Relational Decision Theory

Abstract

In recent years, many representations have been proposed that combine graphical models with aspects of first-order logic, along with learning and inference algorithms for them. However, the problem of extending decision theory to these representations remains largely unaddressed. In this paper, we propose a framework for relational decision theory based on Markov logic, which treats weighted first-order clauses as templates for features of Markov networks. By allowing clauses to have utility weights as well as probability weights, very rich utility functions can be represented. In particular, both classical planning and Markov decision processes are special cases of this framework. Maximizing expected utility in this representation is intractable, but we develop approximate algorithms for it, based on a combination of weighted satisfiability testing for maximization and MCMC for computing expectations. Experiments on a viral marketing domain and a logistics one illustrate the power of our approach and the efficiency of the algorithms.

1 INTRODUCTION

Traditionally, statistical models used in AI have focused on handling uncertainty, while logical models have been used to capture complex relational patterns in data. Over the past few years, these two approaches have been combined into the emerging field of statistical relational learning (Getoor and Taskar, 2007).

Markov logic (Richardson and Domingos, 2006) is a representation that subsumes finite first-order logic as well as certain probabilistic graphical models. In this paper, we extend Markov logic to represent decision-theoretic inference problems, and present an algorithm to maximize expected utility in an uncertain, relational setting.

1.1 RELATED WORK

(TO DO)

2 BACKGROUND

2.1 MARKOV NETWORKS

A *Markov network* is a model for the joint distribution of a set of variables ('nodes') $X = (X_1, X_2, \ldots, X_n)$. A set of *potential functions* (Pearl, 1989) $\{\phi_1 \ldots \phi_p\}$ is defined, each one mapping the states of some subset ('clique') of X to non-negative real numbers. The joint distribution represented by the network is given by:

$$P(X = x) = \frac{1}{Z} \prod_{k=1}^{p} \phi_k(x_{\{k\}})$$

Here, $x_{\{k\}}$ is the state of the variables in the kth clique, and Z is a normalization constant.

It is often convenient to represent a Markov network as a *log-linear model*. Each clique potential is replaced by an exponentiated weighted sum of features of the state, yielding:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x)\right)$$

This paper will focus on the binary features (i.e. $f_j(x) \in \{0, 1\}$).

2.2 FIRST-ORDER LOGIC

Formulae in first-order logic are constructed from logical connectives, quantifiers, and symbols for predicates, constants, variables and functions. A *grounding*

of a predicate is a replacement of all its arguments by constants (or, more generally, ground terms). Similarly, a grounding of a formula is a replacement of all its variables by constants. A *possible world* is an assignment of truth values to all possible groundings of predicates.

First-order logic is a convenient way to represent complex relational structure in a domain. However, it is often difficult to express certain pieces of domain knowledge as non-contradictory formulae that are always true. This makes first-order logic fragile; a truth assignment that violates even one formula has zero probability.

2.3 MARKOV LOGIC NETWORKS

Markov logic is a probabilistic extension of first-order logic. A Markov logic network (MLN) is a set of weighted first-order clauses. Together with a set of constants, it defines a Markov network with one node per ground atom and one feature per ground clause. The weight of a feature is the weight of the first-order clause that originated it. The probability distribution specified by the Markov network defined by the MLN is:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(x)\right)$$

where F is the number of formulas in the MLN, and $n_i(x)$ is the number of true groundings of F_i in x.

A Markov logic network can compactly specify a probability distribution over complex relational domains. Deterministic dependecies are represented by formulae with infinite weight. In this paper, we focus on MLNs whose formulae are function-free clauses and assume domain closure, ensuring that the Markov networks generated are finite.

2.3.1 MAP Inference

One basic inference task is to find the most probable state of the world given some evidence. This is known as maximum a posteriori (MAP) or most likely explanation (MPE) inference. In Markov logic, this reduces to finding the truth assignment that maximizes the sum of the weights of the satisfied clauses. We have used MaxWalkSat, a weighted variant of the WalkSat local-search satisfiability solver (Kautz et al., 1997).

2.3.2 Inferring Probabilities

Another key inference task is computing the probability that a formula holds, given an MLN and a set of constants, and possibly other formulae as evidence. The probability of a formula is the sum of the probabilities of the worlds where it holds. It is not tractable to compute this by brute force. Instead, the probability can be estimated using Markov Chain Monte Carlo (MCMC) inference (Gilks et al., 1996). One samples a sequence of states according to their probabilities, and counts the fraction of sampled states where the formula holds.

Standard MCMC algorithms such as Gibbs sampling and simulated tempering break down in the presence of deterministic or near-deterministic dependencies. Strong dependencies greatly slow inference by creating low probability regions that are very difficult to traverse (or impossible to traverse, if the dependencies are deterministic). The MC-SAT algorithm (Poon and Domingos, 2006) addresses this problem by combining MCMC with satisfiability testing. MC-SAT uses WalkSat to jump between regions of non-zero probability, while keeping the probability estimation process unbiased.

3 DECISION-MLNS

We define a decision Markov logic network (DMLN) as a set of triples (F_i, w_i, u_i), where F_i is a formula in first-order logic, and w_i and u_i are real numbers. As with a normal MLN, a DMLN together with a set of constants defines a Markov network (as described in section 2.3). The probability distribution specified by this network is:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(x)\right)$$

Additionally, the DMLN specifies a utility distribution over possible worlds x:

$$U(X = x) = \sum_{i=1}^{F} u_i n_i(x)$$

This representation allows us to specify very rich utility functions.

Given a DMLN and some evidence, it is straightforward to calculate the expected utility of the world. We can use existing algorithms such as MC-SAT to determine the probability that each grounding of each formula holds. The expected utility is given by:

$$E[U] = \sum_{i=1}^{F} \sum_{j=1}^{g_i} p_{i,j} u_i$$

where g_i is the number of groundings of formula i, and $p_{i,j}$ is the probability that grounding j of formula i is true.

A more interesting problem is to calculate which of several alternative policies yields the highest expected

utility. Let us assume that we are allowed to set some of the predicates in the formulae of our DMLN; we will refer to these as *action predicates*. We refer to predicates we cannot set as *state predicates*. We refer to groundings of action and state predicates as *action groundings* (AGs) and *state groundings* (SGs) respectively. The inference problem is to choose a truth assignment of action groundings that maximizes the expected utility.

This formulation allows us to represent both classical planning problems and Markov decision processes as DMLNs. A classical planning problem can be represented as a set of hard first-order logic formulae, i.e. an DMLN with a weight of infinity on every clause. The formulae include the start and goal states, and the axioms that describe the world.

To represent a Markov decision process, we can define one constant for each state and each action. The transition function can be represented as a set of soft (finite weight) formula of the form $\texttt{inState(s}_1\texttt{)} \wedge \texttt{performAction(a)} \Rightarrow \texttt{inState(s}_2\texttt{)}$. The weight of this formula encodes the probability of this transition, and the utility is the reward.

4 MAXIMIZING EXPECTED UTILITY

The problem of selecting an action grounding assignment that maximizes expected utility has parallels with the problem of selecting a truth assignment that maximizes the probability of the world. Algorithm 1 gives a modified version of the MaxWalkSat algorithm that maximizes expected utility instead of probability. We refer to this algorithm as RMEU (Relational Maximization of Expected Utility).

4.1 SELECTING ACTION GROUNDINGS

For every ground clause, we construct a set of action groundings likely to affect the truth value of the clause. Algorithms 2 and 3 describe how we do this.

This algorithm may not represent the complete set of action groundings relevant to a clause, since for each clause we terminate the search when we encounter a neighboring clause with no action groundings. However, we believe that this is a reasonable approximation to make for tractability.

Once we have a set of relevant groundings for every clause, we can use this to select an action grounding to flip during inference using algorithm 4.

Algorithm 1 RMEU()

for $i \leftarrow 1$ to *num_starts* **do**
  Run WalkSat to satisfy all hard clauses.
  for $j \leftarrow 1$ to *max_mws_tries* **do**
    for $k \leftarrow 1$ to *num_flips* **do**
      $ag \leftarrow$ pickRandom()
      Flip ag.
      Run WalkSat to satisfy hard clauses.
      if flip violates hard clauses **then**
        Flip ag back.
      else
        Infer probabilities of ground clauses.
        $cost \leftarrow \sum_{i=1}^{F} \sum_{j=1}^{g_i} p_{i,j} u_i$
      end if
    end for
  end for
end for
return solution with lowest cost.

Algorithm 2 setAGs()

Mark all clauses as 'not examined'.
Mark all state groundings as 'not examined'.
for all ground clauses, '*clause*' **do**
  if utility weight of *clause* is non-zero **then**
    setAGs(*clause*, $TRUE$)
  end if
end for

Algorithm 3 setAGs(*clause*, *top_level*)

if *clause* has been examined **then**
  return
end if
Mark AGs in *clause* as relevant.
if *clause* contains no AGs **and** $\neg$*top_level* **then**
  return
end if
Mark *clause* as 'examined'.
for all state groundings sg in *clause* **do**
  if sg has not been examined **then**
    for all clauses c containing sg **do**
      Call setAGs(c, $FALSE$).
      Mark AGs relevant to c as relevant to sg.
    end for
    Mark sg as examined.
    Mark AGs relevant to sg as relevant to *clause*.
  end if
end for

Algorithm 4 pickRandom()

for $i \leftarrow 0$ to *max_tries* **do**
  Select clause c at random.
  if Uniform$(0, 1)$ < probability that c is true **then**
    return random AG relevant to c.
  end if
end for

4.2 INFERRING PROBABILITIES

In each iteration of RMEU, we need to infer the probabilities of all ground clauses with non-zero utilities. A natural way to do this is to use MC-SAT. However, it may be expensive to run complete inference at every iteration of RMEU.

It seems unlikely that flipping a single action grounding would affect a large number of clauses. The vast majority of clauses are unlikely to be affected by most flips; it is wasteful to repeatedly infer their probabilities at every iteration. Instead, we use *Iterative MC-SAT* (I-MC-SAT; Algorithm 5) to update only the probabilities of nodes significantly affected by the flip.

We initialize the probability distribution by running MC-SAT over the whole graph before flipping any action groundings. After each flip, we run MC-SAT over a graph containing the neighbors of the flipped node to see if they are significantly affected. We then add the neighbors of the affected nodes to the graph, and run inference again. We repeat this until we find a graph containing all the nodes significantly affected by the flip. Note that we don't need to calculate accurate probabilities during each iteration of I-MC-SAT; we only need to know whether the probability has changed significantly since the last flip. Therefore, we only need a relatively small number of MC-SAT iterations for each iteration of I-MC-SAT. Once we have found the set of nodes affected by the flip, we can run MC-SAT for more iterations to calculate accurate probabilities for those nodes.

Algorithm 5 I-MC-SAT()

$N \leftarrow$ {Changed action predicate}
$Frontier \leftarrow N$
$stable \leftarrow$ FALSE
$iters \leftarrow 0$
while $\neg stable$ **and** $iters < MAX\_ITERS$ **do**
  $iters \leftarrow iters + 1$.
  $Neighbors \leftarrow$ {neighbors of elements of $Frontier$}
  $N' \leftarrow N \cup Neighbors$
  MC-SAT(N')
  $Frontier \leftarrow \emptyset$
  for all $x \in N'$ **do**
    if change in $P(x) >$ THRESHOLD **then**
      $Frontier \leftarrow Frontier \cup \{x\}$
    end if
  end for
  if $Frontier == \emptyset$ **then**
    $stable \leftarrow$ TRUE
  else
    $N \leftarrow N \cup Frontier$
  end if
end while
Run MC-SAT(N) for more iterations.

5 EXPERIMENTS

We implemented RMEU as an extension of the Alchemy system (Kok et al., 2008).

5.1 VIRAL MARKETING

Viral marketing makes use of existing social networks to maximize the effects of an advertising campaign. It is based on the premise that members of a social network may affect each other's purchasing decisions. If so, marketing to influential members of the network would have a greater impact than marketing to members with less influence over their peers' decisions.

The knowledge-sharing website Epinions [1] allows users to maintain a list of peers whose opinions they trust. We extracted the 'web of trust' from the 100 most-trusted members of Epinions, and performed experiments to infer an optimal viral marketing policy.

The DMLN we used in our experiments had the following formulae:

1. `Buys(u1)` $\wedge$ `Trusts(u2, u1)` $\Rightarrow$ `Buys(u2)`

2. `MarketTo(u)` $\Rightarrow$ `Buys(x)`

Formula 1 represents the influence a user u_1 has on u_2 as a result of being in u_2's web of trust. Formula 2 represents the effect of marketing to a user; the higher the weight, the more likely users are to be influenced by a marketing campaign. The strengths of these formulae are hard to determine experimentally; they are parameters that marketers would need to tailor to specific products and social networks.

We can set a prior belief that a user would buy the product by setting the weight of the formula `Buys(u)`. This formula would typically have a negative weight, since most users do not buy most products. We can also incorporate information about users who have already purchased the product by passing it in as evidence to the inference algorithm.

We define our utility function by setting the utilities of `Buys(u)` and `MarketTo(u)`. `Buys(u)` would have a positive utility, representing the returns directly gained from selling a product to a user. The negative weight of `MarketTo(u)` represents the cost of marketing to a single user.

[1] http://www.epinions.com

Methodology

Results

5.2 LOGISTICS

6 CONCLUSION

- Summary. - Future work.

References

Lise Getoor and Ben Taskar, editors. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

Walter R Gilks, Sylvia Richardson, and David J Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996.

Henry Kautz, Bart Selman, and Yueyen Jiang. A general stochastic approach to solving problems with hard and soft constraints. In Dinghzu Gu, Jun Du, and Panos Pardalos, editors, *The Satisfiability Problem: Theory and Applications*. American Mathematical Society, 1997.

Stanley Kok, Parag Singla, Matthew Richardson, Pedro Domingos, Marc Sumner, Hoifung Poon, and Jue Wang. The Alchemy system for statistical relational AI. http://alchemy.cs.washington.edu/, 2008.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1989.

Hoifung Poon and Pedro Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, 2006.

Marc Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 2006.