Language models

- **Language models** answer the question:
  
  \[ \text{How likely is a string of English words good English?} \]

- Help with reordering
  
  \[ p_{\text{LM}}(\text{the house is small}) > p_{\text{LM}}(\text{small the is house}) \]

- Help with word choice
  
  \[ p_{\text{LM}}(\text{I am going home}) > p_{\text{LM}}(\text{I am going house}) \]

---

N-Gram Language Models

- Given: a string of English words \( W = w_1, w_2, w_3, ..., w_n \)
- Question: what is \( p(W) \)?
- Sparse data: Many good English sentences will not have been seen before
  
  \[ p(w_1, w_2, w_3, ..., w_n) = p(w_1) p(w_2|w_1) p(w_3|w_1, w_2) ... p(w_n|w_1, w_2, ..., w_{n-1}) \]

  (not much gained yet, \( p(w_n|w_1, w_2, ..., w_{n-1}) \) is equally sparse)

---

Markov Chain

- **Markov assumption:**
  
  - only previous history matters
  - limited memory: only last \( k \) words are included in history
    
    (older words less relevant)

  \( k \)-th order Markov model

- For instance 2-gram language model:
  
  \[ p(w_1, w_2, w_3, ..., w_n) \approx p(w_1) p(w_2|w_1) p(w_3|w_2) ... p(w_n|w_{n-1}) \]

  - What is conditioned on, here \( w_{i-1} \) is called the **history**
Estimating N-Gram Probabilities

- Maximum likelihood estimation
  \[ p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \]
- Collect counts over a large text corpus
- Millions to billions of words are easy to get (trillions of English words available on the web)

Example: 3-Gram

- Counts for trigrams and estimated word probabilities

<table>
<thead>
<tr>
<th>The green (total: 1748)</th>
<th>The red (total: 225)</th>
<th>The blue (total: 54)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>word</strong></td>
<td><strong>c.</strong></td>
<td><strong>prob.</strong></td>
</tr>
<tr>
<td>paper</td>
<td>801</td>
<td>0.458</td>
</tr>
<tr>
<td>group</td>
<td>640</td>
<td>0.367</td>
</tr>
<tr>
<td>light</td>
<td>110</td>
<td>0.063</td>
</tr>
<tr>
<td>party</td>
<td>27</td>
<td>0.015</td>
</tr>
<tr>
<td>ecu</td>
<td>21</td>
<td>0.012</td>
</tr>
</tbody>
</table>

- 225 trigrams in the Europarl corpus start with **the red**
- 123 of them end with **cross**
- maximum likelihood probability is \( \frac{123}{225} = 0.547 \).

How good is the LM?

- A good model assigns a text of real English \( W \) a high probability
- This can be also measured with cross entropy:
  \[ H(W) = \frac{1}{n} \log p(W^n) \]
- Or, perplexity
  \[ \text{perplexity}(W) = 2^H(W) \]
### Comparison 1–4-Gram

<table>
<thead>
<tr>
<th>word</th>
<th>unigram</th>
<th>bigram</th>
<th>trigram</th>
<th>4-gram</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6.684</td>
<td>3.197</td>
<td>3.197</td>
<td>3.197</td>
</tr>
<tr>
<td>would</td>
<td>8.342</td>
<td>2.884</td>
<td>2.791</td>
<td>2.791</td>
</tr>
<tr>
<td>like</td>
<td>9.129</td>
<td>2.026</td>
<td>1.031</td>
<td>1.290</td>
</tr>
<tr>
<td>to</td>
<td>5.081</td>
<td>0.402</td>
<td>0.144</td>
<td>0.113</td>
</tr>
<tr>
<td>commend</td>
<td>15.487</td>
<td>12.335</td>
<td>8.794</td>
<td>8.633</td>
</tr>
<tr>
<td>the</td>
<td>3.885</td>
<td>1.402</td>
<td>1.084</td>
<td>0.880</td>
</tr>
<tr>
<td>rapporteur</td>
<td>10.840</td>
<td>7.319</td>
<td>2.763</td>
<td>2.350</td>
</tr>
<tr>
<td>on</td>
<td>6.765</td>
<td>4.140</td>
<td>4.150</td>
<td>1.862</td>
</tr>
<tr>
<td>his</td>
<td>10.678</td>
<td>7.316</td>
<td>2.367</td>
<td>1.978</td>
</tr>
<tr>
<td>work</td>
<td>9.993</td>
<td>4.816</td>
<td>3.498</td>
<td>2.394</td>
</tr>
<tr>
<td>.</td>
<td>4.896</td>
<td>3.020</td>
<td>1.785</td>
<td>1.510</td>
</tr>
<tr>
<td>&lt;/s&gt;</td>
<td>4.828</td>
<td>0.005</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>average</td>
<td>8.051</td>
<td>4.072</td>
<td>2.634</td>
<td>2.251</td>
</tr>
<tr>
<td>perplexity</td>
<td>265.136</td>
<td>16.817</td>
<td>6.206</td>
<td>4.758</td>
</tr>
</tbody>
</table>

### Unseen N-Grams
- We have seen *i like to* in our corpus
- We have never seen *i like to smooth* in our corpus

\[ p(\text{smooth}|i \text{ like to}) = 0 \]

- Any sentence that includes *i like to smooth* will be assigned probability 0

### Add-One Smoothing
- For all possible bigrams, add the count of one.

\[ p = \frac{c + 1}{n + v^2} \]

- \( c \) = count of n-gram in corpus
- \( n \) = count of history
- \( v \) = vocabulary size

- But there are many more unseen n-grams than seen n-grams
- Example: Europarl 2-bigrams:
  - \( v \) = 86,700 distinct words
  - \( v^2 \) = 86,700\(^2\) = 7,516,890,000 possible bigrams
  - but only about \( n \) = 30,000,000 words (and bigrams) in corpus

### Add-\( \alpha \) Smoothing
- Add \( \alpha < 1 \) to each count

\[ p = \frac{c + \alpha}{n + \alpha v^2} \]

- What is a good value for \( \alpha \)?
- Could be optimized on held-out set
Adjusted Counts

- Previously, we estimated probabilities based on actual counts
  \[ p = \frac{c}{n} \]

- Now, we change the formula to estimate smoothed probabilities
  \[ p_{\text{smoothed}} = \frac{c + 1}{n + v^2} \]

- Another view: we adjusted the counts
  \[ p_{\text{smoothed}} = \frac{c^*}{n} \Rightarrow c^* = n p_{\text{smoothed}} = (c + 1) \frac{n}{n + v^2} \]

Good-Turing Smoothing

- Adjust actual counts \( c \) to expected counts \( c^* \) with formula
  \[ c^* = (c + 1) \frac{N_{c+1}}{N_c} \]
  - \( N_c \) number of n-grams that occur exactly \( c \) times in corpus
  - \( N_0 \) total number of unseen n-grams

Example: 2-Grams in Europarl

<table>
<thead>
<tr>
<th>Count</th>
<th>Count of counts</th>
<th>Adjusted count</th>
<th>Test count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7,514,941,065</td>
<td>0.00015</td>
<td>0.00016</td>
</tr>
<tr>
<td>1</td>
<td>1,132,844</td>
<td>0.46539</td>
<td>0.46235</td>
</tr>
<tr>
<td>2</td>
<td>263,611</td>
<td>1.40679</td>
<td>1.39946</td>
</tr>
<tr>
<td>3</td>
<td>123,615</td>
<td>2.38767</td>
<td>2.34307</td>
</tr>
<tr>
<td>4</td>
<td>73,788</td>
<td>3.33753</td>
<td>3.35202</td>
</tr>
<tr>
<td>5</td>
<td>49,254</td>
<td>4.36967</td>
<td>4.35234</td>
</tr>
<tr>
<td>6</td>
<td>35,869</td>
<td>5.32928</td>
<td>5.33762</td>
</tr>
<tr>
<td>8</td>
<td>21,693</td>
<td>7.43798</td>
<td>7.15074</td>
</tr>
<tr>
<td>10</td>
<td>14,880</td>
<td>9.31304</td>
<td>9.11927</td>
</tr>
<tr>
<td>20</td>
<td>4,546</td>
<td>19.54487</td>
<td>18.95948</td>
</tr>
</tbody>
</table>

adjusted count fairly accurate when compared against the test count

- Add-\( \alpha \) smoothing with \( \alpha = 0.00017 \)
- \( t_c \) are average counts of n-grams in test set that occurred \( c \) times in corpus
**Derivation of Good-Turing**

- A specific n-gram $\alpha$ occurs with (unknown) probability $p$ in the corpus
- Assumption: all occurrences of an n-gram $\alpha$ are independent of each other
- Number of times $\alpha$ occurs in corpus follows binomial distribution

$$p(c(\alpha) = r) = b(r; N, p) = \binom{N}{r} p^r (1-p)^{N-r}$$

**Derivation of Good-Turing (2)**

- Goal of Good-Turing smoothing: compute expected count $c^*$
- Expected count can be computed with help from binomial distribution:

$$E(c^*(\alpha)) = \sum_{r=0}^{N} r \cdot p(c(\alpha) = r) = \sum_{r=0}^{N} r \binom{N}{r} p^r (1-p)^{N-r}$$

- Note again: $p$ is unknown, we cannot actually compute this

**Derivation of Good-Turing (3)**

- Definition: expected number of n-grams that occur $r$ times: $E_N(N_r)$
- We have $s$ different n-grams in corpus
  - let us call them $\alpha_1, ..., \alpha_s$
  - each occurs with probability $p_1, ..., p_s$, respectively
- Given the previous formulae, we can compute

$$E_N(N_r) = \sum_{i=1}^{s} p(c(\alpha_i) = r) = \sum_{i=1}^{s} \binom{N}{r} p_i^r (1-p_i)^{N-r}$$

- Note again: $p_i$ is unknown, we cannot actually compute this

**Derivation of Good-Turing (4)**

- Reflection
  - we derived a formula to compute $E_N(N_r)$
  - we have $N_r$
  - for small $r$: $E_N(N_r) \simeq N_r$
- Ultimate goal compute expected counts $c^*$, given actual counts $c$

$$E(c^*(\alpha)|c(\alpha) = r)$$
Derivation of Good-Turing (5)

- For a particular n-gram $\alpha$, we know its actual count $r$.
- Any of the n-grams $\alpha_i$ may occur $r$ times.
- Probability that $\alpha$ is one specific $\alpha_i$:
  \[
  p(\alpha = \alpha_i | c(\alpha) = r) = \frac{p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}
  \]
- Expected count of this n-gram $\alpha$:
  \[
  E(c^*(\alpha) | c(\alpha) = r) = \sum_{i=1}^{s} N p_i p(\alpha = \alpha_i | c(\alpha) = r)
  \]

Derivation of Good-Turing (6)

- Combining the last two equations:
  \[
  E(c^*(\alpha) | c(\alpha) = r) = \sum_{i=1}^{s} N p_i p(c(\alpha_i) = r) / \sum_{j=1}^{s} p(c(\alpha_j) = r)
  \]
- We will now transform this equation to derive Good-Turing smoothing.

Derivation of Good-Turing (7)

- Repeat:
  \[
  E(c^*(\alpha) | c(\alpha) = r) = \sum_{i=1}^{s} N p_i p(c(\alpha_i) = r) / \sum_{j=1}^{s} p(c(\alpha_j) = r)
  \]
- Denominator is our definition of expected counts $E_N(N_r)$.

Derivation of Good-Turing (8)

- Numerator:
  \[
  \sum_{i=1}^{s} N p_i p(c(\alpha_i) = r) = \sum_{i=1}^{s} N p_i \left( \frac{N}{r} \right)^r \left( 1 - p_i \right)^{N-r}
  \]
  \[
  = \sum_{i=1}^{s} N p_i \frac{N!}{r!} p_i^{r+1} \left( 1 - p_i \right)^{N-r}
  \]
  \[
  = \left( r + 1 \right) \frac{N + 1}{N + r} E_{N+1}(N_{r+1})
  \]
  \[
  \simeq (r + 1) E_{N+1}(N_{r+1})
  \]
Derivation of Good-Turing (9)

- Using the simplifications of numerator and denominator:

\[ r^* = E(c^*(\alpha)|c(\alpha) = r) = \frac{(r + 1) E_{N+1}(N_{r+1})}{E_N(N_r)} \approx (r + 1) \frac{N_{r+1}}{N_r} \]

- QED