Our goal is to *make decision*. We can assume that it requires to *estimate probabilities of events*. We will talk how to estimate this probabilities and use them in decision making.

We will observe that there are several problems with making decisions based on counts. One problem is how to count – namely, how to make sure that we get correct counts. We will address a few technical issues in this context – *smoothing questions*:

- Technical question - better estimation
- Technical/conceptual question - can we count “something else” (events that are more frequent)?

A second, related, aspect is what to count.

- We want to count more “important” events.
Roadmap

- Probabilistic Models for Sparse Structured Data
- Parameter Estimation: Ngrams
- Smoothing
  - Add-one (Laplace) smoothing
  - Good-Turing Smoothing
  - Backoff / Deleted Interpolation
At a basic level most of low level decisions in NLP are of the form: "Is it more like A than B?"

- **Illinois** bored of education {bored; board}
- *(This Art) (can N) (will MD) (rust V) {V;N}*
- *The dog bit the kid. He was taken to a veterinarian a hospital {dog, kid}*
- **Tiger** was in Washington for the PGA Tour {Person, Location, Animal}

How do we make these decisions?
In the statistical approach, the notion is that we want to figure out which of $A$ or $B$ occurs more often.

- issue of *conditioning* and how to *represent* $A$ and $B$.

- This is really just a shallow description of what is really going on.
We assume a family of models, one of which governs the generation of the data.
We observe the data.
We estimate the most likely parameters given the observed data (Fully Bayesian: posterior distribution of parameters).
We use the chosen model to estimate the likelihood of the newly observed data (Fully Bayesian: sum over the models using the posterior distribution).
If there is a need to choose between two hypotheses, we choose the one that is more likely given the model we have (Fully Bayesian: given the predictive distribution).
Parameter Estimation

- Parameter estimation is the *counting* problem.
- there is also the issue of **What to count?** or, equally, *what is the assumed model?*.
  1. What pieces of information to count?
  2. What structures to count?
What structures to count?

- you can count in terms of
  - words
  - pos tags (paper presented today)
  - relations in the sentence (to be presented later)
- Subtle issues: counting words as they appear in the text *wordform*? or counting the *lemma*? What about capitalization?
Generalization

- You want to count something which will allow you to make good decisions in the future.
- For the examples “Illinois bored of education” we can count:
  - You can count whether *bored* occurs more often than *board*.
  - You can also count whether “*bored of education*” occurs more often than “*board of education*”
  - A few things in between
- Which is better?
  - Intuition: the larger structure you count, the more informative it is.
  - However: it is harder to get good counts for large structures.
  - Need some tradeoff...
- We will talk about this on the example of Ngrams.
  - Much of this is applicable to other examples.
The Ngram model represent an attempt to count structures by counting its substructure. \((N - \text{size})\)

The process involves two stages:

1. **Decomposition stage**: involves an independence assumption on the underlying probabilistic model.
2. **Estimation stage**: involves another assumption and, typically, a hack (smoothing).
We would like to estimate the probability of a complete string \( w_1^n = (w_1 w_2 \ldots w_n) \).

We use the chain rule:

\[
P(w_1 w_2 \ldots w_n) = P(w_n|w_1 w_2 \ldots w_{n-1})P(w_1 \ldots w_{n-1}) \\
= P(w_n|w_1 w_2 \ldots w_{n-1})P(w_{n-1}|w_1 \ldots w_{n-2})P(w_1 \ldots w_{n-2}) \\
= P(w_1)P(w_2|w_1)P(w_3|w_1 w_2) \ldots P(w_{n-1}|w_{n-2})P(w_n|h_{n-1})P(w_n|h_n)
\]

How to compute the conditional probabilities?

- unigram: \( P(w_i|\text{history}) = p(w_i) \)
- bigram: \( P(w_i|\text{history}) = p(w_i|w_{i-1}) \)
- trigram \( P(w_i|\text{history}) = p(w_i|w_{i-1} w_{i-2}) \)
- quadgram
- ...

How do we estimate the model parameters?
Estimation

- We take a *training corpus*.
- For a given bigram (say), we count the number of times it occurs, and divide by the number of all bigrams with the same history (same first word):

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{\sum_w C(w_{i-1}w)}.$$  

Notice that: $\sum_w C(w_{i-1}w) = C(w_{i-1})$, which gives:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}.$$  

- In the general case of $N$-grams:

$$P(w_i|w_{i-1} \ldots w_{i-N+1}) = \frac{C(w_{i-N+1} \ldots w_{i-1}w_i)}{C(w_{i-N+1} \ldots w_{i-1})} = \frac{C(w_{i-N+1}^i)}{C(w_{i-N+1})}.$$
Estimation: maximum likelihood estimate of a Bernoulli distribution.

Assume $n$ independent trials, each with success probability $p$. The maximum likelihood estimate of $p$ is

$$\frac{k}{m}$$

where $k$ is the number of observed successes.

Issues:

- sensitivity to the corpus
- Robustness of the estimate (unreasonable when $k$ or $m$ are small) (e.g., after observing 0 heads in two trials, you unlikely to conclude that the estimate of the probability is 0;)

How related to the taxicab problem?
How to get around these difficulties?

- **Backoff models**: Count something else.
  - If you know that you cannot estimate something well, estimate something else that is "related" to what you want, provided that its estimate is more robust.
  - E.g, Bigrams instead of Trigrams.

- **Smooth**: Make some other assumptions to get more robust estimation of the counts.

- **Interpolate**: Do both of the above, and interpolate.
The sparseness of the data is the key issue
  - larger structures $\Rightarrow$ more severe problem

An extreme case is that of zero counts
  - If we have never seen a word, should we decide that its probability is 0?

We have never seen the word can as a noun. Does it mean it can never happen?

Smoothing techniques to “correct” observed counts and replace them by better, more representative counts.
Add-One Smoothing

- We want to estimate $p(a|bc)$
- Our training data includes
  
  $...bcX....bcY....bcY....$

  but never
  
  $bcZ$

- Should we conclude that
  
  $p(X|bc) = 1/3; p(Y|bc) = 2/3$?

- NO! Absence of $cbZ$ might just be bad luck.
- We want to **discount** the positive counts somewhat and **reallocate** that probability to the zeroes.
- This is especially important if total number of observations is small (denominator)
Add-One Smoothing: Example

<table>
<thead>
<tr>
<th>String</th>
<th>observed</th>
<th>estimate</th>
<th>+1-numerator</th>
<th>new estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcA</td>
<td>1</td>
<td>1/3</td>
<td>2</td>
<td>2/29</td>
</tr>
<tr>
<td>bcB</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/29</td>
</tr>
<tr>
<td>bcC</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/29</td>
</tr>
<tr>
<td>bcD</td>
<td>2</td>
<td>2/3</td>
<td>3</td>
<td>3/29</td>
</tr>
<tr>
<td>bcE</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>bcZ</td>
<td>0</td>
<td>0/3</td>
<td>1</td>
<td>1/29</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>bc</strong></td>
<td><strong>3/3</strong></td>
<td><strong>29</strong></td>
<td><strong>29/29</strong></td>
</tr>
</tbody>
</table>

- By adding 1 to the numerator we are forced to add to the denominators $V$ (probability distribution).
- $V$ - vocabulary: the total number of possible types, the smoothed estimate (*Laplace estimate*) is:

$$P^*(w_i|w_{i-1}) = \frac{C(w_{i-1} w_i) + 1}{C(w_{i-1}) + V}.$$
Add-One Smoothing: Another Example

Note: when you see more observations, say, 300 instead of 3, less smoothing is required, and this method gives it automatically.

<table>
<thead>
<tr>
<th>String</th>
<th>observed</th>
<th>estimate</th>
<th>+1-numerator</th>
<th>new estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>bca</td>
<td>100</td>
<td>1/300</td>
<td>101</td>
<td>101/326</td>
</tr>
<tr>
<td>bcb</td>
<td>0</td>
<td>0/300</td>
<td>1</td>
<td>1/326</td>
</tr>
<tr>
<td>bcc</td>
<td>0</td>
<td>0/300</td>
<td>1</td>
<td>1/326</td>
</tr>
<tr>
<td>bcd</td>
<td>200</td>
<td>2/300</td>
<td>201</td>
<td>201/326</td>
</tr>
<tr>
<td>bce</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>bcZ</td>
<td>0</td>
<td>0/300</td>
<td>1</td>
<td>1/326</td>
</tr>
<tr>
<td>Total</td>
<td>bc</td>
<td>300</td>
<td>300/300</td>
<td>326/326</td>
</tr>
</tbody>
</table>
Theoretical Motivation for the ‘Hack’

- Assume that we have a binary problem (1 or 0)
- We assume that each variable $X_i$ is generated from an unknown binomial distribution: with probability $p$: $X_i = 1$, with $1 - p$: $X_i = 0$
- We want to get posterior distribution of $p$
  - Posterior $P(p|X_i = x_i \text{ for } i = 1, \ldots, m)$ is proportional to the product of the prior $P(p)$ and likelihood $L(p) = P(X_i = x_i \text{ for } i = 1, \ldots, m|p)$

$$L(p) = \prod_{i=1}^{m} p^{x_i}(1 - p)^{1-x_i} = p^k(1 - p)^{m-k}$$

- Let the prior be uniform on $[0, 1]$
Theoretical Motivation for the ‘Hack’

- \( L(p) = \prod_{i=1}^{m} p^{x_i}(1-p)^{1-x_i} = p^k(1-p)^{m-k} \)
- Prior is uniform
- Therefore \( P(p|D) \propto L(p) \)
- Normalization:

\[
\int_0^1 p^k(1-p)^{m-k} \, dp = \frac{k!(m-k)!}{(m+1)!}
\]

- Therefore:

\[
P(p|D) = \frac{(m+1)!}{k!(m-k)!} p^{x_i}(1-p)^{1-x_i}
\]

- Beta distribution with
  - Mean: \( \frac{k}{m} \) (ML estimate)
  - Expected value: \( \frac{k+1}{m+1} \) (Laplace Estimate)
On Wednesday we talked about how to estimate probabilities with sparse data:
- Problems are most severe with rare events, but there are many rare events in language
- We started considered smoothing techniques.
- Considered Add-One (Laplace) smoothing

Today:
- Finish motivation of Laplace smoothing
- Cover Good-Turing and Back-Off smoothing
- Start talking about classifiers
Modification of Add-one Smoothing

- What if add-one smoothing is too strong?
- Maybe we could just add $\alpha$ instead?
- Can we motivate this approach? (the previous motivation does not seem to support it)
Another Theoretical Motivation

- **Binomial distribution**: the probability distribution of the number of "successes" in $n$ independent Bernoulli trials, with the same probability of "success" on each trial.
- In a **multinomial distribution**, the same but – each trial not binary (there $k$ outcomes with probabilities $p_1 \ldots p_k$)
- $X_i$ is the number of times outcome $k$ was observed over $n$ trials
- $X = (X_1, \ldots, X_k)$ follows a multinomial distribution:

$$
\Pr(X_1 = x_1 \text{ and } \ldots \text{ and } X_k = x_k | p) = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k},
$$
on $x_i \geq 0$ summing to $n$
Another Theoretical Motivation

- $X = (X_1, \ldots, X_k)$ follows a multinomial distribution:

$$\Pr(X_1 = x_1 \text{ and } \ldots \text{ and } X_k = x_k | \mathbf{p}) = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k},$$

- Dirichlet distribution is the conjugate prior for the multinomial:

$$P(\mathbf{p} | \mathbf{\beta}) \propto \prod_{j=1}^{k} p_j^{\beta_j - 1}$$

(Recall the conjugate prior definition: posterior is from the same family as prior)

- If we choose beta to be

$$(\alpha p(w_1), \alpha p(w_n), \ldots \alpha p(w_n))$$

The model is given by:

$$P(w_i | w_{i-1}) = \frac{C(w_{i-1} w_i) + \alpha p(w_i)}{\sum_{w \in V} C(w_{i-1} w) + \alpha}$$
The goal of smoothing is to hold out some probability for novel events.

Different methods differ by how do they split the amount of probability mass among novel events.

Most frequent example - Laplace smoothing: when the number of possible elements is large relative to the sample size, add-constant estimators have been found to not be robust.

Recall: this is what we expect with Zipfian distributions – the tail if heavy, there are many rare events.
Key idea, attributed to Turing, but described and refined by Good, was to **re-estimate the amount of probability mass to assign to Ngrams with low counts by looking at the number of Ngrams of higher counts**.

Specifically, judge rate of 0-count events by rate of 1-count events.

This smoothing technique was first studied in the context of cryptography: Good and Turing worked on breaking codes of the Enigma machine during WWII.
Good-Turing Smoothing

- Let $N_r = \# \text{ of ngram types with } r \text{ training tokens}$
  (Called: the frequency of frequency $r$):
  - $n_0$ - the number of bigrams of count 0;
  - $n_1$ - the number of bigrams of count 1, etc.

- Therefore, the total number of training tokens is:

$$N = \sum_r rN_r.$$

- **ML estimate:** if $x$ is an ngram that has $r$ tokens:

$$p_r = r/N.$$

- **Total ML estimate of probability** of all words with $r$ tokens:

$$N_r \frac{r}{N}.$$
**Good-Turing Smoothing**

- **ML estimate:** $p_r = r/N$.
- Good-Turing techniques suggest to replace $r$ with $r^*$, and $r^*$ such that $p_0 > 0$
- It is based on the theorem assuming that all the ngrams have (unknown) binomial distributions:

$$r^* = (r + 1) \frac{E(N_{r+1})}{E(N_r)}$$

- $E(N_r)$ is the expect number of ngrams with frequency $r$ in the corpus
- With Zipfian distribution for low $r$ we get good estimates ($N_r$ are large when $r$ is small), so we can use $N_r$ instead of $E(N_r)$
- For unseen events we have:
  - $r \approx \frac{N_1}{N_0}$, e.g. for bigrams $N_0 = V^2 - \sum_{r=1}^{r_{max}} N_r$
  - I.e., total probability of unseen words is $\frac{N_1}{N}$
Good-Turing Smoothing

- **Total ML estimate of probability** of all words with $r$ tokens:
  \[ N_r \frac{r}{N}. \]

- GT estimate: \( p_r^* = (r + 1) \frac{N_{r+1}}{N_r N} \)

- Thus, **Good-Turing estimate** of this total probability:
  \[ (r + 1) \frac{N_{r+1}}{N}, \]

- The proportion of novel ngrams in the test data is estimated by proportion of singletons in training data.
- the proportion in test data of the \( n_1 \) singletons is estimated by proportion of the \( n_2 \) doubletons in training data.
Problems of Good-Turing estimation

- What are we going to do with $N_r$ for large $r$?
- Examples of prosodic sequences (Gale, 1994)

<table>
<thead>
<tr>
<th>frequency $(r)$</th>
<th>frequency of freq $(N_r)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>120</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
</tr>
</tbody>
</table>

- Recall GT estimate: $p_r^* = (r + 1) \frac{N_{r+1}}{N_r N}$
- What is going to happen for $r = 11$?
Estimates are unreasonable for large $r$

Good (1953) suggested to replace counts $N_r$ with smoothed values: $S(N_r)$ to get: $p_r^* = (r + 1) \frac{S(N_{r+1})}{S(N_r)N}$

There are many ways to smooth it, e.g., you can consider fitting linear function for log $N_r$ with large $r$

Alternatively, you can use GT estimates only for small $r$, use ML estimates for larger $r$, and then renormalize the scores to get a valid probability distribution

Essentially, this is a family of techniques
Table: Rare bigram “Frequencies of frequencies” from 22 million AP bigrams, and Good-Turing re-estimates, after Church and Gale (1991)

<table>
<thead>
<tr>
<th>$r(ML)$</th>
<th>$n_r$</th>
<th>$r^*$ (GT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>74,671,100,000</td>
<td>0.000027</td>
</tr>
<tr>
<td>1</td>
<td>2,018,046</td>
<td>0.446</td>
</tr>
<tr>
<td>2</td>
<td>449,721</td>
<td>1.26</td>
</tr>
<tr>
<td>3</td>
<td>188,933</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>105,668</td>
<td>3.24</td>
</tr>
<tr>
<td>5</td>
<td>68,379</td>
<td>4.22</td>
</tr>
<tr>
<td>6</td>
<td>48,190</td>
<td>5.19</td>
</tr>
<tr>
<td>7</td>
<td>35,709</td>
<td>6.21</td>
</tr>
<tr>
<td>8</td>
<td>27,710</td>
<td>7.24</td>
</tr>
<tr>
<td>9</td>
<td>22,280</td>
<td>8.25</td>
</tr>
</tbody>
</table>

Note: for $r > 0 : r^* < r$
There are many modifications.

Results on convergence of GT estimates.

If interested in additional details (but not theory), see “Good-Turing Smoothing Without Tears” by Gale.
Notice that while both methods are better than allowing zero estimates, there is still a problem.

The methods do not distinguish between non-zero events. E.g, “observe that" and “observe thou” will have the same estimate.

Like previous smoothing methods we looked at, the idea is to hold out same amount of probability mass for novel events.

The key difference is that now we will divide up this amount unevenly: in proportion to backoff probability that we will now discuss.
A hack (sometimes called Jelinek-Mercer model) that has been used and is a special case of what we will discuss later is to take the unigram estimate into account, and interpolate.

\[ P_{\text{interp}}(w_i | w_{i-1}) = \lambda P_{ML}(w_i | w_{i-1}) + (1 - \lambda) P_{ML}(w_i). \]

How to estimate \( \lambda \)?
The idea in a Backoff model is to build an Ngram model based on an (N-1)gram model.

If we have zero counts for Ngrams, we Back off to (N-1)grams.

If there is a non-zero count there, we stop and rely on it.

You can also imagine models that will keep on going and will interpolate lower orders.

The recursive definition:

\[
P_{bo}(w_i|w_{i-N+1}^{i-1}) = \hat{P}(w_i|w_{i-N+1}^{i-1}) + \alpha \theta(P(w_i|w_{i-N+1}^{i-1}))P_{bo}(w_i|w_{i-N+2}^{i-1}),
\]

where

\[
\theta = \begin{cases} 
1, & \text{if } x = 0 \\
0, & \text{otherwise}
\end{cases}
\]

We will talk about \(\alpha\) and \(\hat{P}\) later.
Trigram Case

The recursive definition:

\[
P_{bo}(w_i | w_{i-N+1}^{i-1}) = \hat{P}(w_i | w_{i-N+1}^{i-1}) + \alpha \theta(P(w_i | w_{i-N+1}^{i-1})) P_{bo}(w_i | w_{i-N+2}^{i-1})
\]

where

\[
\theta = \begin{cases} 
1, & \text{if } x = 0 \\
0, & \text{otherwise}
\end{cases}
\]

Specifically, for the case of trigrams:

\[
P_{bo}(w_i | w_{i-2} w_{i-1}) = \begin{cases} 
\hat{P}(w_i | w_{i-2} w_{i-1}), & \text{if } C(w_{i-2} w_{i-1} w_i) > 0 \\
\alpha_1 \hat{P}(w_i | w_{i-1}), & \text{else if } C(w_{i-1} w_i) > 0 \\
\alpha_2 \hat{P}(w_i), & \text{otherwise}
\end{cases}
\]
Why do we need the $\alpha$ and $\hat{P}$?

The original Ngram model, computed using the frequencies, is a probability distribution. That is:

$$\sum_{i,j} P(w_n | w_i w_j) = 1$$

when the sum is over all contexts.

Therefore, when we back off to a lower order model when the count is zero we are adding extra probability mass and the total probability of a word will be greater than 1. Back off probabilities thus need to be *discounted*. 
Backoff: Selecting parameters

- $\hat{P}$ - discount the ML estimate to save some probability mass for lower order Ngrams.
- $\alpha$s - used to ensure that the probability mass from all the lower order Ngrams sums up to the amount saved.
- There are normalization parameters, and computing them is pretty messy:

$$
\hat{P}(w_i|w_{i-N+1}^{i-1}) = \frac{C^*(w_{i-N+1}^i)}{C(w_{i-N+1}^{i-1})},
$$

where $C^*(w_{i-N+1}^i) < C(w_{i-N+1}^{i-1})$.
- This will allow us to leave some probability mass for the lower order Ngrams, which will be done using the $\alpha$. 


Backoff: Selecting parameters

- The total left over probability mass for a given (N-1)gram context is:

\[
\beta(w_{i-N+1}^{i-1}) = 1 - \sum_{w_i : c(w_{i-N+1}^i) > 0} \hat{P}(w_i | w_{i-N+1}^{i-1}).
\]

- This mass need to be distributed to all the (N-1)gram. Each individual (N-1)gram will get a fraction of this mass.

\[
\alpha(w_{i-N+1}^{i-1}) = \frac{\beta(w_{i-N+1}^{i-1})}{\sum_{w_i : c(w_{i-N+1}^i) > 0} \hat{P}(w_i | w_{i-N+2}^{i-1})}.
\]

- Continue: estimate again the probabilities \( \hat{P} \) with the discount and split the remaining mass to the lower order Ngrams again.
\( \alpha \)'s are a function of the preceding string.

That is, the amount by which we discount each Ngram and how we split this mass depends on the (N-1)grams that are the substrings of the Ngram.

The estimate itself, \( C^* \), can be computed in several ways, using any smoothing techniques you want.
Deleted Interpolation

- Instead of just *backing off* to the non-zero Ngram, it is possible to take into account all Ngrams.
- That is, estimate (in the case of a trigram):

\[
\hat{P}(w_i|w_{i-2}w_{i-1}) = \lambda_1 P(w_i|w_{i-2}w_{i-1}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i),
\]

where \(\sum \lambda_i = 1\).

- In principle, the \(\lambda\) are made context dependent:
  \[
  \lambda_i = \lambda_i(w_{i-2}).
  \]

- Training the \(\lambda\)s can be done so as to maximize the likelihood of a *held-out* data, separate from the main training corpus:
  - For each set of \(\lambda\)s, one can compute the likelihood of the given data.
  - Find this set of \(\lambda\)s that gives the highest value.
  - This is conceptually a search procedure, that can be done more efficiently using a version of the EM algorithm (we will talk about this later).
Directly related to the first programming assignment

The key issue to notice when you think in a concrete way about Ngram in an application is that the model is actually quite large. There are *many* bigrams, trigrams, etc. to consider.

**Spelling problem**

- Given a sentence
  \[
  W = \{w_1 w_2 \ldots w_n\}
  \]
  where the word \(w_k\) has, say, two alternatives, \(w_k', w_k''\), we choose the spelling that maximize \(P(W)\), where \(P(W)\) is computed via an Ngram model.

**Trigrams:**

\[
P(w_1 w_2 \ldots w_n) = P(w_n|w_1 w_2 \ldots w_{n-1})P(w_1 \ldots w_{n-1}) \\
= P(w_n|w_1 w_2 \ldots w_{n-1})P(w_{n-1}|w_1 \ldots w_{n-2})P(w_1 \ldots w_{n-2}) \\
= P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)\ldots P(w_{n-1}|h_{n-1})P(w_n|h_n)
\]
However, you end up using only *those three trigrams* that contain the word:

\[
P(w' | w_{i-2}w_{i-1})P(w_{i+1} | w'w_{i-1})P(w_{i+2} | w_{i+1}w') \text{ vs. } \]

\[
P(w'' | w_{i-2}w_{i-1})P(w_{i+1} | w''w_{i-1})P(w_{i+2} | w_{i+1}w'')
\]

or in the logarithmic scale:

\[
\tilde{P}(w' | w_{i-2}w_{i-1}) + \tilde{P}(w_{i+1} | w'w_{i-1}) + \tilde{P}(w_{i+2} | w_{i+1}w')
\]

vs.

\[
\tilde{P}(w'' | w_{i-2}w_{i-1}) + \tilde{P}(w_{i+1} | w''w_{i-1}) + \tilde{P}(w_{i+2} | w_{i+1}w'')
\]

There are two directions to go from here;

1. try to generalize; e.g., why use words?
2. A second is to try to figure out a better functional form for the discriminating function.
We talked about methods how to estimate probabilities:

- **Add-One (Add-$\alpha$) smoothing:**
  - conceptually simple,
  - not very robust when many events are possible

- **Good-Turing Smoothing:**
  - can be complex, but simple modifications exist and work well
  - still does not distinguish different types of rare events

- **Backoff / Deleted Interpolation:**
  - Count statistics about different types of events and combine to estimate probabilities

- **Often we do not need probabilities, but need accurate classifiers**

- **Next: classification tasks in NLP**