estimation

• Want to estimate MQ and/or MD from Q and/or D

• General problem:
  - given a string of text $S (= Q$ or $D)$, estimate its language model $M_S$
  - $S$ is commonly assumed to be an i.i.d. random sample from $M_S$
    • Independent and identically distributed

• Basic Language Models
  - maximum-likelihood estimator and the zero frequency problem
  - discounting, interpolation techniques
  - Bayesian estimation
maximum likelihood

- count relative frequencies of words in $S$
- maximum-likelihood property:
  - assigns highest possible likelihood to the observation
- unbiased estimator:
  - if we repeat estimation an infinite number of times with different starting points $S$, we will get correct probabilities (on average)
  - this is not very useful...

$$P_{ml}(w|M_S) = \frac{\#(w,S)}{|S|}$$

![Diagram showing distribution of probabilities]

- $P$ (●) = 1/3
- $P$ (●) = 1/3
- $P$ (○) = 1/3
- $P$ (□) = 0
- $P$ (●) = 0
zero-frequency problem

• Suppose some event not in our observation $S$
  – Model will assign zero probability to that event
  – And to any set of events involving the unseen event

• Happens very frequently with language

• It is incorrect to infer zero probabilities
  – especially when creating a model from short samples
Laplace smoothing

- count events in observed data
- add 1 to every count
- renormalize to obtain probabilities
- it corresponds to uniform priors

- if event counts are \((m_1, m_2, \ldots, m_k)\) with \(\sum_i m_i = N\) then
  max likelihood estimates are \(\left( \frac{m_1}{N}, \frac{m_2}{N}, \ldots, \frac{m_k}{N} \right)\)
  laplace estimates are \(\left( \frac{m_1+1}{N+k}, \frac{m_2+1}{N+k}, \ldots, \frac{m_k+1}{N+k} \right)\)
discounting methods

- Laplace smoothing
- Lindstone correction
  - add $\varepsilon$ to all count, renormalize
- absolute discounting
  - subtract $\varepsilon$, redistribute probab mass

\[
P(\bullet) = \frac{1 + \varepsilon}{3+5\varepsilon}
\]
\[
P(\circ) = \frac{1 + \varepsilon}{3+5\varepsilon}
\]
\[
P(\bullet) = \frac{0 + \varepsilon}{3+5\varepsilon}
\]
\[
P(\circ) = \frac{0 + \varepsilon}{3+5\varepsilon}
\]
discounting methods

• Held-out estimation
  - Divide data into training and held-out sections
  - In training data, count $Nr$, the number of words occurring $r$ times
  - In held-out data, count $Tr$, the number of times those words occur
  - $r^* = Tr/Nr$ is adjusted count (equals $r$ if training matches held-out)
  - Use $r^*/N$ as estimate for words that occur $r$ times

• Deleted estimation (cross-validation)
  - Same idea, but break data into $K$ sections
  - Use each in turn as held-out data, to calculate $Tr(k)$ and $Nr(k)$
  - Estimate for words that occur $r$ times is average of each

• Good-Turing estimation
  - From previous, $P(w|M) = r^* / N$ if word $w$ occurs $r$ times in sample
  - In Good-Turing, steal total probability mass from next most frequent word
  - Provides probability mass for words that occur $r=0$ times
  - Take what’s leftover from $r>0$ to ensure adds to one

\[
TPM(r + 1) = Nr_{r+1} \cdot \frac{r + 1}{N}
\]

\[
P(w_r|M) = \frac{TPM(r + 1)/Nr}{Nr_{r+1}} \cdot \frac{r + 1}{N}
\]
interpolation methods

- Problem with all discounting methods:
  - discounting treats unseen words equally (add or subtract $\epsilon$)
  - some words are more frequent than others

- Idea: use background probabilities
  - “interpolate” ML estimates with General English expectations
    (computed as relative frequency of a word in a large collection)
  - reflects expected frequency of events

ML estimate

background probability

final estimate = $\lambda$ + $(1-\lambda)$
Jelinek Mercer smoothing

- Correctly setting $\lambda$ is very important

- Start simple
  - set $\lambda$ to be a constant, independent of document, query

- Tune to optimize retrieval performance
  - optimal value of $\lambda$ varies with different databases, query sets, etc.

$$\lambda + (1-\lambda)$$
Dirichlet smoothing

- Problem with Jelinek-Mercer:
  - longer documents provide better estimates
  - could get by with less smoothing

- Make smoothing depend on sample size

- $N$ is length of sample = document length
- $\mu$ is a constant

$$\lambda \left( \frac{N}{N + \mu} \right) + (1-\lambda) \left( \frac{\mu}{N + \mu} \right)$$
Witten-Bell smoothing

- A step further:
  - condition smoothing on “redundancy” of the example
  - long, redundant example requires little smoothing
  - short, sparse example requires a lot of smoothing

- Derived by considering the proportion of new events as we walk through example
  - $N$ is total number of events $=$ document length
  - $V$ is number of unique events $=$ number of unique terms in doc

\[
\lambda \left( \frac{N}{N + V} \right) + \left( 1 - \lambda \right) \frac{V}{N + V}
\]
interpolation vs back-off

• Two possible approaches to smoothing

• Interpolation:
  - Adjust probabilities for all events, both seen and unseen

• Back-off:
  - Adjust probabilities only for unseen events
  - Leave non-zero probabilities as they are
  - Rescale everything to sum to one: rescales “seen” probabilities by a constant

• Interpolation tends to work better
  - And has a cleaner probabilistic interpretation
Two-stage smoothing

Query = “the algorithms for data mining”

<table>
<thead>
<tr>
<th></th>
<th>the</th>
<th>algorithms</th>
<th>for</th>
<th>data</th>
<th>mining</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1:</td>
<td>0.04</td>
<td>0.001</td>
<td>0.02</td>
<td>0.002</td>
<td>0.003</td>
</tr>
<tr>
<td>d2:</td>
<td>0.02</td>
<td>0.001</td>
<td>0.01</td>
<td>0.003</td>
<td>0.004</td>
</tr>
</tbody>
</table>
Two-stage smoothing

Query = “the algorithms for data mining”

<table>
<thead>
<tr>
<th></th>
<th>d1:</th>
<th>d2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(“algorithms”</td>
<td>d1)</td>
<td>0.04</td>
</tr>
<tr>
<td>p(“algorithm”</td>
<td>d2)</td>
<td>0.001</td>
</tr>
<tr>
<td>p(“data”</td>
<td>d1)</td>
<td>0.02</td>
</tr>
<tr>
<td>p(“data”</td>
<td>d2)</td>
<td>0.002</td>
</tr>
<tr>
<td>p(“mining”</td>
<td>d1)</td>
<td>0.003</td>
</tr>
<tr>
<td>p(“mining”</td>
<td>d2)</td>
<td>0.003</td>
</tr>
</tbody>
</table>

p(“algorithms”|d1) = p(“algorithm”|d2)

p(“data”|d1) < p(“data”|d2)

p(“mining”|d1) < p(“mining”|d2)

But p(q|d1) > p(q|d2)!
Two-stage smoothing

Query = “the algorithms for data mining”

<table>
<thead>
<tr>
<th></th>
<th>d1:</th>
<th>d2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(“the”)</td>
<td>0.04</td>
<td>0.02</td>
</tr>
<tr>
<td>p(“algorithm”)</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>p(“for”)</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>p(“data”)</td>
<td>0.002</td>
<td>0.003</td>
</tr>
<tr>
<td>p(“mining”)</td>
<td>0.003</td>
<td>0.004</td>
</tr>
</tbody>
</table>

\[
p(\text{“algorithms”}|d1) = p(\text{“algorithm”}|d2) \\
p(\text{“data”}|d1) < p(\text{“data”}|d2) \\
p(\text{“mining”}|d1) < p(\text{“mining”}|d2)
\]

But \( p(q|d1) > p(q|d2)! \)

We should make \( p(\text{“the”}) \) and \( p(\text{“for”}) \) less different for all docs.
Two-stage smoothing

\[ P(w|d) = \frac{c(w,d)}{|d|} \]
Two-stage smoothing

Stage-1

- Explain unseen words
- Dirichlet prior (Bayesian)

\[ P(w|d) = \frac{c(w,d) + \mu p(w|C)}{|d| + \mu} \]
Two-stage smoothing

Stage-1
- Explain unseen words
- Dirichlet prior (Bayesian)

Stage-2
- Explain noise in query
- 2-component mixture

\[
P(w|d) = (1-\lambda) \frac{c(w,d) + \mu p(w|C)}{|d|} + \mu + \lambda p(w|U)
\]