable not being applicable (a patient did not take a certain medical test) or the variable not being directly measurable (these may or may not have a semantic interpretation, and are usually called latent variables). One way of handling these cases would be to simply ignore the missing cases, and do maximum likelihood on the observed data. This could be a valid approach if only little data is missing, otherwise we risk reducing the data a lot, which will lead to worse estimates. Another approach is to fill in missing values with default values or random values from a given distribution. If we have some prior knowledge on what these values or distributions could be, this might be okay, especially if only few data are missing. Otherwise, the estimates will be biased to whatever default values we use.

An even bigger issue is that even if we have some clever way of handling missing data (such as expectation maximisation covered later), it could fail depending on the process that determines if a value is missing or not. If the process is independent of the data itself, then we are safe; otherwise we could be in trouble. Let's illustrate with a simple example.

Example 1.1 (Missing coin) *Imagine we have a box of 100 identical red/blue coins (one red side, one blue side) that we suspect might be unfair, i.e. the probabilities for red and blue are not equal. We construct an experiment where we empty the box on the floor and count the number of red and blue faces. Unfortunately, some coins are not found (perhaps they rolled under some furniture), and are thus reported as "missing". In this scenario, it should be safe to simply ignore the missing data, and use the remaining ones to estimate the probabilities, since the fact that a coin went missing is independent of its color. However, imagine that we are in an all-red room, and perform the same experiment. Now, it's likely that more red coins go missing, simply because they are more difficult to spot. Because of this, if we ignore the missing data, our estimates will be biased towards blue. As a concrete example, imagine the coins are in fact fair, but that the chance of finding a red-face coin is* 50% *and finding a blue-face coin is* 100%. *We will then find 50 blue and 25 red coins, and our estimates will be* $p(blue) = 1 - p(red) = 2/3$. *Note that in this case, filling missing data with default values will also not work. In fact, there is not much we can do unless we have some insight in the process that leads to missing data. The graph of the first case is*

(Face) (Found)

with no connection. However, for the second case we have

(Face) –> (Found)

since the probability for finding a coin depends on the face-up color.

A sufficient condition for when missing data is okay is that the process that determines whether data is missing or not is independent of the data. This is called missing completely at random (MCAR), e.g., the missing data are caused by random computer errors. This was the case in the first coin example. However, a weaker assumption is actually sufficient, namely that the mechanism leading to missing data is independent of the missing values given the observed values - this is called missing at random (MAR). Consider an example where we have two coins, A and B, and every time B shows "head" we don't record the value of A. Clearly, this is MAR but not MCAR. Sometimes, we can make the data MAR by adding more variables. A small example can illustrate this. Moreover, all the other cases of missing data (that are not MCAR or MAR) fall in to the category, missing not at random (MNAR), which is in general an open problem.

The formal definitions of missingness mechanisms, such as MCAR and MAR, require some further notation, which is deliberately left out of the rest of this tutorial to simplify. Let X be a set of random variables, and denote by R_X (the missingness indicator). R_X are binary variables, indicating whether or not the values of X are observed. We say that the model is MCAR if X is independent of R_X. To define MAR, we further partition the variables X into fully observed variables X_{obs} and partially observed variables X_m. The model is MAR if R_{X_m} is independent from X_m given X_{obs}.

Example 1.2 (Missing grades) *Imagine that we send out a survey to students after a course finished. One question is what grade they received, with one possible answer being "prefer not to say". We can see this as a sort of missing data, so we need to think about how we can deal with it. Maybe we have reason to believe that disclosing the grade or not is not independent of the actual grade (students who get a lower grade may be less inclined to tell it). Thus, we can't ignore these answers, since that would bias our results to high grades. However, it might be that the grade itself is not the deciding factor, but rather a student is likely to reveal his or her grade if he or she is happy with the result. That is, it is not the grade itself that is important, but actually the contentment level of the student (how happy the student is with the result). Formally, we have that the probability for disclosing the grade or not is independent of the grade given the contentment level. By adding this information as a question in the survey ("how happy are you with your grade?"), we get data that is MAR, for which we can use techniques to learn the parameter. Now we can infer from the data the probabilities of being content given a certain grade, and the probability to disclose the grade given a certain contentment level. To summarise, our original setup was not MCAR or MAR, but by adding more variables, we achieved MAR, which is feasible to work with.*

Question 1a. Draw the Bayesian network with the variables, grade and disclose, and the one with the variables, grade, contentment level, and disclose.

Question 1b. Write down the d-separation in the second Bayesian network, and answer that what the missingness mechanisms are in both cases (MCAR, MAR, or MNAR).

1.4 Expectation Maximization

Suppose our data satisfy MAR, we still do not have a good way to do learning. Why can't we use the same approach as with complete data? With complete data, the likelihood function decomposes into small parts, so that in the end we only need to compute frequencies. However, when we have missing data, we need to sum over all possible values of the missing variables. If we again consider the example with two dice and the sum, we have that the log-likelihood $l(\theta, D)$ with complete data is

$$
\begin{aligned}
l(\theta, D) = \log p(D|\theta) &= \sum_{data} \log p(sum, A, B|\theta) &(1)\\
&= \sum_{data} [\log p(sum|A, B, \theta) + \log p(A|\theta) + \log p(B|\theta)]. &(2)
\end{aligned}
$$

However, with missing data (e.g. if B is missing), we need to sum out the missing variables, and get

$$
\begin{aligned}
l(\theta, D) = \log p(D|\theta) &= \sum_{data} \log p(sum, A|\theta) &(3)\\
&= \sum_{data} \log \sum_{B} p(sum|A, B, \theta) p(A|\theta) p(B|\theta). &(4)
\end{aligned}
$$

This is a much harder problem, since now the parameters for different parts are coupled and can not be learned separately. In fact, the number of summations we need to do is exponential in the number of missing values, which makes this approach intractable for large problems. The function will typically not be convex or unimodal anymore, and so one alternative is to use gradient based methods to iteratively find a (local) maximum. Here, we will instead use expectation maximization.

Expecation maximization, or EM, is an iterative algorithm to find maximum likelihood estimates - which is exactly what we are trying to do here! The idea is quite simple. First, we note that the difficulty arose since we were trying to learn parameters with missing values, which is generally difficult, since we need to sum over all cases to compute the likelihood. However, if we somehow knew the missing values, then the solution is easily found in one step. Similarly, if we knew the parameters, then we could easily estimate the missing values, or even find the probabilities for all possible values. EM thus works in two steps, where in the first step we assume that the parameters are known, and in the second we assume that the values are known. So, if we have some guess for the parameters, we can do these two steps to get a new estimate. It can be proved that these two steps will always lead to a better estimate (unless, of course, we are at a local maximum), and so by repeating them we can iteratively get an estimate of the parameters that is a local maximum of the log likelihood.

Why is it called expectations maximization? To understand the name, let's study the two steps in more detail. Pseudocode for the full algorithm is given in the Jupyter notebook.

E-step. The E-step assumes that we have an estimate of the parameters, $\theta^{(t)}$. The goal is to use these to compute the probability for all the missing values. More precisely, we need to compute the expected sufficient statistics. If you are not familiar with the concept, you can think of these as some values that fully summarise the data. For example, if we want to fit a Gaussian distribution with 500 data points, we don't actually need to keep the individual points - all we need is the sample mean and variance of the points. In the case of table-CPDs, the sufficient statistics are the number of occurrences for each (value, value of parents) tuple, i.e. one value for each cell in the tables. The expected number of occurrences of ($A = a$, $Pa(A) = u$) given $\theta^{(t)}$ can be computed as

$$M_{\theta^{(t)}}[a,u] = \sum_{data} p(a,u|data,\theta^{(t)}).$$

Of course, some nodes do not have parent nodes, but we use the same notation and simply ignore the u. To do this efficiently, we need a good way to compute the joint probability, for example with message-passing. However, for the small networks in this tutorial, we can get by with a more naive approach where we compute each probability individually. The $M_{\theta^{(t)}}$ are the expected sufficient statistics, which motivates the name.

M-step. The second step is to estimate the parameters given the sufficient statistics, i.e. the number of occurrences. In the case of fully observed data, this is done by simply computing $\frac{M[a,u]}{M[u]}$, where $M[u] = \sum_a M[a,u]$ denotes the total number of occurrences of u. This gives a number between 0 and 1, and makes sure that the rows in each table sum to 1. Here we instead have expected sufficient statistics given the previous parameters, $M_{\theta^{(t)}}$, which are not necessarily integer values. However, we can use the exact same formula, which gives the following update rule:

$$\theta_{a|u}^{(t+1)} = \frac{M_{\theta^{(t)}}[a,u]}{M_{\theta^{(t)}}[u]}.$$

For nodes that do not have parent nodes, we divide by the total number of data points. This is the maximum likelihood estimate of the parameters given the expected sufficient statistics, which explains the name.

We have now seen how the two steps work, and are almost ready to start coding. First we should just note how to start and how to end. In principle, we could start with either the E-step or the M-step. That is, we could either make an initial guess on the parameters (random or equal probabilities), and use these as $\theta^{(0)}$ in the first E-step. Alternatively, we could make an initial guess on the sufficient statistics (random assignment of all missing values, default values or soft assignments). Generating plausible parameters seems to be the most common approach, i.e. starting with the E-step. (In some cases, the initial parameters can be found from another algorithm, e.g. k-means to initialise Gaussian mixture models; or maximum likelihood on all complete data points.)

How do we know how many iterations to run the EM algorithm? Again, there are multiple options. The simplest way is to use a fixed number of iterations, and hope that it converges in this time. Another approach would be to look at the change in parameters and stop when the change is below a certain threshold. In the same way, we could look at the expected sufficient statistics and stop when these do not differ much between two iterations. It is interesting to note that these two are not equivalent, meaning that there are cases when the parameters do not change much but the sufficient statistics do, and vice versa. Here, we should make one quick note about the issue of overfitting. As stated, each iteration of EM will increase the log likelihood, but only on the training data. There is no guarantee that it increases for a separate test set. This means that if we do too many iterations, we can eventually start overfitting to the training set and make the log likelihood of a test set decrease. To deal with this, one can employ any common regularisation technique, such as early stopping using a separate validation set, or averaging the parameters over multiple k-fold cross validation runs.

2 Part II: Practical

Please Hand in answers to Question 1-5, along with your code.

The goal of this assignment will be to implement the E and M step of the algorithm. We will work with a 3 node network in a v-structure

(X –> Z <– Y).

Each variable is binary, which means that in total there are 12 parameters. To simplify, our implementation will not be general, in that it will only work for our network.

Have a look at the Jupyter notebook ("notebook.ipynb") and Python ("em.py") files provided and try to understand roughly what the code does. Note that parts of the $e\_step(...)$ and $m\_step(...)$ are missing - your first task is to fill in the blanks! The "em.py" file, however, you are not expected to change. Also, do not change the signature of the functions, since they are passed to $expectation\_maximization(...)$.

Question 2. Implement the E-step and M-step for complete data (no missing values). A working solution should run without any exceptions or assertion errors. Of course, the tests are not complete, so a fail-free run is not a guarantee that the implementation is correct. Another sanity check is to use the random seed 1337 to numpy, you should then get the following parameters after 10 iterations:

```
print("Learnt parameters")
print("-----------------")
print_tables(qx, qy, qz)

> Learnt parameters
> -----------------
> p(x) = [0.568  0.432]
> p(y) = [0.294  0.706]
> p(z|x=0, y=0) = [0.24137931  0.75862069]
> p(z|x=0, y=1) = [0.63451777  0.36548223]
> p(z|x=1, y=0) = [0.88333333  0.11666667]
> p(z|x=1, y=1) = [0.1025641  0.8974359]
```

If you get other values (but really think your implementation is correct, and the values look correct), please report them. It might be some rounding errors or the like. Now, change the seed to 2018 and report your values.

Question 3. Modify the implementation of the E-step to work with missing "x" and "y" (M-step should work already). Pass the argument $partially\_observed = True$ to $generate\_data(...)$ in order to generate data with missing values. These are set to "None". Do the same as in the previous task, i.e. run it with random seed 1337 and you should get the following values:

```
print("Learnt parameters")
print("-----------------")
print_tables(qx, qy, qz)

> Learnt parameters
> -----------------
> p(x) = [0.58130798  0.41869202]
> p(y) = [0.28239202 0.71760798]
> p(z|x=0, y=0) = [0.28220124 0.71779876]
> p(z|x=0, y=1) = [0.60052118 0.39947882]
> p(z|x=1, y=0) = [0.92109464 0.07890536]
> p(z|x=1, y=1) = [0.09122205 0.90877795]
```

Again, if you get different values, report them. Change the seed to 2018 and report the values you get.

Question 4. Investigate new distributions. Change the generating distributions (px, py, pz) to new values and see if some combinations are more difficult to learn than others. What if some probabilities are 0? What if we have less data? Report some interesting findings!

Question 5. Change how missing data are generated. Pass the argument $never\_coobserved = True$ to $generate\_data(...)$. This will make it so that y is missing if and only if x is not missing, i.e. all data points are either "(None, y, z)" or "(x, None, z)". How do you think this affects learning? Run EM with this data, and compare the parameters you find with the true ones. What do you observe? Why? Is it still MAR (check the theory part if you forgot the definition)?

Now compute and print the marginal distributions of z given one of x or y (call $print\_marginals(...)$). Are these values closer to the true values? Why/why not? What conclusions about missing data can we draw from this experiment? Does MAR tell the whole story?