1 Introduction

Probabilistic programs provide an appealing language for describing mental theories, because they are Turing complete: any computable process may be described as a program. Program induction is the problem of inferring theories, in the form of (probabilistic) programs, that describe some set of observations. Minimum Description Length, or MDL, is one common approach to program induction [11]. The MDL approach selects the hypothesis (program) such that the sum of the length of the program, along with the length of the data (observations), when encoded with the help of the program, is minimized. Exactly how the data is encoded depends upon the hypothesis; when MDL is used for program induction, the encoding of the data is typically some sort of certificate proving that the program outputs the observations.

Instead of MDL, one may also take a Bayesian approach to program induction. This approach involves placing a prior upon programs and calculating, for a given program, the likelihood of the observations. Typically the prior penalizes longer programs [7 pg 385]. In many situations, such as polynomial curve fitting, the MDL and Bayesian approaches coincide: the MAP hypothesis and the hypothesis with minimal description length are the same [7 pg 392].

In this project, I explore the problem of evaluating candidate programs to explain some number of observations, using both the MDL and MAP criteria. I focus on grammar induction: inferring the process (probabilistic program) that generates phrases found within a corpus. Both MDL and MAP are used
and compared. The hypotheses are described using a variant upon the logic programming language Prolog.

2 Inductive Logic Programming

2.1 Non-deterministic Prolog

Logic programming languages, such as Prolog, are commonly used in inductive logic programming. They are usually preferred because they provide a single, uniform representation for hypotheses, evidence, and background information [3, 8].

I wanted to use Prolog to describe probabilistic context free grammars ("PCFGs"). As described in [4, pg 838], probabilistic context free grammars are CFGs where each production is assigned a probability. For any non-terminal symbol, the probabilities sum to one. The probability assigned to any derivation is the product of the probabilities of each production used in that derivation. The probability of a string being in the language is obtained by marginalizing over derivations.

Wanting to experiment with inductive logic programming and PCFGs, I wrote a Prolog interpreter, but allowed each Horn clause to be assigned a probability, subject to the condition that all the probabilities for a given clause sum to one. When trying to prove a term, my interpreter non-deterministically picks a production with the probability assigned to that production. I call this “non-deterministic Prolog” [1]. Non-deterministic Prolog allows one to readily express, for example, PCFGs. The interpreter can be downloaded at [http://web.mit.edu/~ellisk/www/npl.lisp](http://web.mit.edu/~ellisk/www/npl.lisp)

Other probabilistic extensions to Prolog exist, such as ProbLog, described in [10]. However, I found ProbLog did not suit my applications. ProbLog programs define a distribution over Prolog programs. In ProbLog, each production is given a probability that specifies the probability that that production will be included in a program sampled from the distribution upon programs. The probability of any clause being in a randomly sampled ProbLog program is independent of any other clause being in that program. For example, the following is valid ProbLog program, taken from [10]:

1.0: likes(X,Y) :- friendof(X,Y).

This name is a little ironic, because isn’t Prolog already considered a non-deterministic language?
0.8: \texttt{likes(X,Y):- friendof(X,Z), likes(Z,Y).}

However, it would not be a valid non-deterministic Prolog program, because the probabilities for the \texttt{likes} production do not sum to one.

The non-deterministic Prolog interpreter computes, for a given query, the probability of that query being derived. It does so by marginalizing over all derivations of the query.

2.2 A Probabilistic Context Free Grammar

I used my non-deterministic Prolog interpreter to define the following PCFG. Each term of the form \( :- \ P \ (H \ \text{args}) \ldots \) is a definition of a Horn clause (production) called \( H \) that takes arguments \texttt{args}. The ellipses correspond to the body of the clause; \( P \) is the probability of the Prolog interpreter selecting this particular production when trying to prove \( H \).

\\begin{verbatim}
;; Nouns - Tufa and Blicket
(:- 1/2 (N (tufa . ?tail) ?tail))
(:- 1/2 (N (blicket . ?tail) ?tail))

;; Noun phrases
(:- 1/2 (NP ?head ?tail)
         (det ?head ?det-tail)
         (N-bar ?det-tail ?tail))
(:- 1/2 (NP ?head ?tail)
         (N-bar ?head ?tail))

;; Determiners
(:- 1/2 (det (the . ?tail) ?tail))
(:- 1/2 (det (a . ?tail) ?tail))

;; N-Bar
(:- 1/3 (N-bar ?head ?tail)
         (N ?head ?tail))
(:- 1/3 (N-bar ?head ?tail)
         (N ?head ?N-tail)
         (PP ?N-tail ?tail))
(:- 1/3 (N-bar ?head ?tail)
         (Adj ?head ?adj-tail)
         (N-bar ?adj-tail ?tail))

;; Adjectives (just big and small)
(:- 1/2 (Adj (big . ?tail) ?tail))
\end{verbatim}
Using the above grammar, I can have the interpreter calculate the probability of deriving the sentence “The tufa likes a small blicket”:

\[
\text{CL-USER}> \quad (\text{-} (\text{ip} \text{ (the tufa likes a small blicket)} \text{ nil}))
\]

Probability 1/13824 (7.233796e-5)

### 2.3 Connections to MDL

In [11], Srinivasan et al. investigate the use of the MDL principle applied to logic programs. They consider the following general problem: given a logic program, \( B \), and terms known to be true, \( E_+ \), and terms known to be false, \( E_- \), infer the logic program \( H \) (hypothesis) such that the encoded length of \( H \), along with the encoded length of a proof that \( E_+ \) may be derived from \( H \), is minimized, subject to the condition that \( H \) cannot derive the terms \( E_- \).

This is equivalent to maximizing the negative length of \( H \) minus the proof of \( E_+ \) given \( H \). Such a rephrasing is very suggestive! We can interpret the former term as a log prior and the latter as a log likelihood. Exponentiating, the MDL principle says to select \( H \) such that

\[
\max_H \left\{ \exp(-|H|) \exp(-|\text{Proof}(E_+|H)|) \right\}
\]

"Prior" "Likelihood?"
I have put “prior” in quotes because it strictly speaking isn’t a prior (it’s probably not normalized, depending upon the encoding scheme). I have put “likelihood?” both in quotes and with a question mark because I will show that, although it has a role in MDL analogous to that of the likelihood, under certain conditions it actually differs in an interesting way from the actual likelihood.

In [11], the quantity $\text{Proof}(E_+|H)$ is calculated by encoding the choices that the Prolog interpreter must make in order to prove each term $e_+ \in E_+$. Each choice entails picking from among some number of Horn clauses. So, if the number of different choices for the first Horn clause used in the derivation is $n_1$, and the number of different choices for the second is $n_2$, etc, then the encoding of $\text{Proof}(e_+ \in E_+|H)$ takes length $\ln(n_1 \times n_2 \times \cdots)$ nats. Negating and then exponentiating this quantity gives the “likelihood”, which is

$$\frac{1}{n_1} \times \frac{1}{n_2} \times \cdots$$

which is the probability of the Prolog interpreter proving $e_+ \in E_+$ using this particular derivation if it picks each production rule randomly. Note that it is not the probability of deriving $e_+ \in E_+$. To calculate that quantity would require marginalizing over derivations, as the non-deterministic Prolog interpreter does. This discrepancy does not arise when each term in $E_+$ has only one derivation. In [11], the case of multiple derivations is not addressed. I will assume that, if a theory $H$ may prove a positive example $e_+ \in E_+$ using more than one derivation, the derivation counted towards the encoded length is the shortest (and therefore most likely) one.

To further study this MDL-analogue of the likelihood, I adapted the encoding scheme described in [11] for non-deterministic Prolog. Hypotheses (logic programs) are encoded using a prefix code; the number of nats required to encode a symbol occurring with probability $p$ is close to the $-\ln p$ bound [11]. Natural numbers are encoded using the self-delimiting code described in [11]; the self-delimited encoded length of natural number $n$ is of order $\ln n$. Probabilities were encoded as pairs of self-delimited natural numbers representing a rational number between zero and one. Proof encoding is similar to Srinivasan et al, with one minor variation: the number of nats required to encode the choice of a production with probability $p$ is $-\ln p$. In [11], this choice is encoded with $\ln n$ nats, where $n$ is the number of productions for the clause. When all clauses have the same probability, this proof encoding reduces to exactly that of [11].
This implementation of the encoding scheme allows us to experiment with the difference between the actual likelihood and the likelihood-analogue of MDL. To make this difference concrete, consider the sentence “The tufa saw a small blicket with binoculars.” There are two derivations (parses) of this sentence: either a tufa was using binoculars to see a small blicket, or the small blicket that had binoculars was seen by the tufa. Non-deterministic Prolog computes the likelihood of this sentence ($P$(Sentence|Grammar)) as

\[
\text{CL-USER} > (?- (ip \text{ the tufa saw a small blicket with-binoculars) nil})) \\
\text{Probability 1/6912 (1.4467593e-4)}
\]

The interpreter can also compute the length of the minimal proof encoding, which is

\[
\text{CL-USER} > (m?- (ip \text{ the tufa saw a small blicket with-binoculars) nil})) \\
\text{MDL 9.534162}
\]

for which \( \exp(-9.534162) \approx 7.23 \times 10^{-5} \neq 1/6912 \). Of course, when there is only one derivation, the MDL and the likelihood agree:

\[
\text{CL-USER} > (?- (ip \text{ the tufa likes a blicket) nil})) \\
\text{Probability 1/2304 (4.3402778e-4)}
\]

\[
\text{CL-USER} > (m?- (ip \text{ the tufa likes a blicket) nil})) \\
\text{MDL 7.742402}
\]

for which \( \exp(-7.742402) \approx 4.34 \times 10^{-4} \).

Actually, this discrepancy between the hypothesis selected by MDL and the MAP hypothesis is a general phenomenon that arises whenever the Fundamental Inequality does not hold \[9\]. The Fundamental Inequality is the following bound:

\[
|K(D|H) + K(H) + \ln P(D|H) + \ln P(H)| \leq K(P(\cdot|H)) + K(P)
\]

The $K(H)$ term is the prefix complexity of the hypothesis: the length of the shortest self-delimited encoding of $H$. This quantity is not computable (see \[7\ pg 202\]), but we can approximate it by the length of the encoding of the logic program corresponding to $H$, which I’ll write as $|H|$. If the prior is taken to be $P(H) \propto \exp(-|H|)^2$, then the $K(H)$ and $\ln P(H)$ terms cancel, up to an additive $O(1)$ term. The $K(P)$ term is the prefix complexity of

\[This is justified in, for example, \[7\ sec 5.2\]
the prior, a quantity both not computable and independent of $H$ and $D$. The quantity $K(D|H)$ is the prefix complexity of the data when run on a universal Turing machine constructed from the hypothesized logic program; again, I’ll approximate it by the length of the encoding of the proof of $D$. So, in the tufa-binoculars-blicket example, the Fundamental Inequality reads

$$|\ln P(\text{Sentence}|\text{Grammar}) + |\text{Proof(Sentence}|\text{Grammar})| + O(1)| \leq f(H)$$

and the previous result is recovered: that the Fundamental Inequality does not hold whenever the likelihoods and $e$–proof length differ by a large amount.

It’s interesting to consider the limit of a large number of different proofs for a given piece of positive evidence, as done in [7, pg 385]. Consider 1000 flips of a coin; our hypothesis will be that the coin is fair. Given our hypothesis, the probability of no heads is much less than the probability of 500 heads, but the length of a derivation of no heads is exactly the same as the length of a derivation of 500 heads.

3 Application to Grammar Induction

In [5], Perfors, Tenenbaum, and Regier show that the hierarchical phrase structure of natural language syntax may be inferred from a relatively small corpus. They construct a Bayesian model that gives higher posterior odds to context-free grammars than to regular grammars, after only being trained upon a relatively small number of exemplar sentences. Their model does not have an explicit form of Universal Grammar, showing that an ideal Bayesian learner could infer some language-specific parameters without domain-specific inductive biases.

Inspired by the work of Perfors et al, I used non-deterministic Prolog to compare the compression achieved by both regular and context-free grammars on a small, hand-made corpus. The context-free language is given in section 2.2. The regular language generates a string of randomly picked words, and is defined in non-deterministic Prolog as

```prolog
(:- 4/5 (sentence (?word . ?other-words))
    (word ?word)
    (sentence ?other-words))
(:- 1/5 (sentence nil))
(:- 1/9 (word the))
```
MDL was used to pick between the regular and context-free grammars. The regular language requires 96.65446 nats to be encoded; the context free language requires 333.93967 nats. Each sentence of the corpus was iteratively added to a list of positive evidence, and the encoded length for both regular and context free languages was recomputed using this list. At first, the regular language gave better compression, but after 33 example sentences, the context-free grammar gave better compression. Shown in figure 1 are description lengths after $k$ examples, for $k$ from 1 to 40. This data shows that the MDL principle selected the context-free grammar after only a small number of exemplar sentences. A hierarchical phrase-structure gives better compression of the data.

<table>
<thead>
<tr>
<th>Sentences</th>
<th>Regular encoded length (nats)</th>
<th>Context-free encoded length (nats)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>96.65446</td>
<td>333.93967</td>
</tr>
<tr>
<td>1</td>
<td>110.36574</td>
<td>341.68207</td>
</tr>
<tr>
<td>10</td>
<td>253.1302</td>
<td>418.5307</td>
</tr>
<tr>
<td>20</td>
<td>409.60596</td>
<td>503.12173</td>
</tr>
<tr>
<td>30</td>
<td>566.0817</td>
<td>587.713</td>
</tr>
<tr>
<td>40</td>
<td>722.55756</td>
<td>672.3043</td>
</tr>
</tbody>
</table>

Figure 1: Comparison of compression for regular and context-free languages that I obtained similar results when I instead compared the posterior probabilities of the regular and context-free grammars. The prior for a given logic program $H$ was $P(H) \propto e^{-|H|}$, where $|H|$ is the encoded length of $H$. The likelihood of a given sentence was equal to the probability of the non-deterministic Prolog interpreter deriving that sentence. Using the same example sentences, I computed log posterior odds as the number of sentences increased from zero to forty; this data is shown in figure 2. The log posterior probabilities are correct up to an additive constant; this constant is independent of the hypothesis and so, for picking the MAP grammar, doesn’t matter. The actual quantity shown in each entry of figure 2 is $\ln P(H) + \ln P(D|H) + \text{const}$, where $H$ is the logic program, $D$ is a set of sentences, and the constant is from a normalizing factor in the prior.
Figure 2: Comparison of log posteriors for regular and context-free languages

It’s interesting to note the agreement between the log posteriors for the regular language and their encoded length. This agreement occurs because, for the regular language, the Fundamental Inequality holds, as there is only ever one derivation in the regular language.

4 Further Directions

I would like to experiment with inductive logic programming applied to probabilistic programming languages. Inductive logic programming done via reversing the inference rules of first-order logic is discussed in [3] and implemented in, for example, the Progol system [8]. In [1], inductive logic programming is approached as a form of stochastic search through a hypothesis space consisting of logic programs. The main difference between these two approaches is in the way candidate hypotheses are evaluated. In [1], the problem is analyzed as a form of Bayesian inference, and the posterior is calculated using monte carlo methods. In contrast, [3] and [8] use the MDL or some variant thereof as the criterion for selecting hypotheses. Given my experiments comparing MDL and MAP in section 2.3, I am more interested in using the posterior to select hypotheses; however, the idea of inference as a sort of “inverted deduction” is a very intriguing one, and suggests combining the hypotheses generation methods of [3], [8] with the hypotheses selection criteria of [1].

It would also be interesting to try using a more realistic model of sentence comprehension when computing likelihoods. The model used in this project

<table>
<thead>
<tr>
<th>Sentences</th>
<th>Log posterior, regular</th>
<th>Log posterior, context-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-96.65446</td>
<td>-333.93967</td>
</tr>
<tr>
<td>1</td>
<td>-110.36574</td>
<td>-341.68207</td>
</tr>
<tr>
<td>10</td>
<td>-253.1302</td>
<td>-416.45126</td>
</tr>
<tr>
<td>20</td>
<td>-409.60596</td>
<td>-498.96286</td>
</tr>
<tr>
<td>30</td>
<td>-566.0817</td>
<td>-581.4746</td>
</tr>
<tr>
<td>40</td>
<td>-722.55756</td>
<td>-663.9864</td>
</tr>
</tbody>
</table>
(see section 3) is very unrealistic, and is only used because it is trivial to implement in non-deterministic Prolog. A much better model of human sentence parsing is given in [1].

Lastly, I am interested in apply inductive logic programming to the “To Seek Whence Cometh a Sequence” problem described in [2]. In [2], Hofstadter et al describe a game similar to the number game: given a sequence of numbers, what is the next number? (and the one after that, etc). I find this problem intriguing because it is almost exactly the same problem addressed by Solomonoff induction [7, sec 5.2]. Following the description in [7, sec 5.2], one would approach this problem by searching the space of programs that produce matching sequences, and compute posterior odds for each program using a prior that penalizes longer programs. Predictions could be made via Bayesian model averaging.

References


---

5To Seek Whence Cometh a Bayesian?


