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

27 July 2011
AMMCS, SS-SSD
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”

\[
\begin{align*}
\Pr(T_1 \equiv t_1 | T_0 \equiv t_0) & \quad \Pr(T_1 \equiv t_1 | T_0 \equiv t_0) & \quad \Pr(T_N \equiv t_N | T_{N-1} \equiv t_{N-1}) \\
\Pr(W_1 \equiv w_1 | T_1 \equiv t_1) & \quad \Pr(W_2 \equiv w_2 | T_2 \equiv t_2) & \quad \Pr(W_N \equiv w_N | T_N \equiv t_N)
\end{align*}
\]
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 model is “trained” on some example data
    • i.e., given the data, choose from a family of functions
  ◦ Many different ways to extrapolate from the training data
    • which family do we choose from?
What is a model?

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

\[
p_{MLE}(x \mid y) = \Pr(x \mid y)
\]

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

where

\[
c_{XY}(x, y) = \text{the count of times an } (x \land y) \text{ joint event was observed}
\]

\[
c_Y(y) = \text{the count of times a } y \text{ event was observed}
\]

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

- The MLE model maximizes the likelihood of the training data, but it underestimates the likelihood of unseen events; i.e.,

\[
c_X(x) = 0 \implies p_{MLE}(x \mid y) = 0
\]
What is a model?

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

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

- 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-\(\lambda\) smoothing (aka, Lidstone’s law, add-\(\delta\) smoothing, additive smoothing)

\[
p_{+\lambda}(x \mid y) = \Pr(x \mid y)
\]

\[
p_{+\lambda}(x \mid y) = \frac{c_{XY}(x, y) + \lambda}{c_Y(y) + \lambda \ast |X|}
\]

• Better, but it requires estimating the parameter \(\lambda\), and it still doesn’t solve the problem in principle
What is a model?

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

\[ p_{CG}(x \mid y) = \Pr(x \mid y) \]

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

where

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

\[ p'(x \mid y) = \Pr(x \mid y') \text{ where } y' \subset y \]

- 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
  - \( p(x | y) \models \Pr(x | y) \implies p : X \to Y \to \mathbb{P} \)
  - \( c_X(x) \implies c_X : X \to \mathbb{C} \)

- 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
    - \( \exists (+) : \mathbb{C} \to \mathbb{C} \to \mathbb{C} \)
    - \( \exists (\ast) : \mathbb{C} \to \mathbb{P} \to \mathbb{C} \) (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

unsmoothed : $(X \rightarrow Y \rightarrow \mathbb{C}) \rightarrow (Y \rightarrow \mathbb{C}) \rightarrow (X \rightarrow Y \rightarrow \mathbb{P})$

unsmoothed$(c_{XY}, c_Y) = \lambda x. y. c_{XY}(x, y) \div c_Y(y)$

addOne : $(X \rightarrow Y \rightarrow \mathbb{C}) \rightarrow (Y \rightarrow \mathbb{C}) \rightarrow \mathbb{C} \rightarrow (X \rightarrow Y \rightarrow \mathbb{P})$

addOne$(c_{XY}, c_Y, |X|) = \lambda x. y. \left( c_{XY}(x, y) + 1 \right) \div \left( c_Y(y) + |X| \right)$

addLambda$(c_{XY}, c_Y, \delta, |X|) = \lambda x. y. \left( c_{XY}(x, y) + \delta \right) \div \left( c_Y(y) + \delta \ast |X| \right)$

chenGoodman$(c_{XY}, c_Y, s_{XY}, p') = \lambda x. y. \left( c_{XY}(x, y) + s_{XY}(y) \ast p'(x \mid y) \right) \div \left( c_Y(y) + s_{XY}(y) \right)$

- 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
  - $O(m \times (n + o)) \implies O(n + m \times o)$

- So-called “constant” factors should not be ignored, because parameters are not constant in practice
  - The first-order Forward algorithm is $O(T^2 \times N)$, not $O(N)$

- Idea: use partial evaluation to perform LICM dynamically
  - We know the order of the loops: $y$ is outer, $x$ is inner
    - $p(x \mid y) \models \Pr(x \mid y) \implies p : Y \rightarrow X \rightarrow \mathbb{P}$
  - Now we can take the partial application, $p(y)$, and perform partial evaluation
    - $p(y) : X \rightarrow \mathbb{P} \implies p_y(x) \models \Pr(x \mid y)$
LICM example

\[
\text{chenGoodman } \text{cyx cy syx pyx y} = \text{let}
\]
\[
\begin{align*}
!\text{cyx}_y &= \text{cyx } y \\
!\text{pyx}_y &= \text{pyx } y \\
!\text{syx}_y &= \text{syx } y \\
!z &= \text{cy } y + \text{syx}_y \\
\text{in } \lambda x \rightarrow (\text{cyx}_y x + \text{syx}_y \ast \text{pyx}_y x) &\div z
\end{align*}
\]
Dynamic LICM

- Original benchmark: gives 10% total-runtime reduction
  - Includes extraneous things like I/O (for an I/O-bound program)
  - Actual improvement is superlinear (because of the asymptotic role of $T$)
- All the details are hidden away in the library
  - that 10% improvement required no client code changes

- Caveat lector
  - More recent benchmarks are less impressive
    - only 3%, excluding all extraneous factors
    - no observed non-linear behavior
  - This is due to algorithmic optimizations since then
    - but that’s an orthogonal concern
Conclusions

• Allows us to perform LICM at runtime
  ◦ With a JIT, could fuse the model and the algorithm
    • to remove indirection overhead
  ◦ Or we can do LICM and fusion at compile time
    via {−# INLINE #−} pragma

• Retains separation of concerns
  ◦ Don’t pollute the algorithm with modeling concerns
  ◦ Don’t base the algorithm around a particular model

• Keeps code legible
  ◦ Say what you mean, not how to optimize it
  ◦ All the details are hidden away in the library
~fin.