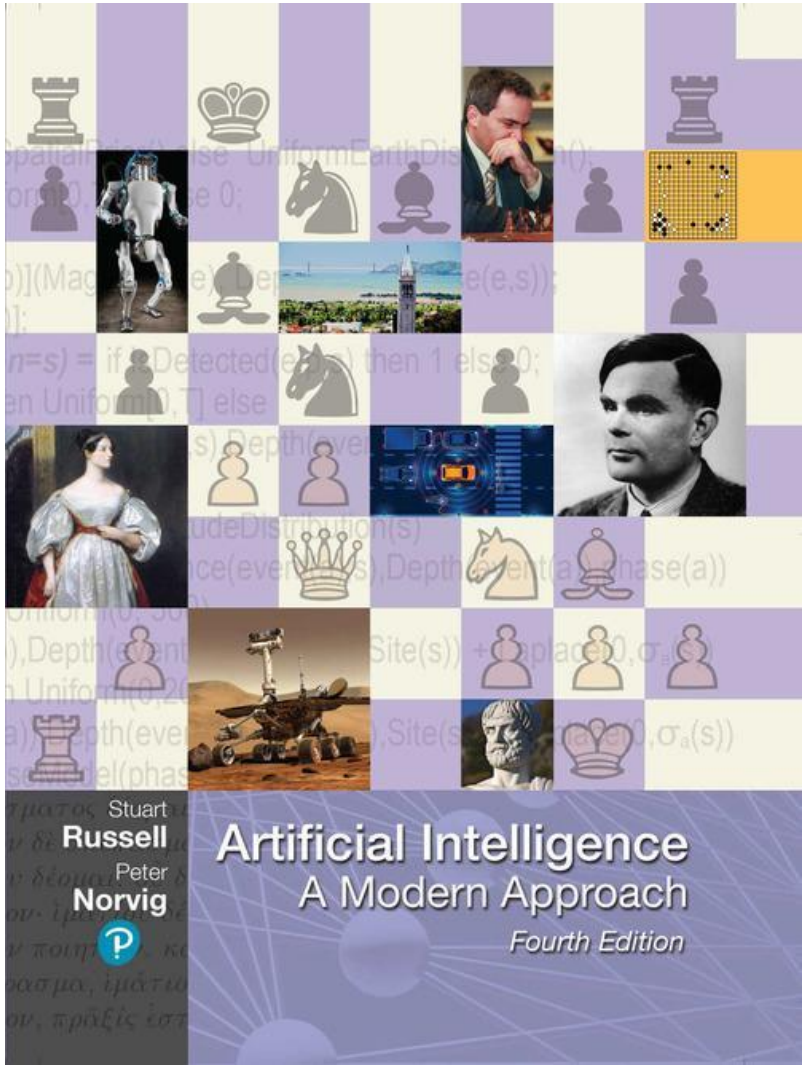
Artificial Intelligence: A Modern Approach

Fourth Edition



Chapter 15

Probabilistic Programming

Outline

- Relational Probability Models
- Open-Universe Probability Models
- Keeping Track of a Complex World
- Programs as Probability Models

Relational Probability Models

- **Relational Probability Models (RPM)** have no closed-world assumptions.

Syntax and semantics

- Constant, function, and predicate symbols
- **type signature** for each function: specification of the type of each argument and the function's value
  - Eliminates possible worlds

- Book-recommendation domain:
  - The types = Customer and Book
  - Type signatures for functions & predicates:

$$Honest : Customer \rightarrow \{true, false\}$$
$$Kindness : Customer \rightarrow \{1, 2, 3, 4, 5\}$$
$$Quality : Book \rightarrow \{1, 2, 3, 4, 5\}$$
$$Recommendation : Customer \times Book \rightarrow \{1, 2, 3, 4, 5\}$$

Relational Probability Models

- **Basic random variables** of the RPM are obtained by instantiating each function with each possible combination of objects

$Honest(C_1)$, $Quality(B_2)$, $Recommendation(C_1, B_2)$,

- One dependency statement for each function

- for any customer *c* and book *b*, the score depends on the honesty and kindness of the customer and the quality of the book:

- $Recommendation(c, b) \sim RecCPT(Honest(c), Kindness(c), Quality(b))$

- *RecCPT* is a separately defined conditional probability table with 2 x 5 x 5 = 50 rows, each with 5 entries.

Relational Probability Models

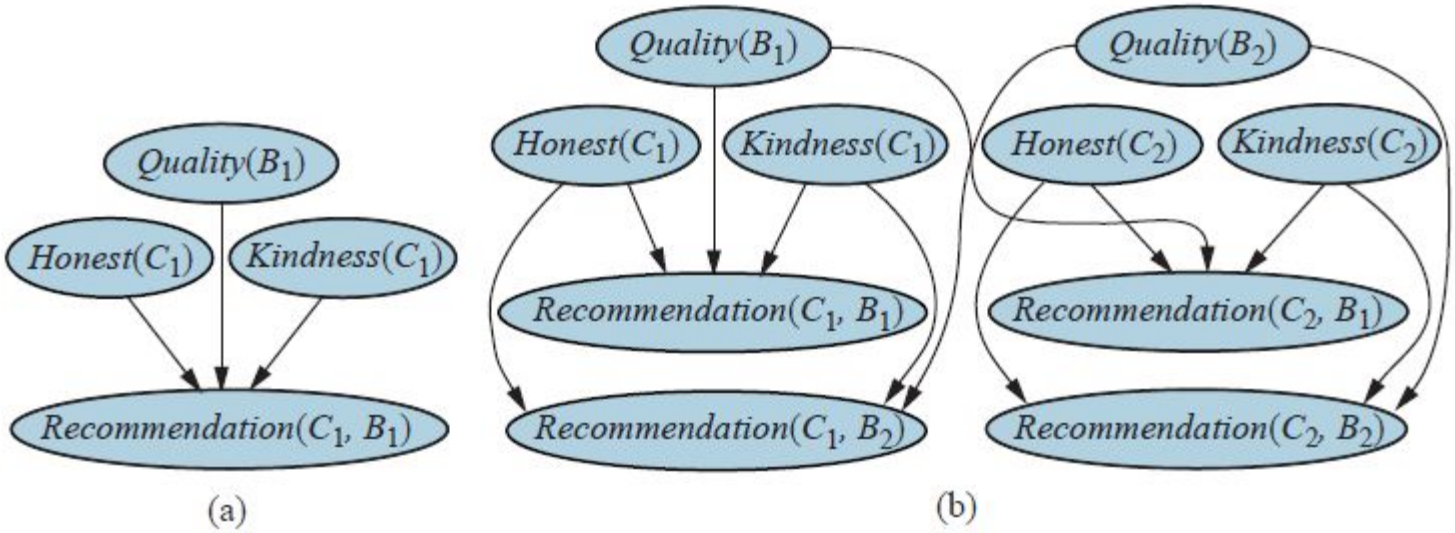
- **context-specific independence**

Recommendation(*c*, *b*) **if** *Honest*(*c*) **then**
HonestRecCPT(*Kindness*(*c*), *Quality*(*b*))
else (0.4, 0.1, 0.0, 0.1, 0.4) .

- **Further elaboration**

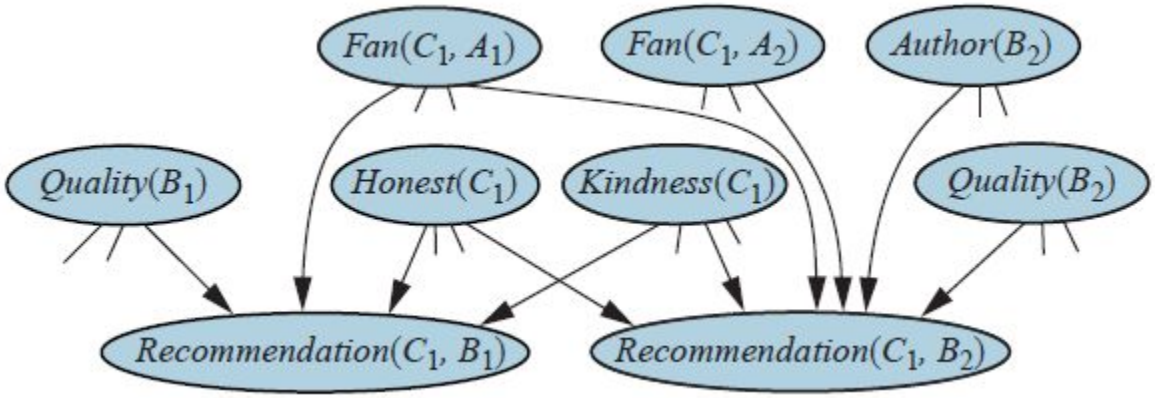
Recommendation(*c*, *b*) **if** *Honest*(*c*) **then**
if *Fan*(*c*, *Author*(*b*)) **then** *Exactly*(5)
else *HonestRecCPT*(*Kindness*(*c*), *Quality*(*b*))
else (0.4, 0.1, 0.0, 0.1, 0.4)

Relational Probability Models



(a) Bayes net for a single customer C_1 recommending a single book B_1. *Honest*(C_1) is Boolean, while the other variables have integer values from 1 to 5. (b) Bayes net with two customers and two books.

Relational Probability Models



Fragment of the equivalent Bayes net for the book recommendation RPM when *Author*(B_2) is unknown.

Inference in relational probability models

RPM Inference approach: construct the equivalent Bayesian network, given the known constant symbols belonging to each type **(grounding/unrolling)**

Example:

for $b = 1$ **to** B *do*
  add node $Quality_b$ with no parents, prior $\langle 0.05, 0.2, 0.4, 0.2, 0.15 \rangle$
for $c = 1$ **to** C *do*
  add node $Honest_c$ with no parents, prior $\langle 0.99, 0.01 \rangle$
  add node $Kindness_c$ with no parents, prior $\langle 0.1, 0.1, 0.2, 0.3, 0.3 \rangle$
  for $b = 1$ **to** B *do*
    add node $Recommendation_{c,b}$ with parents $Honest_c, Kindness_c, Quality_b$
      and conditional distribution $RecCPT(Honest_c, Kindness_c, Quality_b)$

Inference in relational probability models

Drawbacks of grounding:

- Large Bayes net
- Many parents of variables

Countermeasures:

- Elimination algorithm, chaining from query and evidence
- Repeated stubstrucutre
- MCMC inference algorithms: sampling complete possible worlds.
- Resolution theorem provers: instantiating the logical variables only as needed

Open-Universe Probability Models

Example: ISBN for books ID, may have several IDs for same book differentiated by hardcover paperback, large print, etc.

Sybils: multiple IDs

Sybil attack: confound a reputation system

Existence uncertainty eg: what are the real books and customers underlying the observed data?

Identity uncertainty eg: which logical terms really refer to the same object

Open-Universe Probability Models

Open universe probability model (OUPM): allows generative steps compared to RPM

- add objects to the possible world under construction, where the number and type of objects may depend on the objects that are already in that world and their properties and relations

- $\#\,Customer \sim UniformInt(1, 3)$
- $\#\,Book \sim UniformInt(2, 4)$.

- $\#\,LoginID(Owner = c) \sim \textbf{if}\ Honest(c)\ \textbf{then}\ Exactly(1)$
  $\textbf{else}\ UniformInt(2, 5)$.

Owner function is an **origin function**

Open-Universe Probability Models

- *UniformInt*: Uniform distribution used in example, for nonnegative integers, Poisson Distribution is commonly used:
- $P(X = k) = \lambda^k e^{-\lambda}/k!$.

- For large numbers, discrete log-normal distribution and order-of-magnitude distribution where OM(3,1) = 10^3 Standard deviation between 10^2 and 10^4.

- **Number variables** of an OUPM: number objects there are of each type with each possible origin in each possible world

- $\#LoginID_{(Owner,\ (Customer,,2))}(\omega) = 4$, in world ω, customer 2 owns 4 login IDs.

- **Basic random variables** determine the values of predicates and functions for all tuples of objects

- $Honest_{(Customer,,2)}(\omega) = true$ means that in world ω, customer 2 is honest.

Open-Universe Probability Models

Variable	Value	Probability
$\#Customer$	2	0.3333
$\#Book$	3	0.3333
$Honest_{\langle Customer,,1\rangle}$	*true*	0.99
$Honest_{\langle Customer,,2\rangle}$	*false*	0.01
$Kindness_{\langle Customer,,1\rangle}$	4	0.3
$Kindness_{\langle Customer,,2\rangle}$	1	0.1
$Quality_{\langle Book,,1\rangle}$	1	0.05
$Quality_{\langle Book,,2\rangle}$	3	0.4
$Quality_{\langle Book,,3\rangle}$	5	0.15
$\#LoginID_{\langle Owner,\langle Customer,,1\rangle\rangle}$	1	1.0
$\#LoginID_{\langle Owner,\langle Customer,,2\rangle\rangle}$	2	0.25
$Recommendation_{\langle LoginID,\langle Owner,\langle Customer,,1\rangle\rangle,1\rangle,\langle Book,,1\rangle}$	2	0.5
$Recommendation_{\langle LoginID,\langle Owner,\langle Customer,,1\rangle\rangle,1\rangle,\langle Book,,2\rangle}$	4	0.5
$Recommendation_{\langle LoginID,\langle Owner,\langle Customer,,1\rangle\rangle,1\rangle,\langle Book,,3\rangle}$	5	0.5
$Recommendation_{\langle LoginID,\langle Owner,\langle Customer,,2\rangle\rangle,1\rangle,\langle Book,,1\rangle}$	5	0.4
$Recommendation_{\langle LoginID,\langle Owner,\langle Customer,,2\rangle\rangle,1\rangle,\langle Book,,2\rangle}$	5	0.4
$Recommendation_{\langle LoginID,\langle Owner,\langle Customer,,2\rangle\rangle,1\rangle,\langle Book,,3\rangle}$	1	0.4
$Recommendation_{\langle LoginID,\langle Owner,\langle Customer,,2\rangle\rangle,2\rangle,\langle Book,,1\rangle}$	5	0.4
$Recommendation_{\langle LoginID,\langle Owner,\langle Customer,,2\rangle\rangle,2\rangle,\langle Book,,2\rangle}$	5	0.4
$Recommendation_{\langle LoginID,\langle Owner,\langle Customer,,2\rangle\rangle,2\rangle,\langle Book,,3\rangle}$	1	0.4

One particular world for the book recommendation OUPM. The number variables and basic random variables are shown in topological order, along with their chosen values and the probabilities for those values.

Inference in open-universe probability models

Citation matching

type Researcher, Paper,
Citation **random** String Name(Researcher)
random String Title(Paper)
random Paper CitedPaper(Citation)
random String Text(Citation)
random Boolean Professor(Researcher)
origin Researcher Author(Paper)

#Researcher ~ *OM*(3, 1)
Name(*r*) ~ *NamePrior*()
Professor(*r*) ~ *Boolean*(0.2)
#Paper(*Author* = *r*) ~ **if** *Professor*(*r*) **then** *OM*(1.5, 0.5) **else** *OM*(1, 0.5)
Title(*p*) ~ *PaperTitlePrior*()
CitedPaper(*c*) ~ *UniformChoice*({*Paper p*})
Text(*c*) ~ *HMMGrammar*(*Name*(*Author*(*CitedPaper*(*c*))), *Title*(*CitedPaper*(*c*)))

An OUPM for citation information extraction. For simplicity the model assumes
one author per paper and omits details of the grammar and error models.

Inference in open-universe probability models

Nuclear treaty monitoring

$$
\begin{aligned}
&\#SeismicEvents \sim Poisson(T * \lambda_e)\\
&Time(e) \sim UniformReal(0,T)\\
&EarthQuake(e) \sim Boolean(0.999)\\
&Location(e) \sim \textbf{if } Earthquake(e) \textbf{ then } SpatialPrior() \textbf{ else } UniformEarth()\\
&Depth(e) \sim \quad \textbf{if } Earthquake(e) \textbf{ then } UniformReal(0,700) \textbf{ else } Exactly(0)\\
&Magnitude(e) \sim Exponential(log(10))\\
&Detected(e,p,s) \sim Logistic(weights(s,p), Magnitude(e), Depth(e), Dist(e,s))\\
&\#Detections(\sout{site} = s) \sim Poisson(T * \lambda_f(s))\\
&\#Detections(event{=}e, phase{=}p, station{=}s) = \textbf{if } Detected(e,p,s) \textbf{ then } 1 \textbf{ else } 0\\
&OnsetTime(a,s) \textbf{ if } (event(a) = null) \textbf{ then } \sim UniformReal(0,T)\\
&\quad \textbf{else} = Time(event(a)) + GeoTT(Dist(event(a),s), Depth(event(a)), phase(a))\\
&\qquad\qquad + Laplace(\mu_t(s), \sigma_t(s))\\
&Amplitude(a,s) \textbf{ if } (event(a) = null) \textbf{ then } \sim NoiseAmpModel(s)\\
&\quad \textbf{else} = AmpModel(Magnitude(event(a)), Dist(event(a),s), Depth(event(a)), phase(a))\\
&Azimuth(a,s) \textbf{ if } (event(a) = null) \textbf{ then } \sim UniformReal(0,\ 360)\\
&\quad \textbf{else} = GeoAzimuth(Location(event(a)), Depth(event(a)), phase(a), Site(s))\\
&\qquad\qquad + Laplace(0, \sigma_a(s))\\
&Slowness(a,s) \textbf{ if } (event(a) = null) \textbf{ then } \sim UniformReal(0,20)\\
&\quad \textbf{else} = GeoSlowness(Location(event(a)), Depth(event(a)), phase(a), Site(s))\\
&\qquad\qquad + Laplace(0, \sigma_s(s))\\
&ObservedPhase(a,s) \sim CategoricalPhaseModel(phase(a))
\end{aligned}
$$

A simplified version of the NET-VISA model

Keeping Track of a Complex World

Data association problem: problem of associating observation data with the objects that generated them

Multitarget tracking

- A_1 and A_2 are **guaranteed objects**,
- true positions be $X(A_1,t)$ and $X(A_2,t)$, where t is a nonnegative integer that indexes the sensor update times.
- the first observation arrives at $t = 1$, and at time 0 the prior distribution for every aircraft's location is $InitX()$.
- assume that each aircraft moves independently according to a known transition model
- sensor model: again, we assume a linear–Gaussian model where an aircraft at position $\mathbf{x}$ produces a blip b whose observed blip position $Z(b)$ is a linear function of $\mathbf{x}$ with added Gaussian noise.

$$\begin{aligned}
&\textbf{guaranteed } Aircraft\, A_1, A_2 \\
&X(a,t) \sim \textbf{if } t = 0 \textbf{ then } InitX() \textbf{ else } \mathcal{N}(\mathbf{F}\, X(a,t-1), \Sigma_x) \\
&\#Blip(Source{=}a, Time{=}t) = 1 \\
&Z(b) \sim \mathcal{N}(\mathbf{H}\, X(Source(b), Time(b)), \Sigma_z)
\end{aligned}$$

Keeping Track of a Complex World

- n known objects generating n observations

- Real applications of data association are typically much more complicated.
- **False alarms** (clutter): not caused by real objects
- **Detection failures**: no observation reported for real object

- Many research studies simply try to work out the complex mathematical details of the probability calculations

- Practical point of view, complexity of inference of the OUPM

- Simplest approach is to choose a single “best” assignment at each time step, given the predicted positions of the objects at the current time
  - Using **Nearest-neighbor** filter

- However fails under more difficult conditions.
- Solution: MCMC algorithm, explores the space of assignment histories

Keeping Track of a Complex World

$$
\begin{aligned}
&\#Aircraft(EntryTime=t) \sim Poisson(\lambda_a)\\
&Exits(a,t) \sim \textbf{if } InFlight(a,t) \textbf{ then } Boolean(\alpha_e)\\
&InFlight(a,t) = (t=EntryTime(a)) \vee (InFlight(a,t-1) \wedge \neg Exits(a,t-1))\\
&X(a,t) \sim \textbf{if } t = EntryTime(a) \textbf{ then } InitX()\\
&\quad \textbf{else if } InFlight(a,t) \textbf{ then } \mathcal{N}(\mathbf{F}X(a,t-1),\Sigma_x)\\
&\#Blip(Source=a,\ Time=t) \sim \textbf{if } InFlight(a,t) \textbf{ then } Bernoulli(DetectionProb(X(a,t)))\\
&\#Blip(Time=t) \sim Poisson(\lambda_f)\\
&Z(b) \sim \textbf{if } Source(b)=null \textbf{ then } UniformZ(R) \textbf{ else } \mathcal{N}(\mathbf{H}X(Source(b),Time(b)),\Sigma_z)
\end{aligned}
$$

An OUPM for radar tracking of multiple targets with false alarms, detection failure, and entry and exit of aircraft. The rate at which new aircraft enter the scene is λ_a, while the probability per time step that an aircraft exits the scene is a_e. False alarm blips (i.e., ones not produced by an aircraft) appear uniformly in space at a rate of λ_f per time step. The probability that an aircraft is detected (i.e., produces a blip) depends on its current position.

Keeping Track of a Complex World

Example: Traffic monitoring

(a) (b)

Images from (a) upstream and (b) downstream surveillance cameras roughly two miles apart on Highway 99 in Sacramento, California. The boxed vehicle has been identified at both cameras.

Keeping Track of a Complex World

Example: Traffic monitoring

```
function GENERATE-IMAGE() returns an image with some letters
    letters ← GENERATE-LETTERS(10)
    return RENDER-NOISY-IMAGE(letters, 32, 128)

function GENERATE-LETTERS(λ) returns a vector of letters
    n ~ Poisson(λ)
    letters ← []
    for i = 1 to n do
        letters[i] ~ UniformChoice({a, b, c, ···})
    return letters

function RENDER-NOISY-IMAGE(letters, width, height) returns a noisy image of the letters
    clean_image ← RENDER(letters, width, height, text_top = 10, text_left = 10)
    noisy_image ← []
    noise_variance ~ UniformReal(0.1, 1)
    for row = 1 to width do
        for col = 1 to height do
            noisy_image[row, col] ~ N(clean_image[row, col], noise_variance)
    return noisy_image
```

Generative program for an open-universe probability model for optical character recognition. The generative program produces degraded images containing sequences of letters by generating each sequence, rendering it into a 2D image, and incorporating additive
noise at each pixel.

Programs as Probability Models

Probabilistic programming languages (PPL) built on the insight that probability models can be defined using executable code in any programming language that incorporates a source of randomness.

- inherit all of the expressive power
- computationally universal

Example: Reading text

Program that reads degraded text

- built for reading text that has been smudged or blurred due to water damage, or spotted due to aging of the paper on which it is printed.
- can also be built for breaking some kinds of CAPTCHAs.

- A generative program that has two components:
  - a way to generate a sequence of letters
  - a way to generate a noisy, blurry rendering of these letters using an off-the-shelf graphics library.

Programs as Probability Models

Probabilistic programming languages (PPL) built on the insight that probability models can be defined using executable code

- inherit all of the expressive power
- computationally universal

Example: Reading text

Program that reads degraded text

- built for reading text that has been smudged or blurred due to water damage, or spotted due to aging of the paper on which it is printed.
- can also be built for breaking some kinds of CAPTCHAs.

- **A generative program:** executable program in which every random choice defines a random variable in an associated probability model

- For reading text, a generative program that has two components:
  - a way to generate a sequence of letters
  - a way to generate a noisy, blurry rendering of these letters using an off-the-shelf graphics library.

Programs as Probability Models

X_i : random variable corresponding to the *i*th random choice made by the program

x_i : a possible valueof X_i

$\omega = \{x_i\}$: **execution trace**

- sequence of possible values for the random choices

Probability distribution over traces: product of the probabilities of each individual random choice: $P(\omega) = \prod_i P(x_i | x_{1, \ldots,} x_{i-1})$.

PPL modular in a way that makes it easy to explore improvements to the underlying model.

Can be improved by incorporating a Markov Model

Programs as Probability Models

```
function GENERATE-IMAGE() returns an image with some letters
  letters ← GENERATE-LETTERS(10)
  return RENDER-NOISY-IMAGE(letters, 32, 128)

function GENERATE-LETTERS(λ) returns a vector of letters
  n ~ Poisson(λ)
  letters ← []
  for i = 1 to n do
    letters[i] ~ UniformChoice({a, b, c, ···})
  return letters

function RENDER-NOISY-IMAGE(letters, width, height) returns a noisy image of the letters
  clean_image ← RENDER(letters, width, height, text_top = 10, text_left = 10)
  noisy_image ← []
  noise_variance ~ UniformReal(0.1, 1)
  for row = 1 to width do
    for col = 1 to height do
      noisy_image[row, col] ~ N(clean_image[row, col], noise_variance)
  return noisy_image
```

Generative program for an open-universe probability model for optical character recognition. The generative program produces degraded images containing sequences of letters by generating each sequence, rendering it into a 2D image, and incorporating additive noise at each pixel.

Programs as Probability Models

uncertainty
uncertainty
uncertainty

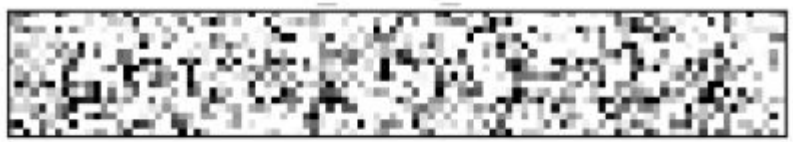
Noisy input image (top) and inference results (bottom) produced by three runs, each of 25 MCMC iterations, with the model from the previous generative program. Note that the inference process correctly identifies the sequence of letters.

Programs as Probability Models

```
function GENERATE-MARKOV-LETTERS(λ) returns a vector of letters
  n ~ Poisson(λ)
  letters ← []
  letter_probs ← MARKOV-INITIAL()
  for i = 1 to n do
      letters[i] ~ Categorical(letter_probs)
      letter_probs ← MARKOV-TRANSITION(letters[i])
  return letters
```

Generative program for an improved optical character recognition model that generates letters according to a letter bigram model whose pairwise letter frequencies are estimated from a list of English words

Programs as Probability Models



qnucitainty	uncertainty
qncurtbinty	uncsitainty
qncuftaintv	unchftsinty

Top: extremely noisy input image. Bottom left: with three inference results from 25 MCMC iterations with the independent-letter model from previous generative program.

Bottom right: three inference results with the letter bigram model. Both models exhibit ambiguity in the results, but the latter model's results reflect prior knowledge of plausible letter sequences

Summary

Relational probability models (RPMs) define probability models on worlds derived from the database semantics for first-order languages

RPMs provide very **concise models for worlds** with large numbers of **objects** and can handle relational **uncertainty**.

Open-universe probability models (OUPMs) build on the full semantics of first-order logic, allowing for new kinds of uncertainty such as identity and existence uncertainty.

Generative programs are representations of probability models—including OUPMs— as executable programs in a probabilistic programming language or PPL.

A generative program represents a distribution over execution traces of the program.

PPLs typically provide universal expressive power for probability models.