Language models, part II: smoothing

Roger Levy

thanks to Dan Klein and Jason Eisner
Recap: Language Models

- Why are language models useful?
- Samples of generated text
- What are the main challenges in building n-gram language models?
  - Discounting versus Backoff/Interpolation
Smoothing

• We often want to make estimates from sparse statistics:

  \[ P(w \mid \text{denied the}) \]
  3 allegations
  2 reports
  1 claims
  1 request
  7 total

  ![Bar chart showing allocations, reports, claims, request, attack, man, outcome.

  • Smoothing flattens spiky distributions so they generalize better

  \[ P(w \mid \text{denied the}) \]
  2.5 allocations
  1.5 reports
  0.5 claims
  0.5 request
  2 other
  7 total

  ![Bar chart showing allocations, reports, claims, request, attack, man, outcome.

  • Very important all over NLP, but easy to do badly!
  • We'll illustrate with bigrams today (h = previous word, could be anything).
Vocabulary Size

- Key issue for language models: open or closed vocabulary?
  - A *closed vocabulary* means you can fix a set of words in advance that may appear in your training set
  - An *open vocabulary* means that you need to hold out probability mass for any possible word
    - Generally managed by fixing a *vocabulary list*; words not in this list are OOVs
  - When would you want an open vocabulary?
  - When would you want a closed vocabulary?

- How to set the vocabulary size $V$?
  - By external factors (e.g. speech recognizers)
  - Using statistical estimates?
  - Difference between estimating unknown token rate and probability of a given unknown word

- Practical considerations
  - In many cases, open vocabularies use multiple types of OOVs (e.g., numbers & proper names)

- For the programming assignment:
  - OK to assume there is only one unknown word type, UNK
  - UNK be quite common in new text!
  - UNK stands for all unknown word type
Five types of smoothing

- Today we’ll cover
  - Add-\(\delta\) smoothing (Laplace)
  - Simple interpolation
  - Good-Turing smoothing
  - Katz smoothing
  - Kneser-Ney smoothing
Smoothing: Add-δ (for bigram models)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>number of word tokens in training data</td>
</tr>
<tr>
<td>c(w)</td>
<td>count of word w in training data</td>
</tr>
<tr>
<td>c(w−1,w)</td>
<td>joint count of the w−1,w bigram</td>
</tr>
<tr>
<td>V</td>
<td>total vocabulary size (assumed known)</td>
</tr>
<tr>
<td>N_k</td>
<td>number of word types with count k</td>
</tr>
</tbody>
</table>

- One class of smoothing functions (*discounting*):
  - Add-one / delta:
    \[ P_{ADD-\delta}(w \mid w_{-1}) = \frac{c(w_{-1}, w) + \delta(1/V)}{c(w_{-1}) + \delta} \]
  - If you know Bayesian statistics, this is equivalent to assuming a uniform prior
  - Another (better?) alternative: assume a unigram prior:
    \[ P_{UNI-PRIOR}(w \mid w_{-1}) = \frac{c(w, w_{-1}) + \delta \hat{P}(w)}{c(w_{-1}) + \delta} \]
  - How would we estimate the unigram model?
Linear Interpolation

• One way to ease the sparsity problem for n-grams is to use less-sparse n-1-gram estimates

• General linear interpolation:

\[ P(w \mid w_{-1}) = [1 - \lambda(w, w_{-1})]\hat{P}(w \mid w_{-1}) + [\lambda(w, w_{-1})]P(w) \]

• Having a single global mixing constant is generally not ideal:

\[ P(w \mid w_{-1}) = [1 - \lambda]\hat{P}(w \mid w_{-1}) + [\lambda]P(w) \]

• A better yet still simple alternative is to vary the mixing constant as a function of the conditioning context

\[ P(w \mid w_{-1}) = [1 - \lambda(w_{-1})]\hat{P}(w \mid w_{-1}) + \lambda(w_{-1})P(w) \]
Held-Out Data

- Important tool for getting models to generalize:
  - When we have a small number of parameters that control the degree of smoothing, we set them to maximize the (log-)likelihood of held-out data.
  
\[
LL(w, \ldots, w_i | M(\lambda_1, \ldots, \lambda_k)) = \sum_i \log P_M(w_i | w_{i-1})
\]
- Can use any optimization technique (line search or EM usually easiest).
- Examples:

\[
P\text{LIN}(\lambda_1, \lambda_2)(w | w_{-1}) = \lambda_1 \hat{P}(w | w_{-1}) + \lambda_2 \hat{P}(w)
\]

\[
P_{\text{UNI-PRIOR}(\delta)}(w | w_{-1}) = \frac{c(w, w_{-1}) + \delta \hat{P}(w)}{c(w_{-1}) + \delta}
\]
Held-Out Reweighting

- What’s wrong with unigram-prior smoothing?
- Let’s look at some real bigram counts [Church and Gale 91]:

<table>
<thead>
<tr>
<th>Count in 22M Words</th>
<th>Actual c* (Next 22M)</th>
<th>Add-one’s c*</th>
<th>Add-0.0000027’s c*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.448</td>
<td>2/7e-10</td>
<td>~1</td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
<td>3/7e-10</td>
<td>~2</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
<td>4/7e-10</td>
<td>~3</td>
</tr>
<tr>
<td>4</td>
<td>3.23</td>
<td>5/7e-10</td>
<td>~4</td>
</tr>
<tr>
<td>5</td>
<td>4.21</td>
<td>6/7e-10</td>
<td>~5</td>
</tr>
</tbody>
</table>

| Mass on New       | 9.2%                 | ~100%        | 9.2%              |
| Ratio 22M/Next    | 2.8                  | 1.5          | ~2                |

- Big things to notice:
  - Add-one vastly overestimates the fraction of new bigrams
  - Add-0.0000027 still underestimates the ratio 2*/1*
- One solution: use held-out data to predict the map of c to c*
Good-Turing smoothing

- Motivation: how can we estimate how likely events we haven’t yet seen are to occur?
- Insight: *singleton events* are our best indicator for this probability
- Generalizing the insight: *cross-validated* models

\[
P(w_i) \quad \text{Training Data (C)}
\]

- We want to estimate \(P(w_i)\) on the basis of the corpus \(C - w_i\)
- But we can’t just do this naively (why not?)
Good-Turing Reweighting I

- Take each of the $c$ training words out in turn
- $c$ training sets of size $c-1$, held-out of size 1
- What fraction of held-out word (tokens) are unseen in training?
  - $N_1/c$
- What fraction of held-out words are seen $k$ times in training?
  - $(k+1)N_{k+1}/c$
- So in the future we expect $(k+1)N_{k+1}/c$ of the words to be those with training count $k$
- There are $N_k$ words with training count $k$
- Each should occur with probability:
  - $(k+1)N_{k+1}/(cN_k)$
- …or expected count $(k+1)N_{k+1}/N_k$
Problem: what about “the”? (say \(c=4417\))
- For small \(k\), \(N_k > N_{k+1}\)
- For large \(k\), too jumpy, zeros wreck estimates

Simple Good-Turing [Gale and Sampson]: replace empirical \(N_k\) with a best-fit regression (e.g., power law) once count counts get unreliable
Good-Turing Reweighting III

- Hypothesis: counts of $k$ should be $k^* = \frac{(k+1)N_{k+1}}{N_k}$

<table>
<thead>
<tr>
<th>Count in 22M Words</th>
<th>Actual $c^*$ (Next 22M)</th>
<th>GT’s $c^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.448</td>
<td>0.446</td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
<td>1.26</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>3.23</td>
<td>3.24</td>
</tr>
</tbody>
</table>

| Mass on New        | 9.2%                    | 9.2%       |

- Not bad!
Katz Smoothing

• Katz (1987) extended the idea of Good-Turing (GT) smoothing to higher models, incorporating backoff
• Here we’ll focus on the backoff procedure
• Intuition: when we’ve never seen an n-gram, we want to back off (recursively) to the lower order n-1-gram
• So we want to do:

\[ P(w \mid w_{-1}) = \begin{cases}  
P(w \mid w_{-1}) & c(w,w_{-1}) > 0 \\
P(w) & c(w,w_{-1}) = 0 
\end{cases} \]

• But we can’t do this (why not?)
Katz Smoothing II

- We can’t do

\[ P(w | w_{-1}) = \begin{cases} 
P(w | w_{-1}) & c(w, w_{-1}) > 0 \\
P(w) & c(w, w_{-1}) = 0 
\end{cases} \]

- But if we use GT-discounted estimates \( P^*(w | w_{-1}) \), we do have probability mass left over for the unseen bigrams

- There are a couple of ways of using this. We could do:

\[ P(w | w_{-1}) = \begin{cases} 
P_{GT}(w | w_{-1}) & c(w, w_{-1}) > 0 \\
\alpha(w_{-1})P(w) & c(w, w_{-1}) = 0 
\end{cases} \]

- or

\[ P(w | w_{-1}) = P_{GT}^*(w | w_{-1}) + \alpha(w_{-1})P(w) \]

see books, Chen & Goodman 1998 for more details
Kneser-Ney Smoothing I

- Something’s been very broken all this time
  - Shannon game: There was an unexpected ____?
    - delay?
    - Francisco?
  - “Francisco” is more common than “delay”
  - … but “Francisco” always follows “San”

- Solution: Kneser-Ney smoothing
  - In the back-off model, we don’t want the unigram probability of $w$
  - Instead, probability given that we are observing a novel continuation
  - Every bigram type was a novel continuation the first time it was seen

$$P_{\text{CONTINUATION}}(w) = \frac{|\{w_{-1} : c(w, w_{-1}) > 0\}|}{|(w, w_{-1}) : c(w, w_{-1}) > 0|}$$
Language models, part II: smoothing

Roger Levy

thanks to Dan Klein and Jason Eisner
What Actually Works?

- Trigrams:
  - Unigrams, bigrams too little context
  - Trigrams much better (when there’s enough data)
  - 4-, 5-grams usually not worth the cost (which is more than it seems, due to how speech recognizers are constructed)
- Good-Turing-like methods for count adjustment
  - Absolute discounting, Good-Turing, held-out estimation, Witten-Bell
- Kneser-Ney equalization for lower-order models
- See [Chen+Goodman] reading for tons of graphs!

[Graphs from Joshua Goodman]
• Having more data is always good…

• … but so is picking a better smoothing mechanism!
• N > 3 often not worth the cost (greater than you’d think)
Beyond N-Gram LMs

- Caching Models
  - Recent words more likely to appear again
    \[ P_{\text{CACHE}}(w \mid \text{history}) = \lambda P(w \mid w_{-1}w_{-2}) + (1 - \lambda) \frac{c(w \in \text{history})}{\mid \text{history} \mid} \]
  - Can be disastrous in practice for speech (why?)

- Skipping Models

  \[ P_{\text{SKIP}}(w \mid w_{-1}w_{-2}) = \lambda \hat{P}(w \mid w_{-1}w_{-2}) + \lambda P(w \mid w_{-1}) + \lambda P(w \mid w_{-2}) \]

- Clustering Models: condition on word classes when words are too sparse
- Trigger Models: condition on bag of history words (e.g., maxent)
- Structured Models: use parse structure (we’ll see these later)