Cross-Entropy and Perplexity

Smoothing \( n \)-gram Models
- Add-one Smoothing
- Additive Smoothing
- Good-Turing Smoothing
- Backoff Smoothing
- Event Space for \( n \)-gram Models
How good is a model

- So far we’ve seen the probability of a sentence: \( P(w_0, \ldots, w_n) \)
- What is the probability of a collection of sentences, that is what is the probability of a corpus
- Let \( T = s_0, \ldots, s_m \) be a text corpus with sentences \( s_0 \) through \( s_m \)
- What is \( P(T) \)?
  - Let us assume that we trained \( P(\cdot) \) on some training data, and \( T \) is the test data

\[ P(T) = P(s_0) \cdot P(s_1) \cdot P(s_2) \cdot \ldots \cdot P(s_m) = \prod_{i=0}^{m} P(s_i) \]

Let \( W_T \) be the length of the text \( T \) measured in words

- Then for the unigram model, \( P(T) = \prod_{w \in T} P(w) \)
- A problem: we want to compare two different models \( P_1 \) and \( P_2 \) on \( T \)
- To do this we use the per word perplexity of the model:

\[
PP_P(T) = P(T)^{-\frac{1}{W_T}} = \sqrt[\text{w}_T]{\frac{1}{P(T)}}
\]
How good is a model

- The \textit{per word} perplexity of the model is:
  \[ PP_P(T) = P(T)^{-\frac{1}{WT}} \]

- Recall that \( PP_P(T) = 2^{H_p(T)} \) where \( H_p(T) \) is the cross-entropy of \( P \) for text \( T \).

- Therefore, \( H_p(T) = \log_2 PP_P(T) = -\frac{1}{WT} \log_2 P(T) \)

- Above we use a unigram model \( P(w) \), but the same derivation holds for bigram, trigram, \ldots

How good is a model

- Lower cross entropy values and perplexity values are better
  Lower values mean that the model is \textit{better}
  Correlation with performance of the language model in various applications

- Performance of a language model is its cross-entropy or perplexity on test data (unseen data)
  corresponds to the number bits required to encode that data

- On various real life datasets, typical perplexity values yielded by \( n \)-gram models on English text range from about 50 to almost 1000 (corresponding to cross entropies from about 6 to 10 bits/word)
Cross-Entropy and Perplexity

Smoothing \( n \)-gram Models
- Add-one Smoothing
- Additive Smoothing
- Good-Turing Smoothing
- Backoff Smoothing
- Event Space for \( n \)-gram Models

Bigram Models

- In practice:

\[
P(Mork \text{ read a book}) =
\begin{align*}
P(Mork \mid < \text{start} >) & \times P(\text{read} \mid Mork) \\
P(a \mid \text{read}) & \times P(\text{book} \mid a) \\
P(< \text{stop} > \mid \text{book}) &
\end{align*}
\]

- \( P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \)

On unseen data, \( c(w_{i-1}, w_i) \) or worse \( c(w_{i-1}) \) could be zero

\[
\sum_{w_i} \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} = ?
\]
Smoothing

- **Smoothing** deals with events that have been observed zero times.
- Smoothing algorithms also tend to improve the accuracy of the model:

\[
P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}
\]

- Not just unobserved events: what about events observed once?

Add-one Smoothing

\[
P(w_i \mid w_{i-1}) = \frac{1 + c(w_{i-1}, w_i)}{V + c(w_{i-1})}
\]

- Add-one Smoothing:
- Let \( V \) be the number of words in our vocabulary
- Assign count of 1 to unseen bigrams
Add-one Smoothing

\[ P(\text{Mindy read a book}) = \]
\[ P(\text{Mindy} \mid < \text{start}>) \times P(\text{read} \mid \text{Mindy}) \times \]
\[ P(a \mid \text{read}) \times P(\text{book} \mid a) \times \]
\[ P(< \text{stop} > \mid \text{book}) \]

- Without smoothing:
  \[ P(\text{read} \mid \text{Mindy}) = \frac{c(\text{Mindy, read})}{c(\text{Mindy})} = 0 \]
- With add-one smoothing (assuming \( c(\text{Mindy}) = 1 \) but \( c(\text{Mindy, read}) = 0 \)):
  \[ P(\text{read} \mid \text{Mindy}) = \frac{1}{V + 1} \]

Additive Smoothing: (Lidstone 1920, Jeffreys 1948)

\[ P(\text{w}_i \mid \text{w}_{i-1}) = \frac{c(\text{w}_{i-1}, \text{w}_i)}{c(\text{w}_{i-1})} \]

- Add-one smoothing works horribly in practice. Seems like 1 is too large a count for unobserved events.
- Additive Smoothing:
  \[ P(\text{w}_i \mid \text{w}_{i-1}) = \frac{\delta + c(\text{w}_{i-1}, \text{w}_i)}{\delta \times V + c(\text{w}_{i-1})} \]
- \( 0 < \delta \leq 1 \)
  Still works horribly in practice, but better than add-one smoothing.
Good-Turing Smoothing: (Good, 1953)

\[ P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \]

- Imagine you’re sitting at a sushi bar with a conveyor belt.
- You see going past you 10 plates of tuna, 3 plates of unagi, 2 plates of salmon, 1 plate of shrimp, 1 plate of octopus, and 1 plate of yellowtail.
- Chance you will observe a new kind of seafood: \( \frac{3}{18} \)
- How likely are you to see another plate of salmon: should be \( < \frac{2}{18} \)

Good-Turing Smoothing

- How many types of seafood (words) were seen once? Use this to predict probabilities for unseen events.
  Let \( n_1 \) be the number of events that occurred once: \( p_0 = \frac{n_1}{N} \)
- The Good-Turing estimate states that for any \( n \)-gram that occurs \( r \) times, we should pretend that it occurs \( r^* \) times.
  \[ r^* = (r + 1) \frac{n_r + 1}{n_r} \]
- \( n_r \): number of different objects seen \( r \) times.
Good-Turing Smoothing

- 10 tuna, 3 unagi, 2 salmon, 1 shrimp, 1 octopus, 1 yellowtail
- How likely is new data? Let $n_1$ be the number of items occurring once, which is 3 in this case. $N$ is the total, which is 18.

$$p_0 = \frac{n_1}{N} = \frac{3}{18} = 0.166$$

Good-Turing Smoothing

- 10 tuna, 3 unagi, 2 salmon, 1 shrimp, 1 octopus, 1 yellowtail
- How likely is octopus? Since $c(\text{octopus}) = 1$ The GT estimate is $1^*$.

$$r^* = (r + 1) \frac{n_{r+1}}{n_r}$$

$$p_{GT} = \frac{r^*}{N}$$

- To compute $1^*$, we need $n_1 = 3$ and $n_2 = 1$

$$1^* = 2 \times \frac{1}{3} = \frac{2}{3}$$

$$p_1 = \frac{1^*}{18} = 0.037$$

- What happens when $n_{r+1} = 0$? (smoothing before smoothing)
Simple Good-Turing: linear interpolation for missing $n_{r+1}$

$$f(r) = a + b \times r$$

$$a = 2.3$$

$$b = -0.17$$

<table>
<thead>
<tr>
<th>$r$</th>
<th>$n_r = f(r)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.14</td>
</tr>
<tr>
<td>2</td>
<td>1.97</td>
</tr>
<tr>
<td>3</td>
<td>1.80</td>
</tr>
<tr>
<td>4</td>
<td>1.63</td>
</tr>
<tr>
<td>5</td>
<td>1.46</td>
</tr>
<tr>
<td>6</td>
<td>1.29</td>
</tr>
<tr>
<td>7</td>
<td>1.12</td>
</tr>
<tr>
<td>8</td>
<td>0.95</td>
</tr>
<tr>
<td>9</td>
<td>0.78</td>
</tr>
<tr>
<td>10</td>
<td>0.61</td>
</tr>
<tr>
<td>11</td>
<td>0.44</td>
</tr>
</tbody>
</table>

Comparison between Add-one and Good-Turing

<table>
<thead>
<tr>
<th>freq num with freq r</th>
<th>NS</th>
<th>Add1</th>
<th>SGT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$</td>
<td>$n_r$</td>
<td>$p_r$</td>
<td>$p_r$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0294</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0.04</td>
<td>0.0588</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.08</td>
<td>0.0882</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.12</td>
<td>0.1176</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.2</td>
<td>0.1764</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.4</td>
<td>0.3235</td>
</tr>
</tbody>
</table>

$N = (1 \times 3) + (2 \times 2) + 3 + 5 + 10 = 25$

$V = 1 + 3 + 2 + 1 + 1 + 1 = 9$

Important: we added a new word type for unseen words. Let’s call it UNK, the unknown word.

Check that: $1.0 = \sum_r n_r \times p_r$

$0.12 + (3 \times 0.03079) + (2 \times 0.06719) + 0.1045 + 0.1797 + 0.3691 = 1.0$
Comparison between Add-one and Good-Turing

<table>
<thead>
<tr>
<th>freq num with freq r</th>
<th>NS</th>
<th>Add1</th>
<th>SGT</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>n_r</td>
<td>p_r</td>
<td>p_r</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.0294</td>
<td>0.12</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0.04</td>
<td>0.0588</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.08</td>
<td>0.0882</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.12</td>
<td>0.1176</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.2</td>
<td>0.1764</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.4</td>
<td>0.3235</td>
</tr>
</tbody>
</table>

- NS = No smoothing: \( p_r = \frac{r}{N} \)
- Add1 = Add-one smoothing: \( p_r = \frac{1+r}{V+N} \)
- SGT = Simple Good-Turing: \( p_0 = \frac{n}{N}, \quad p_r = \frac{(r+1)n_{r+1}}{N} \)

with linear interpolation for missing values where \( n_{r+1} = 0 \)

(Gale and Sampson, 1995) [http://www.grsampson.net/AGtf1.html](http://www.grsampson.net/AGtf1.html)

Simple Backoff Smoothing: incorrect version

\[
P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}
\]

- In add-one or Good-Turing:
  \( P(\text{the} \mid \text{string}) = P(\text{Fonz} \mid \text{string}) \)
- If \( c(w_{i-1}, w_i) = 0 \), then use \( P(w_i) \) (back off)
- Works for trigrams: back off to bigrams and then unigrams
- Works better in practice, but probabilities get mixed up
  (unseen bigrams, for example will get higher probabilities than seen bigrams)
Backoff Smoothing: Jelinek-Mercer Smoothing

\[ P_{ML}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \]

- \[ P_{JM}(w_i \mid w_{i-1}) = \lambda P_{ML}(w_i \mid w_{i-1}) + (1 - \lambda) P_{ML}(w_i) \]
where, \( 0 \leq \lambda \leq 1 \)
- Notice that \( P_{JM}(\text{the string}) > P_{JM}(\text{Fonz string}) \) as we wanted
- Jelinek-Mercer (1980) describe an elegant form of this interpolation:

\[ P_{JM}(\text{n-gram}) = \lambda P_{ML}(\text{n-gram}) + (1 - \lambda) P_{JM}(\text{n-1gram}) \]

- What about \( P_{JM}(w_i) \)?
  For missing unigrams: \( P_{JM}(w_i) = \lambda P_{ML}(w_i) + (1 - \lambda) \frac{\delta}{V} \)

Backoff Smoothing: Many alternatives

\[ P_{JM}(\text{n-gram}) = \lambda P_{ML}(\text{n-gram}) + (1 - \lambda) P_{JM}(\text{n-1gram}) \]

- Different methods for finding the values for \( \lambda \) correspond to variety of different smoothing methods
- Katz Backoff (include Good-Turing with Backoff Smoothing)

\[ P_{katz}(y \mid x) = \begin{cases} 
\frac{c^*(xy)}{c(x)} & \text{if } c(xy) > 0 \\
\alpha(x)P_{katz}(y) & \text{otherwise}
\end{cases} \]

- where \( \alpha(x) \) is chosen to make sure that \( P_{katz}(y \mid x) \) is a proper probability

\[ \alpha(x) = 1 - \sum_y \frac{c^*(xy)}{c(x)} \]
Backoff Smoothing: Many alternatives

\[ P_{JM}(n\text{gram}) = \lambda P_{ML}(n\text{gram}) + (1 - \lambda) P_{JM}(n - 1\text{gram}) \]

- Deleted Interpolation (Jelinek, Mercer)
  compute \( \lambda \) values to minimize cross-entropy on **held-out** data which is deleted from the initial set of training data
- Improved JM smoothing, a separate \( \lambda \) for each \( w_{i-1} \):
  \[ P_{JM}(w_i \mid w_{i-1}) = \lambda(w_{i-1}) P_{ML}(w_i \mid w_{i-1}) + (1 - \lambda(w_{i-1})) P_{ML}(w_i) \]
  where \( \sum_i \lambda(w_i) = 1 \) because \( \sum_{w_i} P(w_i \mid w_{i-1}) = 1 \)

Backoff Smoothing: Many alternatives

\[ P_{JM}(n\text{gram}) = \lambda P_{ML}(n\text{gram}) + (1 - \lambda) P_{JM}(n - 1\text{gram}) \]

- Witten-Bell smoothing
  use the \( n - 1 \) gram model when the \( n \) gram model has too few unique words in the \( n \) gram context
- Absolute discounting (Ney, Essen, Kneser)
  \[ P_{abs}(y \mid x) = \begin{cases} \frac{c(xy) - D}{c(x)} & \text{if } c(xy) > 0 \\ \alpha(x) P_{abs}(y) & \text{otherwise} \end{cases} \]
  compute \( \alpha(x) \) as was done in Katz smoothing
Backoff Smoothing: Many alternatives

\[ P_{JM}(ngram) = \lambda P_{ML}(ngram) + (1 - \lambda) P_{JM}(n-1gram) \]

- Kneser-Ney smoothing
  - \[ P(\text{Francisco} \mid \text{eggplant}) > P(\text{stew} \mid \text{eggplant}) \]
    - \text{Francisco} is common, so interpolation gives \[ P(\text{Francisco} \mid \text{eggplant}) \] a high value
    - But \text{Francisco} occurs in few contexts (only after \text{San})
    - \text{stew} is common, and \text{stew} occurs in many contexts
    - Hence weight the interpolation based on number of contexts for the word using discounting
  - Modified Kneser-Ney smoothing (Chen and Goodman)
    - multiple discounts for one count, two counts and three or more counts
  - Finding \( \lambda \): use Generalized line search (Powell search) or the Expectation-Maximization algorithm
Trigram Models

- Revisiting the trigram model:
  \[ P(w_1, w_2, \ldots, w_n) = \]
  \[ P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times P(w_4 \mid w_2, w_3) \times \]
  \[ \ldots P(w_i \mid w_{i-2}, w_{i-1}) \times P(w_{n-1} \mid w_{n-2}, \ldots, w_{n-1}) \]

- Notice that the length of the sentence \( n \) is variable

The stop symbol

- Let \( \Sigma = \{a, b\} \) and the language be \( \Sigma^* \)
  so \( L = \{\epsilon, a, b, aa, bb, ab, ba \ldots\} \)

- Consider a unigram model: \( P(a) = P(b) = 0.5 \)

- \( P(a) = 0.5, P(b) = 0.5, P(aa) = 0.5^2 = 0.25, P(bb) = 0.25 \)
  and so on.

- But \( P(a) + P(b) + P(aa) + P(bb) = 1.5 \)!!

\[ \sum_{w} P(w) = 1 \]
The stop symbol

- What went wrong?
  No probability for $P(\epsilon)$
- Add a special stop symbol:

  $$P(a) = P(b) = 0.25$$

  $$P(\text{stop}) = 0.5$$

- $P(\text{stop}) = 0.5$, 
  $P(a \text{ stop}) = P(b \text{ stop}) = 0.25 \times 0.5 = 0.125$,
  $P(aa \text{ stop}) = 0.25^2 \times 0.5 = 0.03125$ (now the sum is no longer greater than one)

The stop symbol

- With this new stop symbol we can show that $\sum_w P(w) = 1$
  Notice that the probability of any sequence of length $n$ is $0.25^n \times 0.5$
  Also there are $2^n$ sequences of length $n$

\[
\sum_w P(w) = \\
\begin{align*}
\sum_{n=0}^{\infty} 2^n \times 0.25^n \times 0.5 \\
\sum_{n=0}^{\infty} 0.5^n \times 0.5 &= \sum_{n=0}^{\infty} 0.5^{n+1} \\
\sum_{n=1}^{\infty} 0.5^n &= 1
\end{align*}
\]