Project Report
Graphical Models LAB

Maurice Wenig
Friedrich Schiller University Jena
Germany
maurice.wenig@uni-jena.de

1 INTRODUCTION

Learning graphical models is a big aspect of machine learning. But for the parameters of the model to be learned, it is often assumed that the structure is already given, which includes information about dependencies and independencies between features. However in practice, that is not the case. Sometimes even the opposite might be true, that the structure between features itself is much more relevant than the actual amplitude of their interactions, with the hope of being able to interpret the interactions (e.g. in a causal context). This is also an inspiration for our assignment, where our main task is coming up with a search strategy to find a sparse Bayesian Network that fits the data we were given.

For our assignment, we focused on Gaussian Bayesian Networks (GBN). A GBN is a Bayesian Network where the conditional distributions of a feature given their parents is a Gaussian. Another constraint to the GBN is that the mean paramater of the Gaussian, that is the conditional distribution, can only depend linearly on the values of the feature's parents in the Network Graph. So with n as the total number of features, the problem of learning the distribution $p(x_i|x_{\text{pa}(i)})$ for a feature $i \in [n]$, its value $x_i \in \mathbb{R}$, and the value of their parents $x_{\text{pa}(i)} \in \mathbb{R}^{n_i}$ is equivalent to learning $\hat{\beta}_i^* \in \mathbb{R}, \hat{\beta}_i \in \mathbb{R}^{n_i}, \hat{\sigma}_i \in \mathbb{R}$ such that $p(x_i|x_{\text{pa}(i)}) \sim \mathcal{N}(\hat{\beta}_i^* + \hat{\beta}_i^\top x_{\text{pa}(i)}, \hat{\sigma_i}^2)$. Now for these parameters, the ML-estimates have a simple closed-form solution (it is just linear regression), so finding the best parameters for any given structure is simple and efficient.

The dataset we were given spans different attributes of Portuguese "Vinho Verde" wine, which was originally used in [cite]. It consists of twelve features: eleven continuous physicochemical attributes and one discrete sensory attribute (quality). Here, i will treat quality as a continuous feature.

visualize data?

2 METHODS

For the structure learning algorithm, i implemented a tabu search, because it generally performs well in accuracy and runtime (cite overview). I also combined the tabu search with random restarts. The entire parameterized algorithm is illustrated in algorithm 1. It is important to note that the function arguments are all passed by reference, so mutations inside of functions will affect the passed argument even outside the function.

First we do a hill climb. After the first hill climb, we do t_0 tabu walks followed by another hill climb each. Then we do a random restart and do it all again. The random restart procedure is repeated t_1 times.

- t_0: number of tabu walks
- s_0: maximum number of steps of a tabu walk

Algorithm 1: Tabu Search with Random Restarts

```
input : dataset D, initial DAG G
G_max ← copy of G;
HillClimb(D, G_max);
for t_0 times do
    TabuWalk(D, G_max, s_0, l);
    HillClimb(D, G_max);
end
for t_1 times do
    RandomRestart(D, G_max, s_1);
    HillClimb(D, G_max);
    for t_0 times do
        TabuWalk(D, G_max, s_0, l);
        HillClimb(D, G_max);
    end
end
```

- l: length of the tabu list
- t_1: number of random restarts
- s_1: number of random steps during a random restart

2.1 Hill Climbing

In every iteration, the hill climbing algorithm computes the best possible elementary change among all changes. If this change improves the objective function, it is applied. This algorithm is illustrated in function 2.

Function 2: ClimbHill

```
input : dataset D, current DAG G
while S_max increases do
    S_G ← Score(G, D);
    c_max ← IMPOSSIBLY_BAD_CHANGE;
    C ← AllChanges(G);
    foreach c ∈ C do
        apply the change c to G;
        S_c ← Score(G, D);
        undo the change c in G;
        if S_c > S_max then
            S_max ← S_c;
            c_max ← c;
    end
    if S_c > S_G then
        apply the change c_max to G;
end
```

In order to avoid copying the adjacency matrix every time a change is applied, the adjacency matrix is changed in-place, and then the score is evaluated. This is also improves the efficiency of generating all possible changes, which is illustrated in function 3. The evaluation of the score can be sped up by only evaluating local changes of the changed nodes. This is further discussed in subsubsection 2.4.1.

Function 3: AllChanges(G)

input : DAG G
output : all possible changes to G such that G is still a DAG after the application of the change

$C \leftarrow \emptyset$;
foreach $(u, v) \in E_G$ **do**
  $c \leftarrow \text{FLIP}(u, v)$;
  apply the change c to G;
  if *G does not have a cycle* **then**
    $C \leftarrow C \cup \{c\}$;
  undo the change c in G;
end
foreach $(u, v) \in E_G$ **do**
  $c \leftarrow \text{DELETION}(u, v)$;
  `// deletions do not add cycles`
  $C \leftarrow C \cup \{c\}$;
end
$G^t \leftarrow \text{TransitiveClosure}(G)$;
foreach $u, v \in V_G, u \neq v$ **do**
  if $(u, v) \notin E_G$ *and* $(v, u) \notin E_{G^t}$ **then**
    $c \leftarrow \text{ADDITION}(u, v)$;
    $C \leftarrow C \cup \{c\}$;
end
return C

Another key point to the efficiency of function 3 is the efficiency of the cyclicity check. For additions, this can be done by computing the transitive closure G^t of G, which can be done in $O(|V_G|^3)$. Then if $(v, u) \notin E_{G^t}$, the addition of (u, v) will not create a new cycle. For flips, i didn't find a more efficient way than completely checking the resulting graph for cycles. This can be done in $O(|V_G| + |E_G|)$ with Kosaraju-Sharir's algorithm [cite]. But because we use ajdacency matrices instead of adjacency lists, Kosaraju-Sharir's algorithm takes $O(|V_G|^2)$. Therefore, the resulting time for generating all flips is $O(|E_G| \cdot |V_G|^2)$. It follows that the overall time for generating all possible changes is $O((|V_G| + |E_G|) \cdot |V_G|^2)$.

2.2 Tabu Walks

For the tabu walks, we use a FiFo-Queue that keeps track of hashes of each visited adjacency matrix. This queue is called the tabu list. The rest of the algorithm is very similar to hill climbing, just with minimal adjustments:

- changes that result in visited adjacency matrices are not considered
- in each iteration, the top change is applied, disregarding whether it improves the objective function
- if the current value of the objective function exceeds the initial value of the objective function, the tabu walk is stopped

To efficiently check which changes result in visited adjacency matrices, a set with all tabu hashes is created before each iteration.

2.3 Random Restarts

For the random restarts, a list of all possible changes is generated at each step. Then one of them is chosen from a uniform distribution.

2.4 Choice of Optimized Score

For the score function $p(G|D) \propto p(D|G) \cdot p(G)$, i used the approximation

$$p(D|G) \approx p(D|G, \hat{\theta})$$
$$\hat{\theta} := \arg\max_{\theta} p(D|G, \theta)$$

and the prior

$$p(G) \propto \frac{1}{|E|^\lambda}$$

With this, the objective function can be decomposed into the sum of independent node scores and a regularization term.

$$\begin{aligned}\arg\max_G p(G|D) &= \arg\max_G \log p(D|G) + \log p(G) \\ &\approx \arg\max_G \log p(D|G, \hat{\theta}) - \lambda |E_G| \\ &= \arg\max_G \sum_{i \in [n]} S_i(G) - \lambda |E_G|\end{aligned}$$

where

$$S_i(G) := -|D| \cdot \log \hat{\sigma}_i - \frac{1}{2} \sum_{x \in D} \left(\frac{x_i - (\hat{\beta}_i^\top x_{\text{pa}(i)} + \hat{\beta}_i^*)}{\hat{\sigma}_i} \right)^2$$

and $\hat{\theta} := \left(\hat{\beta}_i, \hat{\beta}_i^*, \hat{\sigma}_i\right)_{i \in [n]}$ are the respective ML estimates for

$$p(x_i | x_{\text{pa}(i)}) \sim \mathcal{N}(\hat{\beta}_i^\top x_{\text{pa}(i)} + \hat{\beta}_i^*, \sigma_i^2)$$

A derivation of this can be found in subsection A.1.

2.4.1 Implications for Hill Climbing. Elementary changes (addition, substraction, flip of an edge) only influence local distributions. That means if we construct a graph G', where G' was made by applying an elementary change to an edge (u, v) in G, the comparison $p(G'|D) > p(G|D)$ can be evaluated locally:

$$\begin{aligned}&\sum_{i \in [n]} S_i(G') - \lambda |E_{G'}| > \sum_{i \in [n]} S_i(G) - \lambda |E_G| \\ \iff &\sum_{i \in [n]} S_i(G') - S_i(G) > \lambda(|E_{G'}| - |E_G|) \\ \iff &\underbrace{\sum_{i \in \{u,v\}} S_i(G') - S_i(G)}_{:= \Delta_S(G', G)} > \underbrace{\lambda (|E_{G'}| - |E_G|)}_{:= \Delta_E(G', G)}\end{aligned}$$

Where the second equivalence holds because $S_i(G)$ only depends on node i and its parents. Therefore $S_i(G') = S_i(G)$ for $i \notin \{u, v\}$.

Note that $\Delta_E(G', G)$ only depends on the type of change applied to G:

$$\Delta_E(G', G) = \begin{cases} 1 & \text{addition} \\ 0 & \text{flip} \\ -1 & \text{deletion} \end{cases}$$

$\Delta_S(G', G)$ measures the improvement of node scores when the change is applied to G.

Similarly, two alterations G_1 and G_2 of G can be compared:

$$\sum_{i \in [n]} S_i(G_1) - \lambda \left|E_{G_1}\right| > \sum_{i \in [n]} S_i(G_2) - \lambda \left|E_{G_2}\right|$$
$$\iff \Delta_S(G_1, G) - \Delta_S(G_2, G) > \lambda \left(\Delta_E(G_1, G) - \Delta_E(G_2, G)\right)$$

A derivation of this can be found in subsection A.2.

The interpretation of this is that a change has to bring an improvement of at least λ per edge in order to improve the whole objective function. But more importantly, this allow a very efficient comparison of two different changes, which we need to efficiently find the best possible change for hill climbing and tabu walks.

2.5 Parameter Fine-Tuning

The parameters with the highest impact on the outcome are λ (in the objective function), s_0 (length of the tabu walks), and s_1 (length of the random walks). The impact of λ will be analyzed in [RESULTS]. With s_0 and s_1, we can fine tune the ability to walk out of slim local maxima, and stay in wider maxima. In order to do this, we can analyze the score history over time after the first hill climb.

In Figure 1, we look at two examples of a score history where the parameters still need some tuning. In Figure 1a the tabu walks do not manage to escape slim maxima, because tabu walks are too short. In Figure 1b the random restarts distort the top adjacency matrix too much and return to worse local maxima, because the random walks are too long.

3 RESULTS

3.1 Generated Structures

The greedy search generates different structures depending on the regularization constant λ in the objective function. The number of paramaters in the generated structures is $n_{params} = 2\,|V_G| + |E_G|$ for any structure G. This decreases when λ increases. This is illustrated in Figure 2.

3.2 Performance

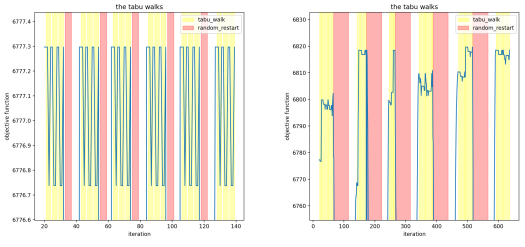
Some performance analysis

3.3 Likelihood

Cross validation time.

Table 1: Error Score Comparison

Recommender	RMSE	MAE
user based	0.670	0.301
item based	0.569	0.222
factorization	0.512	0.207
hybrid	0.496	0.193



(a) Short tabu walks **(b) Long random walks**

Figure 1: Score history after the first hill climb

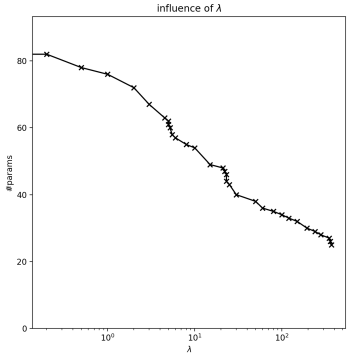


Figure 2: Number of parameters vs λ

3.3.1 *Submission Scores.*

4 CONCLUSION

Some conclusion.

REFERENCES

A CALCULATIONS

A.1 Score Function

Here we derive

$$\max_G \log p(D|G,\hat{\theta}) = \max_G \sum_{i\in[n]} S_i(G)$$

$$\begin{aligned}
&\max_G \log p(D|G,\hat{\theta}) = \max_G \log\left[\prod_{x\in D} p(x|G,\hat{\theta})\right] \\
&= \max_G \log\left[\prod_{x\in D}\prod_{i\in[n]} p(x_i|x_{\mathrm{pa}(i)},\hat{\beta}_i,\hat{\beta}_i^{\,*},\hat{\sigma}_i)\right] \\
&= \max_G \sum_{x\in D}\sum_{i\in[n]} \log p(x_i|x_{\mathrm{pa}(i)},\hat{\beta}_i,\hat{\beta}_i^{\,*},\hat{\sigma}_i) \\
&= \max_G \sum_{x\in D}\sum_{i\in[n]} \log\left[\frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(-\frac{1}{2}\left(\frac{x_i-(\hat{\beta}_i^{\top}x_{\mathrm{pa}(i)}+\hat{\beta}_i^{\,*})}{\sigma_i}\right)^2\right)\right] \\
&= \max_G \sum_{x\in D}\sum_{i\in[n]} \left[-\frac{1}{2}\left(\frac{x_i-(\hat{\beta}_i^{\top}x_{\mathrm{pa}(i)}+\hat{\beta}_i^{\,*})}{\sigma_i}\right)^2 - \log\hat{\sigma}_i - \frac{1}{2}\log 2\pi\right] \\
&= \max_G \sum_{i\in[n]} \underbrace{\left[-|D|\cdot\log\hat{\sigma}_i - \frac{1}{2}\sum_{x\in D}\left(\frac{x_i-(\hat{\beta}_i^{\top}x_{\mathrm{pa}(i)}+\hat{\beta}_i^{\,*})}{\hat{\sigma}_i}\right)^2\right]}_{=S_i(G)} \qquad \square
\end{aligned}$$

Note that $\hat{\theta}\,(\hat{\beta}_i,\hat{\sigma}_i)$ depends on $G\,(\mathrm{pa}(i))$.

A.2 Graph Comparison

For a graph G' that was made by applying one elementary change to a graph G, it holds that

$$\sum_{i\in[n]} S_i(G') = \sum_{i\in[n]} S_i(G) + \Delta_S(G',G) \tag{1}$$

and

$$|E_{G'}| = |E_G| + \Delta_E(G',G) \tag{2}$$

Therefore

$$\begin{aligned}
&\sum_{i\in[n]} S_i(G_1) - \lambda\left|E_{G_1}\right| > \sum_{i\in[n]} S_i(G_2) - \lambda\left|E_{G_2}\right| \\
&\overset{(1),(2)}{\iff} \Delta_S(G_1,G) - \lambda\Delta_E(G_1,G) > \Delta_S(G_2,G) - \lambda\Delta_E(G_2,G) \\
&\iff \Delta_S(G_1,G) - \Delta_S(G_2,G) > \lambda\left(\Delta_E(G_1,G) - \Delta_E(G_2,G)\right)
\end{aligned}$$