Structure Prediction of a Bayesian Network

Deepak Rishi

April 9, 2016

1 Introduction

The purpose of this project is to explore the area of Structure Prediction. Knowing the underlying structure of a dataset can significantly help in the area of supervised learning and feature Engineering. This project explores the gains in performance of supervised learning when the underlying Bayesian Network structure of a dataset is known and is used to do a Bayes Net inference for a classification task. Its performance has been compared to that of Document based Naive Bayes Model.

1.1 What is the application domain?

The application domain is primarily structure prediction for supervised learning problems (both categorical and continuous data). The algorithm used can be applied to a wide variety of datasets -*text*, *music*, *images* and *time series data*.

1.2 What is the problem?

General Purpose Machine Learning Tasks like Image Classification, Sentiment Analysis can be solved with a very high accuracy using Deep Learning. However, the prerequisite of deep learning models is the availability of a large amount of labelled data. While, images and general purpose text (like newsgroups, e.t.c.) are publically available, when making a machine learning model for a custom specific task (say document categorization according to a very specific criteria), we may not have access to a large amount of labelled data. Using Deep learning models is very difficult in such situations. Manual feature engineering is required when we have limited access to labelled data. Although, feature engineering requires a certain amount of domain knowledge, even with that one has to experiment with a number of possible different combinations of features that tend to work out the best.

In such a case, knowing the structure of a dataset that describes how different features are dependent on one another can prove to be really helpful.

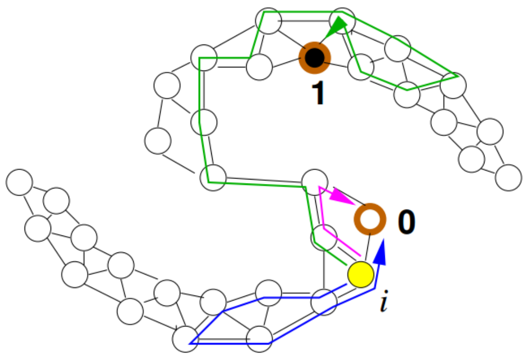
In cases where the structure of a dataset is not known, Naive Bayes Algorithm is applied. Although in many cases it gives good results (Email Spam classification, Text categorization, etcetera) , it does not take into account the different probabilistic dependencies between the features. This project aims to first infer those probabilistic dependencies between the features and exploit them in classification problems.

2 Techniques to tackle the problem

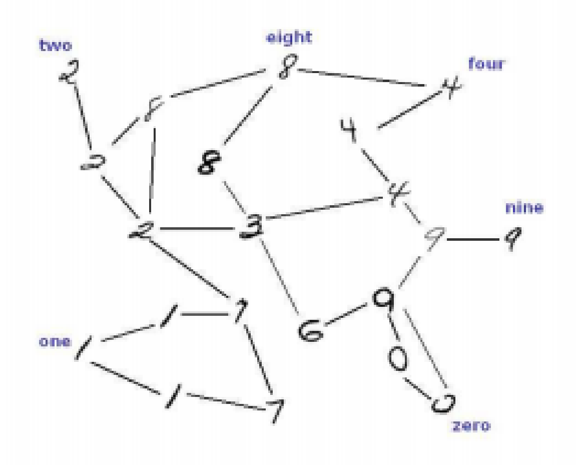
Structure prediction of a dataset is a very wide topic. There are many different ways to infer the structure of a dataset. Some of the techniques include (but not limited to)

2.1 Graph Based Semi Supervised Learning

Graph based Semi Supervised learning first builds a graph of the dataset based on a similarity metric and then applies label propagation to infer the labels of unlabeled points.[12]. Figure 1 shows the application of the technique.



(a) Random Walks on Graph[4]



(b) Similarity Graph on Images[12]

Figure 1: Graph Based Semi Supervised Learning

2.2 Neural Networks

Neural Networks can be also used to infer the structure of a dataset. They have been widely used to identify the protein secondary structure prediction [10]. Figure 2 shows an example of how a neural net might be trained to identify the structure of proteins.

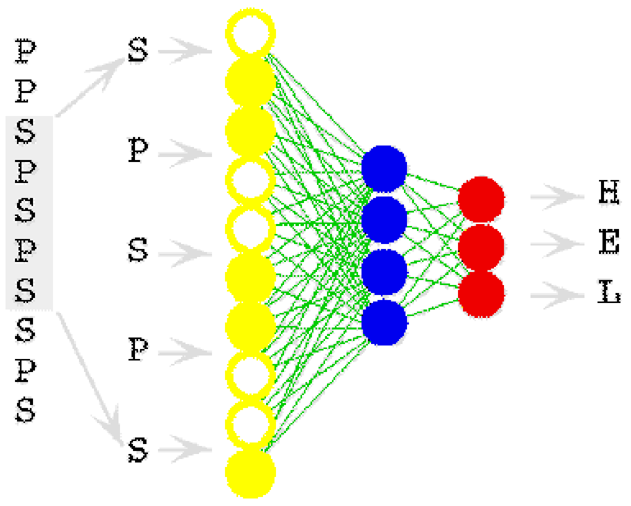


Figure 2: Protein Structure Prediction using Neural Network[3]

2.3 Learning a Bayes Net Structure from a dataset

Another intuitive way to identify the structure of a dataset is to learn a Bayesian Network for that data. A Bayesian network is a probabilistic graphical modelthat represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). Figure 3 shows an example of a Bayes Net.

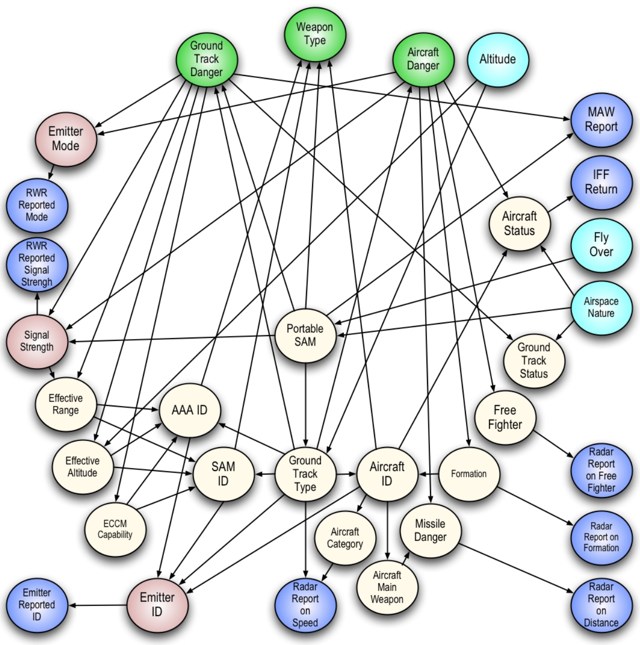


Figure 3: Example of an Bayesian Network[1]

2.3.1 Different Techniques to Learn a Bayes Net

There are three approaches to learn a Bayes Net from a data

• Constraint Based Structure Learning : These approaches view a Bayesian Network as a representation of independencies. They try to test out the conditional dependence and independence in the data and then try to find a network or a class of networks that best fit the data

• Score Based Structure Learning : These approaches view the Bayesian Network as a statistical model and then address the learning problem as a model selection problem.

• Bayesian Model Averaging : This method generates an ensemble of possible structures and then averages the prediction of all the structures (similar to Bayesian Learning)

In this project I explore the Score Based Learning which defines a score function that defines how well a structure matches the data. Even with a scoring function this problem is $NP-Hard$, so certain *heuristic* search techniques are used. Score based methods evaluate the whole structure at once so they are less sensitive to the individual features and better at making compromises between the extent to which variables are dependent in the data and the cost of adding an edge.

For a complete dataset (with no missing values), the maximum likelihood score based on the Bayes Net can be used to infer the correctness of the learnt Bayes Net.

Likelihood Score can be defined as

$$maxScore_L(G, D) = maxl(\theta, G) : D \tag{1}$$

where θ is the set of parameters used to define the values of the random variables in the Bayes Net.

For a general Bayes net graph the likelihood score can be written as

$$Score_L(G, D) = M \sum_{i=1}^{n} I_p(X_i; Pa_{X_i}) - M \sum_{i=1}^{n} H_P(X_i) \tag{2}$$

where I_P is the mutual information between the node and its parent, H is the entropy of the node and M is the number of training instances.

With the above likelihood function it is possible to have overfitting because the score will increase every time we add an edge to our bayes net.

To counter this Bayesian Information Criteria is used which penalizes the amount of edges in the graph. This is done by subtracting $log(\frac{M}{2})Dim(G)$ from the likelihood function.

However the problem is still $NP - Hard$ in the space of possible structures. To help with this we only consider *Tree Structured Networks.* They are less susceptible to overfitting and have sparse parameterization which enables them to generalize well.

2.3.2 Tree Structured Networks

A network G is called a Tree Structured Network if each variable X has at most 1 parent. The advantages of using a tree structured network are

• They can be learnt efficiently - in polynomial time

• They are sparse therefore avoid most overfitting problems.

• They also capture the most important dependencies and can provide some insight into the domain.

• They also provide a better baseline for approximating the distribution than the set of independent marginals of different variables.

2.3.3 Chow Liu algorithm

The Chow Liu algorithm [5] states that the best tree-structured network can be found by minimizing the Kullback-Leibler divergence between the true distribution and the tree-structured network distribution.

$$KL(P(X)||T(X)) = \sum_{k} P(X = k) log(\frac{P(X = k)}{T(X = k)}) \tag{3}$$

where P(X) is the true distribution, T(X) is the tree-structured network.

The algorithm further states that to minimize $KL(P||T)$, it suffices to find the tree network T that maximizes the sum of mutual informations over its edges.

Mathematically,

$$KL(P(X)||T(X)) = \sum_k P(X=k) log(\frac{P(X=k)}{T(X=k)}) = -\sum_i I(X_i, Pa_{X_i}) + \sum_i H(X_i) - H(X_1, X_2......X_N) \quad (4)$$

The second term in 4 does not depend on the Tree network at all. So, all that remains is to maximize the mutual information between the nodes X_i and their parents.

2.3.4 Chow Liu Algorithm using Greedy Heuristic Search

Since from 2.3.3 we inferred that maximizing the mutual information will minimize the KL divergence, we can use the following greedy algorithm to find the undirected Bayes net graph.

• For each pair of vars A,B, use data to estimate P(A,B), P(A), P(B)

• For each pair of vars A,B calculate mutual information

• Calculate the maximum spanning tree over the set of variables, using edge weights I(A,B).

• Transform the resulting undirected tree to a directed one by choosing a root variable and setting the direction of all edges to be outward from it.

Since the selection of root does not affect the log-likelihood of the tree the choice of selection of the root node does not matter. Based on the arrows made, the Chow Liu algorithm states that the given Bayes Net will best represent the data.

2.3.5 Tree Augmented Naive Bayes (TAN)

Naive Bayes suffers from the major drawback of not taking into account the different dependencies between the features given the label of the data. [11] explains Tree Augmented Naive Bayes in which conditioned on the label we learn a bayes net graph among the different features in a dataset and then do a Bayes Net Inference to solve classification problem.

Tree augmented naive Bayes is a semi-naive Bayesian Learning method. It relaxes the naive Bayes attribute independence assumption by employing a tree structure, in which each attribute only depends on the class and one other attribute. A maximum weighted spanning tree that maximizes the likelihood of the training data is used to perform classification. Figure 4 shows TAN.

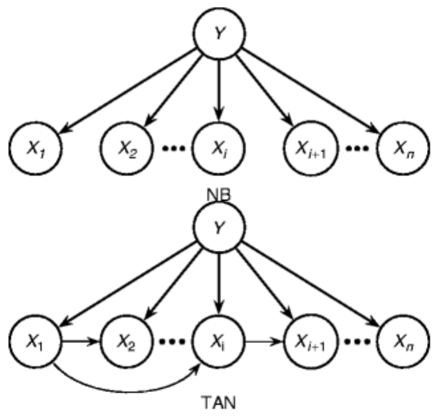


Figure 4: Naive Bayes and Tree Augmented Naive Bayes[11]

3 Empirical evaluation of Naive Bayes and Tree Augmented Naive Bayes

I ran Naive Bayes and TAN on the Chess (King-Rook vs. King-Pawn) End Game Data Set [2]. The properties of the dataset are described below

• Number of Instances: 3196

• Number of Attributes: 36 (all categorical)

• Classes (2): – White-can-win ("won") and White-cannot-win ("nowin").

• Class Distribution: In 1669 of the positions (52%), White can win. In 1527 of the positions (48%), White cannot win.

• The format for instances in this database is a sequence of 37 attribute values. Each instance is a board-descriptions for this chess endgame. The first 36 attributes describe the board. The last (37th) attribute is the classification: "win" or "nowin". There are 0 missing values. A typical board-description is

f,f,f,f,f,f,f,f,f,f,f,f,l,f,n,f,f,t,f,f,f,f,f,f,f,t,f,f,f,f,f,f,f,t,t,n,won

The names of the features do not appear in the board-descriptions. Instead, each feature corresponds to a particular position in the feature-value list. For example, the head of this list is the value for the feature "bkblk". The following is the list of features, in the order in which their values appear in the feature-value list:

[bkblk,bknwy,bkon8,bkona,bkspr,bkxbq,bkxcr,bkxwp,blxwp,bxqsq,cntxt,dsopp,dwipd, hdchk,katri,mulch,qxmsq,r2ar8,reskd,res
skrxp,spcop,stlmt,thrsk,wkcti,wkna8,wknck,wkovl,wkpos,wtoeg]

• Training set size used : 2130 instances. Number of *wons* are 1109 number of *no wins* are 1021.

• Testing set size used : 1066 instances. Number of *wons* are 560 number of *no wins* are 506.

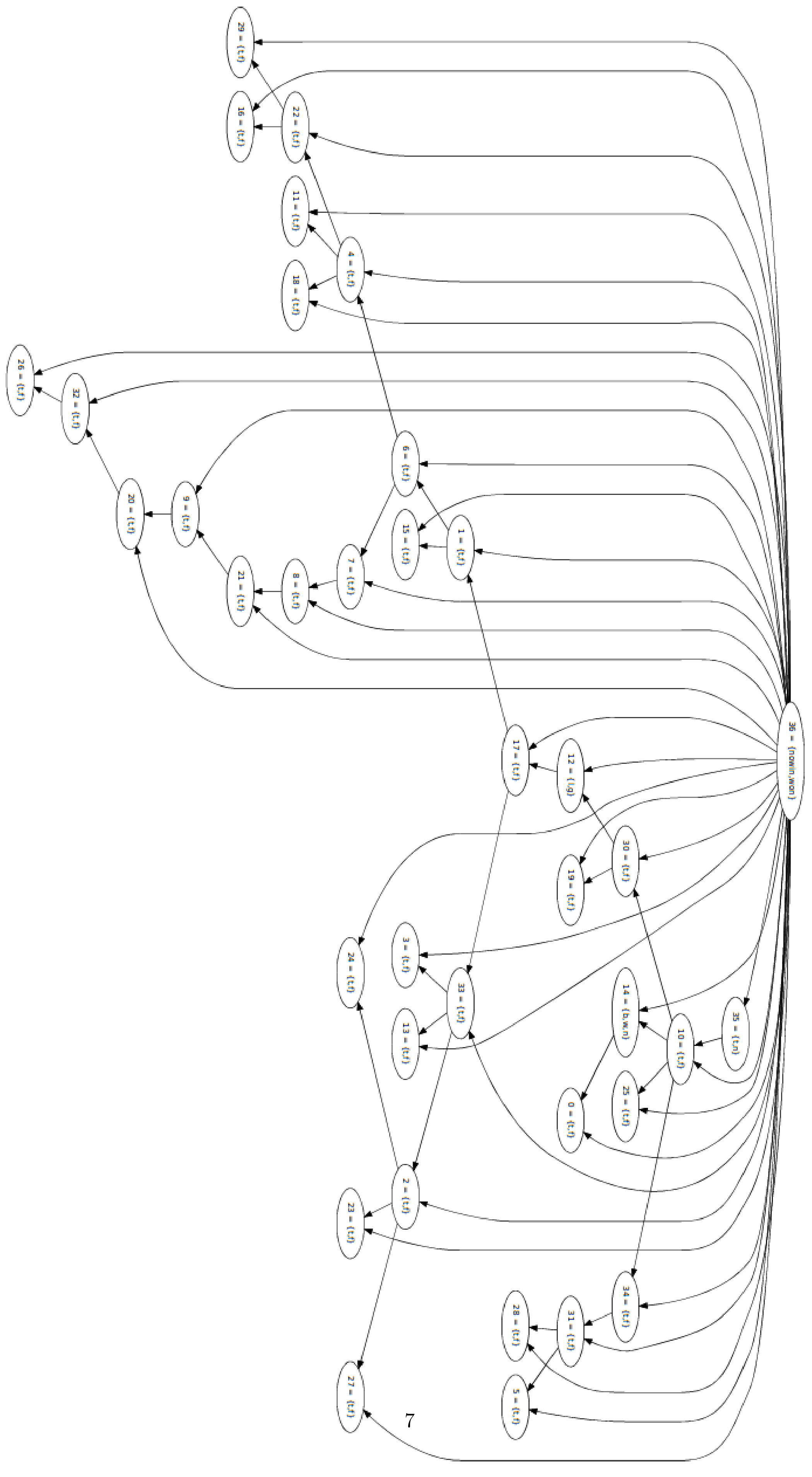
• 35 of the 36 attributes are boolean the parameters of which have a $Beta(2,2)$ Prior and one attribute (number 15) is multinomial with 3 values which has a Dirichlet Prior(2,2,2) over it. The same prior is used for both Naive Bayes and for TAN.

	Naive Bayes	TAN
Test Set Accuracy	87.05 %	93.34%

Table 1: Comparison between Naive Bayes and Tree Augmented Naive Bayes

FIgure **??** shows the comparison between Naive Bayes and Tree Augmented Naive Bayes.

The Bayes Net learnt from the dataset is shown in the Figure 5. The number is the attribute number in the dataset its possible values are also shown. (Image has been rotated in order to fit on the page and read the attributes numbers clearly)


36 = {nowin,won}
29 = {t,f}
16 = {t,f}
22 = {t,f}
11 = {t,f}
18 = {t,f}
4 = {t,f}
26 = {t,f}
32 = {t,f}
6 = {t,f}
20 = {t,f}
9 = {t,f}
15 = {t,f}
1 = {t,f}
7 = {t,f}
21 = {t,f}
8 = {t,f}
17 = {t,f}
12 = {l,g}
30 = {t,f}
19 = {t,f}
24 = {t,f}
3 = {t,f}
33 = {t,f}
13 = {t,f}
14 = {b,w,n}
35 = {t,n}
10 = {t,f}
25 = {t,f}
0 = {t,f}
2 = {t,f}
23 = {t,f}
34 = {t,f}
31 = {t,f}
28 = {t,f}
5 = {t,f}
27 = {t,f}

4 Conclusion

Tree Augmented Naive Bayes 2.3.5 was able to give us a Bayes Net of a dataset in polynomial time. It was also able to capture interesting dependencies between the different features.

Using Neural Nets for this task was not an option because of the lack of labeled data.

Graph based Semi Supervised Learning could have been used here, but that required a similarity matrix to be constructed beforehand and some additional feature engineering such as $tf - idf$ would have been required to first transform the data into a high dimensional space.

Thus, Tree Augmented Naive Bayes proved to be the most efficient algorithm to use in this case.

4.1 Future Research

Tree Augmented Naive Bayes can be further used for feature engineering for Text data. Given how different features are dependent on each other , it can prove to be very important factor while doing Manual Feature Engineering. My thesis will be on Natural Language Processing and Social Network Analysis. I will be using Tree Augmented Naive Bayes to form interesting features for text data. Also, I plan to host this algorithm online so that people can upload their datasets and generate a Bayes Net for it. A couple of things that I will be exploring in this area are

• Generating a Bayes Net with a mixture of categorical and continuous Variables.

• For continuous variables compare the results with Gaussian Naive Bayes(since it can be used for Nonlinear Classification)

• Generating a general and more complicated graph (non tree structured) for a dataset.

References

[1] Bayesian Networks. `http://www.pr-owl.org/basics/bn.php`. Accessed: 2016-04-08.

[2] Chess End Game Database. `https://archive.ics.uci.edu/ml/datasets/Chess+(King-Rook+vs.+King-Pawn)`. Accessed: 2016-04-08.

[3] Evolution teaches neural networks to predict protein structure. `https://www.rostlab.org/papers/1999_honnef/paper.html`. Accessed: 2016-04-08.

[4] Graph Based Semi Supervised Learning. `http://mlg.eng.cam.ac.uk/zoubin/talks/lect3ssl.pdf`. Accessed: 2016-04-08.

[5] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, May 1968.

[6] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[7] Liangxiao Jiang, Zhihua Cai, Dianhong Wang, and Harry Zhang. Improving tree augmented naive bayes for class probability estimation. *Knowledge-Based Systems*, 26:239–245, 2012.

[8] Liangxiao Jiang, Harry Zhang, Zhihua Cai, and Jiang Su. Learning tree augmented naive bayes for ranking. In *Database Systems for Advanced Applications*, pages 688–698. Springer, 2005.

[9] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, volume 96, pages 202–207. Citeseer, 1996.

[10] C. Ruggiero, R. Sacile, and G. Rauch. Peptides secondary structure prediction with neural networks: a criterion for building appropriate learning sets. *IEEE Transactions on Biomedical Engineering*, 40(11):1114–1121, Nov 1993.

[11] Fei Zheng and Geoffrey I. Webb. *Encyclopedia of Machine Learning*, chapter Tree Augmented Naive Bayes, pages 990–991. Springer US, Boston, MA, 2010.

[12] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *IN ICML*, pages 912–919, 2003.