

Learning through failure (extended abstract)

Taisuke SATO¹, Yoshitaka Kameya²,

¹ Tokyo Institute of Technology / CREST, JST
2-12-1 Ōokayama Meguro-ku Tokyo Japan 152-8552

sato@mi.cs.titech.ac.jp

² Tokyo Institute of Technology
2-12-1 Ōokayama Meguro-ku Tokyo Japan 152-8552

kameya@mi.cs.titech.ac.jp

Keywords. PRISM, generative modeling, failure

1 PRISM and Negation

PRISM is a logic-based stochastic language proposed for modeling complex phenomena in the real world such as natural language understanding[1].^{1 2} A PRISM program is a logic program containing probabilistic atoms. It defines a probability measure over the Herbrand interpretations for a possibly infinite domain from which distributions over structured objects with arbitrary complexity are deduced. Parameters in the defined distribution are estimated efficiently from observed data using dynamic programming by a generic EM algorithm called the *gEM (graphical EM) algorithm*. While our approach is extremely general, the generality does not necessarily mean inefficiency compared to specialized approaches. For example parameter learning of PCFGs (probabilistic context free grammars) can be done with the same time complexity as the Inside-outside algorithm, a well-known special EM algorithm for PCFGs [2].

Recently we introduced negation to PRISM for more freedom of modeling. To cope with programs with negation theoretically and practically, we have developed a new semantics and a new parameter learning algorithm below (not described in this extended abstract though):

- a new three-valued probabilistic semantics [3] and
- a new EM algorithm called the *fgEM* algorithm for parameter learning [4].

As a result of allowing negation, or equivalently allowing failure from the view point of logic programming, generative modeling with failure using constraints is made possible. I.e., we write a PRISM program that describes a generation process of an outcome satisfying given constraints where the process is a chain of probabilistic choices and it fails when the constraints turn out to be unsatisfiable. In this talk, we show two examples of such *generative modeling with failure* and their parameter learning.

¹ <http://sato-www.cs.titech.ac.jp/prism/index.html>

² This work is supported in part by the 21st Century COE Program ‘Framework for Systematization and Application of Large-scale Knowledge Resources’.

2 Constrained HMMs

The first example is an extension of HMMs. *Constrained HMMs (Hidden Markov models)* are new HMMs with constraints which may fail if transition paths or emitted symbols do not satisfy given constraints. By the way constrained HMMs can be extended to constrained PCFGs or more general models as the next example suggests.

For illustration, we take a simple case of a two state HMM representing one's probabilistic choice of restaurant and lunch, and impose a constraint on it as follows. Suppose there is a professor who takes lunch at one of two restaurants 'r0' and 'r1' everyday and probabilistically changes the restaurant he visits. As he is on a diet, he tries to satisfy a constraint that the total calories for lunch in a week are less than 4000 calories. He probabilistically orders pizza (900) or sandwich (400) at r0, and hamburger (500) or sandwich (500) at r1 (numbers are calories). He records what he has eaten in a week like [p, s, s, p, h, s, h] and preserves the record if and only if he has succeeded in satisfying the constraint. We have a list of the preserved records and we wish to estimate his behavioral probabilities from it.

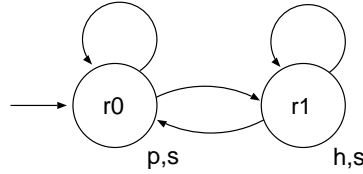


Fig. 1. HMM model for a probabilistic choice of restaurant and lunch

The behavior of the dieting professor is modeled by a constrained HMM described above and coded as a PRISM program in Figure 2 where failing to satisfy the calorie constraint on the seventh day causes failure thereby producing no output. `msw` atoms represent random choices. `msw(lunch(R),D)` for example represents a probabilistic choice of dish 'D' for lunch at a restaurant 'R'.

Negation in this program is compiled away by a full automatic logic program synthesizer called FOC (first-order complier) [5,6]. The correctness of compilation is checked by calculating the sum of probabilities of `success` and `failure` which must be identical to unity as shown below. We used here the 'original value' given in Table 1 as checking parameters.

```
?- prob(success,Ps),prob(failure,Pf),
    X is Ps+Pf.
X = 1.0
Pf = 0.348592596784
Ps = 0.651407403216
```

```

failure:- not(success).
success:- hmmf(L,r0,0,7).

hmmf(L,R,C,N):- N>0,
  msw(tr(R),R2), msw(lunch(R),D),
  ( R == r0, % pizza:900, sandwich:400
    ( D = p, C2 is C+900
      ; D = s, C2 is C+400 )
    ; R == r1, % hanburger:400, sandwich:500
      ( D = h, C2 is C+400
        ; D = s, C2 is C+500 ) ),
  L=[D|L2], N2 is N-1,
  hmmf(L2,R2,C2,N2).
hmmf([],_,C,0):- C < 4000.

```

Fig. 2. Constrained HMM for the dieting professor

We then repeated learning experiments. In an experiment we generated 500 samples by the program with the same parameter values used in the checking, and then let the program estimate them using the fgEM algorithm. Table 1 shows averages of 50 experiments where numbers should be read that for example, the probability of `msw(tr(r0),r0)` is originally 0.7 and the average of estimated ones is 0.697. Estimated parameters look close enough to the original ones.

Table 1. EM learning

sw name	original value	average estimation
tr(r0)	r0 (0.7) r1 (0.3)	r0 (0.697) r1 (0.303)
tr(r1)	r1 (0.7) r0 (0.3)	r1 (0.701) r0 (0.299)
lunch(r0)	p (0.4) s (0.6)	p (0.399) s (0.601)
lunch(r1)	h (0.5) s (0.5)	h (0.499) s (0.501)

We would like to emphasize that the computational strength of our approach is that the probability of `failure` is computed in a dynamic programming manner with no additional cost just like ordinary HMMs thanks to the tabling mechanism of PRISM [7].

3 Stochastic HPSGs

The second example is a stochastic extension of HPSGs. *HPSGs (head-driven phrase structure grammars)* [8] are a class of unification grammars which construct parses of sentences based on unification between feature structures con-

taining attributes and their values organized by a type system. They differ from CFGs in that most of linguistic information is contained in lexical items and there are only a few rules (principles) that specify constraints among feature structures associated with linguistic entities. We here introduce stochastic HPSGs described by PRISM programs which are generative and give mathematically well-defined distributions.

First note that we can imagine stochastic HPSGs as a special type of PCFGs. We can define CFG rules which are derived from the HPSG principles and feature structures and also can define a process of generating HPSG parses as a sequence of probabilistic rewriting of feature structures by the CFG rules just mentioned. This simple approach has fatal drawbacks however. For example feature sharing by feature structures cannot be adequately expressed by PCFGs [9].

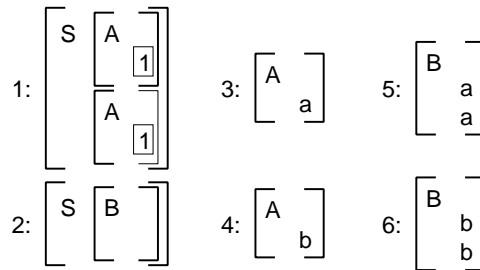


Fig. 3. An attribute-value grammar \mathcal{G}

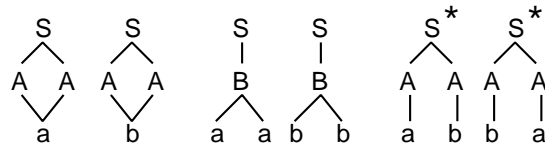


Fig. 4. Possible trees (dags)

Trees (dags) displayed in Figure 4 with no ‘*’ mark are ones licensed by an attribute-value grammar \mathcal{G} shown in Figure 3. In particular the leftmost dags exemplify sharing of a feature ‘a’ by two occurrences of ‘A’. A corresponding (P)CFG $\{S \rightarrow AA \mid B, A \rightarrow a \mid b, B \rightarrow aa \mid bb\}$ which is obtained by abstracting away feature structure from \mathcal{G} can produce these dags and also can produce extra trees marked ‘*’ on the right side which are not legitimate parses by \mathcal{G} but nonetheless take away some PCFG probabilities. So the total probability of observable parses assigned by the PCFG is inevitably less than one.

Kiefer et al. met this problem when they introduced PCFG approximations for stochastic HPSGs [10]. To construct an stochastic HPSG, they first construct an approximating CFG that generates more parses than the given HPSG and then turn the approximating CFG to an approximating PCFG by associating probabilities with CFG rules. Thanks to assigned probabilities, every HPSG parse has a probability computed by the approximating PCFG. But the problem remains that the approximating PCFG generates more parses than the HPSG, hence the probabilities placed on the original HPSG parses will sum to less than one; their probabilistic model is improper and hence consistency is lost.

We solve this problem to obtain a proper distribution for HPSG parses by normalization with $P(\text{success}|\boldsymbol{\theta}) = \sum_{\tau \in \mathcal{L}} P(\tau|\boldsymbol{\theta})$ where τ is a sentence in the language \mathcal{L} defined by the HPSG, and $\boldsymbol{\theta}$ denotes the set of parameters associated with the approximating PCFG. However this summation is infinite and not computable. We therefore further limit sentence length and rule out any sentence longer than the given limit to make the sum finite. The resulting distribution $P(\tau|\text{size} < l, \text{success}, \boldsymbol{\theta})$ is computable and becomes a proper distribution over the HPSG parses where **size** is a random variable representing sentence length and l the maximum sentence length.

The remaining problem is to compute $P(\text{success}|\text{size} < l, \boldsymbol{\theta})$ and learn $\boldsymbol{\theta}$ from data. Because the length constraint $\text{size} < l$ causes failure of the PRISM program describing a given HPSG, we need a *failure program* that computes $P(\text{failure}|\text{size} < l, \boldsymbol{\theta}) = 1 - P(\text{success}|\text{size} < l, \boldsymbol{\theta})$ to learn parameters using the fgEM algorithm. We derived a failure program manually for efficiency reason, unlike the previous section where FOC automatically derives the required failure program.

Using the manually derived failure program, we conducted EM learning of our model for an HPSG taken from the LKB system³ varying the sentence length limit from 8 to 12. We compared each result to those by the improper model proposed by Kiefer et al. We have found that our model achieves a higher likelihood than Kiefer et al.'s model in every case for the data randomly generated from the HPSG program.

References

1. Sato, T., Kameya, Y.: PRISM: a language for symbolic-statistical modeling. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97). (1997) 1330–1335
2. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research* **15** (2001) 391–454
3. Sato, T., Kameya, Y., Zhou, N.F.: Generative modeling with failure in prism. (In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05))
4. Sato, T., Kameya, Y.: A dynamic programming approach to parameter learning of generative models with failure. In: Proceedings of ICML 2004 workshop on Learning Statistical Models from Relational Data (SRL2004). (2004)

³ <http://www.delph-in.net/lkb/>

5. Sato, T.: First order compiler: A deterministic logic program synthesis algorithm. *Journal of Symbolic Computation* **8** (1989) 605–627
6. Sato, T., Kameya, Y.: Negation elimination for finite pcfgs. In: *Proceedings of the International Symposium on Logic-based Program Synthesis and Transformation 2004 (LOPSTR04)*. (2004) 119–134
7. Zhou, N.F., Shen, Y., Sato, T.: Semi-naive Evaluation in Linear Tabling . In: *Proceedings of the Sixth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP2004)*. (2004) 90–97
8. Sag, I., Wasow, T.: *Syntactic Theory: A Formal Introduction*. Stanford: CSLI Publications (1999)
9. Abney, S.: Stochastic attribute-value grammars. *Computational Linguistics* **23** (1997) 597–618
10. Kiefer, B., K.H.U., Prescher, D.: A novel disambiguation method for unification-based grammars using probabilistic context-free approximations. In: *Proceedings of the 19th International Conference on Computational Linguistics (COLING'02)*. (2002)