Recap: Smoothing

- We often want to make predictions from sparse statistics:
  \[ P(w \mid \text{denied the}) \]
  - 3 allegations
  - 2 reports
  - 1 claims
  - 1 request
  - 7 total

- Smoothing flattens spiky distributions so they generalize better

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

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
  \[ LL(w_1 \ldots w_L \mid M(\lambda_1, \ldots, \lambda_k)) = \sum \log P_M(w_{i-1} \mid w_i) \]
  - Can use any optimization technique (line search or EM usually easiest)
  - Examples:
    - \( P_{LH}(w \mid w_{-1}) = \hat{\lambda} P(w \mid w_{-1}) + \hat{\lambda} \hat{P}(w) \)
    - \( P_{UNI-PRIOR}(w \mid w_{-1}) = \frac{c(w, w_{-1}) + \delta}{c(w_{-1}) + \delta} \)

Good-Turing Reweighting I

- We’d like to not need held-out data (why?)
- Idea: leave-one-out validation:
  - 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 words are unseen in training?
    \[ N_{\text{held out}} \]
  - What fraction of held-out words are seen \( k \) times in training?
    \[ N_k \]
  - So in the future we expect \( (k+1)N_k/N_{\text{held out}} \) of the words to be those with training count \( k \)
  - There are \( N_k \) words with training count \( k \)
  - Each should occur with probability:
    - \( (k+1)N_k/N_{\text{held out}} \)
    - … or expected count \((k+1)N_k/N_{\text{held out}}\)
Good-Turing Reweighting II

- 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 power law once count counts get unreliable

Good-Turing Reweighting III

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

Katz Smoothing
- Use GT discounted bigram counts (roughly – Katz left large counts alone)
- Whatever mass is left goes to empirical unigram

\[
P_{\text{GT}}(w | w_j) = \sum_{w_{j+1}}^{} c(W, w_{j+1}) + \alpha(w_j)\hat{P}(w)
\]

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

Good-Turing Reweighting II

\[
P_{\text{Kneser-Ney}}(w | w_j) = \left(1 + \frac{1}{\alpha} \right) P_{\text{GT}}(w | w_j)
\]

- Absolute Discounting
  - Save ourselves some time and just subtract 0.75 (or some \(d\))
  - Maybe have a separate value of \(d\) for very low counts

Kneser-Ney Smoothing II

- One more aspect to Kneser-Ney:
  - Look at the GT counts:

\[
P_{\text{Kneser-Ney}}(w | w_j) = \left(1 + \frac{1}{\alpha} \right) P_{\text{GT}}(w | w_j)
\]

- Absolute Discounting
  - Save ourselves some time and just subtract 0.75 (or some \(d\))
  - Maybe have a separate value of \(d\) for very low counts

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

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

\[
P_{\text{Kneser-Ney}}(w | w_j) = \left(1 + \frac{1}{\alpha} \right) P_{\text{GT}}(w | w_j)
\]

- Absolute Discounting
  - Save ourselves some time and just subtract 0.75 (or some \(d\))
  - Maybe have a separate value of \(d\) for very low counts

Kneser-Ney Smoothing II

- One more aspect to Kneser-Ney:
  - Look at the GT counts:

\[
P_{\text{Kneser-Ney}}(w | w_j) = \left(1 + \frac{1}{\alpha} \right) P_{\text{GT}}(w | w_j)
\]

- Absolute Discounting
  - Save ourselves some time and just subtract 0.75 (or some \(d\))
  - Maybe have a separate value of \(d\) for very low counts

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

Data >> Method?

- 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
    \[ P_{\text{cache}}(w | \text{history}) = \lambda P(w | \text{history}) + (1 - \lambda) P(w | \text{history}) \]
  - Can be disastrous in practice for speech (why?)

- Skipping Models
  \[ P_{\text{skip}}(w | w_1, w_2, \ldots, w_n) = \lambda P(w | w_1, w_2, \ldots, w_n) + \sum P(w | w_{n+1}, \ldots, w_n) \]

- Trigger Models: condition on bag of history words (e.g., maxent)
- Structured Models: use parse structure (we'll see these later)

Text Categorization

- Want to classify documents into broad semantic topics (e.g., politics, sports, etc.)
- Demo: vice presidential candidate John Edwards accused President Bush and Vice President Dick Cheney of misleading Americans by implying a link between deposed Iraqi President Saddam Hussein and the Sept. 11, 2001 terrorist attacks.
- Which one is the politics document? (And how much deep processing did that decision take?)
- One approach: bag-of-words and Naïve-Bayes models
- Another approach next lecture...

Naïve-Bayes Models

- Idea: pick a topic, then generate a document using a language model for that topic.
- Naïve-Bayes assumption: all words are independent given the topic.
- Compare to a unigram language model:
  \[ P(w_1, w_2, \ldots, w_n) = \prod_{i} P(w_i) \]

Using NB for Classification

- We have a joint model of topics and documents
  \[ P(c, w_1, w_2, \ldots, w_n) = P(c) \prod P(w_i | c) \]
- Gives posterior likelihood of topic given a document
  \[ P(c | w_1, w_2, \ldots, w_n) = \frac{P(c) \prod P(w_i | c)}{\sum_{c'} P(c') \prod P(w_i | c')} \]

- What about totally unknown words?
- Can work shockingly well for textcat (especially in the wild)
- How can unigram models be so terrible for language modeling, but class-conditional unigram models work for textcat?

Two NB Formulations

- Two NB models for text categorization
  - The class-conditional unigram model, a.k.a. multinomial model
    - One node per word in the document
    - Driven by words which are present
    - Multiple occurrences, multiple evidence
    - Better overall – plus, know how to smooth
  - The binary model
    - One node for each word in the vocabulary
    - Incorporates explicit negative correlations
    - Know how to do feature selection (e.g. keep words with high mutual information with the class variable)

Example: Sensors

- NB FACTORS:
  - P(s) = 1/2
  - P(+|s) = 1/4
  - P(+|r) = 3/4

- NB Model
  - Incorporates explicit negative correlations
  - Know how to do feature selection (e.g. keep words with high mutual information with the class variable)

- Reality
  - P(+,+,r) = 3/8
  - P(+,+,s) = 1/8
  - P(r,+,+) = 9/10
  - P(s,+,+) = 1/10

- PREDICTIONS:
  - P(r,+,+) = 9/10
  - P(s,+,+) = 1/10

- Overconfidence!
Example: Stoplights

<table>
<thead>
<tr>
<th>Reality</th>
<th>Lights Working</th>
<th>Lights Broken</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(g,r,w) = 3/7</td>
<td>P(g,r,w) = 3/7</td>
<td>P(r,r,b) = 1/7</td>
</tr>
</tbody>
</table>

NB Model

<table>
<thead>
<tr>
<th>Working?</th>
<th>NS</th>
<th>EW</th>
</tr>
</thead>
</table>

NB FACTORS:

- P(w) = 6/7
- P(r|w) = 1/2
- P(g|w) = 1/2
- P(b) = 1/7
- P(r|b) = 1
- P(g|b) = 0

P(r|r,r) = 4/10 (what happened?)

(Non-)Independence Issues

- Mild Non-Independence
  - Evidence all points in the right direction
  - Observations just not entirely independent
  - Results
    - Inflated Confidence
    - Deflated Priors
  - What to do? Boost priors or attenuate evidence

- Severe Non-Independence
  - Words viewed independently are misleading
  - Interactions have to be modeled
  - What to do?
    - Change your model

Language Identification

- How can we tell what language a document is in?
- How to tell the French from the English?
  - Treat it as word-level textcat?
  - Overkill, and requires a lot of training data
  - You don’t actually need to know about words!
  - Σύµµοργος καθειρήτου και ανάπτυξης
  - Option: build a character-level language model

Class-Conditional LMs

- Can have a topic variable for other language models
- Could be characters instead of words, used for language ID (HW1)
- Could sum out the topic variable and use as a language model
- How might a class-conditional n-gram language model behave differently from a standard n-gram model?

History Lattices

- Often we have multinomials which condition on lots of other events
- induce back-off lattices
- Can either pick a linear chain or use multiple mixing weights
- Often a sign that one should use other techniques, such as maximum entropy modeling (next class)

EM for Mixing Parameters

- How to estimate mixing parameters?
  - $P_{EM}(w | w_c) = \tilde{P}(w | w_c)P_c + \lambda \tilde{P}(w)$
  - Sometimes you can just do line search
  - … or the “try a few orders of magnitude” approach

- Alternative: Use EM
  - Think of mixing as a hidden choice between histories:
  - $P_{EM}(w | w_c) = P_1(1)\tilde{P}(w | w_c) + P_2(0)\tilde{P}(w)$
  - Given a guess at $P_{EM}$, we can calculate expectations of which generation route a given token took (over held-out data, why?)
- $P(h = 1 | w, w_c) = \frac{P_1(1)\tilde{P}(w | w_c)}{P_1(1)\tilde{P}(w | w_c) + P_2(0)\tilde{P}(w)}$
- Use these expectations to update $P_{EM}$, rinse and repeat
What’s Next

- Next class:
  - Word Sense Disambiguation
  - Maximum Entropy Models

- Reading: M+S 7