CS447: Natural Language Processing
http://courses.engr.illinois.edu/cs447

Lecture 10: Using (pointwise) mutual information to cluster words

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Class Admin

Midterm exam: Thu, Oct 12, 6:30pm

Location: DCL 1320 (next door)

Closed book exam:

- You are not allowed to use any cheat sheets, computers, calculators, phones etc.(you shouldn't have to anyway)
- Only the material covered in lectures

Format: Short answer questions

We will have a **review session** on **Friday, Oct 6** in class.

- Review the material *before* that class, so that we can clear up any confusions

Exam Question types

Define **X:**
Provide a mathematical/formal definition of X

Explain **X;** ***Explain*** **what X is/does:**
Use plain English to define X and say what X is/does

Compute **X:**
Return X; Show the steps required to calculate it

Draw **X:**
Draw a figure of X

Show/Prove **that X is true/is the case/…:**
This may require a (typically very simple) proof.

Discuss/Argue **whether …**

Use your knowledge (of X,Y,Z) to argue your point

4th Credit Hour

Either a **research project** (alone or with one other student) or a **literature survey** (alone)

Upcoming deadlines:

- **Before Oct 20:** Check your idea with me
- **Oct 20 (Wk 8): Proposal due:**
  (What will you do? What papers will you read?)

Good places to find NLP papers:

- **ACL anthology** http://aclweb.org/anthology
  covers almost everything published in NLP
- **JNLE** http://journals.cambridge.org/action/displayJournal?jid=NLE
  is another big NLP journal that is not part of the ACL
- Standard machine learning/AI conferences (**NIPS, ICML, IJCAI, AAAI**) and journals (**JMLR, JAIR** etc.) are okay as well.
- Other venues: check with me that this is actually NLP

4th Credit hour: Proposal

Upload a **one-page PDF** to Compass by Oct 20

- written in **LaTeX** (not MS Word)
- with **full bibliography** of the papers you want to read or base your project on
  (ideally with **links to online versions**; add url-field to your bibtex file)
- include a **motivation** of why you have chosen those papers
- for a research project: tell me whether you have the **data** you need, what **existing software** you will be using, what you will have to **implement yourself.**
- mention any **questions/concerns** that you may have.

Back to the material…

Wednesday’s key concepts

Sequence labeling tasks:

- POS tagging
- NP chunking
- Shallow Parsing
- Named Entity Recognition

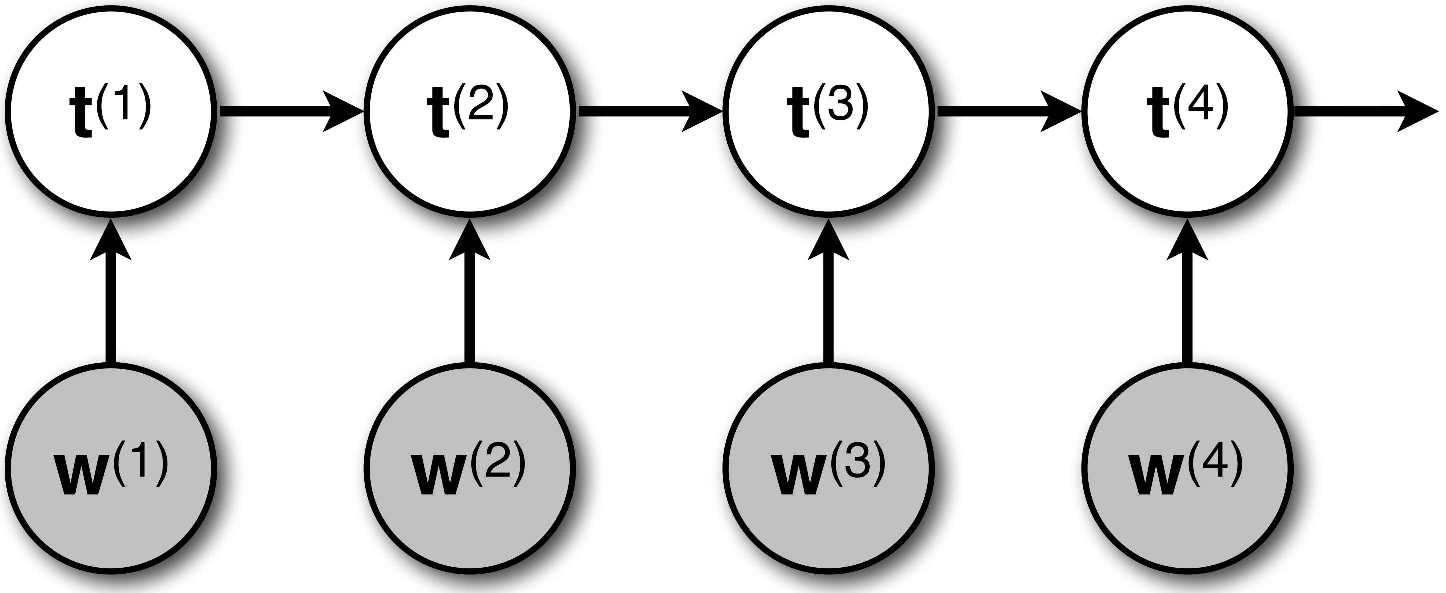
The BIO encoding for sequence labeling

Discriminative models:

- Maximum Entropy classifiers
- MEMMs (and CRFs)

Discriminative probability models

A discriminative or **conditional** model of the labels **t** given the observed input string **w** models $P(\mathbf{t} \mid \mathbf{w}) = \prod_i P(t^{(i)} \mid w^{(i)}, t^{(i-1)})$ directly.



Discriminative models

There are two main types of discriminative probability models:

- Maximum Entropy Markov Models (MEMMs)
- Conditional Random Fields (CRFs)

MEMMs and CRFs:

- are both based on logistic regression
- have the same graphical model
- require the Viterbi algorithm for tagging
- differ in that MEMMs consist of independently learned distributions, while CRFs are trained to maximize the probability of the entire sequence

Probabilistic classification

Classification:

Predict a class (label) c for an input $\mathbf{x}$

There are only a (small) finite number of possible class labels

Probabilistic classification:

– Model the probability $P(c \mid \mathbf{x})$

$P(c|\mathbf{x})$ is a probability if $0 \leq P (c_i \mid \mathbf{x}) \leq 1$, and $\sum_i P(c_i \mid \mathbf{x}) = 1$

– Return the class $c^* = \text{argmax}_i\, P (c_i \mid \mathbf{x})$ that has the highest probability

There are different ways to model $P(c \mid \mathbf{x})$.
MEMMs and CRFs are based on logistic regression

Using features

Think of feature functions as useful questions you can ask about the input $\mathbf{x}$:

- **Binary feature functions**:
  $f_{\text{first-letter-capitalized}}(\textbf{Urbana}) = 1$
  $f_{\text{first-letter-capitalized}}(\textbf{computer}) = 0$

- **Integer (or real-valued) features**:
  $f_{\text{number-of-vowels}}(\textbf{Urbana}) = 3$

Which specific feature functions are useful will depend on your task (and your training data).

From features to probabilities

We associate a real-valued weight w_{ic} with each feature function $f_i(\mathbf{x})$ and output class c

Note that the feature function $f_i(\mathbf{x})$ does not have to depend on c as long as the weight does (note the double index w_{ic})

This gives us a real-valued score for predicting class c for input $\mathbf{x}$: $\text{score}(\mathbf{x},c) = \sum_i w_{ic} f_i(\mathbf{x})$

This score could be negative, so we exponentiate it:

$\text{score}(\mathbf{x},c) = \exp(\sum_i w_{ic} f_i(\mathbf{x}))$

To get a probability distribution over all classes c, we renormalize these scores:

$$P(c \mid \mathbf{x}) = \text{score}(\mathbf{x},c) / \sum_j \text{score}(\mathbf{x},c_j)$$
$$= \exp(\sum_i w_{ic} f_i(\mathbf{x})) / \sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$$

Terminology

Models that are of the form

$$P(c \mid \mathbf{x}) = \text{score}(\mathbf{x},c) / \sum_j \text{score}(\mathbf{x},c_j)$$
$$= \exp(\sum_i w_{ic} f_i(\mathbf{x})) / \sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$$

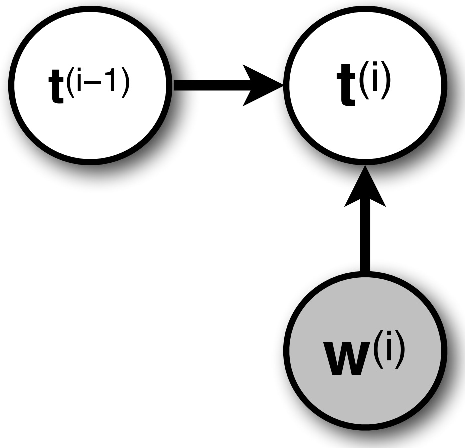
are also called loglinear models, Maximum Entropy (MaxEnt) models, or multinomial logistic regression models.

CS446 and CS546 should give you more details about these.

The normalizing term $\sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$ is also called the partition function and is often abbreviated as Z

Maximum Entropy Markov Models

MEMMs use a MaxEnt classifier for each $P(t^{(i)} | w^{(i)}, t^{(i-1)})$:



Since we use w to refer to words, let's use λ_{jk} as the weight for the feature function $f_j(t^{(i-1)}, w^{(i)})$ when predicting tag t_k:

$$P(t^{(i)} = t_k \mid t^{(i-1)}, w^{(i)}) = \frac{\exp(\sum_j \lambda_{jk} f_j(t^{(i-1)}, w^{(i)}))}{\sum_l \exp(\sum_j \lambda_{jl} f_j(t^{(i-1)}, w^{(i)}))}$$

Viterbi for MEMMs

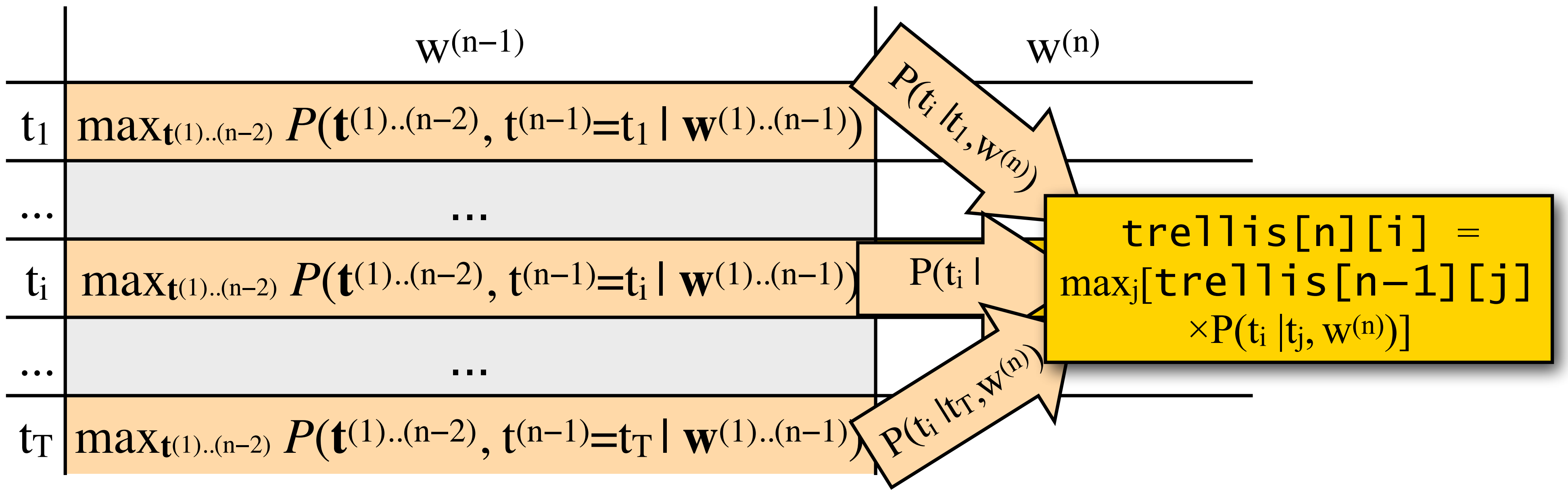
`trellis[n][i]` stores the probability of the most likely (Viterbi) tag sequence $\mathbf{t}^{(1)\dots(n)}$ that ends in tag t_i for the prefix $w^{(1)}\dots w^{(n)}$

Remember that we do not generate $\mathbf{w}$ in MEMMs. So:

$$\texttt{trellis[n][i]} = \max_{\mathbf{t}^{(1)..(n-1)}}\left[P(\mathbf{t}^{(1)\dots(n-1)}, t^{(n)}=t_i \mid \mathbf{w}^{(1)\dots(n)}) \right]$$

$$= \max_j \left[\texttt{trellis[n-1][j]} \times P(t_i \mid t_j, w^{(n)}) \right]$$

$$= \max_j \left[\max_{\mathbf{t}^{(1)..(n-2)}}[P(\mathbf{t}^{(1)..(n-2)}, t^{(n-1)}=t_j \mid \mathbf{w}^{(1)..(n-1)})] \times P(t_i \mid t_j, w^{(n)})\right]$$



$w^{(n-1)}$
$w^{(n)}$
t_1
$\max_{\mathbf{t}^{(1)..(n-2)}} P(\mathbf{t}^{(1)..(n-2)}, t^{(n-1)}=t_1 \mid \mathbf{w}^{(1)..(n-1)})$
...
t_i
$\max_{\mathbf{t}^{(1)..(n-2)}} P(\mathbf{t}^{(1)..(n-2)}, t^{(n-1)}=t_i \mid \mathbf{w}^{(1)..(n-1)})$
t_T
$\max_{\mathbf{t}^{(1)..(n-2)}} P(\mathbf{t}^{(1)..(n-2)}, t^{(n-1)}=t_T \mid \mathbf{w}^{(1)..(n-1)})$
$P(t_i \mid t_1, w^{(n)})$
$P(t_i \mid$
$P(t_i \mid t_T, w^{(n)})$
`trellis[n][i]` $= \max_j[$`trellis[n-1][j]` $\times P(t_i \mid t_j, w^{(n)})]$

Where we're at

So far, we've looked at the structure of words, and at sequences of words (without much structure):

- Finite-State Transducers for morphology
- N-gram models for language modeling
- Hidden Markov Models for POS tagging
- MEMMs for sequence labeling

In the next few lectures, we will go back to looking at words, but now we will consider their meaning:

- Finding groups of similar words by inducing word clusters
- Computing the semantic similarity of words by representing them in a vector space
- Identifying different meanings of words by word sense disambiguation

Mutual information and Pointwise mutual information (PMI)

Today's class

Goals:

- Identify "sticky pairs" of words that always go together
- Identify semantic clusters
- Partition words into clusters such that words that belong to the same cluster are "similar"
- Brown clusters

Mathematical tools:

- Mutual information
- Pointwise mutual information
- Hard (vs. soft clustering)

Discrete random variables

A discrete random variable X can take on values $\{x_1, \ldots, x_n\}$ with probability $p(X = x_i)$

A note on notation:

$p(X)$ refers to the distribution, while $p(X = x_i)$ refers to the probability of a specific value x_i. $p(X = x_i)$ also written as $p(x_i)$

In language modeling, the random variables correspond to words W or to sequences of words $W^{(1)} \ldots W^{(n)}$.

Another note on notation:

We're often sloppy in making the distinction between the *i*-th word [token] in a sequence/string, and the *i*-th word [type] in the vocabulary clear.

Mutual information $I(X;Y)$

Two random variables X, Y are **independent** iff their joint distribution is equal to the product of their individual distributions:

$$p(X, Y) = p(X)p(Y)$$

That is, for all outcomes x, y:

$$p(X{=}x, Y{=}x) = p(X{=}x)p(Y{=}y)$$

$I(X;Y)$, the **mutual information** of two random variables X and Y is defined as

$$I(X;Y) = \sum_{X,Y} p(X=x, Y=y) \log \frac{p(X=x, Y=y)}{p(X=x)p(Y=y)}$$

Pointwise mutual information (PMI)

Recall that two **events** x, y are **independent** if their joint probability is equal to the product of their individual probabilities:

x,y are independent iff $p(x,y) = p(x)p(y)$

x,y are independent iff $p(x,y) / p(x)p(y) = 1$

In NLP, we often use the pointwise mutual information (PMI) of two outcomes/events (e.g. words):

$$PMI(x,y) = \log \frac{p(X=x, Y=y)}{p(X=x)p(Y=y)}$$

Using PMI to cluster words

Using PMI to find related words

Find pairs of words w_i, w_j that have high pointwise mutual information:

$$PMI(w_i, w_j) = \log \frac{p(w_i, w_j)}{p(w_i)p(w_j)}$$

Different ways of defining $p(w_i, w_j)$
give different answers.

Using PMI to find "sticky pairs"

$p(w_i, w_j)$: probability that w_i, w_j are adjacent

Define $p(w_i, w_j) = p(\text{“}w_i w_j\text{”})$

High PMI word pairs under this definition:

Humpty Dumpty, Klux Klan, Ku Klux, Tse Tung, avant garde, gizzard shad, Bobby Orr, mutatis mutandis, Taj Mahal, Pontius Pilate, ammonium nitrate, jiggery pokery, anciens combattants, fuddle duddle, helter skelter, mumbo jumbo

(and a few more)

Using PMI to find "semantic clusters"

$p(w_i, w_j)$: probability that w_i, w_j are near each other

Define $p(w_i, w_j) = p(w_i)p(w_j \mid w_i)$, and $p(w_j \mid w_i)$ as the probability of w_j occurring within a window around w_i

(window = 500 words to the left or right of w_i, excluding two words before w_i and two words after w_i)

Resulting word clusters:

- *we our us ourselves ours*
- *question questions asking answer answers answering*
- *tie jacket suit*
- *attorney counsel trial court judge*
- *morning noon evening night nights midnight bed*
- *wall ceiling walls enclosure roof*
- *sell buy selling buying sold*

Using mutual information to induce word classes (Brown clusters)

Brown clusters

Goal: Partition words into clusters such that words that belong to the same cluster are “similar”.

- Brown clusters use a *hard* clustering approach: each word belongs to exactly one cluster.
- *Soft* clustering approaches allow each word to belong to multiple clusters.
- NB: If you learn HMMs from raw text, you induce *soft* clusters (there is no constraint that each word can only be emitted by a single hidden state)

Motivation: these clusters are useful features for many NLP tasks.

- Instead of dealing with 10K or 20K or more distinct words, replace them with a much smaller number of clusters.

Brown clusters

Instead of defining n-gram models over word types, first assign each word type to one class of words, and then define an n-gram model over these classes.

These word classes are induced automatically, and are chosen to maximize the avg. mutual information of classes whose words appear next to each other.

Words belonging to the same class are syntactically and semantically similar.

Class-Based n-gram Models of Natural Language
P. Brown et al., Computational Linguistics 18(4), December 1992
http://www.aclweb.org/anthology/J/J92/J92-4003.pdf

Class-based bigram models

Define a function $\pi(w) = c$ that assigns each word type w to a *single* class c.

- $\pi(w)$ defines a hard clustering (or a partition) over the words

Define the probability of $w^{(1)} \ldots w^{(n)}$ as

$$p(w^{(1)} \ldots w^{(n)}) = \prod_i p(c^{(i)} \mid c^{(i-1)}) p(w^{(i)} \mid c^{(i)})$$

Estimate the probability of $p(c^{(i)} \mid c^{(i-1)})$ as $\#(c^{(i-1)} c^{(i)}) / \#(c^{(i-1)})$

Estimate the probability of $p(w^{(i)} \mid c^{(i)})$ as $\#(w^{(i)}) / \#(c^{(i)})$

- NB: To avoid confusion between the classes c and count I'm using # for frequencies/counts here.

Class-based bigram models

Define the probability of the data $w^{(1)}\dots w^{(n)}$ as

$$p(w^{(1)}\dots w^{(n)}) = \prod_i p(c^{(i)} \mid c^{(i-1)})p(w^{(i)} \mid c^{(i)})$$

It can be shown that $L(\pi)$, the probability ('likelihood') of the training corpus under the mapping π is

$$L(\pi) = \sum_w p(w)\log p(w) + \sum_i p(c^{(i-1)}, c^{(i)}) \log \frac{p(c^{(i)} \mid c^{(i-1)})}{p(c^{(i)})}$$

$$= \boxed{\sum_w p(w)\log p(w)} + \boxed{\sum_i p(c^{(i-1)}, c^{(i)}) \log \frac{p(c^{(i)}, c^{(i-1)})}{p(c^{(i)})p(c^{(i-1)})}}$$

$-H(w)$
Unigram entropy

$E[\ I(C^{(i-1)}, C^{(i)})\ ]$
Expected (avg.) mutual information of adjacent classes

Class-based bigram models

$$L(\pi) = \boxed{\underbrace{\sum_{w} p(w) \log p(w)}_{-\mathrm{H}(w)}} + \boxed{\sum_{i} p(c^{(i-1)}, c^{(i)}) \log \frac{p(c^{(i)}, c^{(i-1)})}{p(c^{(i)})\, p(c^{(i-1)})}}$$

Unigram entropy:
This is fixed!

$E[\, I(c^{(i-1)}, c^{(i)})\,]$
Expected (avg.) mutual information
of adjacent classes:
This depends on π

A "good" mapping π assigns high probability to the training data

Learning objective: find $\pi^* = \max_{\pi} L(\pi)$

A "good" mapping π has high avg. mutual information of classes that appear next to each other.

Inducing word classes

Inducing word classes = learning a (good) partition of words into classes, $\pi(w) = \mathrm{c}$

What is a good partition, and how can we find it?

- A good language model assigns high probability to the training data ($\mathrm{L}(\pi)$).
- For two partitions π' and π'', $\mathrm{L}(\pi')$ and $\mathrm{L}(\pi'')$ differ only in the mutual information of adjacent classes.
- We want to find a π that maximizes the expected mutual information of adjacent classes.
- Given a training corpus and a partition π, we can compute $\mathrm{L}(\pi)$, but since we can't enumerate all possible partitions, we can't find the best possible π.
- Instead, we use a greedy algorithm

Inducing word classes

Greedy algorithm:

- Initialization: Assign each word to its own class
- Iterative step: Find the pair of classes that has the smallest loss in average mutual information.
- Repeat until C classes remain

This results in a hierarchical (“bottom-up”) clustering

Hierarchical clustering

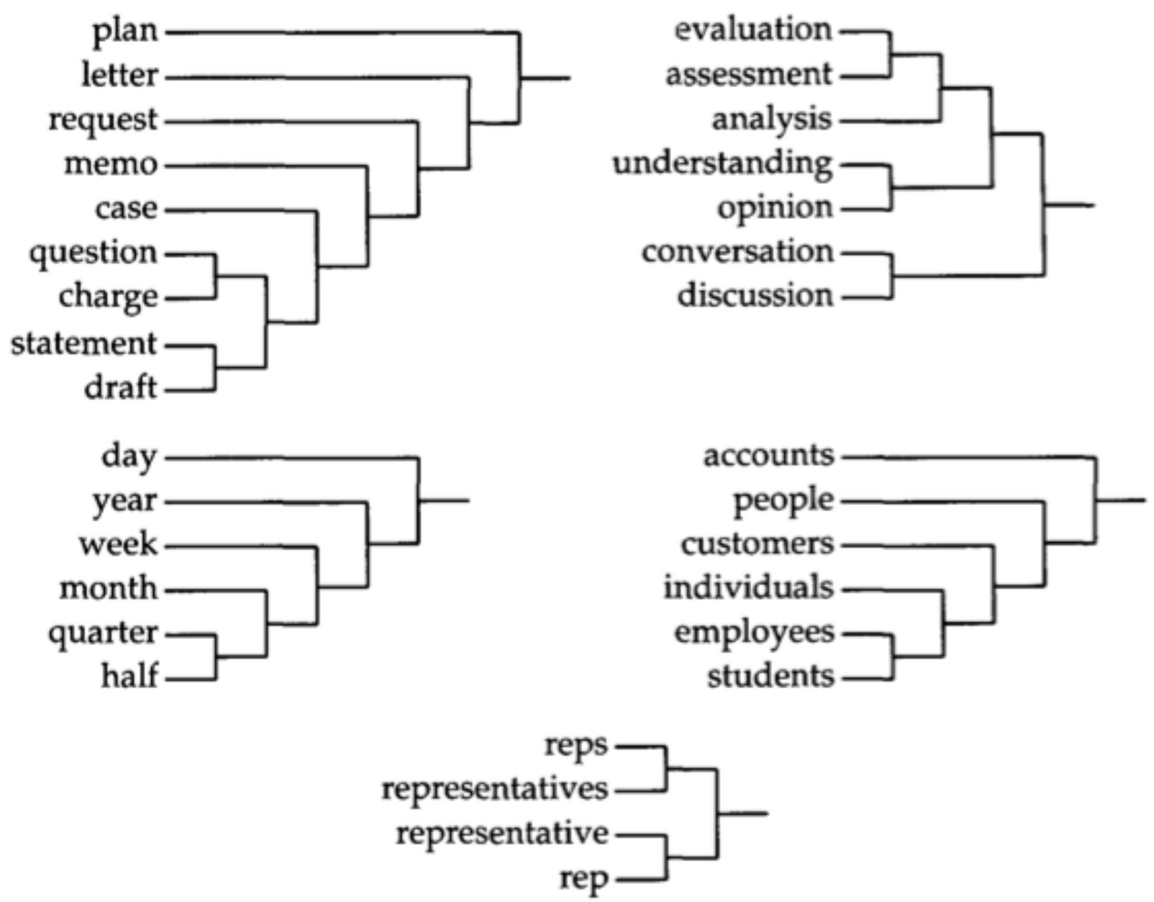


Figure 2
Sample subtrees from a 1,000-word mutual information tree.

Dealing with larger vocabularies

Computationally, this algorithm does not scale to large vocabularies.

Variant:

- Sort words by frequency
- Initialization: Assign first C words to their own classes
- Iterative step: take next most frequent word to get C+1 classes. Find the best merger.

Examples of word classes

- *Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends Sundays Saturdays*
- *people guys folks fellows CEOs chaps doubters commies unfortunates blokes*
- *down backwards ashore sideways southward northward overboard aloft downwards adrift*
- *had hadn't hath would've could've should've must've might've*
- *that tha theta*
- *head body hands eyes voice arm seat eye mouth*
- *rise focus depend rely concentrate dwell capitalize embark intrude typewriting*
- *running moving playing setting holding carrying passing cutting driving fighting*
- *brought moved opened picked caught tied gathered cleared hung lifted*

How can you use these clusters?

Most NLP systems are based on some form of machine learning.

Clusters provide useful **features** that can help deal with the sparsity of words:
Instead of (or in addition to) 10K/20K or more distinct words, use a much smaller number of clusters.

Today's key concepts

Clustering: hard vs. soft

Mutual information
Pointwise mutual information

Clustering:
using PMI of words
class-based bigram models: minimize the mutual information of adjacent classes