Learning with Graphical Models

Jagadish M (07305050)

jagadish@cse.iitb.ac.in

Guide: Prof. Sunita Sarawagi

April 15, 2008

Contents

Abstract

Graphical models provide a powerful framework for probabilistic modelling and reasoning. Although theory behind learning and inference is well understood, most practical applications require approximation to known algorithms. We review learning of thin junction trees–a class of graphical models that permits efficient inference. We discuss particular cases in clique graphs where exact inference is possible in polynomial time and some special cases where good approximation guarantees can be given. We also point out the drawbacks of learning with approximate inference. Finally, a practical application of probabilistic generative model for learning visual attributes from images is discussed.

1 Introduction

Probabilistic Graphical Models(PGMs) encode conditional independencies among random variables and help in reasoning them about collectively. They are particularly useful in applications where its beneficial to classify a group of objects collectively rather than individually. Since most inference problems are computationally hard many approximation techniques have been discovered. Junction tree(JT) is one such approximation method which is attractive due to its modularity and complexity guarantees.

In general a junction tree for a Markov random field $\langle V, G \rangle$ is a tree T where each node n is associated with a set a variables denoted $V(n)$ with $V(n) \subseteq V$ and the following conditions hold.

- **Cover Property:** For each $\Gamma \in G$ there exists a node n where $V(n)$ contains all variables on which Γ depends.
- **Running Intersection Property:** For any variable X, the set of nodes n with $X \in V(n)$ is a subtree of T.

The *width* of a junction tree is the maximum size of the set $V(n)$ over all nodes n of the tree minus 1. Complexity of inference is proportional to the width of the Junction Tree. The tree width of a Markov random field is the minimum width of any junction tree for that field. It is NP-hard to determine tree width. However, good heuristics exists for constructing junction trees of small width. Junction trees with small width (thin-junction trees) aim to trade complexity for accuracy by enforcing a bound on tree-width.

Firstly, we review an algorithm to learn bounded treewidth junction trees in polynomial time. In Section 3, we look at some situations involving clique graphs where exact inference or near-optimal inference is possible in polynomial

time. We review some cases where inexact inference and learning techniques can lead to poor results. In the last section, a generative model for learning visual attributes from images is presented.

2 Learning of thin junction trees

In order to use a probabilistic graphical model for inference one needs to define its structure and parameters. In most applications, it is not easy to manually define the structure of PGM. In such cases, the structure has to be learnt from a set of training instances. In this section, we see how to learn the structure of junction trees of bounded width using only polynomial number of training samples. Let $P(V)$ represent the probability distribution over discrete variables V. Let $\mathbb{C} = \{C_1, \ldots, C_m\}$ be the collection of subsets of V and T be the set of edges such that $(T, \mathbb{C})$ forms a junction tree. A set $S_{ij} \equiv C_i \cap C_j$ is called the separator corresponding to the edge $(i-j)$ from T. The size of the largest clique in a junction tree minus one is the treewidth of that tree. Let $\mathbb{C}^i_{ij}$ be the cliques that can be reached from C_i in the $(T, \mathbb{C})$ without using the edge $(i-j)$ and corresponding reachable variables be denoted by $V^i_{ij} \equiv V^i_{ji} \equiv \bigcup_{C_k \in \mathbb{C}^i_{ij}} C_k \setminus S_{ij}$. The distribution $P(V)$ is representable by tree $(T, \mathbb{C})$ iff $\forall (i-j) \in T, (V^i_{ij} \perp V^j_{ij} \mid S_{ij})$. If $P(V)$ factors according to some junction tree of treewidth k, we say that $P(V)$ is k-**JT-representable**. We consider only those **maximal** junction trees where all separators have size k. If P is k-JT representable, it also factors according to some maximal JT of treewidth k.

If separator S_{ij} splits the tree into into parts T_i and T_j we know that $T_i \perp T_j \mid S_{ij}$. This notion of conditional independence is very strong and usually not found in practice. Hence an alternate way to look at it would be to relax the CI with Conditional Mutual Information $I(A, B \mid S)$. I tells us given S how much does does knowing A or B reduce the entropy of other variable–naturally CI implies $I(A, B \mid S)$ is 0. $(T, \mathbb{C})$ is an $\epsilon -$ **junctiontree** for $P(V)$ iff $\forall (i-j) \in T : I(V^i_{ij} \perp V^j_{ij} \mid S_{ij}) \leq \epsilon$.

For an ϵ-JT for $P(V)$ it holds that

$$KL(P, P_{(T,\mathbb{C})}) \leq n\epsilon$$

This bound means that if we have ϵ-JT for $P(V)$ we can use a tractable principled approximation $P_{(T,\mathbb{C})}$ for inference. [CG08] presents the first polynomial time algorithm for PAC-learning the structure of such junction trees with bounded treewidth. k is assumed to be a constant and not part of the input. The complexity of this approach is exponential in k.

2.1 Junction tree learning algorithm

Let us assume that we have an oracle $I(\cdot, \cdot \mid \cdot)$ that can compute the mutual information $I(A, B \mid C)$ for any subsets $A, B, C \subset V$. Using oracle I the most direct approach to learn the ϵ-JT would be as follows:

- **Step 1** For all possible disjoint sets A,B,S with S=$|k|$ and $|A|+|B|+|S| = |V|$ calculate $I(A, B \mid S)$.
- **Step 2** Store the pairs (S, A) for which $I(A, B \mid S) \leq \epsilon$ in a list $\mathbb{L}$.
- **Step 3** Find a junction tree $(T, \mathbb{C})$ which is consistent with list $\mathbb{L}$.

We say a junction tree $(T, \mathbb{C})$ is consistent with $\mathbb{L}$ iff for every separator S_{ij} of $(T, \mathbb{C})$ it holds that $(S_{ij}, V_{ij}^i) \in \mathbb{L}$. Any consistent junction tree would be an ϵ-JT for $P(V)$.

Step 3 of the algorithm is solved using *constraint-based* approaches. However, there are three main problems with Step 1:

- Prob. 1: Each call to oracle I takes n variables as arguments–unfortunately, computing I is exponential in n
- Prob. 2: Since we are enumerating all possible sets A,B,S in Step 1, the number of calls to I itself is exponential.
- Prob. 3: We have assumed that oracle I exists. In reality, there is no way to compute $I(\cdot, \cdot \mid \cdot)$ exactly since actual distribution is not known but we are given only samples.

These are the ways in which the above problems are solved:

Sol 1: We can provide an upper bound on I by seeing only a subset of variables using the following property:

Lemma 2.1. *Let $P(V)$ be a k-JT ϵ-representable distribution. Let $S \subset V, A \subset V \setminus S$. If $\forall X \subseteq V \setminus S$ s.t $|X| \leq k+1$, it holds that $I(A \cap X, V \setminus \{S, A\} \cap X \mid S) \leq \delta$, then $I(A, V \setminus \{S, A\} \mid S) \leq n(\epsilon + \delta)$*

This provides a way to compute an upper bound on $I(A, V \setminus \{S, A\})$ using $O\left(\binom{n}{k}\right) = O(n^k)$ calls to oracle $I(\cdot, \cdot \mid \cdot)$ and each call will involve atmost $|S| + k + 1 (= 2k + 1)$ calls. The lemma also bounds the quality of approximation of P.

Sol 2: The key observation here is that the function $I(Q, V \setminus \{S, Q\} \mid S) \equiv F_S(Q)$ is **sub modular**: $F_S(A) + F_S(B) \geq F_S(A \cup B) + F_S(A \cap B)$. Queyranne's algorithm allows minimization of a submodular function in $O(n^3)$ time. We can use this fact and the definition of ϵ-junction tree to reduce the number of calls to polynomial order.

Sol 3: Computing I exactly is not possible, instead we can get a probabilistic estimate of $I(A, B \mid C)$, that has accuracy $\pm\triangle$ with probability $1 - \gamma$, using number of samples and computational time polynomial in $\frac{1}{\triangle}$ and $\log \frac{1}{\gamma}$, respectively.

Using the above modifications to the naïve algorithm we can learn junction trees with limited-treewidth in polynomial time with polynomial number of samples. This tree is guaranteed to be to close in KL-divergence to the true distribution.

3 Efficient inference in cliques

In this section we look at particular forms of inference problems where efficient MAP labelling is possible. [GDS07] describes efficient inference algorithms for potential functions of the form

$$f(\mathbf{y}) = np(\mathbf{y}) + cp(\mathbf{y})$$

where $\mathbf{y}$ is the label vector of size n with each y_i taking any one of the m labels. $np(\mathbf{y})$ is the sum of all node potentials and $cp(\mathbf{y})$ is the clique potential with depends only on the cardinality of y_is i.e $cp(\mathbf{y}) = f(n_1, \ldots, n_m)$. Potentials of this form arise in information extraction, hypertext learning, associative networks etc. For example, in hypertext document classification if a document needs to be labelled with a relevance category between 1 to m, the node potential is depends on the content of the document while edge potential depends on the number of its neighbours and their relevance categories.

3.1 Binary Labels

For the case when nodes can takes only binary labels the optimal labelling can be found in $O(n \log(n))$ time. Suppose we know the optimal assignment of labels has k nodes with label 0 and $n-k$ nodes with label 1, the clique potential in this case would be $C(k, n-k)$. Since $C(k, n-k)$ does not depend on the which of the nodes are labelled 0 or 1, we can independently maximize $np(\mathbf{y})$. Which are the best k nodes we need to pick to label them 0? Obviously, we need to pick those nodes with have high node potential for label 0 and low node potential for label 1. Hence, we sort nodes by their $\phi_0 - \phi_1$ values and pick the top k nodes(where ϕ_i is the value of potential for label i for that node). We run the procedure for each k from 0 to n with one-time sorting and a single pass.

3.2 Multi-label Case

In binary case, the clique potential can be any arbitrary function of k and $n-k$. For multi-label case, finding the optimal labelling is hard. Hence, we look at only 3 particular forms of clique potential where efficient inference is possible.

MAX: Clique potential is of the form $cp(\mathbf{y}) = max_y f_y(n_y)$. Suppose in the optimal labelling clique potential takes the value of $f_\alpha(k_\alpha)$, we know that this case corresponds to exactly k nodes getting labelled α. The problem now reduces to picking those k nodes to be labelled α which maximizes the node potential. This can be done by sorting nodes in decreasing order of $(\phi_\alpha - max_{y\neq\alpha}\phi_y)$. Checking this for $k = 1 \dots n$ and $\alpha = 1 \dots m$ can be done in $O(mn \log n)$ time (α-pass algorithm).

SUM: Clique potential is of the form $cp(\mathbf{y}) = \sum_y f_y(n_y)$. Exact inference for the general case is NP-Hard. However, for the case when $f_y(n_y) = \lambda \sum_j n_j^2$ where $\lambda > 0$ the algorithm described above(for the MAX case) gives the solution within a factor of 13/15 of the optimal bound(tight). In this case we modify the clique potential as $cp(\mathbf{y}) = max_y \lambda n_y^2$. When $cp(\mathbf{y})$ is an arbitrary function best known algorithm ($\alpha-$expansion) produces a solution within a factor of 1/2 of the optimal. However, an efficient way to pick nodes during $\alpha-$expansion can be given with a generalized version of α-pass algorithm. Instead of find the best k nodes to take the value α, we can pick the best k nodes that can take values from a subset of labels A. The algorithm grows exponentially with size of A, but with increasing approximation to the optimal.

MAJORITY: Clique potential is of the form $f_a(\mathbf{n})$, $a = \text{argmax}_y n_y$ i.e $cp\mathbf{y}$ depends only on the label with highest cardinality. For the general case, inference can be no better than 1/2 of the optimal. However, when $f_a(n) = \sum_y w_{ay} n_y$(linear majority function) a modified version of $\alpha-$pass algorithm gives a good approximation. The algorithm is similar to one in MAX case. Sort the nodes according to $\phi_\alpha + w_{\alpha\alpha} - max_{y\neq\alpha}\phi_y + w_{\alpha y}$. During the $\alpha-$pass discard all solutions where majority label is not α. Exact algorithm for majority potentials can be found by formulating it as a degree constrained bipartite matching problem. It appears that this problem can be solved in polynomial time using combinatorial approaches.

4 Drawbacks of Approximate Inference

In the previous section, we reviewed some approximate inference algorithms which provide MAP labels within a constant bound of the optimal. Popular

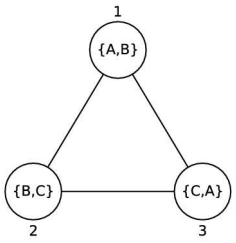

Figure 1: An Example Markov graph

approximate inference methods include LP-inference, loopy belief propagation, tree-reweighted message passing, etc. Here we discuss two problems with approximate inference techniques from [KP08]. They are

1. Even if the inference meets a good approximation guarantee it can reduce the expressive power of the model and make learning of even simple concepts impossible.
2. An inexact inference may misguide a learning algorithm thus making it impossible to find valid model parameters.

4.1 LP-inference

Consider the example shown in Fig. 1 where each node can take any of the two labels having associative edge potentials. Let $\lambda_{ij}(> 0)$ be the score obtained if node i and j take the same label and 0 if they differ. In this setting, the possible non-zero score labelings for nodes $(1, 2, 3)$ are $(B, B, ?)$, $(A, ?, A)$ $(?, C, C)$ with corresponding scores being λ_{12}, λ_{23} and λ_{31}. Optimizing the objective by formulating it as an integer programming problem is NP-Hard–so we use LP-relaxation approximation to solve this. Let λ_{i*j*} be the optimal score (which corresponds to assigning $i*$ and $j*$ nodes the same label and any arbitrary label to the third node) and $(\lambda_{12} + \lambda_{23} + \lambda_{31})/2 > \lambda_{i*j*}$. Since LP-relaxation may generate fractional solutions $(\lambda_{12} + \lambda_{23} + \lambda_{31})/2$ is reported as the answer. This answer is $< 3/2\lambda_{i*j*}$ and thus has a good approximation ratio of 3/2. However, this configuration corresponds to every node having equal probability of taking any of the two labels–this is useless since it basically says that all labelings are equally likely!

Consider the task of learning a model with weight vector $\mathbf{w}$ to predict labelings $\mathbf{y}$ from instances of the MRF in the above example, letting $\lambda_{ij} = w_{ij}x_{ij}$. It is possible that even when the samples are separable, the data cannot be correctly labelled with any choice of w vector using LP-relaxation.

4.2 Incompatible inference and learning

In order to illustrate the second problem, consider a setup where inference is done by loopy belief propagation(LBP) and learning happens by perceptron algorithm. The setup is as follows: We are given labelled training instances; for each instance and a given weight vector we estimate its label using LBP and compare it with the training label. If the predicted label is incorrect then we update the weight vector $\mathbf{w}$ accordingly and continue till we reach a $\mathbf{w}$ that predicts all instances correctly. The data given is separable.

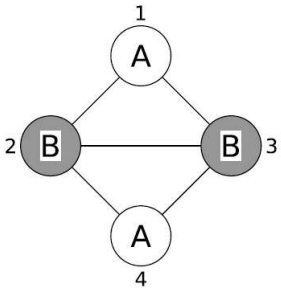

Figure 2: An MRF where LBP is inexact

Consider the MRF given in Fig 2. Each node can take two labels $\{-1, 1\}$ and potentials are assigned by type(A or B):

$\psi_A(-1) = 1 \quad \psi_A(1) = e^{\alpha}$

$\psi_B(-1) = 1 \quad \psi_B(1) = e^{\beta}$

where α and β are real valued parameters.

Let the edge potentials be sufficiently large to make the MAP configurations either $(1, 1, 1, 1)$ or $(-1, -1, -1, -1)$, abbreviated by $\mathbf{1}$ and $-\mathbf{1}$. The solution is $-\mathbf{1}$ if $\alpha + \beta < 0$ and $\mathbf{1}$ otherwise. Hence $\mathbf{y}_{MAP} = sign(\alpha + \beta)$.

We run LBP on this network and wait till convergence. The converged values of LBP are such that, MAP labelling can be done as $\mathbf{y}_{LBP} = sign(\alpha + \gamma\beta)$, where $\gamma \approx 1.0893$. This is not surprising since LBP in an approximate inference algorithm and the obtained answer is very close to the exact answer($\gamma = 1$).

Consider two training instances shown in Fig 3. The potentials of each node type depend on feature vectors $\mathbf{x}_\alpha$ and $\mathbf{x}_\beta$ for node type A and B, respectively. We wish to learn $\mathbf{w}$ that separates the two instances with potentials calculated by setting $\alpha = \mathbf{w} \cdot \mathbf{x}_\alpha$ and $\beta = \mathbf{w} \cdot \mathbf{x}_\beta$. Since, there are only two instances in this case there exists a $\mathbf{w}$ that has 0 training loss. In this case, it best $\mathbf{w}$ would be $(1, -1)$ since is maximizes the margin between the two instances.

- For training instance (a) Feature vector : $x_\alpha = [1, 0]^T$ and $x_\beta = [0, 0.95]^T$ and label = $-\mathbf{1}$

- For training instance (b) Feature vector : $x_\alpha = [0, 1]^T$ and $x_\beta = [0.95, 0]^T$ and label = $\mathbf{1}$
- We initialize $\mathbf{w}$ and update the weight vector for each misclassified instance.
- For a misclassified instance, weight vector is updated as $\mathbf{w} \leftarrow \mathbf{w} - \hat{y}([x_\alpha\ x_\beta])$
- Note that $\hat{y}$ is obtained using $\mathbf{y}_{LBP} = sign(\alpha + \gamma\beta)$

(a) $\mathbf{y} = -\mathbf{1}$ (b) $\mathbf{y} = \mathbf{1}$

Figure 3: Training set with their labelings

Let the initial weight vector be $(0, 0)$.
Iteration 1: For training instance (a) $\alpha+\gamma\beta = 0$ and $\mathbf{y}_{LBP}$ predicts label $\mathbf{1}$ when the training instance has label $-\mathbf{1}$. Hence, $\mathbf{w}$ is updated to $(0, 0) - \mathbf{1}(1, 0.95) = (-1, -0.95)$ For training instance (b) : $\alpha = \beta = -0.95$, therefore, $(\alpha + \gamma\beta) < 0$ and $\mathbf{y}_{LBP}$ predicts label $-\mathbf{1}$ when the training instance has label $\mathbf{1}$. $\mathbf{w}$ is again updated to $(-1, -0.95) - -\mathbf{1}(0.95, 1) = (-0.05, 0.05)$.

Similarly, iterations that follow take $\mathbf{w}$ in the direction of $(-0.05, 0.05)$ which diverges from the actual solution of $w = (1, -1)$. In this case inexact inference causes $\mathbf{w}$ to diverge from the actual solution, thus misguiding the learning algorithm.

Despite these problems with approximate inference techniques, appropriate pairs of inference and learning algorithms do exist and choosing them gives rise to a bound on *true* risk. LP-relaxation with structured perceptron given by Collins show cases where the perceptron makes only finite number of mistakes as long as the data is algorithmically separable.

5 An Application of Graphical Models

Lets look at an application of probabilistic generative model to identify visual attributes from a given image. Attributes refer to visual properties of objects like color, texture, etc. Given training instances we wish to learn a model $\mathcal{M}$ that

can identify attributes in images. This is potentially useful for specific image retrieval, where search queries like "a spotted blue skirt", "grainy brown purse", etc. can be answered more accurately. The model sees an image as collection of 'segments'. A segment is part of image that has uniform appearance. During the training phase a codebook $\mathcal{A}$ of appearances is built as follows:

- Randomly sample 5×5 pixel patches from set of training images.
- Apply k-means to patches to obtain k patch types.
- Represent each segment as a histogram over patch types.
- Cluster the segment histograms to obtain a codebook $\mathcal{A}$ of appearances.

We can think of each appearance as being a prototype segment descriptor.

Characterizing Visual Attributes

There are two types of attributes the model can learn:

Unary Attributes: which are characterized by properties of a single segment. Some unary attributes can be defined either by their appearance such as colors (eg.red, blue) or texture (eg.sand, grainy) and some by a segment shape (eg. round).

Binary Attributes: whose basic element is composed of two segments. 'Striped' attribute is a binary attribute which depend on two segments being adjacent to each other with same shape, nearly parallel and have comparable area.

Essentially, visual attributes can be thought of as repeated segments sharing some common properties.

5.1 Generative Model

Having these basic data-types we can proceed to build a generative model $\mathcal{M}$ (used to denote model parameters) that explains the whole image I. As mentioned earlier, an image is seen as a collection of segments. We can break up the image as foreground and background segments which are regions covered by the attribute and those that are not, respectively. Let f be the latent variable which is 1 if the segment is part of foreground and 0 when its part of background. Let $\mathbf{F}$ be the vector of all fs for the image. If the image has foreground appearance a shared by all foreground segments then the likelihood of the image in terms of each pixel x is given by Eq. 1 :

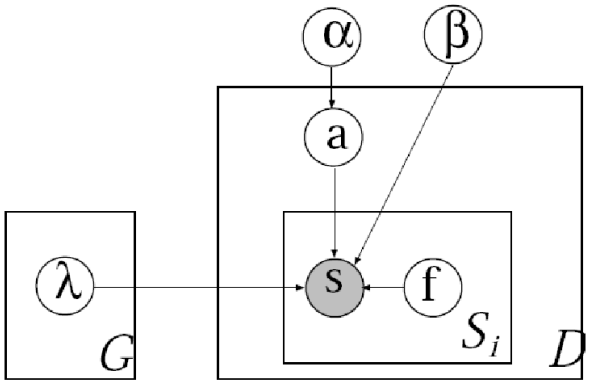

Figure 4: Graphical Model for Unary Attributes. D is the number of images in the dataset. S_i is is the number of segments in the image i and G is the total number of geometric properties.

Segment s is the only observed variable and $\mathbf{F}$ influences s directly. The appearance a and $\mathbf{F}$ is associated with each image the dependency is as shown in Fig. 4.

$$p(I|\mathcal{M};\mathbf{F},a) = \prod_{x\in I} p(x|\mathcal{M};\mathbf{F},a) \tag{1}$$

Term in RHS of Eq. 1 can be written as

$$p(x|\mathcal{M};\mathbf{F},a) = p(s^x|\mathcal{M};f,a) \tag{2}$$

where s^x refers to segment containing x. Eq. 2 follows because the probability of seeing any pixel of a segment is as likely as seeing the segment itself. For example, if an image contains 2 background segments of type s^b and 1 foreground segment s^f with equal area, then $p(s^f) = 1/3$ which is the same as probability of randomly picking a pixel and finding it to belong to s^f.

The image likelihood can be expressed as product over the probability of each segment times the number of pixels each one contains

$$p(I|\mathcal{M};\mathbf{F},a) = \prod_{x\in I} p(s^x|\mathcal{M};f,a) = \prod_{s\in I} p(s|\mathcal{M};f,a)^{N_s} \tag{3}$$

where N_s is the number of pixels in that segment. For simplicity, we focus only on unary attributes. Segments are the only observed variables in unary model. A segment $s = (s_a, {s_g}^i)$ is defined by its appearance s_a and shape which is captured by a set of geometric measurements ${s_g}^i$. For each visual attribute there

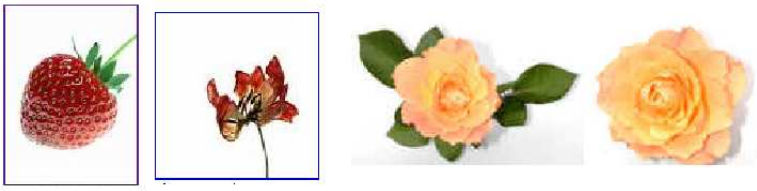

Figure 5: For the attribute red, two images on the left are positive training examples and two on the right are negative

are associated geometric properties which are *active*. The task of learning is to determine which properties are active for a visual attribute and their foreground distribution. For each geometric property $\lambda^j = (\Phi^j, v^j)$, Φ^j defines the distribution over foreground distribution and v^j denotes if that property is active or not. The conditional probability of image segments are therefore

$$p(x|\mathcal{M}; f, a) = \begin{cases} p(s_a|a) \cdot \prod_j p({s_g}^j|\Phi^j)^{v^j} & \text{if f=1} \\ \beta & \text{if f=0} \end{cases} \tag{4}$$

β is a scalar denoting the probability of segment being in background. It is usually set to a small value.

The image likelihood given in Eq. 2 depends on labels $\mathbf{F}$ and on the foreground appearance a. Computing the likelihood only in terms of model parameters $\mathcal{M}$ we get:

$$p(I|\mathcal{M}) = \max_{a \in \alpha} \max_{\mathbf{F}} p(I|\mathcal{M}; \mathbf{F}, a) \tag{5}$$

The objective now reduces to learning the best model parameters. Maximization over $\mathbf{F}$ is achieved by setting each f to the greater of two cases in Eq 4, while maximization over a requires trying out all appearences α in the codebook $\mathcal{A}$. There are only a small number of entries in the codebook of appearences so it is computationally inexpensive. The graphical model for the unary attribute case is shown in Fig. 4.

5.2 Learning

Instead of learning the likelihood directly, the model is learnt in discriminative fashion. The reason for this is best understood by an example. Consider Fig. 5 where we want to learn the attribute *red*. Both the images have prominent color as white and hence positive examples ($\mathcal{I}_+$) cannot be discriminated against the

negative examples($\mathcal{I}_-$) . For this reason, we maximize the *ratio* of the likelihood of images which is:

$$\frac{p(\mathcal{I}_+|\mathcal{M})}{p(\mathcal{I}_-|\mathcal{M})} = \frac{\prod_{I^i{}_+ \in \mathcal{I}_+} p(I_+|\mathcal{M})}{\prod_{I^i{}_- \in \mathcal{I}_-} p(I_-|\mathcal{M})} \tag{6}$$

The optimization of the objective function is done in an EM-like manner, the details of which can be found in [FZ08]. The paper also describes the how the likelihood changes in case of binary attributes.

6 Conclusion

It is important to realize that many approximation techniques in use may not be applicable in their original form. In most cases, a careful understanding of both the problem domain and inference techniques is needed for practical problem solving. Some of the applications where graphical models have proved to be useful are error-correction, speech recognition, segmentation, computer vision, gene regulation networks, etc.

References

[CG08] Anton Chechetka and Carlos Guestrin. Efficient principled learning of thin junction trees. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

[FZ08] Vittorio Ferrari and Andrew Zisserman. Learning visual attributes. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

[GDS07] R. Gupta, A. Diwan, and S. Sarawagi. Efficient inference with cardinality-based clique potentials. pages 329–336, 2007.

[KP08] Alex Kulesza and Fernando Pereira. Structured learning with approximate inference. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.