Probability Smoothing for NLP

A case study for functional programming and little languages

wren ng thornton

wrnthorn@indiana.edu

Cognitive Science & Computational Linguistics
Indiana University Bloomington

Outline of the talk

- What is the domain?
  - Statistical natural-language processing (NLP)
  - More specifically: part-of-speech (POS) tagging
  - More specifically: ...using hidden Markov models (HMMs)
- What is the problem?
  - Keeping models and algorithms separate, modular
  - Specifying different smoothed models quickly and easily
- The solution
  - A little language
- But what is the problem, really?
  - Achieving high performance, despite modularity
- The revised solution
  - Partial evaluation for loop-invariant code motion

What is the domain?

- Statistical NLP
  - But don't worry if you can't follow the stats
- POS (and other) tagging
  - Given a sequence of words, w_1^N, figure out a sequence of tags, t_1^N, one for each word
- (first-order) HMMs for tagging
  - The "noisy channel model"

$$\begin{tikzcd}
\mathbb{T}_0 \arrow[r, "\Pr(\mathbb{T}_1 \equiv t_1 \mid \mathbb{T}_0 \equiv t_0)"] & \mathbb{T}_1 \arrow[r, "\Pr(\mathbb{T}_1 \equiv t_1 \mid \mathbb{T}_0 \equiv t_0)"] \arrow[d, "\Pr(\mathbb{W}_1 \equiv w_1 \mid \mathbb{T}_1 \equiv t_1)"'] & \mathbb{T}_2 \arrow[d, "\Pr(\mathbb{W}_2 \equiv w_2 \mid \mathbb{T}_2 \equiv t_2)"'] & \ldots & \mathbb{T}_{N-1} \arrow[r, "\Pr(\mathbb{T}_N \equiv t_N \mid \mathbb{T}_{N-1} \equiv t_{N-1})"] \arrow[d, dashed] & \mathbb{T}_N \arrow[d, "\Pr(\mathbb{W}_N \equiv w_N \mid \mathbb{T}_N \equiv t_N)"'] \\
& \mathbb{W}_1 & \mathbb{W}_2 & \ldots & \mathbb{W}_{N-1} & \mathbb{W}_N
\end{tikzcd}$$

What is the problem?

- Keeping models and algorithms separate, modular
  - Should be trivial, but noone seems to do it; why?
  - Will be talked about more later
- Specifying different smoothed models quickly and easily
  - Where do we get those probabilities from?
    - From a model
  - What is a model?
    - A function estimating the true probabilities of events
  - The function is “trained” on some example data
    - i.e., given the data we choose from a family of functions
    - Many different ways to extrapolate from the training data

What is a model?

- The MLE (maximum liklihood estimate) model, aka unsmoothed model

$$p_{MLE}(x \mid y) \models \Pr(x \mid y)$$

$$p_{MLE}(x \mid y) = \frac{c_{XY}(x,y)}{c_Y(y)}$$

where

$c_{XY}(x,y) =$ the count of times an $(x \wedge y)$ joint event was observed

$c_Y(y) =$ the count of times a y event was observed

$$c_Y(y) = \sum_{x \in X} c_{XY}(x,y)$$

- The MLE model maximizes the likelihood of the training data, but it underestimates the likelihood of unseen events; i.e., $c_X(x) = 0 \quad \Longrightarrow \quad p_{MLE}(x \mid y) = 0$

What is a model?

- Add-1 smoothing (aka, Laplace's law)

$$p_{+1}(x \mid y) \models \Pr(x \mid y)$$

$$p_{+1}(x \mid y) = \frac{c_{XY}(x,y)+1}{c_Y(y)+|X|}$$

- Nice: guarantees no zero probabilities for novel events
- Bug: for large domains of possible events it gives too much probability to the novel events

What is a model?

- Add-λ smoothing (aka, Lidstone's law, add-δ smoothing, additive smoothing)

$$p_{+\lambda}(x \mid y) \models \Pr(x \mid y)$$

$$p_{+\lambda}(x \mid y) = \frac{c_{XY}(x,y) + \lambda}{c_Y(y) + \lambda * |X|}$$

- Better, but it requires estimating the parameter λ, and it still doesn't solve the problem in principle

What is a model?

- Chen–Goodman smoothing (aka, one-count smoothing)

$$p_{CG}(x \mid y) \models \Pr(x \mid y)$$

$$p_{CG}(x \mid y) = \frac{c_{XY}(x,y) + s_{XY}(y) * p'(x \mid y)}{c_Y(y) + s_{XY}(y)}$$

where

$s_{XY}(y) =$ the count of $x \in X$ such that $c_{XY}(x,y) = 1$

$p'(x \mid y) \models \Pr(x \mid y')$ where $y' \subset y$

- And others: linear interpolation, Good–Turing, Katz backoff, Witten–Bell, Kneser–Ney, Jelinek–Mercer, Church–Gale, Moore–Quick, and numerous variants

The solution, pt. I

- What is a model?
  - A **function** estimating the true probabilities of events
- A statistical take on the Curry–Howard isomorphism: Probabilities as types; distributions as values
  - $p(x \mid y) \models \Pr(x \mid y) \quad \Longrightarrow \quad p : X \to Y \to \mathbb{P}$
  - $c_X(x) \quad \Longrightarrow \quad c_X : X \to \mathbb{C}$
- The types $\mathbb{P}$ and $\mathbb{C}$ are related by a kind of module structure We'll gloss over the details, but suffice it to say that
  - $\exists(+) : \mathbb{C} \to \mathbb{C} \to \mathbb{C}$
  - $\exists(*) : \mathbb{C} \to \mathbb{P} \to \mathbb{C}$ (or $\mathbb{P} \to \mathbb{C} \to \mathbb{C}$)
  - $\exists(\div) : \mathbb{C} \to \mathbb{C} \to \mathbb{P}$
- With these, we can define a combinator library

The solution, pt. I

$$\text{unsmoothed} : (X \to Y \to \mathbb{C}) \to (Y \to \mathbb{C}) \to (X \to Y \to \mathbb{P})$$
$$\text{unsmoothed}(c_{XY}, c_Y) = \lambda x\, y.\; c_{XY}(x,y) \div c_Y(y)$$

$$\text{addOne} : (X \to Y \to \mathbb{C}) \to (Y \to \mathbb{C}) \to \mathbb{C} \to (X \to Y \to \mathbb{P})$$
$$\text{addOne}(c_{XY}, c_Y, |X|) = \lambda x\, y.\; (c_{XY}(x,y) + 1) \div (c_Y(y) + |X|)$$

$$\text{addLambda}(c_{XY}, c_Y, \delta, |X|) = \lambda x\, y.\; (c_{XY}(x,y) + \delta) \div (c_Y(y) + \delta * |X|)$$
$$\text{chenGoodman}(c_{XY}, c_Y, s_{XY}, p') = \lambda x\, y.\; (c_{XY}(x,y) + s_{XY}(y) * p'(x \mid y)) \div (c_Y(y) + s_{XY}(y))$$

- Combinators like these make it easy to specify complex smoothing methods, as well as being clear and explicit about it

But what is the problem really?

- Keeping models and algorithms separate, modular
    - Using HOFs makes this easy
- . . . While achieving high performance
    - These probability distributions will be evaluated inside triply nested loops: $\forall i.\ \forall y_i.\ \forall x_i.\ p(x_i \mid y_i)$; or worse
    - Standard optimizations from imperative programming aren't available; e.g., loop invariant code motion
    - . . . Or are they?

Loop invariant code motion

- Lifting invariant code **can** improve asymptotic performance
  - $O(m*(n+o)) \quad \Longrightarrow \quad O(n+m*o)$
- So-called “constant” factors should not be ignored, because parameters are not constant in practice
  - e.g., the Forward algorithm is $O(T^2*N)$, not $O(N)$
- Idea: use partial evaluation to perform LICM dynamically
  - We know the order of the loops: y is outer, x is inner
  - $p(x \mid y) \models \Pr(x \mid y) \quad \Longrightarrow \quad p : Y \to X \to \mathbb{P}$
  - Now we can take the partial *application*, $p(y)$, and perform partial *evaluation*
  - $p(y) : X \to \mathbb{P} \quad \Longrightarrow \quad p_y(x) \models \Pr(x \mid y)$

LICM example

```
chenGoodman cyx cy syx pyx y = let
    !cyx_y = cyx y
    !pyx_y = pyx y
    !syx_y = syx y
    !z     = cy y + syx_y
    in λ x → (cyx_y x + syx_y * pyx_y x) / z
```

Dynamic LICM

- Lame benchmark: gives 10% total-runtime reduction
  - Includes extraneous things like I/O
    (for an I/O-bound program)
  - Actual improvement is asymptotic
    (the 10% was for a standard corpus)
- Allows us to perform LICM *at runtime*
  - With a JIT, could fuse the model and the algorithm
  - Or we can LICM and fuse at compile time, via INLINE
- We don't need to do it manually
  - Retains separation of concerns
    - don't pollute the algorithm with modeling concerns
  - Keeps code legible
    - say what you mean, not how to optimize it
  - All the details are hidden away in the library
    - that $\geq 10\%$ improvement required **no** client code changes

$\sim$*fin.*