Announcements

- My Office hours: Today at 2pm
  - CS 6395
- Hw 2 will be out today
  - All programming
    - Analyze and plot word frequency distribution
    - Analyze and plot sentence length distribution
    - Implement interpolated, smoothed, language model
- Due: 2/23 (2 weeks from now).
  - **Counts as 2 homeworks.**

Last Time

- Language models
- N-grams
  - What happens when N is too small?
  - What happens when N is too big?
What Do N-Gram Models “Know”?  
- They (sort of) learn:
  - Rare vs. common words and patterns
  - Local syntax (an elephant, a rhinoceros)
  - Words with related meanings (ate apples)
  - Punctuation and spelling
- They have no idea about:
  - Sentence structure
  - Underlying rules of agreement/spelling/etc.
  - Meaning
  - The World

Connecting N-Gram Models with FSAs  
- Ignoring the probabilities, you can think of a bigram model as an FSA whose “alphabet” is the word vocabulary.
- How?
- Any N-gram model (for finite N) can be represented using an FSA!
- With the probabilities, an N-gram model is a weighted or probabilistic FSA.
- The biggest difference is that, where traditional FSAs focus on which strings are in the language, N-gram models (and other probabilistic models) focus on which strings are likely.

Calculating the Probability of Text  
- Assume you have a way to efficiently look up an N-gram probability $p(w_i | w_{i-N+1}, w_{i-N+2}, ..., w_{i-1})$.
- How do you calculate the probability of a sentence?
- Always remember the stop symbol's probability.
**Evaluation**

- How can we objectively judge the quality of a language model?
- Extrinsic evaluation
  - Plug into a larger system, evaluate that system before and after.
  - Can be expensive.
- Intrinsic evaluation?

**Intrinsic Evaluation - Thoughts**

- Language models are all about assigning probability.
- How about the probability of the data?
- How about the probability of new, unseen data?
- Probability is tiny and falls off fast as the data get bigger. Need to normalize somehow.

**Perplexity**

- Given some new, unseen data, how “perplexed” (surprised) is our language model?
- Lower is better.

\[
\text{perplexity}(w) = p(w) \prod_{i=1}^{\|w\|} \frac{1}{p(w_i | w_{i-1})} = \sqrt{\frac{1}{p(w)}}
\]

- For a bigram model:

\[
\text{perplexity}(w) = \sqrt{\frac{1}{\prod_{i=1}^{\|w\|} p(w_i | w_{i-1})}}
\]
Perplexity is Like A Branching Factor

- Back to the huge many-sided die.
- Imagine it’s fair, with $|\Sigma|$ different words possible.
- The number of outcomes is $|\Sigma|$, and each is equally likely.

$$\text{perplexity}(w) = \sqrt[n]{\prod_{i=1}^{n} p(w_i)} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{|\Sigma|}} = \sqrt[n]{|\Sigma|^{n}} = |\Sigma|$$

- Claim: We should be able to do better than this by making the words that actually occur more likely!

Perplexity Rules

- Only useful on unseen test data. (Otherwise, cheating is possible - how?)
- Claim: relative frequency estimation on a corpus is equivalent to minimizing perplexity on that corpus!
- Perplexity is only comparable when two models have the same vocabulary.

Types of Datasets

1. Training data: used for parameter estimation (e.g., getting N-gram counts)
2. Held-out (or “tuning”) data: used for making higher-level decisions (e.g., deciding which “N”)
3. Development (or “dev-test”) data: used for intrinsic evaluation during development
4. Test data: saved until the end, used to report “clean” numbers in a report or publication
Perplexity’s Pet Peeve

\[
\text{perplexity}(w) = \sqrt[|w|]{\frac{1}{\prod_{i=1}^{|w|} p(w_i | w_{i-1})}}
\]

\[\exists i, p(w_i | w_{i-1}) = 0 \Rightarrow \text{perplexity}(w) = +\infty\]

Some Perplexity Observations

• Training-set perplexity will improve as N gets bigger.
• Test-set perplexity will eventually stop improving as N gets bigger.
• If the probability distribution does not add up to 1, you are either hurting yourself (if the total < 1) or cheating (if the total > 1).

Smoothing Language Models
**Smoothing: Motivation**

- Relative frequencies are **sparse**, giving 0 probability to many things that are actually possible, even if we never saw them in the training data.
- Maximum likelihood estimation (MLE) is the fancy, general term for what relative frequency estimates accomplish.

\[
\hat{p}_{\text{MLE}}(w_N | \langle w_1, ..., w_{N-1} \rangle) = \frac{\text{count}(\langle w_1 w_2 ... w_{N-1} w_N \rangle)}{\sum_{v \in \Sigma} \text{count}(\langle w_1 w_2 ... w_{N-1} v \rangle)}
\]

- **Smoothing** means spreading out the counts/probability across N-grams more evenly, even to ones that were never seen.

**Smoothing**

- Sparse data:
  - \(P(w | \text{denied the})\)
  - 3 allegations
  - 2 reports
  - 1 claims
  - 1 request
  - 7 total

Smoothing flattens spiky distributions so they generalize better

- \(P(w | \text{denied the})\)
  - 2.5 allegations
  - 1.5 reports
  - 0.5 claims
  - 0.5 request
  - 2 other
  - 7 total

**Smoothing Method: Laplace**

- Add 1 to every N-gram’s count.
- Compare:

\[
\hat{p}_{\text{MLE}}(w_N | \langle w_1, ..., w_{N-1} \rangle) = \frac{\text{count}(\langle w_1 w_2 ... w_{N-1} w_N \rangle)}{\sum_{v \in \Sigma} \text{count}(\langle w_1 w_2 ... w_{N-1} v \rangle)}
\]

\[
\hat{p}_{\text{Laplace}}(w_N | \langle w_1, ..., w_{N-1} \rangle) = \frac{1 + \text{count}(\langle w_1 w_2 ... w_{N-1} w_N \rangle)}{\vert \Sigma \vert + \sum_{v \in \Sigma} \text{count}(\langle w_1 w_2 ... w_{N-1} v \rangle)}
\]
Laplace Smoothing

- Example in textbook: “want to” had its effective count reduced from 608 to 238.
- Usually, this is “too much smoothing.”
- Any other ideas?
  - Add-λ smoothing.
  - How to pick λ?

Smoothing Method: Good-Turing

- Key idea: use the count of things you’ve seen once to help estimate counts for the things you’ve seen zero times.
- Let \( N_0 \) be the number of N-grams (types) with count 0.
- In general, \( N_c \) is the number of N-grams with count \( c \).

\[
c^* = \frac{(c+1) \cdot N_{c+1}}{N_c}
\]

Good-Turing’s Discounted Counts

<table>
<thead>
<tr>
<th></th>
<th>AP Newsreel</th>
<th>Berkeley Restaurant</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c )</td>
<td>( \lambda c )</td>
<td>( \lambda c )</td>
</tr>
<tr>
<td>MLE</td>
<td>( N_c )</td>
<td>( N_c^* )</td>
</tr>
<tr>
<td>0</td>
<td>14,017,196</td>
<td>6,000,020</td>
</tr>
<tr>
<td>1</td>
<td>2,118,040</td>
<td>0.466</td>
</tr>
<tr>
<td>2</td>
<td>449,522</td>
<td>1.26</td>
</tr>
<tr>
<td>3</td>
<td>109,308</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>24,770</td>
<td>3.24</td>
</tr>
<tr>
<td>5</td>
<td>5,436</td>
<td>4.22</td>
</tr>
<tr>
<td>6</td>
<td>1,106</td>
<td>5.19</td>
</tr>
<tr>
<td>7</td>
<td>225</td>
<td>6.18</td>
</tr>
</tbody>
</table>

Smith Thesis, 2018
“Simple” Good-Turing

- What if $N_{c+1} = 0$?
  - There are no bigrams with count 35. How do we get $c'(34)$?

Smoothing Method: Linear Interpolation

- Key Idea: Use simpler models (smaller $N$) to help smooth more complex ones.

$$
\hat{p}_{\text{interp}}(w_N \mid \langle w_1 \ldots w_{N-1} \rangle) = \lambda_N \hat{p}_{\text{MLE}}(w_N \mid \langle w_1 \ldots w_{N-1} \rangle) + \lambda_{N-1} \hat{p}_{\text{MLE}}(w_N \mid \langle w_2 \ldots w_{N-1} \rangle) + \lambda_{N-2} \hat{p}_{\text{MLE}}(w_N \mid \langle w_3 \ldots w_{N-1} \rangle) + \ldots + \lambda_2 \hat{p}_{\text{MLE}}(w_N \mid \langle w_{N-1} \rangle) + \lambda_1 \hat{p}_{\text{MLE}}(w_N)
$$

Interpretation of Linear Interpolation

- (This is the story we use when generating text from an interpolated model.)

- Do:
  - Randomly choose an $N$-gram order $n$, according to a distribution given by $\lambda_1, \ldots, \lambda_{N+1}$.
  - Use the $N$-gram probability with the previous $n-1$ words as history to generate the next word $w$.
  - While $w \neq \text{stop}$. 
Linear Interpolation Variations

• Make $\lambda$ depend on the history. Some histories are well-supported and we can trust the trigram model; others aren’t and we should use the bigram or unigram model.
• Where are these $\lambda$s going to come from?

Advanced Smoothing

• Katz backoff
• instead of interpolation, only use lower order N-grams when count is low
• Kneser-Ney smoothing

A Bit of Information Theory
Probability Review

• Probability is a function of event subsets to [0, 1]
  • \( p(\emptyset) = 0 \)
  • \( A \subseteq B \) implies that \( p(A) \leq p(B) \)
  • \( A \cap B = \emptyset \) implies that \( p(A \cup B) = p(A) + p(B) \)
• Conditional probability: \( p(B \mid A) = \frac{p(A, B)}{p(A)} \)
• From last time:
  • \( p(A, B, C) = p(A) \times p(B \mid A) \times p(C \mid A, B) \)
  • Independence of \( A \) and \( B \) implies \( p(A, B) = p(A) \times p(B) \)

A Horse Race

• Suppose that there are eight horses running in an upcoming race.
• Your friend, who cares deeply about this race, is on the moon.
• It’s really expensive to send a bit to the moon!
• You want to send him the results.

<table>
<thead>
<tr>
<th></th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clinton</td>
<td>0</td>
<td>Edwards</td>
<td>000</td>
<td>001</td>
<td>Huckabee</td>
<td>100</td>
<td>101</td>
<td>010</td>
</tr>
</tbody>
</table>

\( E[\text{bits}] = 3 \)
Another View of the Code Table

Horse Race with Probability

• Suppose you agree that the probabilities over the outcome of the race are not at all even.

<table>
<thead>
<tr>
<th></th>
<th>Clinton</th>
<th>Huckabee</th>
<th>Edwards</th>
<th>McCain</th>
<th>Kucinich</th>
<th>Paul</th>
<th>Obama</th>
<th>Romney</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clinton</td>
<td>1/4</td>
<td>1/64</td>
<td>1/16</td>
<td>1/8</td>
<td>1/64</td>
<td>1/64</td>
<td>1/2</td>
<td>1/64</td>
</tr>
<tr>
<td>Edwards</td>
<td>1/16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kucinich</td>
<td>1/64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obama</td>
<td>1/2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

probabilities
codes

E[bits] = 2
Entropy

- The entropy of a distribution:
  \[ H(p) = - \sum_{x \in X} p(X = x) \log_2 p(X = x) \]
- Always \( \geq 0 \); maximal for the uniform distribution.
  \[ H(p_{\text{uniform}}) = - \sum_{x \in X} \frac{1}{|X|} \log_2 \frac{1}{|X|} = \log_2 |X| \]
- Lower bound on the average number of bits needed to encode the outcome in the best possible encoding scheme.

Entropy of a Coin

- The entropy of a coin flip:
  \[ H(p_{\text{coin}}) = - \frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1 \]

- The entropy of a biased coin:
  \[ H(p_{\text{biased coin}}) = - \frac{p(X = \text{heads})}{2} \log_2 \frac{p(X = \text{heads})}{2} - \frac{1-p(X = \text{heads})}{2} \log_2 \frac{1-p(X = \text{heads})}{2} \]

- The entropy of a fair six-sided die:
  \[ H(p_{\text{6-sided die}}) = - \sum_{x \in \{1, 2, 3, 4, 5, 6\}} \frac{1}{6} \log_2 \frac{1}{6} = \log_2 6 \]
Cross-Entropy

- Cross-entropy uses one distribution to tell us something about another distribution.

\[ H(p; q) = - \sum_{x \in X} p(X = x) \log q(X = x) \]

- The difference \( H(p; q) - H(p) \) tells us how many extra bits (on average) we waste by using \( q \) instead of \( p \).
- Extra bits make us sad (or poor); we can therefore think of this as a measure of regret.
- We want to choose \( q \) so that \( H(p; q) \) is small.

Cross-Entropy and Codes

- Key result from information theory:
  - Given true distribution \( p \) over outcomes, the best average code-length (bits per message) you can hope for is \( E_p[-\log_2 p] \), which is the entropy of \( p \).
  - There are exist methods for getting within 1 bit of the entropy, if you know \( p \! \! \).
  - But you don’t know \( p \), so you approximate it with \( q \).
  - **Cross-entropy** is an estimate of \( E_q[-\log_2 q] \), the average code-length (bits per message) when using \( q \) as a proxy for \( p \).

Perplexity, Again

- Perplexity = \( 2^H(p_{true}; q) \)
- Think of it as an "average branching factor."
  - As surprised, on average, as if choosing among \( P \) equiprobable choices.
  - Some people prefer cross-entropy (because we like bits)

  - Manning and Schütze:
    - "we reduced perplexity from 950 to 540"
    - "we reduced cross-entropy from 9.9 to 9.1 bits"
Perplexity Example

- True distribution, \( p_{true} \):
  
<table>
<thead>
<tr>
<th>Candidate</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clinton</td>
<td>1/4</td>
</tr>
<tr>
<td>Huckabee</td>
<td>1/64</td>
</tr>
<tr>
<td>Edwards</td>
<td>1/16</td>
</tr>
<tr>
<td>McCain</td>
<td>1/8</td>
</tr>
<tr>
<td>Kucinich</td>
<td>1/64</td>
</tr>
<tr>
<td>Paul</td>
<td>1/64</td>
</tr>
<tr>
<td>Obama</td>
<td>1/2</td>
</tr>
<tr>
<td>Romney</td>
<td>1/64</td>
</tr>
</tbody>
</table>

- Perplexity if we built our code/model around the true distribution? \( H(p; p) = H(p) = 2 \) bits.
- Perplexity if we built our code/model around the uniform distribution? \( H(p; \text{uniform}) = 3 \) bits.

Back to Language Models

Three Distributions

- \( p_{train} \): training sample (which horses we’ve seen before)
- \( p_{test} \): test sample (which horse will win today)
- \( q \): our model (or code)

Real goal when training: make \( H(p_{test}; q) \) small.
- We don't know \( p_{test} \). The closest we have is \( p_{train} \).
- So make \( H(p_{train}; q) \) small.
- But that overfits and can lead to infinite perplexity.
- Smoothing hopefully makes \( q \) more like \( p_{test} \).
What is a **model**? Here are some examples:

- FSA encoding of word lists
- FST encoding of surface/morphological analysis relation
- Probabilistic language models
- Unigram (bag of words)
- N-gram
- Entire history