Hill Climbing and Expectation Maximization for Learning Network Structure and Learning with Missing Data
ECSE 6810: Probabilistic Graphical Models
Andrés Vargas

# 1 Introduction

In this project, I implemented in Python the EM algorithm for parameter learning under incomplete data and the Hill Climbing algorithm for structure learning under complete data. The true network structure is as follows:

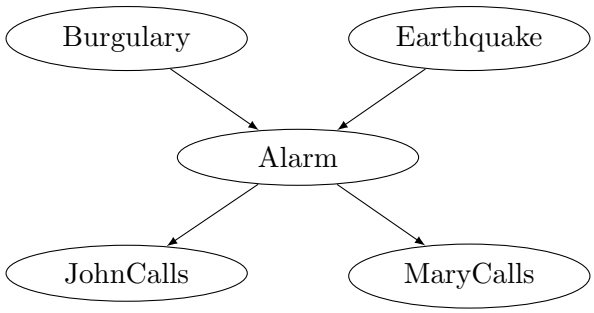



Figure 1

For the EM algorithm, incomplete data was supplied and the links and their directions were provided. For the Hill Climbing algorithm, complete data was supplied, but the links and their directions were unknown.

# 2 Programmatic Graph Representation

To represent the graph structure I used a dictionary of Factor objects, where "Factor" is a class that I created to represent the conditional probability tables (CPTs) . Each Factor object is nothing but a Python dictionary with the additional attributes "parents", "children", "query", and "order". To be clear, the query node in the CPT for $\mathbb{P}(A|B,E)$ would be A, the query node for $\mathbb{P}(B)$ would be B, etc. "order" is up to the user as long as it is consistent with the ordering of the elements of the keys of the factor. The keys are binary tuples that represent the possible realizations in the corresponding CPT. Thus, each factor has $2^k$ keys and each of these keys has $k$ elements, where $k$ is the scope of the factor. The keys of the outer dictionary are characters that correspond to the name of each node. So, in this case, the outer dictionary has five keys: "B", "E", "A", "J", "M". Representing the CPTs as such made it easier to write functions that perform CPT operations. For the remainder of this paper, we use the convention that $M$ represents the number of samples and $N$ represents the number of nodes in the network.

# 3 Theory

## 3.1 EM Algorithm for Incomplete Data with Known Structure

The EM algorithm for incomplete data is a generalization of the maximum likelihood method for complete data. In the maximum likelihood method, the estimate for $\theta_{ijk}$ is $\hat{\theta}_{ijk} = \frac{M_{ijk}}{M_{ij}}$, where $M_{ijk}$ is the number of observations such that node $i$ takes the value $k$ and the parents of node $i$ have configuration $j$. $M_{ij}$ is the number of observations such the parents of node $i$ have configuration $j$. We can think of this method as incrementing the elements of pseudo-CPTs. These pseudo-CPTs correspond to the actual CPTs for the nodes of the network. After seeing an observation, we increment the corresponding element of each pseudo CPT by one. After we have seen all the observations and incremented the corresponding elements in the pseudo CPTs accordingly, the counts contained in each pseudo-CPT will represent the joint distribution (except for a normalizing factor) of the variables in the scope of the corresponding CPT. From here, we can obtain the distribution of each query node given its parents by dividing corresponding pseudo-CPT by its marginalization over realizations of the

query node (see introduction to see what I mean by "query node").

If some samples are missing for certain variables, how do we increment the counts of the pseudo-CPTs whose scopes include the missing variables for these samples? The solution is that, whenever this situation occurs, instead of incrementing one element of each pseudo-CPT by one, we increment the elements of the pseudo-CPT by the probability of their corresponding joint realization given the observed variables. If all the variables in the scope of a factor are present in a particular observation, then this process is the equivalent to the maximum likelihood method. After we have finished incrementing all counts (now they are more like pseudo-counts), the procedure for obtaining the parameters is the same as the maximum likelihood method: we divide each pseudo-CPT by its marginalization over realizations of its query node.

Mathematically, we write the pseudo-count for parameter $\theta_{ijk}$ as

$$\bar{M}_{ijk} = \sum_{m=1}^{M} \mathbb{P}(X_i^m = k, Pa(X_i^m) = j | \boldsymbol{o}[m]) \ ,$$

where $\boldsymbol{o}[m]$ is the set of realizations of observed variables in the $m^{th}$ sample. After we have obtained $\bar{M}_{ijk}$, we marginalize to obtain

$$\bar{M}_{ij} = \sum_{k \in Val(X_i)} \bar{M}_{ijk} \ .$$

The EM estimate is then $\hat{\theta}_{ijk} = \frac{\bar{M}_{ijk}}{\bar{M}_{ij}}$. Why is this algorithm called "EM", which stands for *expectation maximization*? The answer becomes readily apparent when we realize that

$$\mathbb{P}(X_i^m = k, Pa(X_i^m) = j | \boldsymbol{o}[m]) = E[\mathbb{I}\{X_i^m = k, Pa(X_i^m) = j | \boldsymbol{o}[m]\}] \ .$$

Thus, $\bar{M}_{ijk}$ is the expected value of the number of samples that have the $i$, $j$, $k$ configuration, and similarly, $\bar{M}_{ij}$ is the expected value of the number of samples that have the $i$, $j$ configuration. It can be shown that the the estimator obtained from the quotient of these two quantities maximizes the lower bound of the marginal log likelihood, and hence we have the reason behind the name of this algorithm. See Appendix A for a pseudo code of the EM algorithm.

### 3.2 Hill Climbing Algorithm for Complete Data with Unknown Structure

If we don't know the configuration of the links in a Bayesian network, we can initialize the links randomly, learn the parameters for the initialization using the data, and then update the links to a configuration that maximizes the BIC score. Then, we re-learn the parameters using the updated link structure, etc. until convergence.

Unfortunately, enumerating all link configurations is intractable in all except for the smallest of networks. To reduce computational complexity, we resort to a greedy approach that considers the nodes one by one. For each node, we change one and only one link so as to maximize the BIC score of the network. This change can be either (1) a reversal, (2) an addition, or (3) a removal. We then recalculate the parameters for the resultant network and repeat the process for the next node in an order chosen by the user. We continue looping through the network in this order until the BIC score stops improving. The BIC score of a graph $G$ is defined as follows:

$$scoreBIC(G) = \sum_{n=1}^{N} \{ \sum_{m=1}^{M} log\mathbb{P}(G(X_n^m) | \hat{\theta}_n) - \frac{logM}{2} Dim(G(X_n^m)) \} \ ,$$

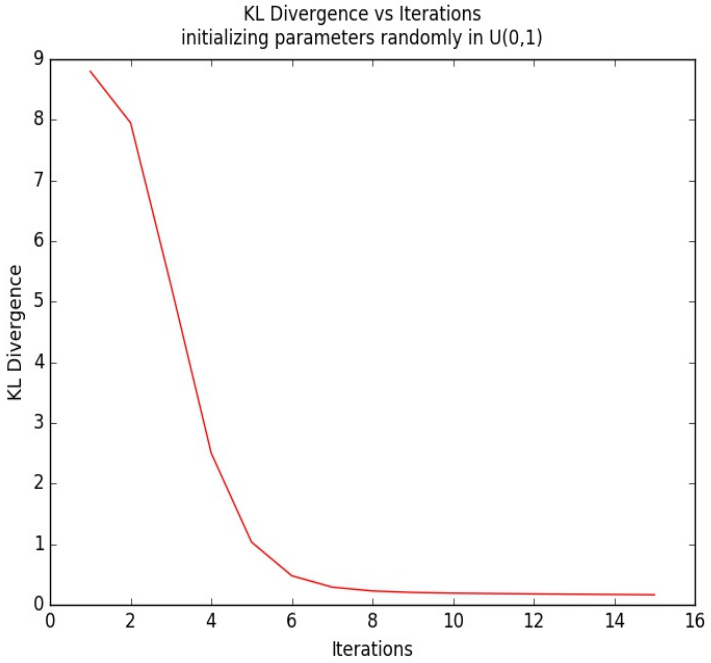
where $G(X_n^m)$ is the structure of node $n$ and its parents in sample $m$ and $Dim(G(X_n^m))$ is the number of independent parameters in the CPT for $X_n^m$. See appendix B for a pseudo code of the Hill Climbing algorithm.

# 4 Experimental Results

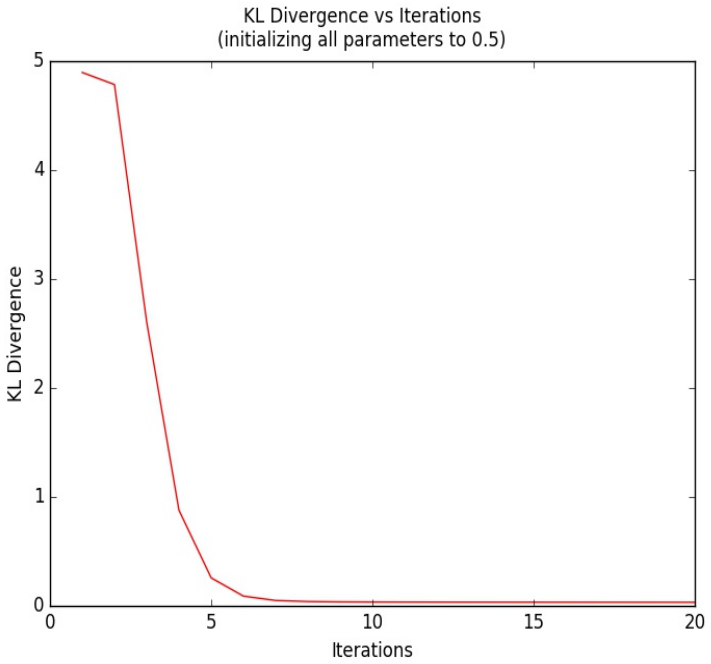
## 4.1 EM Algorithm

I implemented the EM algorithm on the first supplied dataset (data1) using two different initial guesses. The results are shown in Figure 2 (a) and (b) below.

Figure 2




(a)




(b)

We see that, for both initializations, the parameters converge to values that are very close to the true parameters in less than 10 iterations. Despite the success of both initializations, initializing all the parameters to 0.5 appears

to outperform initializing the parameters in U(0,1).

### 4.2 Hill Climbing Algorithm

I implemented the Hill Climbing Algorithm on the second supplied dataset (data2), initializing the network to an empty BN i.e. no links between any two nodes. Figure 3 (below) is a plot of Score vs Iteration for the iterations of the middle for-loop. Each black asterisk marks the beginning of an iteration of the outer while-loop (see appendix B).

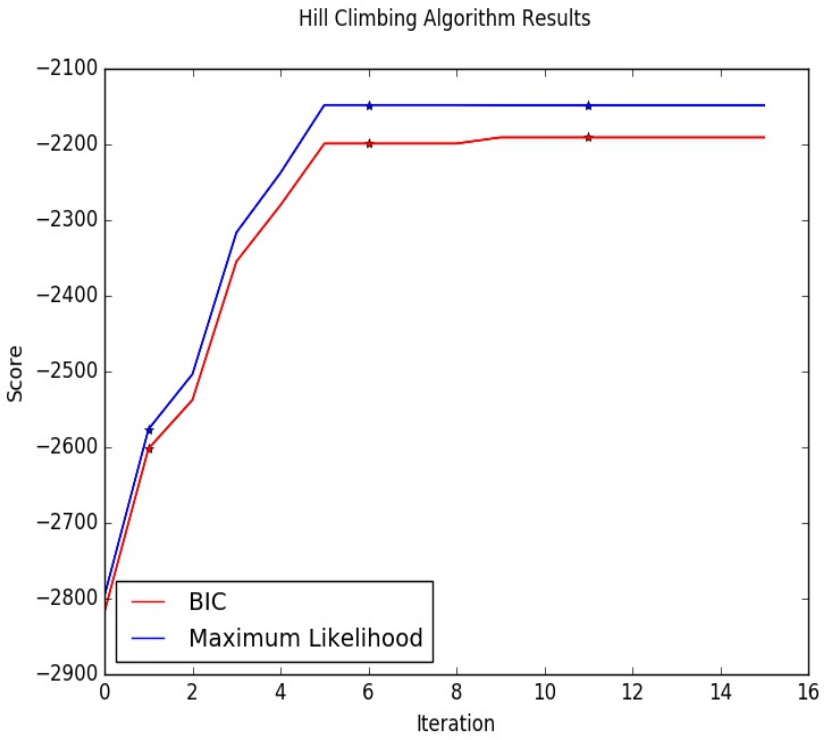



Figure 3

We see that the BIC score is always slightly less than the Maximum Likelihood score because the BIC score is attained by subtracting an overfitting penalty from the Maximum Likelihood score. In Figure 4 below, I have displayed the learned network structure for the Hill Climbing Algorithm.

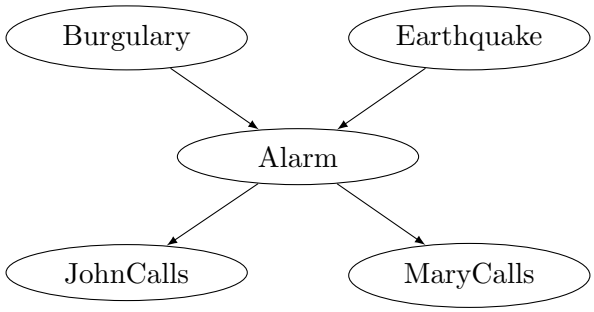



Figure 4

As can be seen, the algorithm learned the correct network structure

## 5 Conclusions and Summary

In this project, I learned parameters under incomplete data and structure under complete data. It is also possible to learn structure under incomplete data via an extension of the EM algorithm. The most difficult

part of this project was understanding how to allocate the pseudo counts for the EM algorithm. The second most difficult aspect was debugging my code. I spent days trying to figure out what was wrong with my code before finally realizing that I was making the mistake of modifying a reference to my graph instead of modifying a copy of it.

# 6 Appendix A: EM Algorithm

**procedure** ESTEP(Set of factors $\{\phi_i\}_{i=1}^N$ that correspond to the given CPTs (i.e. the factors contain probability of each state $k$ of each node $i$ given its parents $j$) , partially observed data $\mathcal{D} = \{\boldsymbol{o}[m], \boldsymbol{h}[m]\}_{m=1}^M$)
  Initialize each $\bar{M}_{ijk}$ to zero.
  **for** m in 1:M **do**
    **for** i in 1:N **do**
      **for** j in 1: $2^{|Scope(\phi_i)|-1}$ **do**
        **for** k in 1:$|Val(X_i)|$ **do**
          $\bar{M}_{ijk} \leftarrow \bar{M}_{ijk} + \mathbb{P}(X_i^m = k, Pa(X_i^m) = j \mid \boldsymbol{o}[m])$
        **end for**
      **end for**
    **end for**
  **end for**
  **return** $\bar{M}_{ijk}$ for all $i$, $j$, $k$
**end procedure**

**procedure** MSTEP(All $\bar{M}_{ijk}$ computed in Estep)
  **for** each node-parent configuration $i$, $j$ **do**
    $\bar{M}_{ij} \leftarrow \sum_{k \in Val(X_i)} \bar{M}_{ijk}$
  **end for**
  **for** each state-node-parent configuration $k$, $i$, $j$ **do**
    $\hat{\theta}_{ijk} \leftarrow \frac{\bar{M}_{ijk}}{\bar{M}_{ij}}$
  **end for**
  **return** $\hat{\theta}_{ijk}$ for all $i$, $j$, $k$
**end procedure**

**procedure** EM(Initial guesses for factors $\Phi^0 = \{\phi_i^0\}_{i=1}^N$, partially observed data $\mathcal{D} = \{\boldsymbol{o}[m], \boldsymbol{h}[m]\}_{m=1}^M$, $\epsilon$)
  $t \leftarrow 0$
  **while** Absolute Difference of Log likelihood of consecutive estimates is larger than $\epsilon$ **do**
    $\{\bar{M}_{ijk}\}_{i,j,k} \leftarrow$ Estep($\Phi^t$, $\mathcal{D}$)
    $\Phi^{t+1} \leftarrow$ Mstep($\{\bar{M}_{ijk}\}_{i,j,k}$)
    $t \leftarrow t + 1$
  **end while**
  **return** $\Phi^t$
**end procedure**

# 7 Appendix B: Hill Climbing Algorithm

**procedure** CLIMB(Fully observed Data $\mathcal{D}$, initial guess for graph structure $\mathcal{G}^0$, function to calculate BIC score $scoreBIC$, $\epsilon > 0$ )
  Learn the Maximum Likelihood Estimates, $\boldsymbol{\theta}^0$, for the parameters that correspond to $\mathcal{G}^0$
  $\{\mathcal{G}, \boldsymbol{\theta}\} = \{\mathcal{G}^0, \boldsymbol{\theta}^0\}$
  $oldscore \leftarrow 0$
  $newscore \leftarrow 999$ #to enter loop

**while** $newscore - oldscore > \epsilon$ **do**
  $oldscore \leftarrow scoreBIC(\{\mathcal{G},\ \boldsymbol{\theta}\})$
  Initialize $n_{\text{store}}$ to an empty container.
  **for** each node $n$ **do**
    $w \leftarrow 0$
    **for** each possible way of reversing/adding/removing one and only one link connected to $n$ **do**
      $w \leftarrow w + 1$
      Obtain the resultant graph structure, $\mathcal{G}^w$
      Learn the Maximum Likelihood Estimates, $\boldsymbol{\theta}^w$, for the parameters that correspond to $\mathcal{G}^w$
      Store $\{\mathcal{G}^w,\ \boldsymbol{\theta}^w\}$
    **end for**
    $\{\mathcal{G},\ \boldsymbol{\theta}\}^n = \text{argmax}_w\, scoreBIC(\{\mathcal{G}^w,\ \boldsymbol{\theta}^w\})$
    **if** $scoreBIC(\{\mathcal{G},\ \boldsymbol{\theta}\}^n) > oldscore$ **then**
      $n_{\text{store}} \leftarrow append(n_{\text{store}}, n)$ ▷ add $n$ to $n_{\text{store}}$.
    **end if**
  **end for**
  **if** $length(n_{\text{store}}) > 0$ **then**
    $\{\mathcal{G},\ \boldsymbol{\theta}\} = \text{argmax}_{n \in n_{\text{store}}}\, scoreBIC(\{\mathcal{G},\ \boldsymbol{\theta}\}^n)$
    $newscore \leftarrow scoreBIC(\{\mathcal{G},\ \boldsymbol{\theta}\})$
  **else** break
  **end if**
**end while**
**return** $\{\mathcal{G},\ \boldsymbol{\theta}\}$

# References


[1] D. Koller, N. Friedman; *Probabilistic Graphical Models: Principles and Techniques*