Automatic Speech Recognition (CS753)
Lecture 12: Language Models (Part II)

Instructor: Preethi Jyothi
Aug 31, 2017
Recap

- Ngram language models are popularly used in various ML applications

- Language models are evaluated using the *perplexity* (normalized per-word cross-entropy) measure.
  - For a uniform unigram model over $L$ words, perplexity $= L$.

- MLE estimates for Ngram models assume there are no unseen Ngrams

- Smoothing algorithms: **Discount** some probability mass from seen Ngrams and redistribute discounted mass to unseen events
Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
  - Katz Backoff Smoothing
  - Absolute Discounting Interpolation
- Kneser-Ney Smoothing
Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
  - Katz Backoff Smoothing
  - Absolute Discounting Interpolation
- Kneser-Ney Smoothing
Recall add-1/add-\(\alpha\) smoothing (also viewed as discounting)

- Smoothing can be viewed as **discounting** (lowering) some probability mass from seen Ngrams and redistributing discounted mass to unseen events

- i.e. probability of a bigram with Laplace (add-1) smoothing

\[
\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i) + 1}{\pi(w_{i-1}) + V}
\]

- can be written as

\[
\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi^*(w_{i-1}, w_i)}{\pi(w_{i-1})}
\]

- where discounted count \(\pi^*(w_{i-1}, w_i) = (\pi(w_{i-1}, w_i) + 1)\frac{\pi(w_{i-1})}{\pi(w_{i-1}) + V}\)
Problems with Add-\(\alpha\) Smoothing

- What’s wrong with add-\(\alpha\) smoothing?

- Assigns too much probability mass away from seen Ngrams to unseen events

- Does not discount high counts and low counts correctly

- Also, \(\alpha\) is tricky to set

- Is there a more principled way to do this smoothing?  
  A solution: Good-Turing estimation
Good-Turing estimation
(uses held-out data)

<table>
<thead>
<tr>
<th>r</th>
<th>(N_r)</th>
<th>(r^*) in heldout set</th>
<th>add-1 (r^*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(2 \times 10^6)</td>
<td>0.448</td>
<td>2.8\times10^{-11}</td>
</tr>
<tr>
<td>2</td>
<td>(4 \times 10^5)</td>
<td>1.25</td>
<td>4.2\times10^{-11}</td>
</tr>
<tr>
<td>3</td>
<td>(2 \times 10^5)</td>
<td>2.24</td>
<td>5.7\times10^{-11}</td>
</tr>
<tr>
<td>4</td>
<td>(1 \times 10^5)</td>
<td>3.23</td>
<td>7.1\times10^{-11}</td>
</tr>
<tr>
<td>5</td>
<td>(7 \times 10^4)</td>
<td>4.21</td>
<td>8.5\times10^{-11}</td>
</tr>
</tbody>
</table>

\(r = \text{Count in a large corpus} \& N_r \text{ is the number of bigrams with } r \text{ counts}
\(r^* \) is estimated on a different held-out corpus

- Add-1 smoothing hugely overestimates fraction of unseen events
- Good-Turing estimation uses held-out data to predict how to go from \(r\) to the heldout-\(r^*\)

[CG91]: Church and Gale, “A comparison of enhanced Good-Turing…”, CSL, 1991
Good-Turing Estimation

- Intuition for Good-Turing estimation using leave-one-out validation:
  - Let $N_r$ be the number of word types that occur $r$ times
  - Split a given set of $N$ word tokens into a training set of $(N-1)$ samples + 1 sample as the held-out set; repeat this process $N$ times so that all $N$ samples appear in the held-out set
  - In what fraction of these $N$ trials is the held-out word unseen during training? $N_1/N$
  - In what fraction of these $N$ trials is the held-out word seen exactly $k$ times during training? $(k+1)N_{k+1}/N$
  - There are $(\approx)N_k$ words with training count $k$. Each should occur with probability: $(k+1)N_{k+1}/(N \times N_k)$
  - Expected count of each of the $N_k$ words: $k^* = \theta(k) = (k+1) N_{k+1}/N_k$
Good-Turing Estimates

<table>
<thead>
<tr>
<th>r</th>
<th>$N_r$</th>
<th>$r^*-\text{GT}$</th>
<th>$r^*-\text{heldout}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$7.47 \times 10^{10}$</td>
<td>.0000270</td>
<td>.0000270</td>
</tr>
<tr>
<td>1</td>
<td>$2 \times 10^6$</td>
<td>0.446</td>
<td>0.448</td>
</tr>
<tr>
<td>2</td>
<td>$4 \times 10^5$</td>
<td>1.26</td>
<td>1.25</td>
</tr>
<tr>
<td>3</td>
<td>$2 \times 10^5$</td>
<td>2.24</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>$1 \times 10^5$</td>
<td>3.24</td>
<td>3.23</td>
</tr>
<tr>
<td>5</td>
<td>$7 \times 10^4$</td>
<td>4.22</td>
<td>4.21</td>
</tr>
<tr>
<td>6</td>
<td>$5 \times 10^4$</td>
<td>5.19</td>
<td>5.23</td>
</tr>
<tr>
<td>7</td>
<td>$3.5 \times 10^4$</td>
<td>6.21</td>
<td>6.21</td>
</tr>
<tr>
<td>8</td>
<td>$2.7 \times 10^4$</td>
<td>7.24</td>
<td>7.21</td>
</tr>
<tr>
<td>9</td>
<td>$2.2 \times 10^4$</td>
<td>8.25</td>
<td>8.26</td>
</tr>
</tbody>
</table>

Table showing frequencies of bigrams from 0 to 9
In this example, for $r > 0$, $r^*-\text{GT} \approx r^*-\text{heldout}$ and $r^*-\text{GT}$ is always less than $r$

[CG91]: Church and Gale, “A comparison of enhanced Good-Turing…”, CSL, 1991
Good-Turing Smoothing

• Thus, Good-Turing smoothing states that for any Ngram that occurs \( r \) times, we should use an adjusted count \( r^* = \theta(r) = (r + 1)N_{r+1}/N_r \)

• Good-Turing smoothed counts for unseen events: \( \theta(0) = N_1/N_0 \)

• Example: 10 bananas, 5 apples, 2 papayas, 1 melon, 1 guava, 1 pear
  • How likely are we to see a guava next? The GT estimate is \( \theta(1)/N \)
  • Here, \( N = 20 \), \( N_2 = 1 \), \( N_1 = 3 \). Computing \( \theta(1) \): \( \theta(1) = 2 \times 1/3 = 2/3 \)
  • Thus, \( Pr_{GT}(\text{guava}) = \theta(1)/20 = 1/30 = 0.0333 \)
Good-Turing Estimation

- One issue: For large $r$, many instances of $N_{r+1} = 0!$
  - This would lead to $\theta(r) = (r + 1)N_{r+1}/N_r$ being set to 0.

- Solution: Discount only for small counts $r \leq k$ (e.g. $k = 9$) and $\theta(r) = r$ for $r > k$

- Another solution: Smooth $N_r$ using a best-fit power law once counts start getting small

- Good-Turing smoothing tells us how to discount some probability mass to unseen events. Could we redistribute this mass across observed counts of lower-order Ngram events?
Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
  - Katz Backoff Smoothing
  - Absolute Discounting Interpolation
- Kneser-Ney Smoothing
Backoff and Interpolation

• General idea: It helps to use lesser context to generalise for contexts that the model doesn’t know enough about

• Backoff:
  • Use trigram probabilities if there is sufficient evidence
  • Else use bigram or unigram probabilities

• Interpolation
  • Mix probability estimates combining trigram, bigram and unigram counts
Interpolation

- Linear interpolation: Linear combination of different Ngram models

\[
\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\
+ \lambda_2 P(w_n|w_{n-1}) \\
+ \lambda_3 P(w_n)
\]

where \( \lambda_1 + \lambda_2 + \lambda_3 = 1 \)

How to set the \( \lambda \)'s?
Interpolation

- Linear interpolation: Linear combination of different Ngram models

\[
\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\
+ \lambda_2 P(w_n|w_{n-1}) \\
+ \lambda_3 P(w_n)
\]

where \( \lambda_1 + \lambda_2 + \lambda_3 = 1 \)

1. Estimate N-gram probabilities on a training set.

2. Then, search for \( \lambda \)'s that maximises the probability of a held-out set
Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
  - Katz Backoff Smoothing
  - Absolute Discounting Interpolation
- Kneser-Ney Smoothing
Katz Smoothing

- Good-Turing discounting determines the volume of probability mass that is allocated to unseen events.

- Katz Smoothing distributes this remaining mass proportionally across “smaller” Ngrams.

  - i.e. no trigram found, use backoff probability of bigram and if no bigram found, use backoff probability of unigram.
Katz Backoff Smoothing

- For a Katz bigram model, let us define:
  \[ \Psi(w_{i-1}) = \{ w : \pi(w_{i-1}, w) > 0 \} \]

- A bigram model with Katz smoothing can be written in terms of a unigram model as follows:

\[
P_{\text{Katz}}(w_i | w_{i-1}) = \begin{cases} 
\frac{\pi^*(w_{i-1}, w_i)}{\pi(w_{i-1})} & \text{if } w_i \in \Psi(w_{i-1}) \\
\alpha(w_{i-1}) P_{\text{Katz}}(w_i) & \text{if } w_i \not\in \Psi(w_{i-1})
\end{cases}
\]

where \( \alpha(w_{i-1}) = \frac{1 - \sum_{w \in \Psi(w_{i-1})} \frac{\pi^*(w_{i-1}, w)}{\pi(w_{i-1})}}{\sum_{w_i \not\in \Psi(w_{i-1})} P_{\text{Katz}}(w_i)} \)
Katz Backoff Smoothing

\[ P_{\text{Katz}}(w_i|w_{i-1}) = \begin{cases} \frac{\pi^*(w_{i-1},w_i)}{\pi(w_{i-1})} & \text{if } w_i \in \Psi(w_{i-1}) \\ \alpha(w_{i-1})P_{\text{Katz}}(w_i) & \text{if } w_i \notin \Psi(w_{i-1}) \end{cases} \]

where \( \alpha(w_{i-1}) = \frac{1 - \sum_{w \in \Psi(w_{i-1})} \frac{\pi^*(w_{i-1},w)}{\pi(w_{i-1})}}{\sum_{w_i \notin \Psi(w_{i-1})} P_{\text{Katz}}(w_i)} \)

- A bigram with a non-zero count is discounted using Good-Turing estimation
- The left-over probability mass from discounting for the unigram model ...
- ... is distributed over \( w_i \notin \Psi(w_{i-1}) \) proportionally to \( P_{\text{Katz}}(w_i) \)
Smoothing for Web-scale N-grams

- “Stupid backoff” [B07]
- Don’t apply any discounting and instead directly use relative counts
- Works well on very large web-scale datasets

\[
S(w_i | w_{i-k+1}^{i-1}) = \begin{cases} 
\frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^i) > 0 \\
0.4S(w_i | w_{i-k+2}^{i-1}) & \text{otherwise}
\end{cases}
\]

\[
S(w_i) = \frac{\text{count}(w_i)}{N}
\]

Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
  - Katz Backoff Smoothing
- Absolute Discounting Interpolation
- Kneser-Ney Smoothing
Recall Good-Turing estimates

<table>
<thead>
<tr>
<th>r</th>
<th>$N_r$</th>
<th>$\theta(r)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$7.47 \times 10^{10}$</td>
<td>0.0000270</td>
</tr>
<tr>
<td>1</td>
<td>$2 \times 10^{6}$</td>
<td>0.446</td>
</tr>
<tr>
<td>2</td>
<td>$4 \times 10^{5}$</td>
<td>1.26</td>
</tr>
<tr>
<td>3</td>
<td>$2 \times 10^{5}$</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>$1 \times 10^{5}$</td>
<td>3.24</td>
</tr>
<tr>
<td>5</td>
<td>$7 \times 10^{4}$</td>
<td>4.22</td>
</tr>
<tr>
<td>6</td>
<td>$5 \times 10^{4}$</td>
<td>5.19</td>
</tr>
<tr>
<td>7</td>
<td>$3.5 \times 10^{4}$</td>
<td>6.21</td>
</tr>
<tr>
<td>8</td>
<td>$2.7 \times 10^{4}$</td>
<td>7.24</td>
</tr>
<tr>
<td>9</td>
<td>$2.2 \times 10^{4}$</td>
<td>8.25</td>
</tr>
</tbody>
</table>

For $r > 0$, we observe that $\theta(r) \approx r - 0.75$ i.e. an absolute discounting

[CG91]: Church and Gale, “A comparison of enhanced Good-Turing…”, CSL, 1991
Absolute Discounting Interpolation

- Absolute discounting motivated by Good-Turing estimation
- Just subtract a constant $d$ from the non-zero counts to get the discounted count
- Also involves linear interpolation with lower-order models

$$\text{Pr}_{\text{abs}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1})\text{Pr}(w_i)$$
Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
  - Katz Backoff Smoothing
  - Absolute Discounting Interpolation
- Kneser-Ney Smoothing
Kneser-Ney discounting

\[ Pr_{KN}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda_{KN}(w_{i-1})Pr_{\text{cont}}(w_i) \]

c.f., absolute discounting

\[ Pr_{abs}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1})Pr(w_i) \]
Kneser-Ney discounting

\[
\Pr_{\text{KN}}(w_i | w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda_{\text{KN}}(w_{i-1}) \Pr_{\text{cont}}(w_i)
\]

Consider an example: “Today I cooked some yellow curry”

Suppose \(\pi(\text{yellow, curry}) = 0\). \(\Pr_{\text{abs}}[w | \text{yellow}] = \lambda(\text{yellow}) \Pr(w)\)

Now, say \(\Pr[\text{Francisco}] >> \Pr[\text{curry}]\), as San Francisco is very common in our corpus.

But Francisco is not as common a “continuation” (follows only San) as curry is (red curry, chicken curry, potato curry, ...)

Moral: Should use probability of being a continuation!

c.f., absolute discounting

\[
\Pr_{\text{abs}}(w_i | w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1}) \Pr(w_i)
\]
Kneser-Ney discounting

\[ \Pr_{\text{KN}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda_{\text{KN}}(w_{i-1})\Pr_{\text{cont}}(w_i) \]

\[ \Pr_{\text{cont}}(w_i) = \frac{|\Phi(w_i)|}{|B|} \quad \text{and} \quad \lambda_{\text{KN}}(w_{i-1}) = \frac{d}{\pi(w_{i-1})} |\Psi(w_{i-1})| \]

where

\[ \Phi(w_i) = \{w_{i-1}: \pi(w_{i-1}, w_i) > 0\} \]

\[ B = \{(w_{i-1}, w_i): \pi(w_{i-1}, w_i) > 0\} \]

c.f., absolute discounting

\[ \Pr_{\text{abs}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1})\Pr(w_i) \]
Kneser-Ney: An Alternate View

- A mix of bigram and unigram models
- A bigram $ab$ could be generated in two ways:
  - In context $a$, output $b$, or
  - In context $a$, forget context and then output $b$ (i.e., as “$a\varepsilon b$”)
- In a given set of bigrams, for each bigram $ab$, assume that $d_{ab}$ of its occurrences were produced in the second way
- Will compute probabilities for each transition under this assumption
Kneser-Ney: An Alternate View

- Assuming \( \pi(a,b) - d_{ab} \) occurrences as “\( ab \)”, and \( d_{ab} \) occurrences as “\( a\varepsilon b \)"
  - \( \text{Pr}[b|a] = [\pi(a,b) - d_{ab}] / \pi(a) \)
  - \( \text{Pr}[\varepsilon|a] = [\sum_y d_{ay}] / \pi(a) \)
  - \( \text{Pr}[b|\varepsilon] = [\sum_x d_{xb}] / [\sum_{xy} d_{xy}] \)
  - \( \text{Pr}_{KN}[b|a] = \text{Pr}[b|a] + \text{Pr}[\varepsilon|a] \cdot \text{Pr}[b|\varepsilon] \)

- Kneser-Ney: Take \( d_{xy} = d \) for all bigrams \( xy \) that do appear (assuming they all appear at least \( d \) times — kosher, e.g., if \( d = 1 \))

- Then \( \sum_y d_{ay} = d \cdot |\Psi(a)| \), \( \sum_x d_{xb} = d \cdot |\Phi(b)| \), and \( \sum_{xy} d_{xy} = d \cdot |B| \)
  where \( \Psi(a) = \{ y : \pi(a,y) > 0 \} \), \( \Phi(b) = \{ x : \pi(x,b) > 0 \} \), \( B = \{ xy : \pi(x,y) > 0 \} \)

\[
\text{Pr}_{KN}(b|a) = \frac{\max\{\pi(a,b) - d, 0\}}{\pi(a)} + \frac{d \cdot |\Psi(a)| \cdot |\Phi(b)|}{\pi(a) \cdot |B|}
\]
Ngram models as WFSAs

- With no optimizations, an Ngram over a vocabulary of \( V \) words defines a WFSA with \( V^{N-1} \) states and \( V^N \) edges.

- Example: Consider a trigram model for a two-word vocabulary, A B.
  - 4 states representing bigram histories, A_A, A_B, B_A, B_B
  - 8 arcs transitioning between these states

- Clearly not practical when \( V \) is large.
  - Resort to backoff language models
WFSA for backoff language model
Next class: Beyond Ngram LMs