Weight Learning for Partial Observation in Markov Logic Networks

A thesis submitted in partial fulfillment
of the requirements for the degree of

MASTER OF TECHNOLOGY

in

Computer Science & Engineering

by

Aswin

Entry No. 2011MCS2587

Under the guidance of

Dr. Parag Singla

Department of Computer Science and Engineering,
Indian Institute of Technology Delhi.
May 2013.

Certificate

This is to certify that the thesis titled **Weight Learning for Partial Observation in Markov Logic Networks** being submitted by **Aswin Sashidharan** for the award of **Master of Technology** in **Computer Science & Engineering** is a record of bona fide work carried out by him under my guidance and supervision at the **Department of Computer Science & Engineering**. The work presented in this thesis has not been submitted elsewhere either in part or full, for the award of any other degree or diploma.

Dr. Parag Singla
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Abstract

Machine learning and data analytics are of widespread use in areas ranging from Social networks to Medicine. Approaches such as Markov Logic Networks have already succeeded in handling uncertainty and representing a wide variety of knowledge compactly.

Even with concretely proven algorithms and methods, the biggest issue faced is shortage of labeled data for training. The inherent characteristics of many of the domains does not allow it to be observed completely.

We present methods to approximately learn weights of a Markov Logic Network in the case of partially observed data. Weight learning in Markov Logic Networks is done by maximizing the likelihood of one or more relational databases. The gradient used in the maximization has been optimized to compensate for the missing data. Also, the standard Expectation Maximization algorithm has been tailored to work with Markov Logic Networks.

The two algorithms have been incorporated and tested in Alchemy, an implementation of Markov Logic Network.

Experiments were run using real world databases to illustrate the promise of this approach. The database contained fully observed data which was then partially removed, so that the results could be compared.

Acknowledgments

I express my sincere gratitude to my guide Dr. Parag Singla, for introducing me to this problem and for his constant support, guidance and motivation throughout this work.

I would like to thank my parents for always believing in me, and standing by me through all the decisions I have made.

I would also like to thank all my friends; both here at IIT Delhi and back home, for all their help and support, both technical and otherwise.

I would like to thank the creators of Alchemy, the cora database and the uw-cse database without which this work could not have been completed.

I would also like thank the creators of Google and Wikipedia for enabling us to have a miraculous amount of information at our fingertips.

Aswin Sashidharan

Contents

List of Figures

Chapter 1

Introduction

1.1 Machine Learning

Machine learning has long since moved out of research labs into the real world, solving real world problems. Many of the prototype algorithms has been scaled to fit terabytes of data and to run on hundreds of server nodes on cloud.[1]

For problems in various domains, many machine learning algorithms have been developed, tried and teseted extensivesly.

Extensive research is still happening in the field. Many of our day-to-day applications include Machine learning. Applications like Facebook have face recognition algorithms[2] on the photos uploaded buy users. The training data is generated by the users who tag their friends in photos. Facebook can filter out many of the wrong tags using a face detection algorithm which in turn is another machine learning algorithm.

Image processing and computer vision are now used extensively in detecting anomalies in medical diagnostic[17][8] methods such as Computed Tomography scan and Magnetic resonance imaging.

Email providers have hundreds of servers running machine learning algorithms for spam classification[3] and personalised advertising.

Machine learning has even enabled the creation of self-driving-cars[16]. It is also applied in robot locomotion.

1.2 Motivation

Development of machine learning algorithms require clean and labeled data for training and testing purposes. This has always been a hurdle for research groups. There exist many demographics which may impossible to completely observe. A naive example would be a study of incidence of cancer in smokers and non-smokers in a social circle. Assume a person conducts a survey to collect information about this. People might be reluctant to reveal information like this.

A more real world example would be to building an anonymous chat app on a social network such as Facebook. The app could look at a person's "likes" on books, music, movies, people etc, to make suggestions. Now a user can choose not to reveal information about the books he has read. The app now has only partial data. This could completely break the suggestion algorithm if this issue has not been taken care of.

1.3 Markov Logic Network

Machine learning problems have to handle the challenges of uncertainty and complexity. Many approaches have been proposed to solve these problems Efficiently. Markov Logic Networks[13] combine both Probabilistic Graphical Models[7][10] and First order logic.

Markov Logic Network can be viewed as a template for generating Markov Network. It is possible to represent Markov networks of extremely high number of nodes and edges compactly using Markov Logic Network. Parameters of the model are learned[11] my maximizing the likelihood of a relational database. Approaches have been proposed to learn weights by optimizing the conditional likelihood[14] of the query predicates given the evidence ones, rather than the joint likelihood of all predicates.

In weight learning, these approaches make a closed world assumption. i.e. the truth value of every predicate is known at learning time. Absence of completely observed data results in the failure of these learning algorithms. It is still possible to use the rest of the available data to estimate the model parameters which could give results comparable to that of complete observation.

Note that this work does not propose any new algorithms. The algorithms used are already well known in literature.
One method will be to directly optimize the gradient[9] within the framework of Markov Logic Network. This would help us to learn weights even in the presence of hidden data.

Another approach will be to take the standard Expectation Maximization[4] Algorithm and tailor it to the Markov Logic Network Framework. The expectation maximization algorithm enables parameter estimation in probabilistic models with incomplete data. It is an iterative procedure. The Expectation Maximization algorithm has the following steps.

1. Randomly initialize parameters.
2. Find the best value of hidden values using these parameters.
3. Use these computed values to re-estimate the model parameters.
4. Repeat 2 and 3 until convergence.

The direct gradient method and the expectation maximization algorithm has been incorporated into an implementation of Markov Logic Networks. The training and testing has been done on real world databases.

Chapter 2

Background

2.1 Markov Logic Network

First order Logic has always been a viable candidate for modeling complex worlds. The formulas in first order logic are universally quantified. The constraints in a first order knowledge base are hard, in the sense that if a world violates even one formula, then it is impossible. Markov Logic Networks attempts to soften the constraints of First order Logic. i.e. When a world violates one formula in the knowledge base, it becomes less probable and not impossible. If the world continues violating more and more formulas, it becomes less and less probable.

Markov Logic Networks[13] softens the constraints by associating a weight with each of the rules. The weight is an indication of how strong the constraint is.the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal.

Definition 2.1.1. *A Markov logic network L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, ..., c_{|C|}\}$, it defines a Markov network $M_{L,C}$ (Equations 1 and 2) as follows:*

1. *$M_{L,C}$ contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1 if the ground atom is true, and 0 otherwise.*

2. *$M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.*

Each state of $M_{L,C}$ represents a possible world. The following assumptions ensure that the set of possible worlds for (L, C) is finite, and that ML,C represents a unique, well-defined probability distribution over those worlds, irrespective of the interpretation and domain. These assumptions are quite reasonable in most practical applications, and greatly simplify the use of MLNs.

- **Unique names**. Different constants refer to different objects[6].
- **Domain closure.** The only objects in the domain are those representable using the constant and function symbols in (L, C) [6].
- **Known functions.** For each function appearing in L, the value of that function applied to every possible tuple of arguments is known, and is an element of C.

The probability distribution over a possible domain x, specified by the ground Markov Network $M_{L,C}$ is

$$P(X = x) = \frac{1}{Z} exp\left(\sum_i w_i n_i(x)\right) = \frac{1}{Z}\prod_i \phi_i(x_i)^{n_i(x)} \tag{2.1}$$

where $n_i(x)$ is the number of true groundings of F_i in x, x_i is the state (truth values) of the atoms appearing in F_i , and $\phi_i(x_i) = e^{w_i}$.

2.1.1 Inference

Markov Logic Networks are able to answer arbitrary queries of the form "What is the probability that formula F1 holds given that formula F2 does?"

$$
\begin{aligned}
P(F_1|F_2, L, C) &= P(F_1|F_2, M_{L,C}) \\
&= \frac{P(F_1 \wedge F_2|M_{L,C})}{P(F_2|M_{L,C})} \\
&= \frac{\sum_{x \in X_{f_1} \bigcap X_{f_2}} P(X = x|M_{L,C})}{\sum_{x \in X_{f_2}} P(X = x|M_{L,C})}
\end{aligned} \tag{2.2}
$$

Where F_1 and F_2 are 2 first order logic formulas, C is a finite set of constants including any constants that appear in F_1 or F_2 , and L is an MLN

On non-trivial domains, computing Equation (2.2) will be intractable. Markov Chain Monte Carlo methods like Gibbs Sampling[5] and MC-SAT[12] are used in approximating it. The basic Gibbs step consists of sampling one ground atom given its Markov blanket. The Markov blanket of a ground atom is the set of ground atoms that appear in some grounding of a formula with it.

2.1.2 Learning

Weights for a Markov Logic Network can be learned using one or more relational databases. A database is effectively a vector $x = (x_1, ..., x_l, ..., x_n)$ where x_l is the truth value of the lth ground atom ($x_l = 1$ if the atom appears in the database, and $x_l = 0$ otherwise). This is if me make a closed world assumption. i.e. every ground atom not in the database is assumed false. Learning weights[11] includes maximizing the likelihood of the database and can be done using gradient ascent.

The gradient can be calculated as follows.

$$
\begin{aligned}
P(X = x) &= \frac{1}{Z} e^{\sum_i w_i n_i(x)} \\
logP(X = x) &= log\frac{1}{Z} + \sum_i w_i n_i(x) \\
\frac{\partial}{\partial w_i} logP(X = x) &= \frac{\partial}{\partial w_i} \sum_i w_i n_i(x) - \frac{\partial}{\partial w_i} logZ \\
&= n_i(x) - \frac{\partial}{\partial w_i} logZ \\
&= n_i(x) - \frac{1}{Z} . \frac{\partial}{\partial w_i} Z \\
&= n_i(x) - \frac{1}{Z} \frac{\partial}{\partial w_i} \sum_{x' \in X} e^{\sum_i w_i n_i(x')} \\
&= n_i(x) - \frac{1}{Z} \sum_{x' \in X} e^{\sum_i w_i n_i(x')} . \frac{\partial}{\partial w_i} \sum_i w_i n_i(x') \\
&= n_i(x) - \sum_{x' \in X} \frac{1}{Z} e^{\sum_i w_i n_i(x')} . n_i(x) \\
&= n_i(x) - \sum_{x' \in X} P_w(X = x') . n_i(x) \qquad (2.3)
\end{aligned}
$$

where the sum is over all possible worlds x' , and $P_w(X = x)$is $P(X = x)$ computed using the current weight vector $w = (w_1, ..., w_i, ...)$. The i^{th} component of the gradient is the difference between the number of true groundings of the i^{th} formula in the data and its expectation according to the current model.

2.2 Expectation Maximization

The EM algorithm is used to approximate a probability function. EM is typically used to compute maximum likelihood estimates given incomplete samples. Let X be the set of hidden variables, Y the set of observable variables, and Ω_X and Ω_Y the set of values or sample space for these variables. The product of Ω_X and Ω_Y can be thought of as the space of complete samples and Ω_Y the set of incomplete samples. If (x, y) is a complete sample,

then y is the observable part.

Let $f(x, y|\theta)$ specify a family of functions one of which governs the generation of complete samples and let $g(y|\theta)$ specify a family of functions one of which governs the generation of incomplete samples. The functions f and g are related by the following equation.

$$g(y|theta) = \int_{x \in \Omega_X} f(x, y|\theta) dx$$

The EM algorithm attempts to find a value for θ which maximizes $g(y|\theta)$ given an observed y.

Chapter 3

Algorithms

Probabilistic Graphical models[10] are widely used to model data. The existence of efficient and robust algorithms to learn parameters from observations make them popular. But incomplete or partial observations of data make the parameter learning difficult. This occurs very often in real world. For example in medical diagnosis, patient histories are recorded only using a limited set of tests.

The frequent occurrences of partial observation makes it crucial to create algorithms which enable parameter estimation for incomplete data.

3.1 Weight Learning for complete data

The joint probability distribution specified by a Markov Logic network is :

$$P(X = x) = \frac{1}{Z} \prod_i \phi_i(x_i)^{n_i(x)} \tag{3.1}$$

$x_i$is the state of the ith clique.

$$Z = \sum_{x \in X} \prod_i \phi_i(x_i)$$

The log linear form is:

$$P(X = x) == \frac{1}{Z} exp\left(\sum_i w_i n_i(x) \right) \tag{3.2}$$

Along with the rules written in first order logic, a relational database consisting of one or more possible worlds are given to the learning algorithm. Weights are learned by maximizing the likelihood of the relational database.

To do that, we need to perform gradient ascent on the gradient of the joint probability distribution. The gradient works out to be:

$$\frac{\partial}{\partial w_i} log P_w(X = x) = n_i(x) - \sum_{x'} P_w(X = x') n_i(x') \tag{3.3}$$

It simply means that the i^{th} component of gradient is the difference between the number of true groundings in the data and its expectation according to the current model. To calculate the expected counts of groundings, inference needs to be performed on the current model. For this random initial weights are assigned to the rules. The Learning Algorithm is given in 1:

Algorithm 1: Weight learning for complete observation

Data: MLN, Ground clause database, non-evidence predicates
Result: Weights for the MLN
Create the ground network from the MLN and the database;
Assign random weights to MLN;
Find out the actual count from the database;
while *Not converged* **do**
    Run inference for non-evidence predicates;
    Find out the expected count from the inference;
    Find the difference of the actual and expected counts(Gradient);
    Update the weights;
end

This can be explained with the help of an example. Consider the following Markov Logic Network.

- $Smokes(x) \Rightarrow Cancer(x)$ 1.5
- $Friends(x, y) \Rightarrow (Smokes(x) \Rightarrow Smokes(y))$ 1.1

The numbers represents the weights of the rules. The database consists of the following two constants.

- Anna(A)

- Bob(B)

Let "Smokes" and "Cancer" be the non-evidence predicates. The ground network generated is given in figure 3.1.

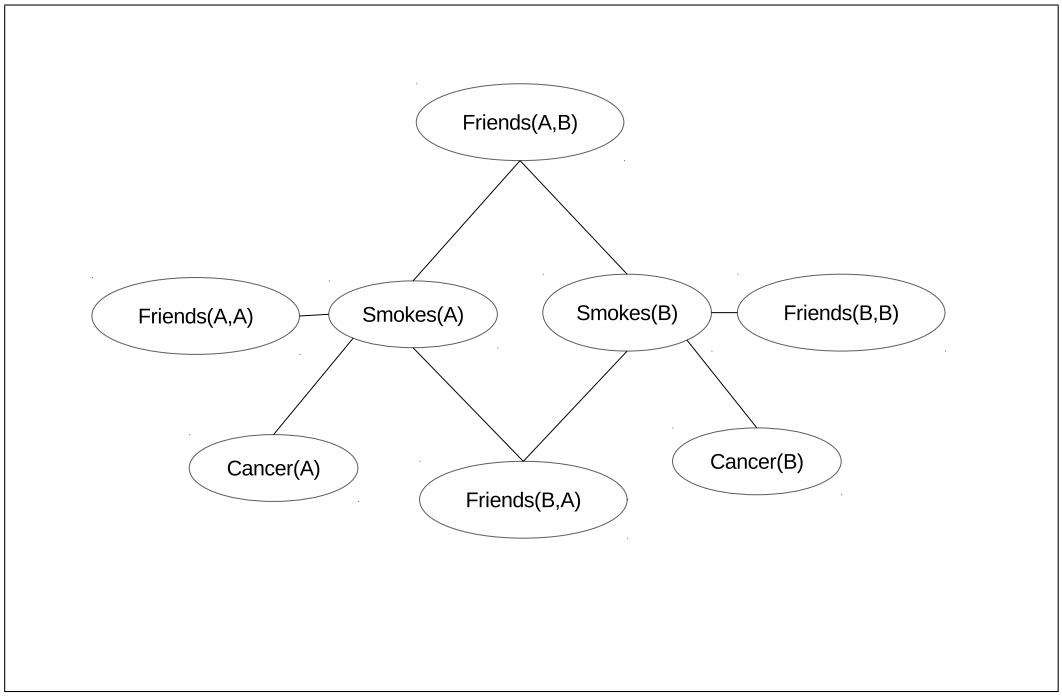

Figure 3.1: Ground network generated for Smokers

After creating the ground network, assign random weights to the clauses in the MLN. Now predict the values for Smokes(Anna), Smokes(Bob), Cancer(Anna) and Cancer(Bob) using the current model. (Figure 3.2).

Now, the true groundings are obtained from the database by finding out the actual values of Smokes(Anna), Smokes(Bob), Cancer(Anna) and Cancer(Bob). The inferred values give the expected count. Now the gradient can be calculated using equation (3.3). The gradient is simply the difference of these counts.

The weights are now updated using the gradient. This process is repeated until convergence.

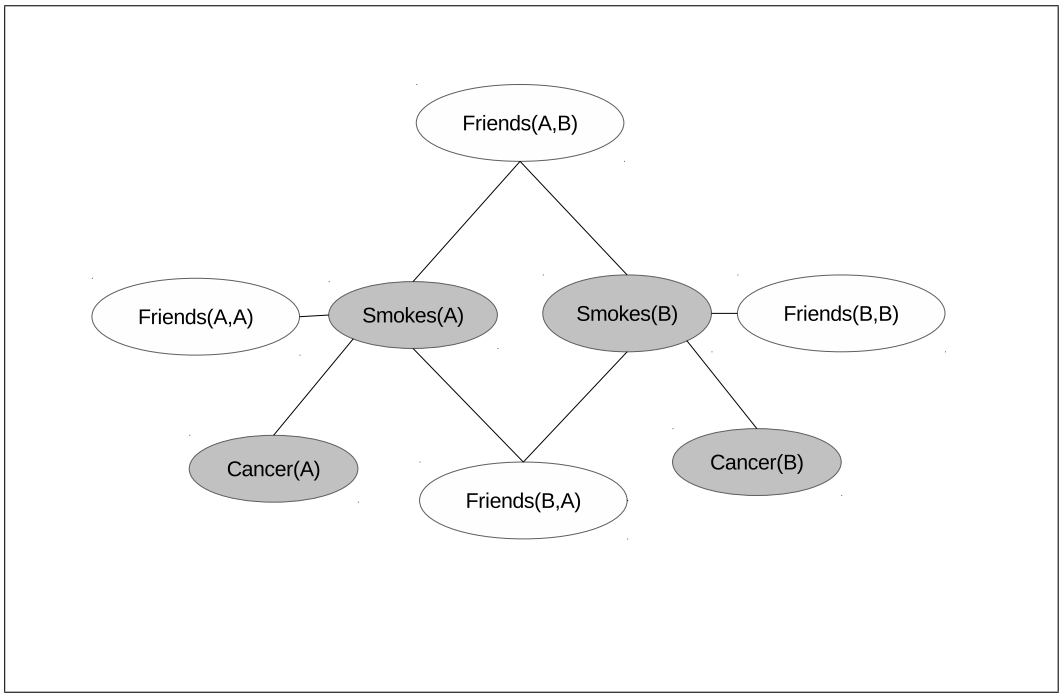

Figure 3.2: Predicting the non-evidence predicates

3.2 Weight Learning for Partial Observation

3.2.1 Direct Gradient

Algorithm 1 makes a closed world assumption. i.e. It assumes that the ground predicates which are not present in the database are false. This gives the truth values of all the ground predicates.

Now assume that the evidence is incomplete. i.e. The truth value of some predicates not present in the database are unknown. This assumption is called open world. This assumption create problems in calculating the gradient using (3.3). This is because it is now not possible to get the true count from the database.

We need to calculate $P(Y = y|X = x)$ where X is the evidence, and Y the query predicates.

$$P(Y = y|X = x) = \sum_{h \in H} P(Y = y, H = h|X = x)$$

where H is the data not observed.

$$\sum_{h \in H} P(Y = y, H = h|X = x) = \sum_{h \in H} \frac{1}{Z_x} e^{\sum_i w_i n_i(y,h|x)}$$

$$\text{where} Z_x = \sum_{y',h'} e^{\sum_i w_i n_i(y,h|x)}$$

Taking log on both sides

$$log \sum_{h \in H} P(Y = y, H = h|X = x) = log \sum_{h \in H} \frac{1}{Z_x} e^{\sum_i w_i n_i(y,h|x)}$$

Differentiating w.r.t w_i

$$\begin{aligned}
\frac{\partial}{\partial_{w_i}} log \sum_{h \in H} P(Y = y, H = h|X = x) &= \frac{1}{\sum_{h \in H} \frac{1}{Z_x} e^{\sum_i w_i n_i(y,h|x)}} \frac{\partial}{\partial_{w_i}} \sum_{h \in H} \frac{1}{Z_x} e^{\sum_i w_i n_i(y,h|x)} \\
&= \frac{1}{P(Y = y|X = x)} \frac{\partial}{\partial_{w_i}} \sum_{h \in H} \frac{1}{Z_x} e^{\sum_i w_i n_i(y,h|x)} \\
&= \frac{1}{P(Y = y|X = x)} \sum_{h \in H} \frac{\partial}{\partial_{w_i}} \frac{1}{Z_x} e^{\sum_i w_i n_i(y,h|x)} \\
&= \frac{1}{P(Y = y|X = x)} \sum_{h \in H} \left\{ \frac{1}{Z_x} \frac{\partial}{\partial_{w_i}} e^{\sum_i w_i n_i(y,h|x)} \right. \\
&\qquad \left. + e^{\sum_i w_i n_i(y,h|x)} \frac{\partial}{\partial_{w_i}} \frac{1}{Z_x} \right\} \\
&= \frac{1}{P(Y = y|X = x)} \sum_{h \in H} \left\{ \frac{1}{Z_x} e^{\sum_i w_i n_i(y,h|x)} . n_i(y, h|x) \right. \\
&\quad \left. + e^{\sum_i w_i n_i(y,h|x)} . (-1) \frac{1}{Z_x^2} . \frac{\partial}{\partial_{w_i}} Z_x \right\}
\end{aligned}$$

$$\begin{aligned}\text{But } \frac{\partial}{\partial_{w_i}} Z_x &= \frac{\partial}{\partial_{w_i}} \sum_{y',h'} e^{\sum_i w_i n_i(y,h|x)} \\ &= \sum_{y',h'} \frac{\partial}{\partial_{w_i}} e^{\sum_i w_i n_i(y,h|x)} \\ &= \sum_{y',h'} e^{\sum_i w_i n_i(y,h|x)}.n_i(y,h|x)\end{aligned}$$

Substituting in (2)

$$\begin{aligned}\frac{\partial}{\partial_{w_i}} log \sum_{h\in H} P(Y=y,H=h|X=x) &= \frac{1}{P(Y=y|X=x)} \sum_{h\in H} \Bigg[\frac{1}{Z_x} e^{\sum_i w_i n_i(y,h|x)}.n_i(y,h|x) + \\ &\quad e^{\sum_i w_i n_i(y,h|x)}.(-1)\frac{1}{Z_x^2}.\sum_{y',h'} e^{\sum_i w_i n_i(y,h|x)}.n_i(y,h|x)\Bigg] \\ &= \frac{1}{P(Y=y|X=x)} \sum_{h\in H} \Bigg[P(Y=y,H=h|X=x).n_i(y,h|x) - \\ &\quad e^{\sum_i w_i n_i(y,h|x)}.\frac{1}{Z_x}.\frac{1}{Z_x}.\sum_{y',h'} e^{\sum_i w_i n_i(y,h|x)}.n_i(y,h|x)\Bigg] \\ &= \frac{1}{P(Y=y|X=x)} \sum_{h\in H} \Bigg[P(Y=y,H=h|X=x).n_i(y,h|x) - \\ &\quad P(Y=y,H=h|X=x).\sum_{y',h'} \frac{1}{Z_x} e^{\sum_i w_i n_i(y,h|x)}.n_i(y,h|x)\Bigg] \\ &= \frac{1}{P(Y=y|X=x)} \sum_{h\in H} \Bigg[P(Y=y,H=h|X=x).n_i(y,h|x) - \\ P(Y=y,H=h|X=x).\sum_{y',h'} .P(Y=y,H=h|X=x).n_i(y,h|x)\Bigg]\end{aligned} \tag{3.4}$$

It can be observed that the set H being empty, Equation (3.4) becomes same as Equation (3.3). The algorithmic implementation is given in 2

Algorithm 2: Direct Gradient

Data: MLN, Ground clause database, non-evidence predicates, Open World predicates

Result: Weights for the MLN

Create the ground network from the MLN and the database;

Assign random weights to MLN;

while *Not converged* **do**

Run inference for open world predicates using the current weights;

Fill in the missing values using these inferred values;

Use this to find out the "actual" count;

Run inference for non-evidence predicates;

Find out the expected count from the inference;

Find the difference(Gradient);

Update the weights;

end

Consider the "smokers" example. Assume the truth values of Cancer(Anna) and Cancer(Bob) are unknown. This makes it impossible to calculate gradient. So we estimate the values of Cancer(Anna) and Cancer(Bob). Now, run inference to predict these values.[Figure 3.3]. The predicted values of Open world predicates are now used in finding out the true counts.

3.2.2 Expectation Maximization

Expectation-maximization Algorithm is also used in estimating the parameters of a Markov Logic Network under Open world assumptions.[Algorithm 3]

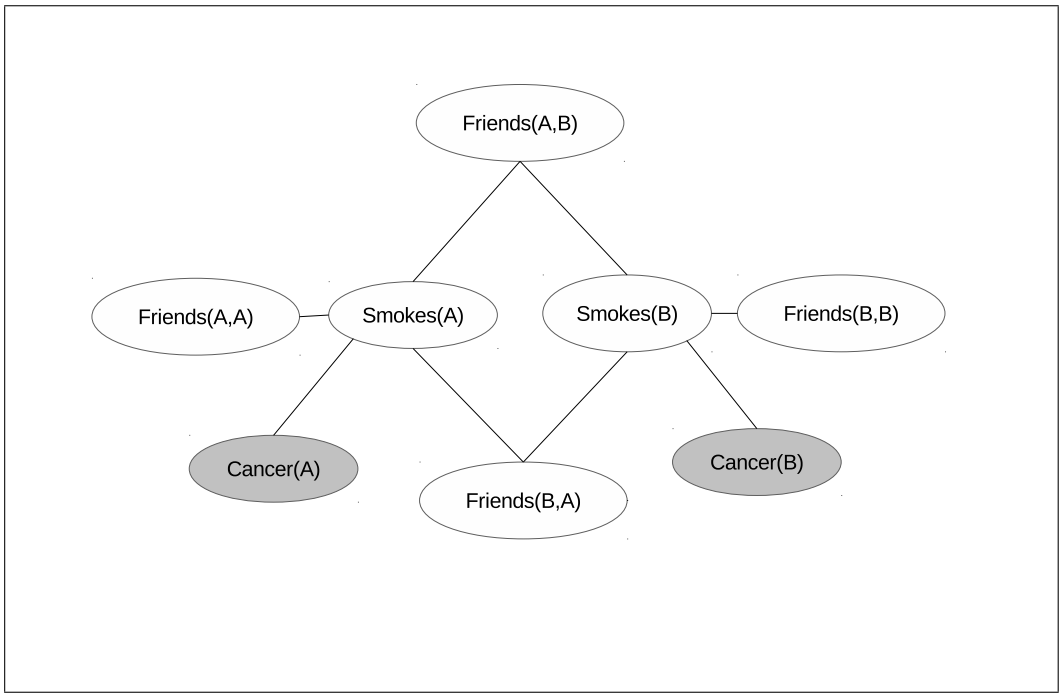

Figure 3.3: Ground network generated for Filling in values of Cancer(x)

Algorithm 3: Expectation Maximization

Data: MLN, Ground clause database, non-evidence predicates, Open World predicates

Result: Weights for the MLN

Create the ground network from the MLN and the database;

Assign random weights to MLN;

for $i = 1 \rightarrow n$ **do**

  Run inference for open world predicates using the current weights;

  Fill in the missing values using these inferred values;

  Find out the "actual" count from this;

  for $i = 1 \rightarrow m$ **do**

    Run inference for non-evidence predicates;

    Find out the expected count from the inference;

    Find the difference(Gradient);

    Update the weights;

  end

end

In Expectation-Maximization, there are 2 steps:

1. **E-Step -** Estimate the missing values using the current model
2. **M-Step -** Optimize the weights according to the estimated values

The two steps are repeated until convergence of weights.

3.2.3 Comparison of Expectation Maximization and Direct Gradient

Both Expectation Maximization and Direct Gradient are very similar. But they have some very small but crucial differences.

Expectation Maximization uses the filled in values to optimize the parameters of the model. It then uses these parameters to perform estimation again and then again perform optimization.

The direct gradient method estimates the missing values after every iteration of the gradient ascent.

Chapter 4

Experiments

The experiments conducted here were performed using a tool called alchemy, an implementation of Markov Logic Network. The Expectation Maximization and Direct Gradient Method were incorporated into the latest version of alchemy code for performing the experiments.

4.1 Dataset Descriptions

The experiments were performed on real world databases. The following databases were used for experimentation.

4.1.1 Cora

The Cora database is a collection of 1295 different citations to computer science research papers. A cleaned up version has been used in which some labels were corrected and missing values were filled in for. This cleaned-up version contains references to 132 different research papers. We used the author, venue, and title fields for experimenting. In this version TF-IDF scores has been assigned to pairs of constants. The goal is to perform entity resolution[15] on this database. Entity resolution is the problem of determining which records in a database refer to the same entities. It is a crucial and expensive step in the data mining process. We used alchemy to determine which pairs of citations, author fields, and venue fields refer to the same underlying paper, author and venue, respectively).

The Markov Logic network used for entity resolution in Cora has many rules that can perform entity resolution. Few examples are given below.

- AuthorScore80(a1,a2) ⇒ SameAuthor(a1,a2)

- AuthorScore60(a1,a2)⇒ SameAuthor(a1,a2)
- AuthorScore40(a1,a2) ⇒ SameAuthor(a1,a2)
- Author(b1,a1) ∧ Author(b2,a2) ∧ SameBib(b1,b2) ⇒ SameAuthor(a1,a2)

Alchemy will be able to learn weights for these rules given the database.

4.1.2 uw-cse

The uw-cse database is a representation of the CSE department of the University of Washington. It contains names of people and their designation(Teacher, student etc.), coursers and the names of teacher who teaches it, students and the teacher who advises them etc.
The goal is to find out which teacher advises which student. Some example rules in the Markov Logic Network for this purpose are given below.

- advisedBy(p, s) ⇒ student(s)
- advisedBy(p, s) ⇒ professor(p)
- student(p) ∧ professor(p)

4.2 Methodology

4.2.1 Complete Observation

The experiments were done on Cora database with SameBib,SameAuthor,SameTitle and SameVenue as non-evidence predicates. All the predicates in the database was assumed to be closed world. This was so that there will not be any missing values. The inference was done on a testing database with SameBib,SameAuthor,SameTitle and SameVenue as query predicates. The inferred values were then compared to the original database to measure the accuracy. The parameter that was being measured was area under the precision-recall curve.

4.2.2 Partial Observation

Multiple Experiments were conducted on partial observation. The following changes were made to the database and the experiments were run without filling in values, with Expectation Maximization and with Direct Gradient.

- Removing all of SameAuthor, SameTitle, SameVenue ground predicates from the database
- Removing all of SameAuthor ground predicates from the database
- Removing all of SameVenue ground predicates from the database
- Removing all of SameTitle ground predicates from the database

Weights were learned using SameBib,SameAuthor,SameTitle and SameVenue as non-evidence predicates and the hidden Predicates as open-world.
Another experiment was run in which 20,40,60,80 and 100 percentage of SameAuthor predicate was randomly removed from the database. SameAuthor was specified as open-world in this.

4.3 Results

The results for Partial Observability by removing all of SameAuthor, SameTitle, SameVenue ground predicates from the database is shown in Figure 4.1. Removing all these data during learning drastically reduces the accuracy during testing. Filling in for the missing data is seen to be providing better results. The results are even comparable to those when data is completely present. Expectation Maximization is found to give slightly better results than direct gradient in the experiments performed.

The results for Partial Observability by individually removing SameAuthor, SameTitle, SameVenue ground predicates from the database is shown in Figures 4.2,4.3 and 4.4 respectively. Removing SameTitle produces worse results than when removing SameAuthor. This means that having SameTitle

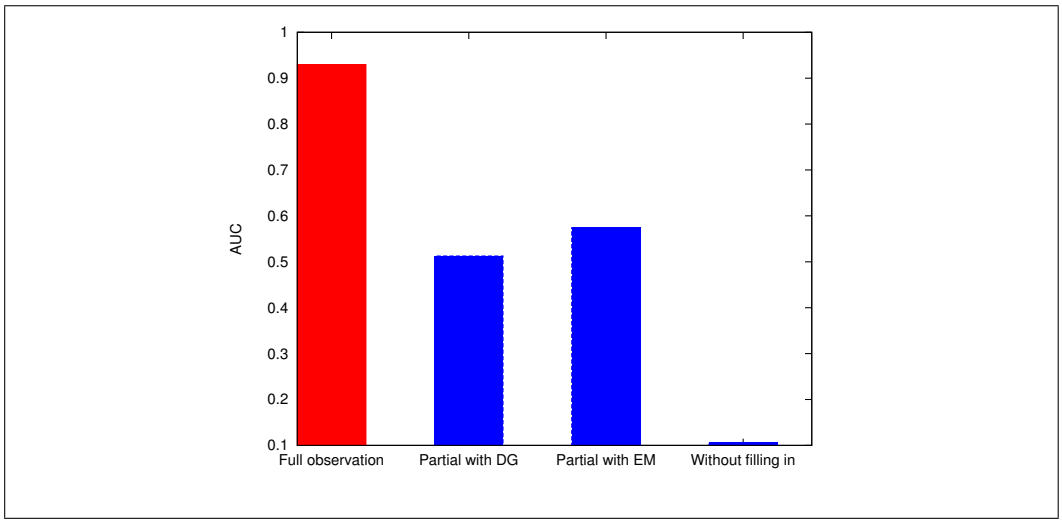

Figure 4.1: AUCs for Removing all Same predicates

clauses in the database helps entity resolution better than having SameAuthor clauses.

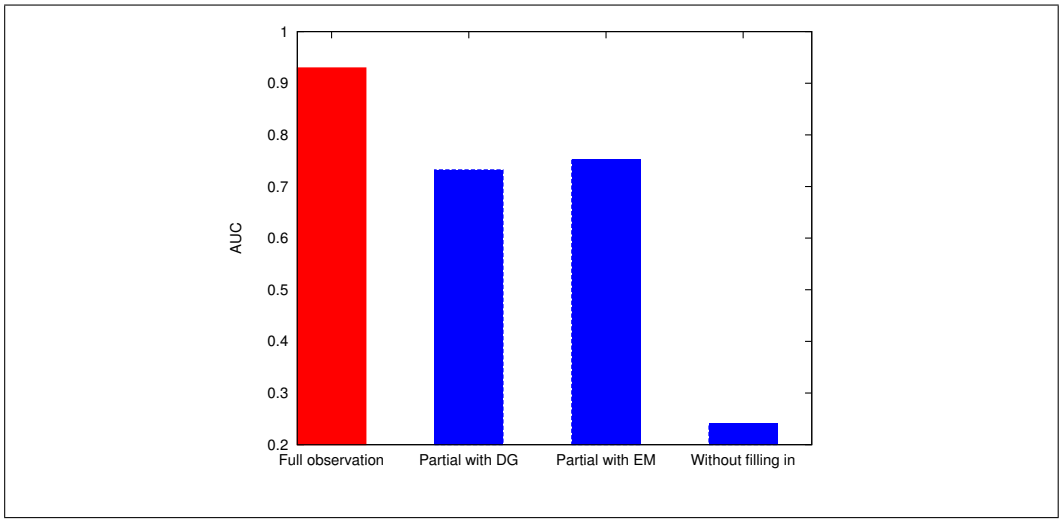

Figure 4.2: AUCs for Removing SameAuthor

The Precision-Recall Curve of Weight Learning under Complete observation is given in figure 4.5.

The Precision-Recall Curve of Weight Learning under Partial observation with Expectation Maximization is given in figure 4.6.

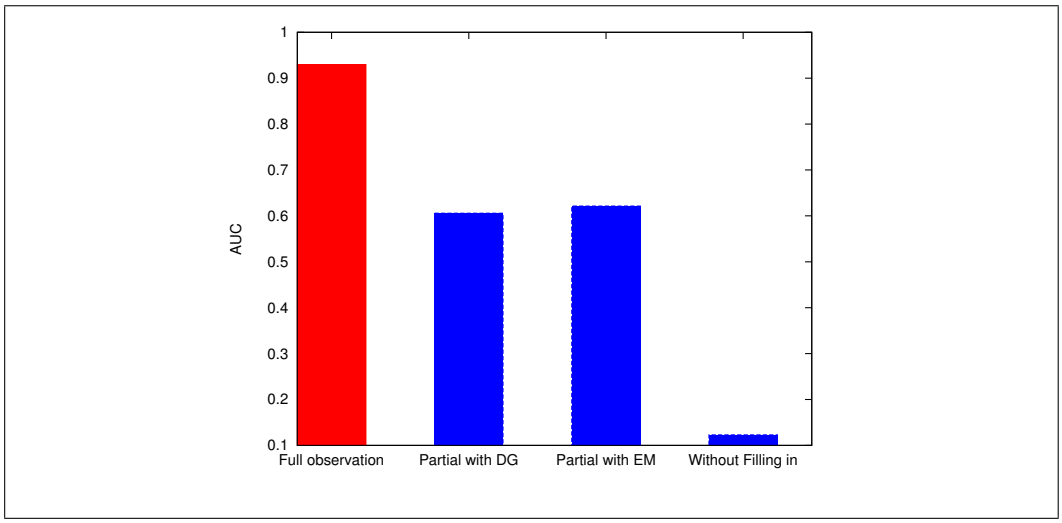

Figure 4.3: AUCs for Removing SameTitle

The Precision-Recall Curve of Weight Learning under Partial observation with Direct Gradient is given in figure 4.7.

The results of randomly removing 20,40,60,80 and 100 percentage of sameAuthor predicate is shown in Figure 4.8. The area under the precision-recall curve is found to be strictly decreasing with more and more data missing. With more missing data the predictions get worse.

Due to some difficulties with the dataset and the code, we were not able to produce good results for the uw-cse dataset.

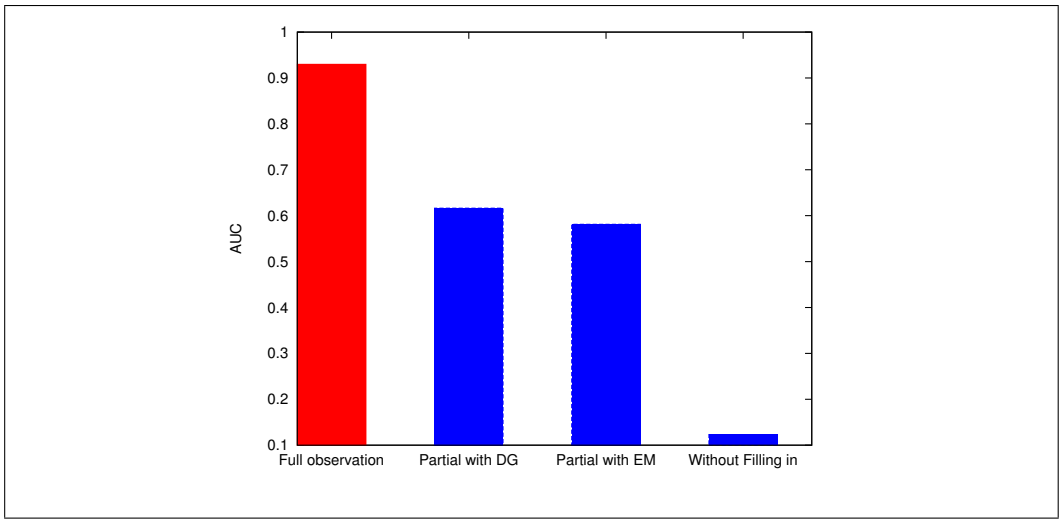

Figure 4.4: AUCs for Removing SameVenue

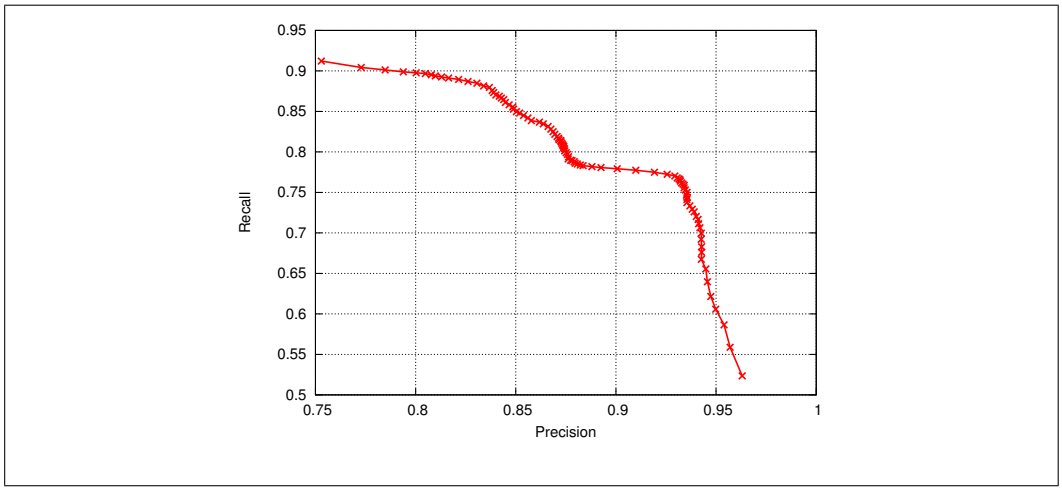

Figure 4.5: Precision-Recall Curve for Full Observation

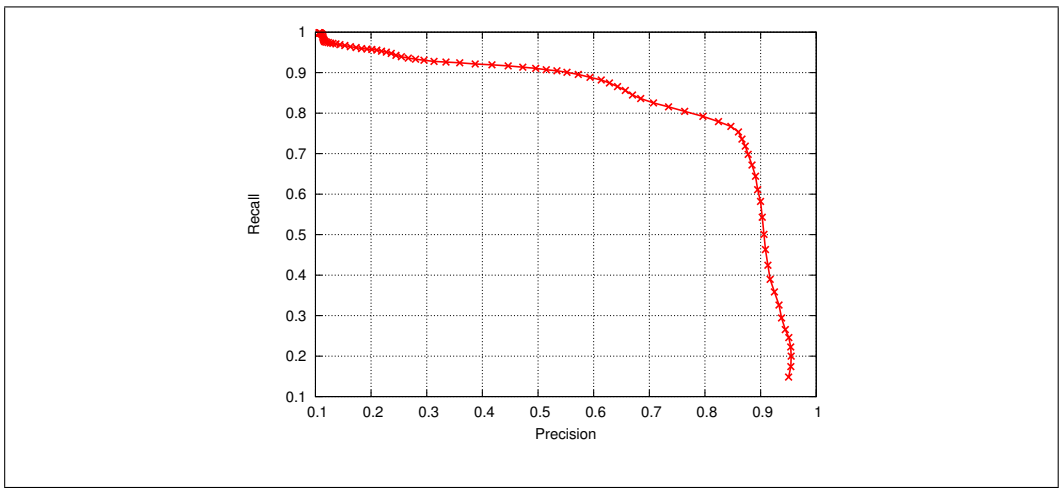

Figure 4.6: Precision-Recall Curve for Partial Observation - Expectation Maximization

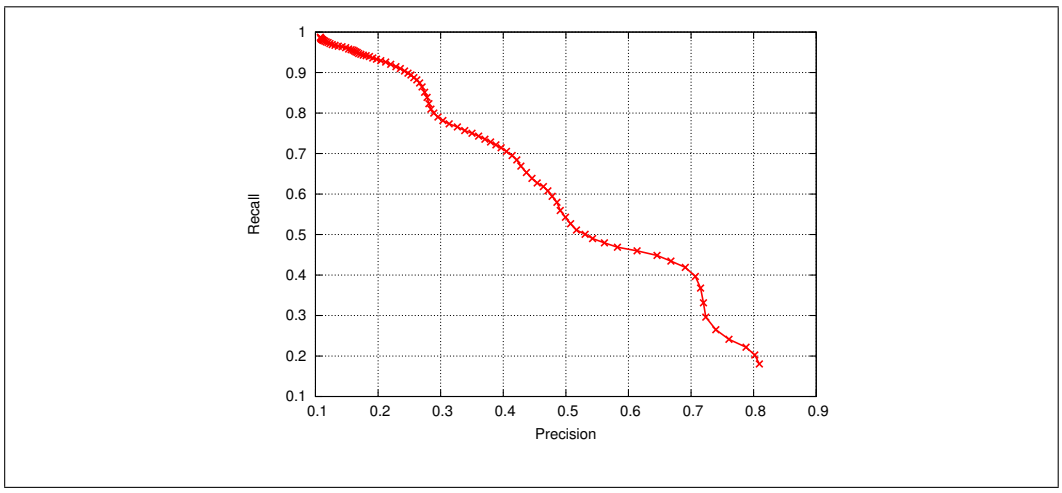

Figure 4.7: Precision-Recall Curve for Partial Observation - Direct Gradient

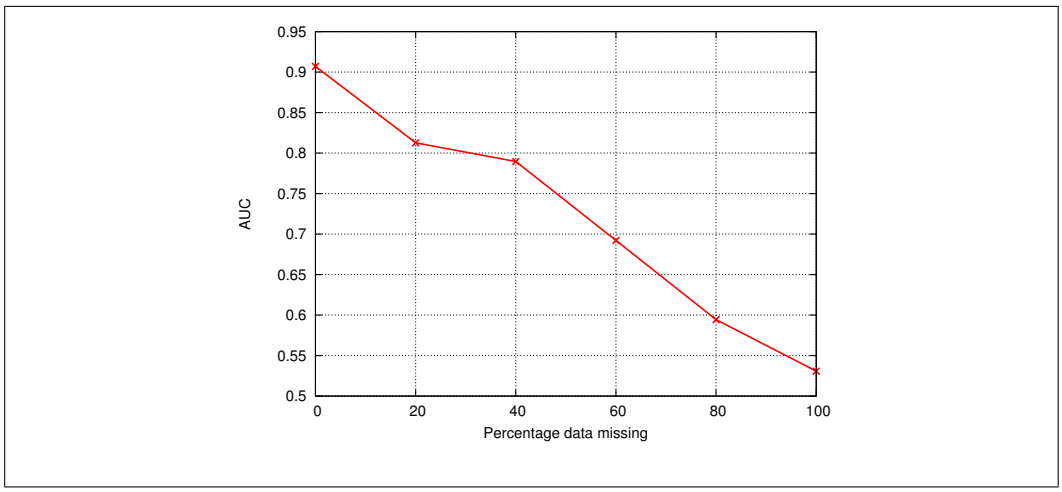

Figure 4.8: AUC for missing data

Chapter 5

Conclusion

In this work, we have extensively performed experiments for partial observation in Markov Logic Networks. It has been empirically verified that it is actually possible to perform approximate learning even in the absence of complete data.

The direct gradient method with hidden variables has been formally derived for the Markov Logic Framework. Proving the direct gradient method also turned out to be a formal proof for why the Expectation Maximization algorithm works correctly for Markov Logic Networks.

It has been seen that both the methods could produce better results by approximately filling in the missing data than without it.

Bibliography

[1] Improving photo search: A step across the semantic gap. http://googleresearch.blogspot.in/2013/06/improving-photo-search-step-across.html. Accessed: 2013-07-07.

[2] Making photo tagging easier. https://www.facebook.com/blog/blog.php?post=467145887130. Accessed: 2013-07-07.

[3] Enrico Blanzieri and Anton Bryl. A survey of learning-based techniques of email spam filtering. *Artif. Intell. Rev.*, 29(1):63–92, March 2008.

[4] Frank Dellaert College and Frank Dellaert. The expectation maximization algorithm. Technical report, 2002.

[5] Alan E. Gelfand. Gibbs sampling. *Journal of the American Statistical Association*, 452:13001304, 1995.

[6] Michael R. Genesereth and Nils J. Nilsson. *Logical foundations of artificial intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[7] Lise Getoor, Nir Friedman, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In Sašo Džeroski and Nada Lavrač, editors, *Relational Data Mining*. 2001.

[8] Benjamín Béjar Haro, Luca Zappella, and René Vidal. Surgical gesture classification from video data. In Nicholas Ayache, Hervé Delingette, Polina Golland, and Kensaku Mori, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2012*, volume 7510 of *Lecture Notes in Computer Science*, pages 34–41. Springer, 2012.

[9] Tushar Khot, Sriraam Natarajan, Kristian Kersting, and Jude Shavlik. Learning markov logic networks via functional gradient boosting. *Data Mining, IEEE International Conference on*, 0:320–329, 2011.

[10] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[11] Daniel Lowd and Pedro Domingos. Efficient weight learning for Markov logic networks. pages 200–211, Warsaw, Poland, 2007.

[12] Hoifung Poon, Pedro Domingos, and Marc Sumner. A general method for reducing the complexity of relational inference and its application to mcmc.

[13] Matthew Richardson and Pedro Domingos. Markov logic networks. 62:107–136, 2006.

[14] Parag Singla and Pedro Domingos. Discriminative training of Markov logic networks. pages 868–873, Pittsburgh, PA, 2005.

[15] Parag Singla and Pedro Domingos. Entity resolution with Markov logic. pages 572–582, Hong Kong, 2006.

[16] Sebastian Thrun. Toward robotic cars. *Commun. ACM*, 53(4):99–106, April 2010.

[17] C.-F. Westin, S. E. Maier, H. Mamata, A. Nabavi, F. A. Jolesz, and R. Kikinis. Processing and visualization of diffusion tensor MRI. *Medical Image Analysis*, 6(2):93–108, 2002.