Acting Under Uncertainty

Read AIMA
Chapter 13 "Quantifying Uncertainty" (13.1-13.5)
and
Chapter 15 "Probabilistic Reasoning Over Time"
(15.2-15.5)
Probability

Slides courtesy of Dan Klein and Pieter Abbeel – University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]
Today

- Probability
  - Random Variables
  - Joint and Marginal Distributions
  - Conditional Distribution
  - Product Rule, Chain Rule, Bayes’ Rule
  - Inference
  - Independence
Uncertainty

- **General situation:**
  - **Observed variables (evidence):** Agent knows certain things about the state of the world (e.g., sensor readings or symptoms)
  - **Unobserved variables:** Agent needs to reason about other aspects (e.g. where an object is or what disease is present)
  - **Model:** Agent knows something about how the known variables relate to the unknown variables

- Probabilistic reasoning gives us a framework for managing our beliefs and knowledge
Random Variables

- A random variable is some aspect of the world about which we (may) have uncertainty
  - \( R = \text{Is it raining?} \)
  - \( T = \text{Is it hot or cold?} \)
  - \( D = \text{How long will it take to drive to work?} \)
  - \( L = \text{Where is the ghost?} \)

- We denote random variables with capital letters

- Like variables in a CSP, random variables have domains
  - \( R \in \{\text{true, false}\} \) (often write as \(+r, -r\))
  - \( T \in \{\text{hot, cold}\} \)
  - \( D \in [0, \infty) \)
  - \( L \in \text{possible locations, maybe \{(0,0), (0,1), \ldots\}} \)
Probability Distributions

- Associate a probability with each value

- Temperature:

  \[ P(T) \]

<table>
<thead>
<tr>
<th>T</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>0.5</td>
</tr>
<tr>
<td>cold</td>
<td>0.5</td>
</tr>
</tbody>
</table>

- Weather:

  \[ P(W) \]

<table>
<thead>
<tr>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun</td>
<td>0.6</td>
</tr>
<tr>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>fog</td>
<td>0.3</td>
</tr>
<tr>
<td>meteor</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Probability Distributions

- Unobserved random variables have distributions

\[ P(T) \]

<table>
<thead>
<tr>
<th>T</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>0.5</td>
</tr>
<tr>
<td>cold</td>
<td>0.5</td>
</tr>
</tbody>
</table>

\[ P(W) \]

<table>
<thead>
<tr>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun</td>
<td>0.6</td>
</tr>
<tr>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>fog</td>
<td>0.3</td>
</tr>
<tr>
<td>meteor</td>
<td>0.0</td>
</tr>
</tbody>
</table>

- A distribution is a TABLE of probabilities of values

- A probability (lower case value) is a single number

\[ P(W = \text{rain}) = 0.1 \]

- Must have: \( \forall x \ P(X = x) \geq 0 \) and \( \sum_x P(X = x) = 1 \)

Shorthand notation:

\[ P(\text{hot}) = P(T = \text{hot}), \]
\[ P(\text{cold}) = P(T = \text{cold}), \]
\[ P(\text{rain}) = P(W = \text{rain}), \]

... OK if all domain entries are unique
Joint Distributions

- A joint distribution over a set of random variables: \( X_1, X_2, \ldots X_n \) specifies a real number for each assignment (or outcome):

\[
P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n)
\]

\[
P(x_1, x_2, \ldots x_n)
\]

- Must obey:

\[
P(x_1, x_2, \ldots x_n) \geq 0
\]

\[
\sum_{(x_1, x_2, \ldots x_n)} P(x_1, x_2, \ldots x_n) = 1
\]

- Size of distribution if \( n \) variables with domain sizes \( d \)?

  - For all but the smallest distributions, impractical to write out!

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Probabilistic Models

- A probabilistic model is a joint distribution over a set of random variables

- Probabilistic models:
  - (Random) variables with domains
  - Assignments are called outcomes
  - Joint distributions: say whether assignments (outcomes) are likely
  - *Normalized*: sum to 1.0
  - Ideally: only certain variables directly interact

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Events

- An event is a set $E$ of outcomes

$$P(E) = \sum_{(x_1 \ldots x_n) \in E} P(x_1 \ldots x_n)$$

- From a joint distribution, we can calculate the probability of any event
  - Probability that it’s hot AND sunny?
  - Probability that it’s hot?
  - Probability that it’s hot OR sunny?

- Typically, the events we care about are *partial assignments*, like $P(T=\text{hot})$
Quiz: Events

- $P(+x, +y)$?

- $P(+x)$?

- $P(-y \text{ OR } +x)$?

\[ P(X, Y) \]

\[
\begin{array}{ccc}
X & Y & P \\
+X & +Y & 0.2 \\
+X & -Y & 0.3 \\
-X & +Y & 0.4 \\
-X & -Y & 0.1 \\
\end{array}
\]
Marginal Distributions

- Marginal distributions are sub-tables which eliminate variables
- Marginalization (summing out): Combine collapsed rows by adding

\[
P(T, W)
\]

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

\[
P(t) = \sum_s P(t, s)
\]

\[
P(s) = \sum_t P(t, s)
\]

\[
P(X_1 = x_1) = \sum_{x_2} P(X_1 = x_1, X_2 = x_2)
\]

\[
P(T)
\]

<table>
<thead>
<tr>
<th>T</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>0.5</td>
</tr>
<tr>
<td>cold</td>
<td>0.5</td>
</tr>
</tbody>
</table>

\[
P(W)
\]

<table>
<thead>
<tr>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun</td>
<td>0.6</td>
</tr>
<tr>
<td>rain</td>
<td>0.4</td>
</tr>
</tbody>
</table>
Quiz: Marginal Distributions

### $P(X, Y)$

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+x$</td>
<td>$+y$</td>
<td>0.2</td>
</tr>
<tr>
<td>$+x$</td>
<td>$-y$</td>
<td>0.3</td>
</tr>
<tr>
<td>$-x$</td>
<td>$+y$</td>
<td>0.4</td>
</tr>
<tr>
<td>$-x$</td>
<td>$-y$</td>
<td>0.1</td>
</tr>
</tbody>
</table>

$$P(x) = \sum_y P(x, y)$$

$$P(y) = \sum_x P(x, y)$$

### $P(X)$

<table>
<thead>
<tr>
<th>$X$</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+x$</td>
<td></td>
</tr>
<tr>
<td>$-x$</td>
<td></td>
</tr>
</tbody>
</table>

### $P(Y)$

<table>
<thead>
<tr>
<th>$Y$</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+y$</td>
<td></td>
</tr>
<tr>
<td>$-y$</td>
<td></td>
</tr>
</tbody>
</table>
A simple relation between joint and conditional probabilities

- In fact, this is taken as the definition of a conditional probability

\[ P(a|b) = \frac{P(a, b)}{P(b)} \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>W</td>
<td>P</td>
</tr>
<tr>
<td>hot</td>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

\[
P(W = s | T = c) = \frac{P(W = s, T = c)}{P(T = c)} = \frac{0.2}{0.5} = 0.4
\]

\[
= P(W = s, T = c) + P(W = r, T = c)
= 0.2 + 0.3 = 0.5
\]
Quiz: Conditional Probabilities

- $P(+x \mid +y)$?

- $P(-x \mid +y)$?

- $P(-y \mid +x)$?
Conditional Distributions

- Conditional distributions are probability distributions over some variables given fixed values of others.

\[
\begin{align*}
P(W|T = \text{hot}) & \\
| W & P |
|---|---|
| sun & 0.8 |
| rain & 0.2 |
\end{align*}
\]

\[
\begin{align*}
P(W|T = \text{cold}) & \\
| W & P |
|---|---|
| sun & 0.4 |
| rain & 0.6 |
\end{align*}
\]

\[
P(T, W)
\]

\[
\begin{array}{ccc}
T & W & P \\
hot & sun & 0.4 \\
hot & rain & 0.1 \\
cold & sun & 0.2 \\
cold & rain & 0.3 \\
\end{array}
\]
Normalization Trick

\[ P(W = s|T = c) = \frac{P(W = s, T = c)}{P(T = c)} \]
\[ = \frac{P(W = s, T = c)}{P(W = s, T = c) + P(W = r, T = c)} \]
\[ = \frac{0.2}{0.2 + 0.3} = 0.4 \]

\[ P(W = r|T = c) = \frac{P(W = r, T = c)}{P(T = c)} \]
\[ = \frac{P(W = r, T = c)}{P(W = s, T = c) + P(W = r, T = c)} \]
\[ = \frac{0.3}{0.2 + 0.3} = 0.6 \]
**Normalization Trick**

\[
P(W = s|T = c) = \frac{P(W = s, T = c)}{P(T = c)} = \frac{P(W = s, T = c)}{\frac{P(W = s, T = c)}{0.2} + \frac{P(W = r, T = c)}{0.4}} = \frac{0.2}{0.2 + 0.3} = 0.4
\]

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

**SELECT** the joint probabilities matching the evidence

**NORMALIZE** the selection (make it sum to one)

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

\[
P(W = r|T = c) = \frac{P(W = r, T = c)}{P(T = c)} = \frac{P(W = r, T = c)}{\frac{P(W = r, T = c)}{0.3} + \frac{P(W = r, T = c)}{0.6}} = \frac{0.3}{0.2 + 0.3} = 0.6
\]

<table>
<thead>
<tr>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>rain</td>
<td>0.6</td>
</tr>
</tbody>
</table>
### Normalization Trick

**$P(T, W)$**

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

**SELECT the joint probabilities matching the evidence**

**$P(c, W)$**

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

**NORMALIZE the selection (make it sum to one)**

**$P(W|T = c)$**

<table>
<thead>
<tr>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>rain</td>
<td>0.6</td>
</tr>
</tbody>
</table>

- **Why does this work?** Sum of selection is $P$(evidence)! ($P(T=c)$, here)

\[
P(x_1|x_2) = \frac{P(x_1, x_2)}{P(x_2)} = \frac{P(x_1, x_2)}{\sum_{x_1} P(x_1, x_2)}
\]
Quiz: Normalization Trick

- $P(X \mid Y=-y)$?

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+x$</td>
<td>$+y$</td>
<td>0.2</td>
</tr>
<tr>
<td>$+x$</td>
<td>$-y$</td>
<td>0.3</td>
</tr>
<tr>
<td>$-x$</td>
<td>$+y$</td>
<td>0.4</td>
</tr>
<tr>
<td>$-x$</td>
<td>$-y$</td>
<td>0.1</td>
</tr>
</tbody>
</table>

**SELECT** the joint probabilities matching the evidence

**NORMALIZE** the selection (make it sum to one)
To Normalize

- (Dictionary) To bring or restore to a normal condition

- Procedure:
  - Step 1: Compute $Z = \sum$ over all entries
  - Step 2: Divide every entry by $Z$

- Example 1

<table>
<thead>
<tr>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Normalize

$Z = 0.5$

<table>
<thead>
<tr>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>rain</td>
<td>0.6</td>
</tr>
</tbody>
</table>

- Example 2

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sun</td>
<td>20</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>5</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>10</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>15</td>
</tr>
</tbody>
</table>

Normalize

$Z = 50$

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Probabilistic Inference

- Probabilistic inference: compute a desired probability from other known probabilities (e.g. conditional from joint)

- We generally compute conditional probabilities
  - \( P(\text{on time} \mid \text{no reported accidents}) = 0.90 \)
  - These represent the agent’s beliefs given the evidence

- Probabilities change with new evidence:
  - \( P(\text{on time} \mid \text{no accidents, 5 a.m.}) = 0.95 \)
  - \( P(\text{on time} \mid \text{no accidents, 5 a.m., raining}) = 0.80 \)
  - Observing new evidence causes beliefs to be updated
Inference by Enumeration

- **General case:**
  - Evidence variables: \( E_1 \ldots E_k = e_1 \ldots e_k \)
  - Query* variable: \( Q \)
  - Hidden variables: \( H_1 \ldots H_r \)

\[
\{ X_1, X_2, \ldots X_n \}
\]

- **We want:**

\[
P(Q | e_1 \ldots e_k)
\]

* Works fine with multiple query variables, too

- **Step 1:** Select the entries consistent with the evidence

- **Step 2:** Sum out \( H \) to get joint of Query and evidence

\[
P(Q, e_1 \ldots e_k) = \sum_{h_1 \ldots h_r} P(Q, h_1 \ldots h_r, e_1 \ldots e_k)
\]

- **Step 3:** Normalize

\[
Z = \sum_q P(Q, e_1 \ldots e_k)
\]

\[
P(Q | e_1 \ldots e_k) = \frac{1}{Z} P(Q, e_1 \ldots e_k)
\]
Inference by Enumeration

- \( P(W) \)?

- \( P(W \mid \text{winter}) \)?

- \( P(W \mid \text{winter}, \text{hot}) \)?

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>summer</td>
<td>hot</td>
<td>sun</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td>summer</td>
<td>hot</td>
<td>rain</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>summer</td>
<td>cold</td>
<td>sun</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>summer</td>
<td>cold</td>
<td>rain</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>winter</td>
<td>hot</td>
<td>sun</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>winter</td>
<td>hot</td>
<td>rain</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>winter</td>
<td>cold</td>
<td>sun</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>winter</td>
<td>cold</td>
<td>rain</td>
<td>0.20</td>
<td></td>
</tr>
</tbody>
</table>
Obvious problems:

- Worst-case time complexity $O(d^n)$
- Space complexity $O(d^n)$ to store the joint distribution
The Product Rule

- Sometimes have conditional distributions but want the joint

\[ P(y)P(x|y) = P(x, y) \quad \iff \quad P(x|y) = \frac{P(x, y)}{P(y)} \]
The Product Rule

\[ P(y)P(x|y) = P(x, y) \]

Example:

| \( P(W) \) | \( P(D|W) \) | \( P(D, W) \) |
|-----|-----|-----|
| R   | P   |     |
| sun | 0.8 |     |
| rain| 0.2 |     |

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>wet</td>
<td>sun</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>dry</td>
<td>sun</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>wet</td>
<td>rain</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>dry</td>
<td>rain</td>
<td>0.3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>wet</td>
<td>sun</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dry</td>
<td>sun</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wet</td>
<td>rain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dry</td>
<td>rain</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More generally, can always write any joint distribution as an incremental product of conditional distributions

\[ P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \]

\[ P(x_1, x_2, \ldots x_n) = \prod_{i} P(x_i|x_1 \ldots x_{i-1}) \]

Why is this always true?
Bayes Rule
Bayes’ Rule

- Two ways to factor a joint distribution over two variables:

\[ P(x, y) = P(x | y)P(y) = P(y | x)P(x) \]

- Dividing, we get:

\[ P(x | y) = \frac{P(y | x)}{P(y)}P(x) \]

- Why is this at all helpful?
  - Lets us build one conditional from its reverse
  - Often one conditional is tricky but the other one is simple
  - Foundation of many systems we’ll see later (e.g. ASR, MT)

- In the running for most important AI equation!
Inference with Bayes’ Rule

- **Example: Diagnostic probability from causal probability:**

\[
P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})P(\text{cause})}{P(\text{effect})}
\]

- **Example:**
  - M: meningitis, S: stiff neck

\[
\begin{align*}
P(+m) &= 0.0001 \\
P(+s|+m) &= 0.8 \\
P(+s|-m) &= 0.01
\end{align*}
\]

\[
\begin{align*}
P(+m|+s) &= \frac{P(+s|m)P(+m)}{P(+s)} \\
&= \frac{P(+s|m)P(+m)}{P(+s|m)P(+m) + P(+s|-m)P(-m)} \\
&= \frac{0.8 \times 0.0001}{0.8 \times 0.0001 + 0.01 \times 0.999}
\end{align*}
\]

- Note: posterior probability of meningitis still very small
- Note: you should still get stiff necks checked out! Why?
 Quiz: Bayes’ Rule

- **Given:**
  
  \[
P(W)\]
  
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>sun</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>rain</td>
<td>0.2</td>
<td></td>
</tr>
</tbody>
</table>

  \[
P(D|W)\]
  
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>W</td>
<td>P</td>
</tr>
<tr>
<td>wet</td>
<td>sun</td>
<td>0.1</td>
</tr>
<tr>
<td>dry</td>
<td>sun</td>
<td>0.9</td>
</tr>
<tr>
<td>wet</td>
<td>rain</td>
<td>0.7</td>
</tr>
<tr>
<td>dry</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

- What is \(P(W \mid \text{dry})\)?
Language Modeling with N-Grams

Speech and Language Processing (3rd Edition draft)

Chapter 3 “Language Modeling with N-Grams”
iOS Autocomplete Song | Song A Day #2110

https://www.youtube.com/watch?v=M8MJFrdfGe0
Probabilistic Language Models

• Today’s goal: assign a probability to a sentence

• Autocomplete for texting

• Machine Translation

• Spelling Correction

• Speech Recognition

• Other NLG tasks: summarization, question-answering, dialog systems
Probabilistic Language Modeling

• Goal: compute the probability of a sentence or sequence of words

• Related task: probability of an upcoming word

• A model that computes either of these

• Better: the grammar  But language model or LM is standard
Probabilistic Language Modeling

• Goal: compute the probability of a sentence or sequence of words
  \[ P(W) = P(w_1, w_2, w_3, w_4, w_5 \ldots w_n) \]

• Related task: probability of an upcoming word
  \[ P(w_5 | w_1, w_2, w_3, w_4) \]

• A model that computes either of these
  \[ P(W) \quad \text{or} \quad P(w_n | w_1, w_2 \ldots w_{n-1}) \]
  is called a language model.

• Better: the grammar  But language model or LM is standard
How to compute $P(W)$

• How to compute this joint probability:
  
  • $P(\text{the, underdog, Philadelphia, Eagles, won})$

• Intuition: let’s rely on the Chain Rule of Probability
The Chain Rule
The Chain Rule

• Recall the definition of conditional probabilities
\[ p(B \mid A) = \frac{P(A, B)}{P(A)} \quad \text{Rewriting:} \quad P(A, B) = P(A)P(B \mid A) \]

• More variables:
\[ P(A, B, C, D) = P(A)P(B \mid A)P(C \mid A, B)P(D \mid A, B, C) \]

• The Chain Rule in General
\[ P(x_1, x_2, x_3, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2) \ldots P(x_n \mid x_1, \ldots, x_{n-1}) \]
The Chain Rule applied to compute joint probability of words in sentence
The Chain Rule applied to compute joint probability of words in sentence

\[
P(w_1w_2\ldots w_n) = \prod_{i} P(w_i \mid w_1w_2\ldots w_{i-1})
\]

\[
P(“\text{the underdog Philadelphia Eagles won}”) = \]
\[
P(\text{the}) \times P(\text{underdog} \mid \text{the}) \times P(\text{Philadelphia} \mid \text{the underdog}) \times P(\text{Eagles} \mid \text{the underdog Philadelphia}) \times P(\text{won} \mid \text{the underdog Philadelphia Eagles})
\]
How to estimate these probabilities

• Could we just count and divide?
How to estimate these probabilities

• Could we just count and divide? Maximum likelihood estimation (MLE)

\[ P(\text{won} | \text{the underdog team}) = \frac{\text{Count}(\text{the underdog team won})}{\text{Count}(\text{the underdog team})} \]

• Why doesn’t this work?
Simplifying Assumption = Markov Assumption
Simplifying Assumption = Markov Assumption

- \( P(\text{won} | \text{the underdog team}) \approx P(\text{won} | \text{team}) \)
- Only depends on the previous \( k \) words, not the whole context
- \( \approx P(\text{won} | \text{underdog team}) \)
- \( \approx P(w_i | w_{i-2} w_{i-1}) \)
- \( P(w_1 w_2 w_3 w_4... w_n) \approx \prod_{i=1}^{n} P(w_i | w_{i-k} ... w_{i-1}) \)
- \( K \) is the number of context words that we take into account
How much history should we use?

<table>
<thead>
<tr>
<th></th>
<th>History</th>
</tr>
</thead>
<tbody>
<tr>
<td>unigram</td>
<td>no history</td>
</tr>
<tr>
<td>bigram</td>
<td>1 word as history</td>
</tr>
<tr>
<td>trigram</td>
<td>2 words as history</td>
</tr>
<tr>
<td>4-gram</td>
<td>3 words as history</td>
</tr>
<tr>
<td>Year</td>
<td>Event</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>1913</td>
<td>Andrei Markov counts 20k letters in <em>Eugene Onegin</em></td>
</tr>
<tr>
<td>1948</td>
<td>Claude Shannon uses n-grams to approximate English</td>
</tr>
<tr>
<td>1956</td>
<td>Noam Chomsky decries finite-state Markov Models</td>
</tr>
<tr>
<td>1980s</td>
<td>Fred Jelinek at IBM TJ Watson uses n-grams for ASR, think about 2 other ideas for models: (1) MT, (2) stock market prediction</td>
</tr>
<tr>
<td>1993</td>
<td>Jelinek at team develops statistical machine translation</td>
</tr>
</tbody>
</table>
|       | \[
|       | \quad \text{argmax}_e p(e|f) = p(e) p(f|e) \]                       |
|       | Jelinek left IBM to found CLSP at JHU                                |
|       | Peter Brown and Robert Mercer move to Renaissance Technology         |
Simplest case: Unigram model

\[ P(w_1w_2\ldots w_n) \approx \prod_{i} P(w_i) \]

Some automatically generated sentences from a unigram model

fifth an of futures the an incorporated a a the inflation most dollars quarter in is mass

thrift did eighty said hard 'm july bullish

that or limited the
Bigram model

- Condition on the previous word:

\[ P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1}) \]

texaco rose one in this issue is pursuing growth in a boiler house said mr. gurria mexico 's motion control proposal without permission from five hundred fifty five yen outside new car parking lot of the agreement reached this would be a record november
N-gram models

• We can extend to trigrams, 4-grams, 5-grams
• In general this is an insufficient model of language
  • because language has **long-distance dependencies**:
    “The computer(s) which I had just put into the machine room on the fifth floor is (are) crashing.”

• But we can often get away with N-gram models
Language Modeling

Estimating N-gram Probabilities
Estimating bigram probabilities

- The Maximum Likelihood Estimate

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

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

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

\[
\begin{align*}
P(I \mid <s>) &= \frac{2}{3} = .67 \\
P(Sam \mid <s>) &= \frac{1}{3} = .33 \\
P(am \mid I) &= \frac{2}{3} = .67 \\
P(\langle/s\rangle \mid Sam) &= \frac{1}{2} = 0.5 \\
P(Sam \mid am) &= \frac{1}{2} = .5 \\
P(do \mid I) &= \frac{1}{3} = .33
\end{align*}
\]
Problems for MLE

• Zeros

<table>
<thead>
<tr>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>denied the allegations</td>
<td>denied the memo</td>
</tr>
<tr>
<td>denied the reports</td>
<td></td>
</tr>
<tr>
<td>denied the claims</td>
<td></td>
</tr>
<tr>
<td>denied the requests</td>
<td></td>
</tr>
</tbody>
</table>

• $P(\text{memo} | \text{denied the}) = 0$

• And we also assign 0 probability to all sentences containing it!
Problems for MLE

• Out of vocabulary items (OOV)
• \texttt{<unk>} to deal with OOVs
• Fixed lexicon $L$ of size $V$
• Normalize training data by replacing any word not in $L$ with \texttt{<unk>}

• Avoid zeros with smoothing
Practical Issues

• We do everything in log space
  • Avoid underflow
  • (also adding is faster than multiplying)

\[
\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4
\]
Language Modeling Toolkits

• SRILM

• KenLM
  • [https://kheafield.com/code/kenlm/](https://kheafield.com/code/kenlm/)
All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, ...

That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.
Google N-Gram Release

• serve as the incoming 92
• serve as the incubator 99
• serve as the independent 794
• serve as the index 223
• serve as the indication 72
• serve as the indicator 120
• serve as the indicators 45
• serve as the indispensible 111
• serve as the indispensable 40
• serve as the individual 234

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html
Google Book N-grams

Language Modeling

Evaluation and Perplexity
Evaluation: How good is our model?

• Does our language model prefer good sentences to bad ones?
  • Assign higher probability to “real” or “frequently observed” sentences
    • Than “ungrammatical” or “rarely observed” sentences?

• We train parameters of our model on a training set.

• We test the model’s performance on data we haven’t seen.
  • A test set is an unseen dataset that is different from our training set, totally unused.
  • An evaluation metric tells us how well our model does on the test set.
Training on the test set

• We can’t allow test sentences into the training set
• We will assign it an artificially high probability when we set it in the test set
• “Training on the test set”
• Bad science!
• And violates the honor code
Extrinsic evaluation of N-gram models
Difficulty of extrinsic (task-based) evaluation of N-gram models

• Extrinsic evaluation
  • Time-consuming; can take days or weeks

• So
  • Sometimes use intrinsic evaluation: perplexity
  • Bad approximation
    • unless the test data looks just like the training data
    • So generally only useful in pilot experiments
  • But is helpful to think about.
Intuition of Perplexity

• The Shannon Game:
  • How well can we predict the next word?

  I always order pizza with cheese and ____
Intuition of Perplexity

• The Shannon Game:
  • How well can we predict the next word?
    
    I always order pizza with cheese and ____
    
    The 33rd President of the US was ____
    
    I saw a ____

  • Unigrams are terrible at this game. (Why?)

• A better model of a text
  • is one which assigns a higher probability to the word that actually occurs

{ mushrooms 0.1
  pepperoni 0.1
  anchovies 0.01
  ....
  fried rice 0.0001
  ....
  and 1e-100 }
Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of words

Minimizing perplexity is the same as maximizing probability
Perplexity

The best language model is one that best predicts an unseen test set
• Gives the highest $P($sentence$)$
Perplexity is the inverse probability of the test set, normalized by the number of words

$$PP(W) = \frac{1}{N} \sqrt[1/N]{P(w_1 w_2 \ldots w_N)}$$

Chain rule:

For bigrams:

$$PP(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i|w_1 \ldots w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability
Perplexity as branching factor

• Let’s suppose a sentence consisting of random digits
• What is the perplexity of this sentence according to a model that assign $P=1/10$ to each digit?
Perplexity as branching factor

• Let’s suppose a sentence consisting of random digits
• What is the perplexity of this sentence according to a model that assign $P=1/10$ to each digit?

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-1/N}$$

$$= \left( \frac{1}{10} \right)^{-1/N}$$

$$= \frac{1}{10^{-1}}$$

$$= \frac{1}{10}$$

$$= 10$$
Lower perplexity = better model

Minimizing perplexity is the same as maximizing probability

• Training 38 million words, test 1.5 million words, WSJ

<table>
<thead>
<tr>
<th>N-gram Order</th>
<th>Unigram</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perplexity</td>
<td>962</td>
<td>170</td>
<td>109</td>
</tr>
</tbody>
</table>
Language Modeling

Generalization and zeros
The Shannon Visualization Method

- Choose a random bigram \((<s>, w)\) according to its probability
- Now choose a random bigram \((w, x)\) according to its probability
- And so on until we choose \(<s>\)
- Then string the words together

\(<s>\ I\ want\ to\ eat\ Chinese\ food\ </s> \quad I\ want\ to\ eat\ Chinese\ food
Imagine all the words of English covering the probability space between 0 and 1, each word covering an interval proportional to its frequency. We choose a random value between 0 and 1 and print the word whose interval includes this chosen value. We continue choosing random numbers and generating words until we randomly generate the sentence-final token \(</s>\). We can use the same technique to generate bigrams by first generating a random bigram that starts with \(</s>\) (according to its bigram probability), then choosing a random bigram to follow (again, according to its bigram probability), and so on.

To give an intuition for the increasing power of higher-order N-grams, Fig. 4.3 shows random sentences generated from unigram, bigram, trigram, and 4-gram models trained on Shakespeare's works.

| 1 | –To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have |
|   | –Hill he late speaks; or! a more to leg less first you enter |
| 2 | –Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. |
|   | –What means, sir. I confess she? then all sorts, he is trim, captain. |
| 3 | –Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. |
|   | –This shall forbid it should be branded, if renown made it empty. |
| 4 | –King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; |
|   | –It cannot be but so. |
Shakespeare as corpus

• $V=29,066$ types, $N=884,647$ tokens
Shakespeare as corpus

• N=884,647 tokens, V=29,066
• Shakespeare produced 300,000 bigram types out of $V^2 = 844$ million possible bigrams.
  • So 99.96% of the possible bigrams were never seen (have zero entries in the table)
• 4-grams worse: What's coming out looks like Shakespeare because it is Shakespeare
The Wall Street Journal is not Shakespeare (no offense)

<table>
<thead>
<tr>
<th>1 gram</th>
<th>Months the my and issue of year foreign new exchange’s September were recession exchange new endorsed a acquire to six executives</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 gram</td>
<td>Last December through the way to preserve the Hudson Corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. Corporation of living on information such as more frequently fishing to keep her</td>
</tr>
<tr>
<td>3 gram</td>
<td>They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions</td>
</tr>
</tbody>
</table>
Can you guess the author of these random 3-gram sentences?

They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and gram Brazil on market conditions

This shall forbid it should be branded, if renown made it empty.

“You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.
The perils of overfitting

• N-grams only work well for word prediction if the test corpus looks like the training corpus
  • In real life, it often doesn’t
  • We need to train robust models that generalize!
• One kind of generalization: Zeros!
  • Things that don’t ever occur in the training set
  • But occur in the test set
Zero probability bigrams

• Bigrams with zero probability
  • mean that we will assign 0 probability to the test set!
• And hence we cannot compute perplexity (can’t divide by 0)!
Language Modeling

Smoothing: Add-one (Laplace) smoothing
The intuition of smoothing (from Dan Klein)

When we have sparse statistics:

\[
P(w \mid \text{denied the})
\]

- 3 allegations
- 2 reports
- 1 claims
- 1 request
- 7 total

Steal probability mass to generalize better

\[
P(w \mid \text{denied the})
\]

- 2.5 allegations
- 1.5 reports
- 0.5 claims
- 0.5 request
- 2 other
- 7 total
Add-one estimation

Also called Laplace smoothing
Pretend we saw each word one more time than we did
Just add one to all the counts!

MLE estimate:

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

Add-1 estimate:

\[ P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V} \]
Maximum Likelihood Estimates

- The maximum likelihood estimate
  - of some parameter of a model M from a training set T
  - maximizes the likelihood of the training set T given the model M

- Suppose the word “bagel” occurs 400 times in a corpus of a million words

- What is the probability that a random word from some other text will be “bagel”?
  - MLE estimate is 400/1,000,000 = .0004

- This may be a bad estimate for some other corpus
  - But it is the estimate that makes it most likely that “bagel” will occur 400 times in a million word corpus.
Add-1 estimation is a blunt instrument

- So add-1 isn’t used for N-grams:
  - We’ll see better methods
- But add-1 is used to smooth other NLP models
  - For text classification
  - In domains where the number of zeros isn’t so huge.
Language Modeling

Interpolation, Backoff, and Web-Scale LMs
Backoff and Interpolation

Sometimes it helps to use **less** context

Condition on less context for contexts you haven’t learned much about

**Backoff:**

use trigram if you have good evidence,
otherwise bigram, otherwise unigram

**Interpolation:**

mix unigram, bigram, trigram

Interpolation works better
Linear Interpolation

Simple interpolation

\[ \hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n) \]

\[ \sum_i \lambda_i = 1 \]

Lambdas conditional on context:

\[ \hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 (w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) + \lambda_2 (w_{n-2}^{n-1})P(w_n|w_{n-1}) + \lambda_3 (w_{n-2}^{n-1})P(w_n) \]
How to set the lambdas?

- Use a **held-out** corpus

![Training Data](image)

| Training Data | Held-Out Data | Test Data |

- Choose $\lambda$s to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for $\lambda$s that give largest probability to held-out set:

$$
\log P(w_1...w_n \mid M(\lambda_1...\lambda_k)) = \sum_i \log P_{M(\lambda_1...\lambda_k)}(w_i \mid w_{i-1})
$$
Unknown words: Open versus closed vocabulary tasks

- If we know all the words in advanced
  - Vocabulary V is fixed
  - Closed vocabulary task

- Often we don’t know this
  - **Out Of Vocabulary** = OOV words
  - Open vocabulary task

- Instead: create an unknown word token `<UNK>`
  - Training of `<UNK>` probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to `<UNK>`
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training
Huge web-scale n-grams

• How to deal with, e.g., Google N-gram corpus
• Pruning
  • Only store N-grams with count > threshold.
    • Remove singletons of higher-order n-grams
  • Entropy-based pruning
• Efficiency
  • Efficient data structures like tries
  • Bloom filters: approximate language models
  • Store words as indexes, not strings
    • Use Huffman coding to fit large numbers of words into two bytes
  • Quantize probabilities (4-8 bits instead of 8-byte float)
Smoothing for Web-scale N-grams

• “Stupid backoff” (Brants et al. 2007)
• No discounting, just use relative frequencies

\[
S(w_i | w_{i-k+1}^{i-1}) = \begin{cases} 
\frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^i) > 0 \\
0.4S(w_i | w_{i-k+2}^{i-1}) & \text{otherwise}
\end{cases}
\]

\[
S(w_i) = \frac{\text{count}(w_i)}{N}
\]
N-gram Smoothing Summary

• Add-1 smoothing:
  • OK for text categorization, not for language modeling

• The most commonly used method:
  • Extended Interpolated Kneser-Ney

• For very large N-grams like the Web:
  • Stupid backoff
Advanced Language Modeling

• Discriminative models:
  • choose n-gram weights to improve a task, not to fit the training set

• Parsing-based models

• Caching Models
  • Recently used words are more likely to appear

\[
P_{\text{CACHE}}(w \mid \text{history}) = \lambda P(w_i \mid w_{i-2}w_{i-1}) + (1 - \lambda) \frac{c(w \in \text{history})}{|\text{history}|}
\]
Language Modeling

Advanced:
Kneser-Ney Smoothing
Absolute discounting: just subtract a little from each count

• Suppose we wanted to subtract a little from a count of 4 to save probability mass for the zeros
• How much to subtract?

• Church and Gale (1991)’s clever idea
• Divide up 22 million words of AP Newswire
  • Training and held-out set
  • for each bigram in the training set
  • see the actual count in the held-out set!

<table>
<thead>
<tr>
<th>Bigram count in training</th>
<th>Bigram count in heldout set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.0000270</td>
</tr>
<tr>
<td>1</td>
<td>0.448</td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>3.23</td>
</tr>
<tr>
<td>5</td>
<td>4.21</td>
</tr>
<tr>
<td>6</td>
<td>5.23</td>
</tr>
<tr>
<td>7</td>
<td>6.21</td>
</tr>
<tr>
<td>8</td>
<td>7.21</td>
</tr>
<tr>
<td>9</td>
<td>8.26</td>
</tr>
</tbody>
</table>

• It sure looks like $c^* = (c - .75)$
Absolute Discounting Interpolation

• Save ourselves some time and just subtract 0.75 (or some d)!

\[ P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(\hat{w}_{i-1})P(w) \]

• (Maybe keeping a couple extra values of d for counts 1 and 2)

• But should we really just use the regular unigram P(w)?
Better estimate for probabilities of lower-order unigrams!

- Shannon game: I can’t see without my reading Francisco?
- “Francisco” is more common than “glasses”
- ... but “Francisco” always follows “San”

The unigram is useful exactly when we haven’t seen this bigram!

Instead of \( P(w) \): “How likely is \( w \)”

\( P_{\text{continuation}}(w) \): “How likely is \( w \) to appear as a novel continuation?"

- For each word, count the number of bigram types it completes
- Every bigram type was a novel continuation the first time it was seen

\[
P_{\text{CONTINUATION}}(w) \propto \left| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} \right|
\]
Kneser-Ney Smoothing II

• How many times does \( w \) appear as a novel continuation:

\[
P_{\text{CONTINUATION}}(w) \propto \left| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} \right|
\]

• Normalized by the total number of word bigram types

\[
\left| \{ (w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0 \} \right|
\]

\[
P_{\text{CONTINUATION}}(w) = \frac{\left| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} \right|}{\left| \{ (w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0 \} \right|}
\]
Kneser-Ney Smoothing III

• Alternative metaphor: The number of word types seen to precede w

\[ | \{ w_{i-1} : c(w_{i-1}, w) > 0 \} | \]

• normalized by the number of words preceding all words:

\[ P_{\text{CONTINUATION}}(w) = \frac{| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} |}{\sum_{w'} | \{ w'_{i-1} : c(w'_{i-1}, w') > 0 \} |} \]

• A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability.
Kneser-Ney Smoothing IV

\[ P_{KN}(w_i | w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{CONTINUATION}(w_i) \]

\( \lambda \) is a normalizing constant; the probability mass we’ve discounted

\[ \lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{ w : c(w_{i-1}, w) > 0 \} \right| \]

the normalized discount

The number of word types that can follow \( w_{i-1} \)

= # of word types we discounted

= # of times we applied normalized discount
Kneser-Ney Smoothing: Recursive formulation

\[
P_{KN}(w_i | w_{i-n+1}) = \frac{\max(c_{KN}(w_{i-n+1}) - d, 0)}{c_{KN}(w_{i-n+1})} + \lambda(w_{i-n+1})P_{KN}(w_i | w_{i-n+2})
\]

\[
c_{KN}(\bullet) = \begin{cases} 
  \text{count}(\bullet) & \text{for the highest order} \\
  \text{continuationcount}(\bullet) & \text{for lower order}
\end{cases}
\]

Continuation count = Number of unique single word contexts for \(\bullet\)
Vector Space Semantics

Read Chapter 6 in the draft 3rd edition of Jurafsky and Martin
What does ongchoi mean?

Suppose you see these sentences:
Ong choi is delicious sautéed with garlic.
Ong choi is superb over rice
Ong choi leaves with salty sauces

And you've also seen these:
...spinach sautéed with garlic over rice
Chard stems and leaves are delicious
Collard greens and other salty leafy greens

Conclusion:
Ongchoi is a leafy green like spinach, chard, or collard greens
Ong choi: *Ipomoea aquatica* "Water Spinach"
If we consider optometrist and eye-doctor we find that, as our corpus of utterances grows, these two occur in almost the same environments. In contrast, there are many sentence environments in which optometrist occurs but lawyer does not...

It is a question of the relative frequency of such environments, and of what we will obtain if we ask an informant to substitute any word he wishes for oculist (not asking what words have the same meaning).

These and similar tests all measure the probability of particular environments occurring with particular elements... If A and B have almost identical environments we say that they are synonyms.

—Zellig Harris (1954)
We'll build a new representation of words that encodes their similarity

• Each word = a vector
• Similar words are "nearby in space"
We define a word as a vector

• Called an "embedding" because it's embedded into a space
• The standard way to represent meaning in NLP
• Fine-grained model of meaning for similarity
  • NLP tasks like sentiment analysis
    • With words, requires same word to be in training and test
    • With embeddings: ok if similar words occurred!!!
  • Question answering, conversational agents, etc
We'll introduce 2 kinds of embeddings

• **Tf-idf**
  • A common baseline model
  • Sparse vectors
  • Words are represented by a simple function of the counts of nearby words

• **Word2vec**
  • Dense vectors
  • Representation is created by training a classifier to distinguish nearby and far-away words