Introduction To Discrete Markov Networks

Francisco Mendonça (F.Mendonca@sms.ed.ac.uk)

Abstact

The focus of this paper is proving an understanding of the concepts involved in Markov Networks and addressing practical implementation. Markov Networks are represented with undirected graphs and a set of probability factors from a Gibbs Distribution. A clique is a complete subgraph and corresponds to the sets of variables the factors operate on. The representation of factors is most easily done in table form. The log-linear representation enables the reduction of parameters needed to represent the factor by separating it into features and weights. The GraphViz library can be used to obtain png files of the Markov Networks. A JSON file structure is suggested for storing the networks on disk. The types of information that we normally want to extract from a joint probability distribution are probability queries and MAP queries. The sum-product and max-product variable elimination algorithms are a direct way of extracting such information. For multi-queries a structure called a clique tree is used and we explain how to create it. For learning we suggest the greedy structure search with L1-Regularization. If a model is to be created based on experts a Bayesian network is probably the better approach. For the cases where those are not appropriate we hope to have provided enough knowledge for a basic implementation and a better understanding of the concepts involved in Markov Networks.

1 Introduction

Markov Networks have several applications in areas ranging from Text Analysis to the prediction of idea dissemination in social networks. The motivation behind this paper was the desire to be able to understand how Markov Logic Networks [1] work. Such understanding requires knowledge of First Order Logic and of Markov Networks. The focus will be in providing an understanding of the concepts involved in Markov Networks and addressing practical implementation problems that we faced when trying to implement some of the algorithms and ideas in the main reference book we used, Probabilistic Graphical Models: Principles and Techniques [2].

2 What are Markov Networks?

Markov Networks are a type of Probabilistic Graphical Models. Probabilistic Graphical Models are a graph representation models for joint probability functions. Markov Networks are represented with undirected graphs and a set of probability factors.

Markov Networks are distinct from Hidden Markov Models. These are included in another type of Probability Graphical Models, that is very widely used and known, called Bayesian Networks. They are represented with directed graphs and a set of conditional probabilities.

The graph representation provides us a map of the probabilistic independencies in the joint probability function. The set of factors that go along with the graph representation are the factors

from a Gibbs Distribution. These factors can be seen to create cliques in the graph representation and are therefore also called clique potentials.

We now explain what some of the concepts used are. We however assume some basic knowledge on graphs and probability and explain only the less common used terms.

If unfamiliar with some of the basics chapter 2 of [2] provides an introduction to both graphs and probability theory. Also a good introduction for graphs and probability are chapters 2 and 3 respectively of [3]. See also chapter 22.1 of [4] for a resource focused more on graph representations on a computer.

2.1 Gibbs Distributions

A Gibbs Distribution is a joint probability function that factors out into a product of functions over subsets of the variables. These subset need not be disjoint, i.e. they can overlap on some of the variables. These functions need not be normalized to sum to 1 like probability functions so the unnormalized probability function is normalized by the partition function Z.

Set of factors: $\Phi=\{\phi_1(D_1),\ldots,\phi_m(D_m)\}$
Unnormalized probability: $\tilde{P}_\Phi(X_{1,}\ldots,X_n)=\phi_1(D_1)\times\ldots\times\phi_m(D_m)$
Partition function: $Z=\sum_{X_{1,}\ldots,X_n}\tilde{P}_\Phi(X_{1,}\ldots,X_n)$
Probability distribution: $P_\Phi(X_{1,}\ldots,X_n)$

2.2 Markov Blankets

In a Gibbs Distribution a variable X_i is therefore connected to all variables with which it shares a factor or, to put it another way, with the union of all the sets D_j that contain X_i. This is what is called the Markov Blanket of a variable. It corresponds in the graph representation to all the nodes connected to the one for variable X_i and in normally denoted as $MB(X_i)$.

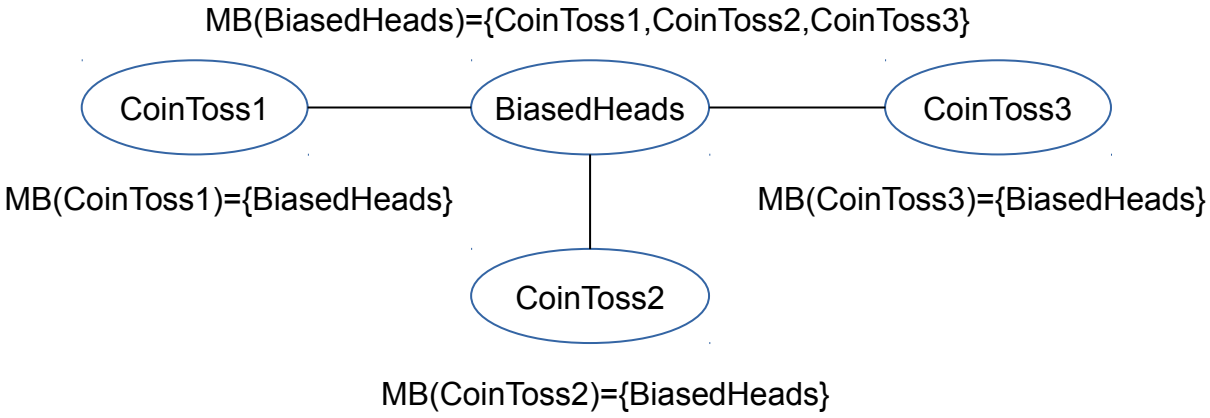

Figure 1: Markov Network for three tosses of a coin that may be biased heads and it's markov blankets for each variable

A variable is independent from any other variable not in its Markov Blanket given the variables in its Markov Blanket. This is called Local independence and is one of the independences that a Markov Network is a map for.

2.3 Graph Cliques

A clique is a complete subgraph. Where a subgraph means a set of its nodes and the edges between them and complete means that there is an edge between a node and every other node. It can be seen in Figure 2 any subset of clique is also a clique. This leads the definition for a maximal clique as clique that cannot be increase by expanding the subgraph.

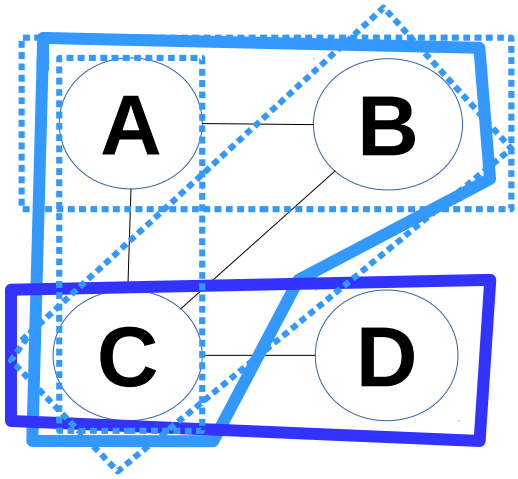

Figure 2: Graph with maximal cliques as thick continuous lines and other cliques as dashed lines

3 How are they represented?

A Markov Network has several levels at which it can be represented. The graphical way to represent it is through graphs and a mathematical way through a set of factors. But these can be represented in more than one way. Also important is how to represent them as a file for use in your program as input or to save a learned network.

3.1 Graphical

The graph representation that is called Markov Network has nodes that represent variables and edges that represent probabilistic influence between the variables. To go from a set of factors to a graph we can first create a node for each variable and then for each factor add edges between each of the variables in the factor.

A separate graph representation that also represents the factors in the graph is called a Factor Graph. In this representation circular (or eliptical) nodes that represent variables and square (or rectangular) nodes that represent factors. The edges in this case are always between factors and variables and not directly between variables.

It may be useful when working with Markov Networks to be able to create an image representation of the graphs. For this we used the GraphViz library [5] and an example of its output when using the ‘fdp’ layout can be seen in figure 3

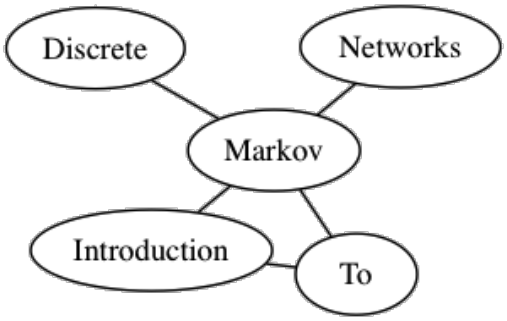

Figure 3: A Markov Network drawn to png with GraphViz library

3.2 Mathematical

The representation of factors is most easily done in table form. For each factor we have a table with entries for each combination combination of the variables in the factor (see table 1).

Variable A	Variable B	Value
a1	b1	100.0
a1	b2	1.0
a2	b1	1.0
a2	b2	1.0

Table 1: Example of table representation of factor

Another way in which factors could be represented is called the log-linear model and this is the representation used in Markov Logic Networks [1]. The log-linear representation enables the reduction of parameters needed to represent the factor by separating it into features and weights. This representations had its birth in physics as the Isling Model and was one of the first types of Markov network models. Features are real valued functions over a set of variables and one of particular interest is the indicator feature that has value 1 when the variables take a specific combination of assignments and 0 otherwise. The weights allow the expression of the level of influence of each specific feature on the factor. As the name would indicate the unnormalised probability is then an exponential over the linear combination of the features.

features: $F=\{f_1(D_1),...,f_k(D_k)\}$
weights: $w_1,...,w_k$

$$P_\Phi \propto \exp(\sum w_i f_i)$$

Figure 4: Log-Linear model equation

indicator features: $f_{a1b1}(A,B)=\begin{cases}1 & if\ A=a1 \wedge B=b1\\ 0 & otherwise\end{cases}$

weights: $w_{a1b1} \approx 4.605$

Figure 5: Factor in table 1 expressed as log-linear model

Since an exponential is only 0 when its exponential is negative infinity this model only applies to a type of probability distributions called positive distributions. These are distributions in which the probability is never 0 for any combination of variable values.

3.3 File storage

There seems to be no standard for Markov Network file storage. We recommend the json format due it's ease of reading and writing with libraries such as JSON for Modern C++ [6]. Since we

know how to create the graph component from the factors and they are needed for any calculation of probabilities we designed the file format to have a list of variable definitions and a list of factor definitions. A list in json is represented by “ListName” : [*element_1, … , element_n*]. For example of our definitions of variables and factors see Figures 6 and 7.

```
{
    "name" : "A",
    "values" : ["a0", "a1", "a2"]
}
```

Figure 7: Example of variable in json file

```
{
    "type" : "TableFactor",
    "variables" : ["A"],
    "table" : [0.1, 0.5, 0.4]
}
```

Figure 6: Example of table factor in json file

4 Inference in Markov Networks

Due to the structure of Gibbs distributions calculating even the straightforward probability of a fully instantiated set of variables requires us to take all possible values into account to calculate the partition function. This is an exponentially large amount of calculations to perform. To reduce the amount of calculations the independences that the graph maps can be taken advantage of. This reduction can be significant but will still end up exponential. Because of this approximate methods have been developed however they give no guarantees in the general case. Therefore we will focus here on the exact methods.

4.1 Types of Query

The types of information that we normally want to extract from a joint probability distribution are probability queries and MAP queries.

For probability queries we are interested in the probability distribution of a reduced set of variables given some evidence on a set of variables. That is called a marginal conditional probabilities query and can be seen as the general form for probability queries. It can be written as P(X \ Y,E | E=e), meaning the probability of variables X except the set Y of variables to marginalize and the set E of evidence. If the evidence is null then we have a simple marginal probability query, if the set of variables to marginalize is null we have a simple conditional probability query.

MAP queries involve discovering the variables with maximum aposteriori probability. This can once again be on a reduced set of variables given some evidence. That is argmax P(X \ Y,E | E=e). This is the general marginal conditional MAP query and can be reduced to a simple marginal MAP query by having null evidence and a simple conditional MAP query by having a null set of variables to marginalize.

4.2 Variable Elimination

The simplest way to reduce the amount of calculations for single queries is by taking advantage of the properties of sums (or maximizations) and multiplications to push sums over a variable to the minimum amount of factors that involve them (see Figure 8 and 9).

$$\sum_{A,B,C} \phi_{A,B} \times \phi_{B,C} = \sum_{A,B} \phi_{A,B} \times \sum_{C} \phi_{B,C}$$

Figure 8: Pushing sum over C to be done first

$$\max_{A,B,C} \phi_{A,B} \times \phi_{B,C} = \max_{A,B} \phi_{A,B} \times \max_{C} \phi_{B,C}$$

Figure 9: Pushing maximization over variable C to be done first

The algorithms that take advantage of this directly are called the sum-product variable elimination and the max-product variable elimination (see chapter 9 of [2]). These algorithms require an order for the elimination. Finding the optimal order for elimination in a general graph is a combinatorial problem. However for a subset of graphs called chordal graphs and optimal algorithm that runs in near linear time consist of the MaxCardinality algorithm (see chapter 9 of [2]). Chordal graphs are graphs in which any loop greater than 3 has a short cut through a loop size 3. This can be identified be triangles of nodes inside any loop and that is why the process of turning a non-chordal graph into a chordal one is called triangulation.

For probability queries the marginalization is done by sum-product algorithms and for MAP queries the MaxMarginalization is done by the max-product algorithm. That leaves the conditional part. Conditioning on a set of variables consists on an operation called factor reduction. This is where factor in table format are reduced to the entries that have the specific instantiation of the evidence variable. By applying factor reduction to all factors before the sum and max product elimination algorithms we can satisfy the conditional part of the queries.

4.3 Clique Trees

When multiple queries will be preformed a structure that can take advantage of local independencies and avoid duplicating variable eliminations can be used. This is called the clique tree. The calculations involved in the product eliminations are the same as the ones involved in clique trees. And indeed a product elimination in a certain order induces a clique tree structure. This is important because for non-chordal graphs the cliques in the graph do not for a tree but for instead a graph of cliques. And although there are approximate methods that use the clique graphs they, as previously stated, offer no guarantees on convergence. An example of a graph and a corresponding clique tree can be see in Figure 10.

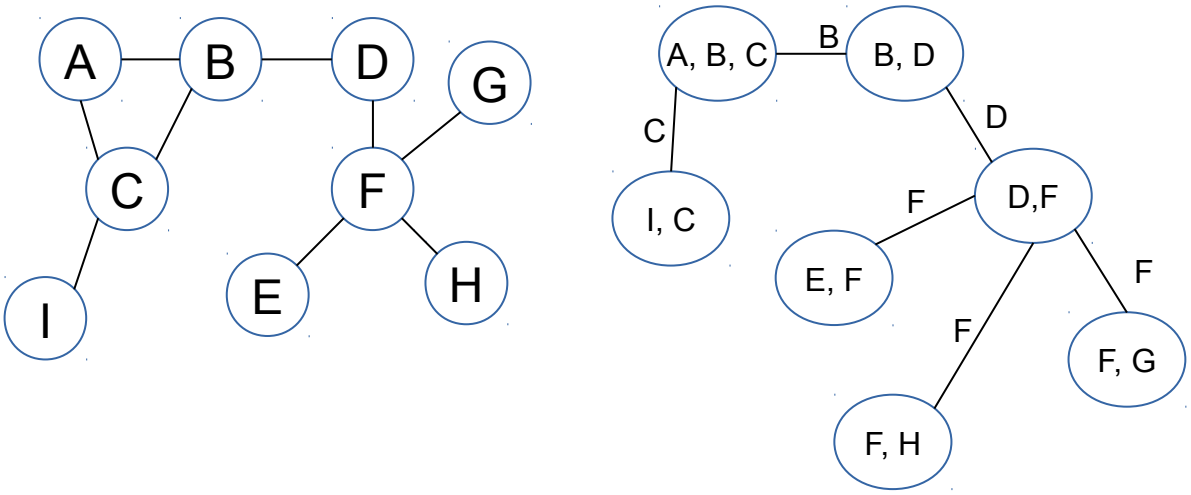

Figure 10: Example graph and a corresponding clique tree

So how to create a clique tree? Like for the ordering problem, creating an optimal clique tree is a combinatorial problem. However given an ordering it is possible to create the induced clique tree making the heuristics for deciding the elimination order also useful for constructing good clique trees. To understand how to make a clique tree from an ordering it is useful to know how the calculations would go when we have a clique tree for the graph already and how these match the ones in the sum-product elimination.

The clique tree can be used to calculate a set of beliefs (corresponding to marginal probabilities) for the cliques (nodes) and the separation sets (edges) in the clique tree. Each factor (or clique potential) is uniquely associated with one clique in the clique tree. The factors created by the multiplication of the factors belonging to a clique are called the initial factors. To calculate the beliefs in a way that they are consistent across the cliques is called calibrating the clique tree. The calibration of a clique tree consists of a two pass algorithms. The first pass is the bottom up message passing where "messages" are passed up to the parent cliques. These messages correspond to the sum-product elimination of variables in the clique that are not in the egde (separation set) to the parent node. Once a clique node has all the messages from child nodes it multiplies them by the initial factors and again creates a message to send to its parent node. This is done until all messages reach the root node.

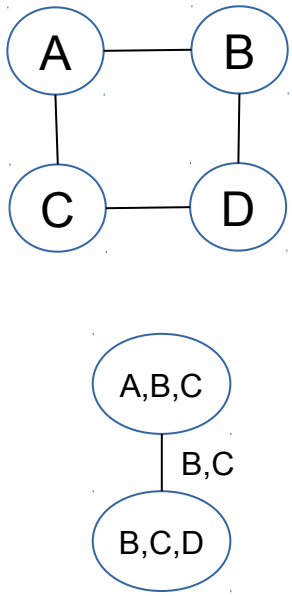

Figure 12: Graph and clique tree induced by elimination order D, B, C, A

set of factors: $\Phi=\{\phi_1(A,B),\phi_2(A,C),\phi_3(B,D),\phi_4(C,D)\}$
elimination order: D,B,C,A
Step 1 eliminate D
$\psi_2(B,C)=\sum_D \psi_1(B,C,D)$, $\psi_1(B,C,D)=\phi_3(B,D)\phi_4(C,D)$
set of factors: $\Phi=\{\phi_1(A,B),\phi_2(A,C),\psi_2(B,C)\}$
create new clik: B,C,D with factors $\phi_3(B,D),\phi_4(C,D)$
Step 2 eliminate B
$\psi_4(A,C)=\sum_B \psi_3(A,B,C)$, $\psi_3(A,B,C)=\phi_1(A,B)\psi_2(B,C)$
set of factors: $\Phi=\{\phi_2(A,C),\psi_4(A,C)\}$
create new clik: A,B,C with factor $\phi_1(A,B)$
Step 3 eliminate C
$\psi_6(A)=\sum_C \psi_5(A,C)$, $\psi_5(A,C)=\phi_2(A,C)\psi_4(A,C)$
set of factors: $\Phi=\{\phi_6(A)\}$
update clik: A,B,C with factor $\phi_2(A,C)$
Step 4 eliminate A
$Z=\sum_A \psi_6(A)$
set of factors: $\Phi=\{\}$

Figure 11: Creation of a clique tree from a set of factors and an elimination order (Graph and Clique tree in Figure 12)

The second pass is the top down message passing. It involves sending messages from the root node. These messages consist of multiplying all incoming messages that did not come from the edge the message is being sent through. A more in depth description of the clique tree calibration can be found in chapter 10 of [2].

Taking the example in Figure 11 can help us illustrate how the message passing in the clique tree corresponds to the sum-product elimination and to help you understand our clique tree creation algorithm. The message passed between the B,C,D clique and the A,B,C clique is ψ_2 from Figure 11. So here is the algorithm:

1) gather all current factors with the current variable to eliminate

2) create a new elimination factor by summing over the elimination variable the product of the gathered factors

3.1) if only original factors are present create a new clique and associate the new elimination factor as coming from this clique

3.2) if there is only one elimination factor and original factors with all variables contained in the clique associated with it, expand the clique's factors to include these original factors

3.3) if there is more than one elimination factor or the original factors variables are not contained in the associate clique, create a new clique with the original factors present and associate the new elimination factor as coming from this clique and the old cliques (from the gathered elimination factors) as being children of this clique

3.4) if there is only one elimination factor simply associate the new elimination factor with the clique of the original elimination factor

4) update the current factors by removing the gathered factors and adding the elimination factor

5) if there are more variables to eliminate go back to 1 else tree is complete.

5 Learning Markov Networks

Learning in Markov Networks has two main areas. Learning the parameters for a given structure and learning the structure. These can be done by looking at it as an optimization over a *hypothesis space*, our set of possible models, of an *objective function*, a score for how well fitted the model is (chapter 16.3 of [2]).

In our opinion the best learning algorithm for Markov Networks is the greedy structure search with L1-Regularization for the score function (for full details see chapter 20.7 of [2]). This approach uses the log-linear model and is based on having a set of possible features and learning the features active in the training data and their weight based on the L1-Regularization score. L1 scoring leads the weights of extraneous features to zero meaning there is no need for an elimination step in the search. It also has a convex structure so there is no chance of getting stuck in local optimums.

The algorithm adds a feature at a time to the active set. It then optimizes the parameters (i.e. weights for all active features) according to the L1 score. It then removes any feature whose weight went to zero from the active set. It loops through all possible features until adding any new feature does not improve the score.

6 Conclusions

We found that Markov Networks are used in several applications. Unfortunately working with Markov Networks means either accepting exponential complexity of the algorithms or risking the non-convergence of the approximate algorithms. If a model is to be created based on experts a Bayesian network with a causal structure is probably the best approach and allows causal queries and algorithms that can results with polynomial complexity.

For the cases where Bayesian networks are not appropriate then we hope to have provided enough knowledge to get a basic implementation and a better understanding of the basic concepts involved in Markov Networks. This paper, by focusing on the basics and introducing the log-linear model is hoped to have convened the knowledge required to get a grasp on Markov Logic Networks. These use the rules in the knowledge base as the features of the log-linear model and weights associated as confidence on the rules application [1].

References

1: Matthew Richardson and Pedro Domingos, Markov Logic Networks, 2006, http://homes.cs.washington.edu/~pedrod/papers/mlj05.pdf
2: Daphne Koller and Nir Freidman, Probabilistic Graphical Models: Principles and Techniques, 2009
3: Uffe B. Kjaerulff and Anders L. Madsen, Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis, 2008
4: Thomas H. Cormen et al., Introduction To Algorithms (Second Edition), 2001
5: Emden R. Gansner and Stephen C. North, An open graph visualization system and its applications to software engineering, 2000
6: Niels Lohmann, JSON for Modern C++, 2015, https://github.com/nlohmann/json