Smoothing

This dark art is why NLP is taught in the engineering school.

There are more principled smoothing methods, too. We'll look next at log-linear models, which are a good and popular general technique.

But the traditional methods are easy to implement, run fast, and will give you intuitions about what you want from a smoothing method.
Never trust a sample under 30
Never trust a sample under 30

Smooth out the bumpy histograms to look more like the truth (we hope!)
Smoothing reduces variance

Different samples of size 20 vary considerably (though on average, they give the correct bell curve!)
Smoothing reduces variance

unsmoothed estimates from different samples

estimates are correct on average:
such an estimation method is called unbiased

estimates are typically far from truth
(high variance = mean squared error)

smoothed estimates from different samples

estimates are incorrect on average:
such an estimation method is called biased

but estimates are typically close to average
(high variance = mean squared distance)
and so may tend to be closer to truth, too
Parameter Estimation

\[ p(x_1=h, x_2=o, x_3=r, x_4=s, x_5=e, x_6=s, \ldots) \]

\[ \approx p(h \mid \text{BOS, BOS}) \]
\[ \times p(o \mid \text{BOS, h}) \]
\[ \times p(r \mid h, o) \]
\[ \times p(s \mid o, r) \]
\[ \times p(e \mid r, s) \]
\[ \times p(s \mid s, e) \]
\[ \times \ldots \]

trigram model’s parameters

\[
\begin{align*}
&4470/52108 \\
&395/4470 \\
&1417/14765 \\
&1573/26412 \\
&1610/12253 \\
&2044/21250
\end{align*}
\]

values of those parameters, as naively estimated from Brown corpus.
**Terminology: Types vs. Tokens**

- **Word type** = distinct vocabulary item
  - A **dictionary** is a list of types (once each)
- **Word token** = occurrence of that type
  - A **corpus** is a list of tokens (each type has many tokens)

<table>
<thead>
<tr>
<th>Type</th>
<th>Tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>100</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>200</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>300</td>
</tr>
</tbody>
</table>

- We’ll estimate probabilities of the dictionary **types** by counting the corpus **tokens** (in context)
How to Estimate?

- $p(z \mid xy) = ?$
- Suppose our training data includes
  
  ... xya ..
  ... xyd ...
  ... xyd ...
  but never xyz
- Should we conclude
  
  $p(a \mid xy) = 1/3$?
  $p(d \mid xy) = 2/3$?
  $p(z \mid xy) = 0/3$?
- NO! Absence of xyz might just be bad luck.
Smoothing the Estimates

- Should we conclude
  \[ p(a \mid xy) = 1/3? \]
  \[ p(d \mid xy) = 2/3? \]
  \[ p(z \mid xy) = 0/3? \]

- **Discount** the positive counts somewhat
- **Reallocate** that probability to the zeroes
- Especially if the denominator is small ...
  - 1/3 probably too high, 100/300 probably about right
- Especially if numerator is small ...
  - 1/300 probably too high, 100/300 probably about right
## Add-One Smoothing

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>xya</td>
<td>1</td>
<td>1/3</td>
<td>2</td>
<td>2/29</td>
<td></td>
</tr>
<tr>
<td>xyb</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/29</td>
<td></td>
</tr>
<tr>
<td>xyc</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/29</td>
<td></td>
</tr>
<tr>
<td>xyd</td>
<td>2</td>
<td>2/3</td>
<td>3</td>
<td>3/29</td>
<td></td>
</tr>
<tr>
<td>xye</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/29</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xyz</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/29</td>
<td></td>
</tr>
<tr>
<td>Total xy</td>
<td>3</td>
<td>3/3</td>
<td>29</td>
<td>29/29</td>
<td></td>
</tr>
</tbody>
</table>
# Add-One Smoothing

300 observations instead of 3 - better data, less smoothing

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>100/300</th>
<th>101</th>
<th>101/326</th>
</tr>
</thead>
<tbody>
<tr>
<td>xya</td>
<td>100</td>
<td>100/300</td>
<td>101</td>
<td>101/326</td>
</tr>
<tr>
<td>xyb</td>
<td>0</td>
<td>0/300</td>
<td>1</td>
<td>1/326</td>
</tr>
<tr>
<td>xyc</td>
<td>0</td>
<td>0/300</td>
<td>1</td>
<td>1/326</td>
</tr>
<tr>
<td>xyd</td>
<td>200</td>
<td>200/300</td>
<td>201</td>
<td>201/326</td>
</tr>
<tr>
<td>xye</td>
<td>0</td>
<td>0/300</td>
<td>1</td>
<td>1/326</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xyz</td>
<td>0</td>
<td>0/300</td>
<td>1</td>
<td>1/326</td>
</tr>
</tbody>
</table>

| Total xy | 300 | 300/300 | 326 | 326/326 |
## Problem with Add-One Smoothing

We’ve been considering just 26 letter types ...

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>xya</td>
<td>1</td>
<td>1/3</td>
<td>2</td>
<td>2/29</td>
</tr>
<tr>
<td>xyb</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/29</td>
</tr>
<tr>
<td>xyc</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/29</td>
</tr>
<tr>
<td>xyd</td>
<td>2</td>
<td>2/3</td>
<td>3</td>
<td>3/29</td>
</tr>
<tr>
<td>xye</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/29</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xyz</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/29</td>
</tr>
<tr>
<td>Total xy</td>
<td>3</td>
<td>3/3</td>
<td>29</td>
<td>29/29</td>
</tr>
</tbody>
</table>
## Problem with Add-One Smoothing

Suppose we’re considering 20000 word types, not 26 letters

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1</td>
<td>1/3</td>
<td>2</td>
<td>2/20003</td>
</tr>
<tr>
<td>see the abbot</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/20003</td>
</tr>
<tr>
<td>see the abduct</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/20003</td>
</tr>
<tr>
<td>see the above</td>
<td>2</td>
<td>2/3</td>
<td>3</td>
<td>3/20003</td>
</tr>
<tr>
<td>see the Abram</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/20003</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>see the zygote</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/20003</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>3</td>
<td>3/3</td>
<td><strong>20003</strong></td>
<td><strong>20003/20003</strong></td>
</tr>
</tbody>
</table>
Problem with Add-One Smoothing

Suppose we’re considering 20000 word types, not 26 letters

<table>
<thead>
<tr>
<th>Event</th>
<th>Count</th>
<th>Probability</th>
<th>Add-One-Smoothed Count</th>
<th>Add-One-Smoothed Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1</td>
<td>1/3</td>
<td>2</td>
<td>2/20003</td>
</tr>
<tr>
<td>see the abbot</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/20003</td>
</tr>
<tr>
<td>see the abduct</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/20003</td>
</tr>
</tbody>
</table>

“Novel event” = 0-count event (never happened in training data).

Here: 19998 novel events, with total estimated probability 19998/20003.

So add-one smoothing thinks we are extremely likely to see novel events, rather than words we’ve seen in training data.

It thinks this only because we have a big dictionary: 20000 possible events.

Is this a good reason?
# Infinite Dictionary?

In fact, aren't there infinitely many possible word types?

<table>
<thead>
<tr>
<th></th>
<th>Category 1</th>
<th>Category 2</th>
<th>Category 3</th>
<th>Category 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the aaaaa</td>
<td>1</td>
<td>1/3</td>
<td>2</td>
<td>2/(∞ + 3)</td>
</tr>
<tr>
<td>see the aaaaab</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/(∞ + 3)</td>
</tr>
<tr>
<td>see the aaaaac</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/(∞ + 3)</td>
</tr>
<tr>
<td>see the aaaaad</td>
<td>2</td>
<td>2/3</td>
<td>3</td>
<td>3/(∞ + 3)</td>
</tr>
<tr>
<td>see the aaaaae</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/(∞ + 3)</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>see the zzzzzz</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/(∞ + 3)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>3</td>
<td>3/3</td>
<td>(∞ + 3)</td>
<td>(∞ + 3)/(∞ + 3)</td>
</tr>
</tbody>
</table>
Add-Lambda Smoothing

- A large dictionary makes novel events too probable.

- To fix: Instead of adding 1 to all counts, add $\lambda = 0.01$?
  - This gives much less probability to novel events.

- But how to pick best value for $\lambda$?
  - That is, how much should we smooth?
## Add-0.001 Smoothing

Doesn’t smooth much (estimated distribution has high variance)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>xya</td>
<td>1</td>
<td>1/3</td>
<td>1.001</td>
<td>0.331</td>
</tr>
<tr>
<td>xyb</td>
<td>0</td>
<td>0/3</td>
<td>0.001</td>
<td>0.0003</td>
</tr>
<tr>
<td>xyc</td>
<td>0</td>
<td>0/3</td>
<td>0.001</td>
<td>0.0003</td>
</tr>
<tr>
<td>xyd</td>
<td>2</td>
<td>2/3</td>
<td>2.001</td>
<td>0.661</td>
</tr>
<tr>
<td>xye</td>
<td>0</td>
<td>0/3</td>
<td>0.001</td>
<td>0.0003</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xyz</td>
<td>0</td>
<td>0/3</td>
<td>0.001</td>
<td>0.0003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total xy</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3/3</td>
<td></td>
<td>3.026</td>
<td>1</td>
</tr>
</tbody>
</table>
**Add-1000 Smoothing**

Smoothes too much (estimated distribution has high bias)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$xya$</td>
<td>1</td>
<td>1/3</td>
<td>1001</td>
<td>1/26</td>
<td></td>
</tr>
<tr>
<td>$xyb$</td>
<td>0</td>
<td>0/3</td>
<td>1000</td>
<td>1/26</td>
<td></td>
</tr>
<tr>
<td>$xc$</td>
<td>0</td>
<td>0/3</td>
<td>1000</td>
<td>1/26</td>
<td></td>
</tr>
<tr>
<td>$xd$</td>
<td>2</td>
<td>2/3</td>
<td>1002</td>
<td>1/26</td>
<td></td>
</tr>
<tr>
<td>$ye$</td>
<td>0</td>
<td>0/3</td>
<td>1000</td>
<td>1/26</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$yz$</td>
<td>0</td>
<td>0/3</td>
<td>1000</td>
<td>1/26</td>
<td></td>
</tr>
<tr>
<td><strong>Total $xy$</strong></td>
<td>3</td>
<td>3/3</td>
<td>26003</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Add-Lambda Smoothing

- A large dictionary makes novel events too probable.

- To fix: Instead of adding 1 to all counts, add $\lambda = 0.01$?
  - This gives much less probability to novel events.

- But how to pick best value for $\lambda$?
  - That is, how much should we smooth?
  - E.g., how much probability to “set aside” for novel events?
    - Depends on how likely novel events really are!
    - Which may depend on the type of text, size of training corpus, ...
  - Can we figure it out from the data?
    - We’ll look at a few methods for deciding how much to smooth.
Setting Smoothing Parameters

- How to pick best value for $\lambda$? (in add- $\lambda$ smoothing)
- Try many $\lambda$ values & report the one that gets best results?

- How to measure whether a particular $\lambda$ gets good results?
- Is it fair to measure that on test data (for setting $\lambda$)?
  - Story: Stock scam ... Also, tenure letters ...
  - Moral: Selective reporting on test data can make a method look artificially good. So it is unethical.
  - Rule: Test data cannot influence system development. No peeking! Use it only to evaluate the final system(s). Report all results on it.

**General Rule of Experimental Ethics:**
Never skew anything in your favor.
Applies to experimental design, reporting, analysis, discussion.
**Feynman's Advice:** “The first principle is that you must not fool yourself, and you are the easiest person to fool.”
Setting Smoothing Parameters

- How to pick best value for $\lambda$?
  - Try many $\lambda$ values & report the one that gets best results?

- How to fairly measure whether a $\lambda$ gets good results?
  - Hold out some “development data” for this purpose

Now use that $\lambda$ to get smoothed counts from all 100% ...

... and report results of that final model on test data.

... when we collect counts from this 80% and smooth them using add-$\lambda$ smoothing.

Pick $\lambda$ that gets best results on this 20% ...
Here we held out 20% of our training set (yellow) for development. Would like to use > 20% yellow: ☹️ 20% not enough to reliably assess $\lambda$. Would like to use > 80% blue: ☹️ Best $\lambda$ for smoothing 80% $\neq$ best $\lambda$ for smoothing 100%

Could we let the yellow and blue sets overlap? ☹️ Ethical, but foolish

---

Pick $\lambda$ that gets best results on this 20% ...

... when we collect counts from this 80% and smooth them using add-$\lambda$ smoothing.

Now use that $\lambda$ to get smoothed counts from all 100% ...

... and report results of that final model on test data.
5-fold Cross-Validation ("Jackknifing")

Would like to use > 20% yellow: 😞 20% not enough to reliably assess λ
Would like to use > 80% blue: ☻ Best λ for smoothing 80%
≠ best λ for smoothing 100%

- If 20% yellow too little: try 5 training/dev splits as below
  - Pick λ that gets best average performance

- ☑ Tests on all 100% as yellow, so we can more reliably assess λ
- 😞 Still picks a λ that’s good at smoothing the 80% size, not 100%.
  - But now we can grow that 80% without trouble ...
Cross-Validation Pseudocode

- for $\lambda$ in {0.01, 0.02, 0.03, ... 9.99}
  - for each of the 5 blue/yellow splits
    - train on the 80% blue data, using $\lambda$ to smooth the counts
    - test on the 20% yellow data, and measure performance
    - goodness of this $\lambda = \text{average performance over the 5 splits}$

  ![Dev. Dev. Dev. Dev. Dev.]

- using best $\lambda$ we found above:
  - train on 100% of the training data, using $\lambda$ to smooth the counts
  - test on the red test data, measure performance & report it

  ![Training Test]
N-fold Cross-Validation ("Leave One Out")

- To evaluate a particular $\lambda$ during dev, test on all the training data: test each sentence with smoothed model from other N-1 sentences
- 🟢 Still tests on all 100% as yellow, so we can reliably assess $\lambda$
- 🤩 Trains on nearly 100% blue data ($(N-1)/N$) to measure whether $\lambda$ is good for smoothing that much data: nearly matches true test conditions
- 🤩 Surprisingly fast: why?
  - Usually easy to retrain on blue by adding/subtracting 1 sentence’s counts

Test

(more extreme version of strategy from last slide)
Smoothing reduces variance

Remember: So does backoff (by increasing size of sample). Use both?
Use the backoff, Luke!

- Why are we treating all novel events as the same?
- \( p(zygote \mid see\ the) \) vs. \( p(baby \mid see\ the) \)
  - Unsmoothed probs: \( \frac{\text{count}(see\ the\ zygote)}{\text{count}(see\ the)} \)
  - Smoothed probs: \( \frac{\text{count}(see\ the\ zygote) + 1}{\text{count}(see\ the) + V} \)
  - What if \( \text{count}(see\ the\ zygote) = \text{count}(see\ the\ baby) = 0? \)

- baby beats zygote as a unigram
- the baby beats the zygote as a bigram
- \( \therefore \) see the baby beats see the zygote?
  (even if both have the same count, such as 0)

- Backoff introduces bias, as usual:
  - Lower-order probabilities (unigram, bigram) aren’t quite what we want
  - But we do have enuf data to estimate them & they’re better than nothing.
Early idea: Model averaging

- Jelinek-Mercer smoothing ("deleted interpolation"): Use a weighted average of backed-off naïve models:
  \[
  p_{\text{average}}(z \mid xy) = \mu_3 p(z \mid xy) + \mu_2 p(z \mid y) + \mu_1 p(z)
  \]
  where \( \mu_3 + \mu_2 + \mu_1 = 1 \) and all are \( \geq 0 \)

- The weights \( \mu \) can depend on the context \( xy \)
  - If we have "enough data" in context \( xy \), can make \( \mu_3 \) large. E.g.:
    - If count(\( xy \)) is high
    - If the entropy of \( z \) is low in the context \( xy \)
  - Learn the weights on held-out data w/ jackknifing
    - Different \( \mu_3 \) when \( xy \) is observed 1 time, 2 times, 3-5 times, ...

- We’ll see some better approaches shortly
More Ideas for Smoothing

- Cross-validation is a general-purpose wrench for tweaking any constants in any system.
  - Here, the system will train the counts from blue data, but we use yellow data to tweak how much the system smooths them ($\lambda$) and how much it backs off for different kinds of contexts ($\mu_3$ etc.)

- Is there anything more specific to try in this case?
- Remember, we’re trying to decide how much to smooth.
  - E.g., how much probability to “set aside” for novel events?
    - Depends on how likely novel events really are ...
    - Which may depend on the type of text, size of training corpus, ...
  - Can we figure this out from the data?
How likely are novel events?

<table>
<thead>
<tr>
<th>20000 types</th>
<th>300 tokens</th>
<th>300 tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>150</td>
<td>0</td>
</tr>
<tr>
<td>both</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>candy</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>donuts</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>every</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>farina</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>grapes</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>his</td>
<td>38</td>
<td>0</td>
</tr>
<tr>
<td>ice cream</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

Which zero would you expect is really rare?
## How likely are novel events?

<table>
<thead>
<tr>
<th>20000 types</th>
<th>300 tokens</th>
<th>300 tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>150</td>
<td>0</td>
</tr>
<tr>
<td>both</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>candy</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>donuts</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>every</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>farina</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>grapes</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>his</td>
<td>38</td>
<td>0</td>
</tr>
<tr>
<td>ice cream</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

Determiners: a closed class
## How likely are novel events?

<table>
<thead>
<tr>
<th>20000 types</th>
<th>300 tokens</th>
<th>300 tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>150</td>
<td>0</td>
</tr>
<tr>
<td>both</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>candy</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>donuts</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>every</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>farina</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>grapes</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>his</td>
<td>38</td>
<td>0</td>
</tr>
<tr>
<td>ice cream</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

(food) nouns: an open class
How common are novel events?

Counts from Brown Corpus (N ≈ 1 million tokens)

- $N_0 * 0$: novel words (in dictionary but never occur)
- $N_1 * 1$: singletons (occur once)
- $N_2 * 2$: doubletons (occur twice)
- $N_3 * 3$
- $N_4 * 4$
- $N_5 * 5$
- $N_6 * 6$

$N_2 = \# \text{ doubleton types}$

$N_2 * 2 = \# \text{ doubleton tokens}$

$\sum_r N_r = \text{ total } \# \text{ types } = T$ (purple bars)

$\sum_r (N_r * r) = \text{ total } \# \text{ tokens } = N$ (all bars)
How common are novel events?

- $1\times69836$: abdomen, bachelor, Caesar ...
- $1\times52108$: aberrant, backlog, cabinets ...
- $N_6\times6$: abdominal, Bach, cabana ...
- $N_5\times5$: Abbas, babel, Cabot ...
- $N_4\times4$: aback, Babbitt, cabanas ...
- $N_3\times3$: abaringe, Babatinde, cabaret ...
- $N_2\times2$: the
- $N_1\times1$: EOS
- $N_0\times0$: 0
Witten-Bell Smoothing Idea

If \( T/N \) is large, we’ve seen lots of novel types in the past, so we expect lots more.

- Imagine scanning the corpus in order.
- Each type’s first token was novel.
- So we saw \( T \) novel types (purple).

**Intuition:** When we see a new type \( w \) in training, \( ++\text{count}(w); \quad ++\text{count(novel)} \)

So \( p(\text{novel}) \) is estimated as \( T/(N+T) \), divided among \( N_0 \) specific novel types
Good-Turing Smoothing Idea

Partition the type vocabulary into classes (novel, singletons, doubletons, ...) by how often they occurred in training data.

Use observed total probability of class \( r+1 \) to estimate total probability of class \( r \).

\[
\frac{r}{N} = \frac{(N_r \cdot r/N)}{N_r} \rightarrow \frac{(N_{r+1} \cdot (r+1)/N)}{N_r}
\]
Justification of Good-Turing

- Justified by leave-one-out training! (Leave out 1 word at a time.)
- Instead of just tuning $\lambda$, we will tune
  - $p(\text{novel}) = 0.02$  
    [= frac. of yellow dev. words that were novel in blue training]
  - $p(\text{singleton}) = 0.015$  
    [= frac. of yellow dev. words that were singletons in blue training]
  - $p(\text{doubleton}) = 0.012$  
    [= frac. of yellow dev. words that were doubletons in blue training]
  - $i.e.,$
    - $p(\text{novel}) = \text{fraction of singletons in \textit{full} training}$
    - $p(\text{singleton}) = \text{fraction of doubletons in \textit{full} training}$, etc.
- Example: $c(\text{aback})=2$.  
  On the 2 folds where yellow=$\text{aback}$, aback was a singleton in blue data, so we’d be rewarded for assigning a high prob to training singletons.  Overall, we’ll get such a reward on 1.5% of the folds.
Witten-Bell vs. Good-Turing

- Estimate $p(z \mid xy)$ using just the tokens we’ve seen in context $xy$. Might be a small set ...

- Witten-Bell intuition: If those tokens were distributed over many \textit{different types}, then novel types are likely in future.
  - Formerly covered on homework 3

- Good-Turing intuition: If many of those tokens came from \textit{singleton types}, then novel types are likely in future.
  - Very nice idea (but a bit tricky in practice)
  - See the paper “Good-Turing smoothing without tears”
Good-Turing (old slides)

- Intuition: Can judge rate of novel events (in a context) by rate of singletons (in that context)

- Let $N_r = \# \text{ of word types with } r \text{ training tokens}$
  - e.g., $N_0 = \text{number of unobserved words}$
  - e.g., $N_1 = \text{number of singletons}$

- Let $N = \sum r \ N_r = \text{total } \# \text{ of training tokens}$
Good-Turing (old slides)

- Let \( N_r \) = \# of word types with \( r \) training tokens
- Let \( N = \sum r \ N_r \) = total \# of training tokens

Naïve estimate: if \( x \) has \( r \) tokens, \( p(x) = ? \)
  - Answer: \( r/N \)

Total naïve probability of all word types with \( r \) tokens?
  - Answer: \( N_r \ r / N \).

Good-Turing estimate of this total probability:
  - Defined as: \( N_{r+1} \ (r+1) / N \)
  - So proportion of novel words in test data is estimated by proportion of singletons in training data.
  - Proportion in test data of the \( N_1 \) singletons is estimated by proportion of the \( N_2 \) doubletons in training data.  Etc.

So what is Good-Turing estimate of \( p(x) \)?
Smoothing + backoff

- Basic smoothing (e.g., add-$\lambda$, Good-Turing, Witten-Bell):
  - Holds out some probability mass for novel events
  - E.g., Good-Turing gives them total mass of $N_1/N$
  - Divided up **evenly** among the novel events

- Backoff smoothing
  - Holds out same amount of probability mass for novel events
  - But divide up **unevenly** in proportion to backoff prob.
  - When defining $p(z \mid xy)$, the backoff prob for novel $z$ is $p(z \mid y)$
    - Novel events are types $z$ that were never observed after $xy$.
  - When defining $p(z \mid y)$, the backoff prob for novel $z$ is $p(z)$
    - Here novel events are types $z$ that were never observed after $y$.
    - Even if $z$ was never observed after $xy$, it may have been observed after the shorter, more frequent context $y$. Then $p(z \mid y)$ can be estimated without further backoff. If not, we back off further to $p(z)$.
  - When defining $p(z)$, do we need a backoff prob for novel $z$?
    - What are novel $z$ in this case? What could the backoff prob be? What if the vocabulary is known and finite? What if it’s potentially infinite?
Smoothing + backoff

- **Note:** The best known backoff smoothing methods:
  - modified Kneser-Ney (smart engineering)
  - Witten-Bell + one small improvement (Carpenter 2005)
  - hierarchical Pitman-Yor (clean Bayesian statistics)
  - All are about equally good.

- **Note:**
  - A given context like xy may be quite rare - perhaps we’ve only observed it a few times.
  - Then it may be hard for Good-Turing, Witten-Bell, etc. to accurately guess that context’s novel-event rate as required
  - We could try to make a better guess by aggregating xy with other contexts (all contexts? similar contexts?).
  - This is another form of backoff. By contrast, basic Good-Turing, Witten-Bell, etc. were limited to a single implicit context.
  - Log-linear models accomplish this very naturally.
There are more principled smoothing methods, too. We’ll look next at log-linear models, which are a good and popular general technique.
Smoothing as Optimization

There are more principled smoothing methods, too. We'll look next at log-linear models, which are a good and popular general technique.
Conditional Modeling

- Given a context $x$
- Which outcomes $y$ are likely in that context?
- We need a conditional distribution $p(y \mid x)$
  - A black-box function that we call on $x, y$
  - $p(\text{NextWord}=y \mid \text{PrecedingWords}=x)$
    - $y$ is a unigram
    - $x$ is an $(n-1)$-gram
  - $p(\text{Category}=y \mid \text{Text}=x)$
    - $y \in \{\text{personal email, work email, spam email}\}$
    - $x \in \Sigma^*$ (it’s a string: the text of the email)

- Remember: $p$ can be any function over $(x,y)$!
  - Provided that $p(y \mid x) \geq 0$, and $\sum_y p(y \mid x) = 1$
Linear Scoring

- We need a **conditional** distribution \( p(y | x) \)
- Convert our linear scoring function to this distribution \( p \)
  - Require that \( p(y | x) \geq 0 \), and \( \sum_y p(y | x) = 1 \); not true of score\( (x,y) \)

How well does \( y \) go with \( x \)?

Simplest option: a linear function of \( (x,y) \). But \( (x,y) \) isn’t a number.
So **describe** it by one or more numbers: “numeric features” that **you** pick.
Then just use a linear function of **those** numbers.

**Weight** of feature \( k \)
To be learned ...

\[
\text{score}(x, y) = \sum_k \theta_k f_k(x, y)
\]

Ranges over all features, \( k \),
e.g., \( k=5 \) (numbered features)
or \( k=\text{“see Det Noun”} \) (named features)

**Whether** \( (x,y) \) has feature \( k(0 \text{ or } 1) \)
**Or how many times** it fires \( (\geq 0) \)
**Or how strongly** it fires \( (\text{real #}) \)
What features should we use?

\[
\text{score}(x, y) = \sum_{k} \theta_k f_k(x, y)
\]

Ranges over all features, e.g., \(k=5\) (numbered features) or \(k=\text{"see Det Noun"}\) (named features)

\begin{itemize}
  \item \(p(\text{NextWord}=y \mid \text{PrecedingWords}=x)\)
    \begin{itemize}
      \item \(y\) is a unigram
      \item \(x\) is an \((n-1)\)-gram
    \end{itemize}
  \item \(p(\text{Category}=y \mid \text{Text}=x)\)
    \begin{itemize}
      \item \(y \in \{\text{personal email, work email, spam email}\}\)
      \item \(x \in \Sigma^*\) (it’s a string: the text of the email)
    \end{itemize}
\end{itemize}

\textbf{Weight} of feature \(k\)
To be learned ...

Whether \((x,y)\) has feature \(k\) \((0 \text{ or } 1)\)
Or how many times it fires \((\geq 0)\)
Or how strongly it fires \(\) (real #)
Log-Linear Conditional Probability
(intertepret score as a log-prob, up to a constant)

\[
p_{\theta}(y \mid x) = \frac{1}{Z(x)} \exp(\text{score}(x, y))
= \frac{1}{Z(x)} \exp \left( \sum_{k} \theta \cdot f(x, y) \right)
\]

where we choose \(Z(x)\) to ensure that \(\sum_{y} p_{\theta}(y \mid x) = 1\)

Thus, \(Z(x) = \sum \exp \text{score}(x, y')\)

Sometimes just written as \(Z\)
Training $\theta$

- $n$ training examples $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
- feature functions $f_1, f_2, \ldots$
- Want to maximize $p(\text{training data}|\theta)$

$$\left( \prod_{i=1}^{n} p_{\theta}(y_i | x_i) \right)$$

- Easier to maximize the log of that:

$$\left( \sum_{i=1}^{n} \log p_{\theta}(y_i | x_i) \right)$$

This version is “discriminative training”: to learn to predict $y$ from $x$, maximize $p(y|x)$. Whereas “joint training” learns to model $x$, too, by maximizing $p(x,y)$.

Alas, some weights $\theta_i$ may be optimal at $-\infty$ or $+\infty$. When would this happen? What’s going “wrong”?
Training $\theta$

- n training examples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- feature functions $f_1, f_2, \ldots$
- Want to maximize $p(\text{training data}|\theta) \cdot p_{\text{prior}}(\theta)$

$$
\left( \prod_{i=1}^{n} p_{\tilde{\theta}}(y_i | x_i) \right) \cdot p_{\text{prior}}(\theta)
$$

- Easier to maximize the log of that:

$$
\left( \sum_{i=1}^{n} \log p_{\tilde{\theta}}(y_i | x_i) \right) - ||\tilde{\theta}||^2
$$

Encourages weights close to 0: "L2 regularization" (other choices possible)

Corresponds to a Gaussian prior, since Gaussian bell curve is just $\exp(\text{quadratic})$. 

This version is “discriminative training”: to learn to predict $y$ from $x$, maximize $p(y|x)$.
Whereas “joint training” learns to model $x$, too, by maximizing $p(x,y)$. 
Gradient-based training

\[
\left( \sum_{i=1}^{n} \log p_{\theta}(y_i \mid x_i) \right) - ||\theta||^2
\]

- Gradually adjust \( \theta \) in a direction that increases this.

- For this, use your favorite function maximization algorithm.
  - gradient descent, conjugate gradient, variable metric, etc.
    - (Go take an optimization course: 550.{361,661,662}.)
    - (Or just download some software!)

nasty non-differentiable cost function with local minima

nice smooth and convex cost function: pick one of these
Gradient-based training

\[
\left( \sum_{i=1}^{n} \log p_{\theta}(y_i \mid x_i) \right) - \| \theta \|^2
\]

- Gradually adjust \( \theta \) in a direction that improves this

Gradient ascent to gradually increase \( f(\theta) \):

while \( (\nabla f(\theta) \neq 0) \) // not at a local max or min
\[
\theta = \theta + \varepsilon \cdot \nabla f(\theta) \quad // \text{for some small } \varepsilon > 0
\]

Remember: \( \nabla f(\theta) = (\partial f(\theta)/\partial \theta_1, \partial f(\theta)/\partial \theta_2, \ldots) \)

So update just means: \( \theta_k += \partial f(\theta)/\partial \theta_k \)

This takes a little step “uphill”
(direction of steepest increase).

This is why you took calculus. 😊
Gradient-based training

$$\left( \sum_{i=1}^{n} \log p_{\theta}(y_i \mid x_i) \right) - \| \tilde{\theta} \|^2$$

- Gradually adjust $\theta$ in a direction that improves this

- The key part of the gradient works out as …

$$\nabla_{\theta} \log p_{\theta}(y \mid x) = \nabla_{\theta} \text{score}(x, y) - \nabla_{\theta} \log Z$$

$$= \tilde{f}(x, y) - \sum_{y'} p_{\theta}(y' \mid x) \tilde{f}(x, y')$$

$$= \tilde{f}(x, y) - \mathbb{E}_{p_{\theta}}[\tilde{f}(x, y)]$$
Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don’t give you any other information.
- Question: Given message m: what is your guess for \( p(C \mid m) \)?

- Suppose I tell you that 55% of all messages are in class A.
- Question: Now what is your guess for \( p(C \mid m) \)?

- Suppose I also tell you that 10% of all messages contain *Buy* and 80% of these are in class A or C.
- Question: Now what is your guess for \( p(C \mid m) \), if m contains *Buy*?

- OUCH!
## Maximum Entropy

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy</td>
<td>.051</td>
<td>.0025</td>
<td>.029</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
</tr>
<tr>
<td>Other</td>
<td>.499</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
</tr>
</tbody>
</table>

- Column A sums to 0.55  (“55% of all messages are in class A”)
### Maximum Entropy

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy</td>
<td>.051</td>
<td>.0025</td>
<td>.029</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
</tr>
<tr>
<td>Other</td>
<td>.499</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
</tr>
</tbody>
</table>

- Column A sums to 0.55
- Row **Buy** sums to 0.1 (“10% of all messages contain Buy”)
Maximum Entropy

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy</td>
<td>.051</td>
<td>.0025</td>
<td>.029</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
</tr>
<tr>
<td>Other</td>
<td>.499</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
</tr>
</tbody>
</table>

- Column A sums to 0.55
- Row Buy sums to 0.1
- \((\text{Buy}, \ A)\) and \((\text{Buy}, \ C)\) cells sum to 0.08 ("80% of the 10%")
- Given these constraints, fill in cells “as equally as possible”: maximize the entropy (related to cross-entropy, perplexity)

\[
\text{Entropy} = - .051 \log .051 - .0025 \log .0025 - .029 \log .029 - \ldots
\]

Largest if probabilities are evenly distributed
### Maximum Entropy

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy</td>
<td>.051</td>
<td>.0025</td>
<td>.029</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
</tr>
<tr>
<td>Other</td>
<td>.499</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
</tr>
</tbody>
</table>

- Column A sums to 0.55
- Row Buy sums to 0.1
- (Buy, A) and (Buy, C) cells sum to 0.08 (“80% of the 10%”)

Given these constraints, fill in cells “as equally as possible”: maximize the entropy

- Now $p(\text{Buy, C}) = .029$ and $p(C | \text{Buy}) = .29$
- We got a compromise: $p(C | \text{Buy}) < p(A | \text{Buy}) < .55$
Maximum Entropy

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy</td>
<td>.051</td>
<td>.0025</td>
<td>.029</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
<td>.0025</td>
</tr>
<tr>
<td>Other</td>
<td>.499</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
<td>.0446</td>
</tr>
</tbody>
</table>

- Given these constraints, fill in cells “as equally as possible”: maximize the entropy
- Now \( p(\text{Buy}, C) = .029 \) and \( p(C | \text{Buy}) = .29 \)
- We got a compromise: \( p(C | \text{Buy}) < p(A | \text{Buy}) < .55 \)

- **Punchline:** This is exactly the maximum-likelihood log-linear distribution \( p(y) \) that uses 3 binary feature functions that ask: Is \( y \) in column A? Is \( y \) in row Buy? Is \( y \) one of the yellow cells? So, find it by gradient ascent.