Algorithms for NLP

Parsing I

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley
MEMM Taggers

- Idea: left-to-right local decisions, condition on previous tags and also entire input

\[ P(t|w) = \prod_i P_{\text{ME}}(t_i|w, t_{i-1}, t_{i-2}) \]

- Train up \( P(t_i|w, t_{i-1}, t_{i-2}) \) as a normal maxent model, then use to score sequences
- This is referred to as an MEMM tagger [Ratnaparkhi 96]
- Beam search effective! (Why?)
- What about beam size 1?

- Subtle issues with local normalization (cf. Lafferty et al 01)
Decoding

- Decoding MEMM taggers:
  - Just like decoding HMMs, different local scores
  - Viterbi, beam search, posterior decoding

- Viterbi algorithm (HMMs):
  \[
  \delta_i(s) = \arg \max_{s'} P(s|s') P(w_{i-1}|s') \delta_{i-1}(s')
  \]

- Viterbi algorithm (MEMMs):
  \[
  \delta_i(s) = \arg \max_{s', w} P(s|s', w) \delta_{i-1}(s')
  \]

- General:
  \[
  \delta_i(s) = \arg \max_{s'} \phi_i(s', s) \delta_{i-1}(s')
  \]
Conditional Random Fields (and Friends)
Perceptron Review
- Linear model:
  \[ \text{score}(t|w) = \lambda^T f(t, w) \]
- ... that decompose along the sequence
  \[ = \lambda^T \sum_i f(t_i, t_{i-1}, w, i) \]
- ... allow us to predict with the Viterbi algorithm
  \[ t^* = \arg \max_t \text{score}(t|w) \]
- ... which means we can train with the perceptron algorithm
  (or related updates, like MIRA)
Conditional Random Fields

- Make a maxent model over entire taggings
  - MEMM
    \[
    P(t|w) = \prod_i \frac{1}{Z(i)} \exp \left( \lambda^T f(t_i, t_{i-1}, w, i) \right)
    \]
  - CRF
    \[
    P(t|w) = \frac{1}{Z(w)} \exp \left( \lambda^T \sum_i f(t_i, t_{i-1}, w, i) \right) = \frac{1}{Z(w)} \prod_i \phi_i(t_i, t_{i-1})
    \]
CRFs

- Like any maxent model, derivative is:

\[
\frac{\partial L(\lambda)}{\partial \lambda} = \sum_k \left( f_k(t^k) - \sum_t P(t \mid w_k) f_k(t) \right)
\]

- So all we need is to be able to compute the expectation of each feature (for example the number of times the label pair \( DT-NN \) occurs, or the number of times \( NN-interest \) occurs) under the model distribution.

- Critical quantity: counts of posterior marginals:

\[
\text{count}(w, s) = \sum_{i : w_i = w} P(t_i = s \mid w)
\]

\[
\text{count}(s \rightarrow s') = \sum_i P(t_{i-1} = s, t_i = s' \mid w)
\]
Computing Posterior Marginals

- How many (expected) times is word \( w \) tagged with \( s \)?

\[
\text{count}(w, s) = \sum_{i:w_i=w} P(t_i = s | w)
\]

- How to compute that marginal?

\[
\alpha_i(s) = \sum_{s'} \phi_i(s', s) \alpha_{i-1}(s')
\]
\[
\beta_i(s) = \sum_{s'} \phi_{i+1}(s, s') \beta_{i+1}(s')
\]
\[
P(t_i = s | w) = \frac{\alpha_i(s) \beta_i(s)}{\alpha_N(\text{END})}
\]
Global Discriminative Taggers

- **Newer, higher-powered discriminative sequence models**
  - CRFs (also perceptrons, M3Ns)
  - Do not decompose training into independent local regions
  - Can be deathly slow to train – require repeated inference on training set
- **Differences tend not to be too important for POS tagging**
- **Differences more substantial on other sequence tasks**
- **However: one issue worth knowing about in local models**
  - “Label bias” and other explaining away effects
  - MEMM taggers’ local scores can be near one without having both good “transitions” and “emissions”
  - This means that often evidence doesn’t flow properly
  - Why isn’t this a big deal for POS tagging?
  - Also: in decoding, condition on predicted, not gold, histories
Domain Effects

- Accuracies degrade outside of domain
  - Up to triple error rate
  - Usually make the most errors on the things you care about in the domain (e.g. protein names)

- Open questions
  - How to effectively exploit unlabeled data from a new domain (what could we gain?)
  - How to best incorporate domain lexica in a principled way (e.g. UMLS specialist lexicon, ontologies)
Unsupervised Tagging
Unsupervised Tagging?

- AKA part-of-speech induction

Task:
- Raw sentences in
- Tagged sentences out

Obvious thing to do:
- Start with a (mostly) uniform HMM
- Run EM
- Inspect results
EM for HMMs: Process

- Alternate between recomputing distributions over hidden variables (the tags) and reestimating parameters.
- Crucial step: we want to tally up how many (fractional) counts of each kind of transition and emission we have under current params:

\[
\text{count}(w, s) = \sum_{i: w_i = w} P(t_i = s | w)
\]

\[
\text{count}(s \rightarrow s') = \sum_{i} P(t_{i-1} = s, t_i = s' | w)
\]

- Same quantities we needed to train a CRF!
Some (discouraging) experiments [Merialdo 94]

Setup:
- You know the set of allowable tags for each word
- Fix k training examples to their true labels
  - Learn $P(w|t)$ on these examples
  - Learn $P(t|t_{-1},t_{-2})$ on these examples
- On n examples, re-estimate with EM

Note: we know allowed tags but not frequencies
## Merialdo: Results

<table>
<thead>
<tr>
<th>Iter</th>
<th>0</th>
<th>100</th>
<th>2000</th>
<th>5000</th>
<th>10000</th>
<th>20000</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>77.0</td>
<td>90.0</td>
<td>95.4</td>
<td>96.2</td>
<td>96.6</td>
<td>96.9</td>
<td>97.0</td>
</tr>
<tr>
<td>1</td>
<td>80.5</td>
<td>92.6</td>
<td>95.8</td>
<td>96.3</td>
<td>96.6</td>
<td>96.7</td>
<td>96.8</td>
</tr>
<tr>
<td>2</td>
<td>81.8</td>
<td>93.0</td>
<td>95.7</td>
<td>96.1</td>
<td>96.3</td>
<td>96.4</td>
<td>96.4</td>
</tr>
<tr>
<td>3</td>
<td>83.0</td>
<td>93.1</td>
<td>95.4</td>
<td>95.8</td>
<td>96.1</td>
<td>96.2</td>
<td>96.2</td>
</tr>
<tr>
<td>4</td>
<td>84.0</td>
<td>93.0</td>
<td>95.2</td>
<td>95.5</td>
<td>95.8</td>
<td>96.0</td>
<td>96.0</td>
</tr>
<tr>
<td>5</td>
<td>84.8</td>
<td>92.9</td>
<td>95.1</td>
<td>95.4</td>
<td>95.6</td>
<td>95.8</td>
<td>95.8</td>
</tr>
<tr>
<td>6</td>
<td>85.3</td>
<td>92.8</td>
<td>94.9</td>
<td>95.2</td>
<td>95.5</td>
<td>95.6</td>
<td>95.7</td>
</tr>
<tr>
<td>7</td>
<td>85.8</td>
<td>92.8</td>
<td>94.7</td>
<td>95.1</td>
<td>95.3</td>
<td>95.5</td>
<td>95.5</td>
</tr>
<tr>
<td>8</td>
<td>86.1</td>
<td>92.7</td>
<td>94.6</td>
<td>95.0</td>
<td>95.2</td>
<td>95.4</td>
<td>95.4</td>
</tr>
<tr>
<td>9</td>
<td>86.3</td>
<td>92.6</td>
<td>94.5</td>
<td>94.9</td>
<td>95.1</td>
<td>95.3</td>
<td>95.3</td>
</tr>
<tr>
<td>10</td>
<td>86.6</td>
<td>92.6</td>
<td>94.4</td>
<td>94.8</td>
<td>95.0</td>
<td>95.2</td>
<td>95.2</td>
</tr>
</tbody>
</table>
DistribuGonal Clustering

- The president said that the downturn was over.

| president | the __ of |
| president | the __ said |
| governor | the __ of |
| governor | the __ appointed |
| said | sources __ |
| said | president __ that |
| reported | sources __ |

[Finch and Chater 92, Shuetze 93, many others]
Distributional Clustering

- **Three main variants on the same idea:**
  - Pairwise similarities and heuristic clustering
    - E.g. [Finch and Chater 92]
    - Produces dendrograms
  - Vector space methods
    - E.g. [Shuetze 93]
    - Models of ambiguity
  - Probabilistic methods
    - Various formulations, e.g. [Lee and Pereira 99]
Nearest Neighbors

<table>
<thead>
<tr>
<th>word</th>
<th>nearest neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>accompanied</td>
<td>submitted banned financed developed authorized headed canceled awarded barred</td>
</tr>
<tr>
<td>almost</td>
<td>virtually merely formally fully quite officially just nearly only less</td>
</tr>
<tr>
<td>causing</td>
<td>reflecting forcing providing creating producing becoming carrying particularly</td>
</tr>
<tr>
<td>classes</td>
<td>elections courses payments losses computers performances violations levels pictures</td>
</tr>
<tr>
<td>directors</td>
<td>professionals investigations materials competitors agreements papers transactions</td>
</tr>
<tr>
<td>goal</td>
<td>mood roof eye image tool song pool scene gap voice</td>
</tr>
<tr>
<td>japanese</td>
<td>chinese iraqi american western arab foreign european federal soviet indian</td>
</tr>
<tr>
<td>represent</td>
<td>reveal attend deliver reflect choose contain impose manage establish retain</td>
</tr>
<tr>
<td>think</td>
<td>believe wish know realize wonder assume feel say mean bet</td>
</tr>
<tr>
<td>york</td>
<td>angeles francisco sox rouge kong diego zone vegas inning layer</td>
</tr>
<tr>
<td>on</td>
<td>through in at over into with from for by across</td>
</tr>
<tr>
<td>must</td>
<td>might would could cannot will should can may does helps</td>
</tr>
<tr>
<td>they</td>
<td>we you i he she nobody who it everybody there</td>
</tr>
</tbody>
</table>
A Probabilistic Version?

\[
P(S, C) = \prod_i P(c_i) P(w_i \mid c_i) P(w_{i-1}, w_{i+1} \mid c_i)
\]

\[\text{the president said that the downturn was over}\]
Syntax
The move followed a round of similar increases by other lenders, reflecting a continuing decline in that market.
Phrase Structure Parsing

- Phrase structure parsing organizes syntax into *constituents* or *brackets*
- In general, this involves nested trees
- Linguists can, and do, argue about details
- Lots of ambiguity
- Not the only kind of syntax...

new art critics write reviews with computers
Constituency Tests

- How do we know what nodes go in the tree?

- Classic constituency tests:
  - Substitution by *proform*
  - Question answers
  - Semantic grounds
    - Coherence
    - Reference
    - Idioms
  - Dislocation
  - Conjunction

- Cross-linguistic arguments, too
Conflicting Tests

- Constituency isn’t always clear
  - Units of transfer:
    - think about ~ penser à
    - talk about ~ hablar de
  - Phonological reduction:
    - I will go → I’ll go
    - I want to go → I wanna go
    - a le centre → au centre

- Coordination
  - He went to and came from the store.
Classical NLP: Parsing

- Write symbolic or logical rules:
  
<table>
<thead>
<tr>
<th>Grammar (CFG)</th>
<th>Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOT $\rightarrow$ S</td>
<td>NP $\rightarrow$ NP PP</td>
</tr>
<tr>
<td>S $\rightarrow$ NP VP</td>
<td>VP $\rightarrow$ VBP NP</td>
</tr>
<tr>
<td>NP $\rightarrow$ DT NN</td>
<td>VP $\rightarrow$ VBP NP PP</td>
</tr>
<tr>
<td>NP $\rightarrow$ NN NNS</td>
<td>PP $\rightarrow$ IN NP</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

- Use deduction systems to prove parses from words
  - Minimal grammar on “Fed raises” sentence: 36 parses
  - Simple 10-rule grammar: 592 parses
  - Real-size grammar: many millions of parses

- This scaled very badly, didn’t yield broad-coverage tools
Ambiguities
Ambiguities: PP Attachment

The board approved [its acquisition] [by Royal Trustco Ltd.] [of Toronto] [for $27 a share] [at its monthly meeting].
Attachments

- I cleaned the dishes from dinner
- I cleaned the dishes with detergent
- I cleaned the dishes in my pajamas
- I cleaned the dishes in the sink
Syntactic Ambiguities I

- **Prepositional phrases:**
  
  They cooked the beans in the pot on the stove with handles.

- **Particle vs. preposition:**
  
  The puppy tore up the staircase.

- **Complement structures**
  
  The tourists objected to the guide that they couldn’t hear.
  
  She knows you like the back of her hand.

- **Gerund vs. participial adjective**
  
  Visiting relatives can be boring.
  
  Changing schedules frequently confused passengers.
Syntactic Ambiguities II

- Modifier scope within NPs
  impractical design requirements
  plastic cup holder

- Multiple gap constructions
  The chicken is ready to eat.
  The contractors are rich enough to sue.

- Coordination scope:
  Small rats and mice can squeeze into holes or cracks in the wall.
Dark Ambiguities

- **Dark ambiguities**: most analyses are shockingly bad (meaning, they don’t have an interpretation you can get your mind around)

  This analysis corresponds to the correct parse of

  “This will panic buyers!”

- **Unknown words and new usages**

- **Solution**: We need mechanisms to focus attention on the best ones, probabilistic techniques do this
PCFGs
Probabilistic Context-Free Grammars

- A context-free grammar is a tuple \(<N, T, S, R>\)
  - \(N\): the set of non-terminals
    - Phrasal categories: S, NP, VP, ADJP, etc.
    - Parts-of-speech (pre-terminals): NN, JJ, DT, VB
  - \(T\): the set of terminals (the words)
  - \(S\): the start symbol
    - Often written as ROOT or TOP
    - \(Not\) usually the sentence non-terminal \(S\)
  - \(R\): the set of rules
    - Of the form \(X \rightarrow Y_1 Y_2 \ldots Y_k\), with \(X, Y_i \in N\)
    - Examples: \(S \rightarrow NP \ VP\), \(VP \rightarrow VP \ CC \ VP\)
    - Also called rewrites, productions, or local trees

- A PCFG adds:
  - A top-down production probability per rule \(P(Y_1 Y_2 \ldots Y_k | X)\)
( (S (NP-SBJ The move)
  (VP followed
   (NP (NP a round)
     (PP of
      (NP (NP similar increases)
        (PP by
          (NP other lenders))
        (PP against
          (NP Arizona real estate loans))))))

(ADJ (S-ADV (NP-SBJ *)
  (VP reflecting
   (NP (NP a continuing decline)
     (PP-LOC in
      (NP that market))))))

.)
Treebank Grammars

- Need a PCFG for broad coverage parsing.
- Can take a grammar right off the trees (doesn’t work well):

```
ROOT
  |        ROOT → S 1
  |  S → NP VP . 1
  |  NP → PRP 1
  |  NP    | VP
  |  PRP   | VBD ADJP
  |  He    | was JJ
  |  right |
```

- Better results by enriching the grammar (e.g., lexicalization).
- Can also get state-of-the-art parsers without lexicalization.
- Treebank grammars can be enormous
  - As FSAs, the raw grammar has ~10K states, excluding the lexicon
  - Better parsers usually make the grammars larger, not smaller
Chomsky Normal Form

- **Chomsky normal form:**
  - All rules of the form $X \rightarrow Y Z$ or $X \rightarrow w$
  - In principle, this is no limitation on the space of (P)CFGs
    - N-ary rules introduce new non-terminals
  - Unaries / empties are “promoted”
  - In practice it’s kind of a pain:
    - Reconstructing n-aries is easy
    - Reconstructing unaries is trickier
    - The straightforward transformations don’t preserve tree scores
  - Makes parsing algorithms simpler!
CKY Parsing
A Recursive Parser

Will this parser work?
Why or why not?
Memory requirements?

\begin{align*}
\text{bestScore}(X,i,j,s) &= \\
&\begin{cases} 
\text{if } (j = i+1) & \text{return } \text{tagScore}(X,s[i]) \\
\text{else} & \text{return } \max \text{ score}(X\rightarrow YZ) \times \text{bestScore}(Y,i,k) \times \text{bestScore}(Z,k,j)
\end{cases}
\end{align*}
One small change:

```java
bestScore(X, i, j, s)
    if (scores[X][i][j] == null)
        if (j = i+1)
            score = tagScore(X, s[i])
        else
            score = max score(X->YZ) * 
                     bestScore(Y, i, k) * 
                     bestScore(Z, k, j)
    scores[X][i][j] = score
return scores[X][i][j]
```
Can also organize things bottom-up

\[
\text{bestScore}(s) \\
\text{for } (i : [0,n-1]) \\
\quad \text{for } (X : \text{tags}[s[i]]) \\
\quad \quad \text{score}[X][i][i+1] = \text{tagScore}(X,s[i]) \\
\quad \text{for } (\text{diff} : [2,n]) \\
\quad \quad \text{for } (i : [0,n-\text{diff}]) \\
\quad \quad \quad j = i + \text{diff} \\
\quad \quad \quad \text{for } (X->YZ : \text{rule}) \\
\quad \quad \quad \quad \text{for } (k : [i+1, j-1]) \\
\quad \quad \quad \quad \quad \text{score}[X][i][j] = \max \ \text{score}[X][i][j], \\
\quad \quad \quad \quad \quad \text{score}(X->YZ) \ * \\
\quad \quad \quad \quad \quad \text{score}[Y][i][k] \ * \\
\quad \quad \quad \quad \quad \text{score}[Z][k][j]
\]
Unary Rules

 Unary rules?

\[
\text{bestScore}(X, i, j, s) = \\
\quad \text{if } (j = i+1) \quad \text{return } \text{tagScore}(X, s[i]) \\
\quad \text{else} \quad \text{return } \max \left( \text{max score}(X-YZ) \times \text{bestScore}(Y, i, k) \times \text{bestScore}(Z, k, j), \text{max score}(X-Y) \times \text{bestScore}(Y, i, j) \right)
\]
We need unaries to be non-cyclic

- Can address by pre-calculating the \textit{unary closure}
- Rather than having zero or more unaries, always have exactly one

Alternate unary and binary layers
Reconstruct unary chains afterwards
Alternating Layers

bestScoreB(X, i, j, s)
return max max score(X->YZ) *
    bestScoreU(Y, i, k) *
    bestScoreU(Z, k, j)

bestScoreU(X, i, j, s)
if (j = i+1)
    return tagScore(X, s[i])
else
    return max max score(X->Y) *
        bestScoreB(Y, i, j)
Analysis
Memory

- How much memory does this require?
  - Have to store the score cache
  - Cache size: $|\text{symbols}| \times n^2$ doubles
  - For the plain treebank grammar:
    - $X \sim 20K$, $n = 40$, double $\sim 8$ bytes $\sim 256$MB
    - Big, but workable.

- Pruning: Beams
  - $\text{score}[X][i][j]$ can get too large (when?)
  - Can keep beams (truncated maps $\text{score}[i][j]$) which only store the best few scores for the span $[i,j]$  

- Pruning: Coarse-to-Fine
  - Use a smaller grammar to rule out most $X[i,j]$  
  - Much more on this later…
How much time will it take to parse?

For each diff (<= n)
  For each i (<= n)
    For each rule X → Y Z
      For each split point k
        Do constant work

Total time: |rules| * n^3

Something like 5 sec for an unoptimized parse of a 20-word sentence
Parsing with the vanilla treebank grammar:

Why’s it worse in practice?
- Longer sentences “unlock” more of the grammar
- All kinds of systems issues don’t scale

~ 20K Rules
(not an optimized parser!)

Observed exponent: 3.6
Same-Span Reachability

ADJP ADVP
FRAG INTJ NP
PP PRN QP S
SBAR UCP VP
WHNP

NX
SQ
X
RRC
SINV
WHADJP
SBARQ
WHPP
WHADVP

LST
CONJP
NAC
Many states are more likely to match larger spans!
Lots of tricks to make CKY efficient

- Some of them are little engineering details:
  - E.g., first choose \( k \), then enumerate through the \( Y: [i, k] \) which are non-zero, then loop through rules by left child.
  - Optimal layout of the dynamic program depends on grammar, input, even system details.

- Another kind is more important (and interesting):
  - Many \( X[i,j] \) can be suppressed on the basis of the input string
  - We’ll see this next class as figures-of-merit, A* heuristics, coarse-to-fine, etc
Agenda-Based Parsing
Agenda-Based Parsing

- Agenda-based parsing is like graph search (but over a hypergraph)
- Concepts:
  - Numbering: we number fenceposts between words
  - “Edges” or items: spans with labels, e.g. PP[3,5], represent the sets of trees over those words rooted at that label (cf. search states)
  - A chart: records edges we’ve expanded (cf. closed set)
  - An agenda: a queue which holds edges (cf. a fringe or open set)
Building an item for the first time is called discovery. Items go into the agenda on discovery.

To initialize, we discover all word items (with score 1.0).

AGENDA

critics[0,1], write[1,2], reviews[2,3], with[3,4], computers[4,5]

CHART [EMPTY]

0 1 2 3 4 5

critics write reviews with computers
When we pop a word item, the lexicon tells us the tag item successors (and scores) which go on the agenda.
Item Successors

- When we pop items off of the agenda:
  - Graph successors: unary projections (NNS → critics, NP → NNS)

\[
Y[i,j] \text{ with } X \rightarrow Y \text{ forms } X[i,j]
\]

- Hypergraph successors: combine with items already in our chart

\[
Y[i,j] \text{ and } Z[j,k] \text{ with } X \rightarrow Y Z \text{ form } X[i,k]
\]

- Enqueue / promote resulting items (if not in chart already)
- Record backtraces as appropriate
- Stick the popped edge in the chart (closed set)

- Queries a chart must support:
  - Is edge X[i,j] in the chart? (What score?)
  - What edges with label Y end at position j?
  - What edges with label Z start at position i?
An Example

critics write reviews with computers

Empty Elements

- Sometimes we want to posit nodes in a parse tree that don’t contain any pronounced words:
  
  I want you to parse this sentence
  
  I want [ ] to parse this sentence

- These are easy to add to a agenda-based parser!
  
  - For each position i, add the “word” edge $\varepsilon[i,i]$
  
  - Add rules like $NP \rightarrow \varepsilon$ to the grammar
  
  - That’s it!
With weighted edges, order matters
- Must expand optimal parse from bottom up (subparses first)
- CKY does this by processing smaller spans before larger ones
- UCS pops items off the agenda in order of decreasing Viterbi score
- A* search also well defined

You can also speed up the search without sacrificing optimality
- Can select which items to process first
- Can do with any “figure of merit” [Charniak 98]
- If your figure-of-merit is a valid A* heuristic, no loss of optimality [Klein and Manning 03]
There was nothing magical about words spanning exactly one position.

When working with speech, we generally don’t know how many words there are, or where they break.

We can represent the possibilities as a lattice and parse these just as easily.
Unsupervised Tagging
Unsupervised Tagging?

- AKA part-of-speech induction

Task:
- Raw sentences in
- Tagged sentences out

Obvious thing to do:
- Start with a (mostly) uniform HMM
- Run EM
- Inspect results
EM for HMMs: Process

- Alternate between recomputing distributions over hidden variables (the tags) and reestimating parameters.
- Crucial step: we want to tally up how many (fractional) counts of each kind of transition and emission we have under current params:

\[
\text{count}(w, s) = \sum_{i: w_i = w} P(t_i = s | w)
\]

\[
\text{count}(s \rightarrow s') = \sum_i P(t_{i-1} = s, t_i = s' | w)
\]

- Same quantities we needed to train a CRF!
Some (discouraging) experiments [Merialdo 94]

Setup:
- You know the set of allowable tags for each word
- Fix k training examples to their true labels
  - Learn $P(w|t)$ on these examples
  - Learn $P(t|t_{-1}, t_{-2})$ on these examples
- On n examples, re-estimate with EM

Note: we know allowed tags but not frequencies
Merialdo: Results

<table>
<thead>
<tr>
<th>Iter</th>
<th>Number of tagged sentences used for the initial model</th>
<th>Correct tags (% words) after ML on 1M words</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>0</td>
<td>77.0</td>
<td>90.0</td>
</tr>
<tr>
<td>1</td>
<td>80.5</td>
<td>92.6</td>
</tr>
<tr>
<td>2</td>
<td>81.8</td>
<td>93.0</td>
</tr>
<tr>
<td>3</td>
<td>83.0</td>
<td>93.1</td>
</tr>
<tr>
<td>4</td>
<td>84.0</td>
<td>93.0</td>
</tr>
<tr>
<td>5</td>
<td>84.8</td>
<td>92.9</td>
</tr>
<tr>
<td>6</td>
<td>85.3</td>
<td>92.8</td>
</tr>
<tr>
<td>7</td>
<td>85.8</td>
<td>92.8</td>
</tr>
<tr>
<td>8</td>
<td>86.1</td>
<td>92.7</td>
</tr>
<tr>
<td>9</td>
<td>86.3</td>
<td>92.6</td>
</tr>
<tr>
<td>10</td>
<td>86.6</td>
<td>92.6</td>
</tr>
</tbody>
</table>
the president said that the downturn was over
DistribuGonal Clustering

- Three main variants on the same idea:
  - Pairwise similarities and heuristic clustering
    - E.g. [Finch and Chater 92]
    - Produces dendrograms
  - Vector space methods
    - E.g. [Shuetze 93]
    - Models of ambiguity
  - Probabilistic methods
    - Various formulations, e.g. [Lee and Pereira 99]
**Nearest Neighbors**

<table>
<thead>
<tr>
<th>word</th>
<th>nearest neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>accompanied</td>
<td>submitted banned financed developed authorized headed canceled awarded barred</td>
</tr>
<tr>
<td>almost</td>
<td>virtually merely formally fully quite officially just nearly only less</td>
</tr>
<tr>
<td>causing</td>
<td>reflecting forcing providing creating producing becoming carrying particularly</td>
</tr>
<tr>
<td>classes</td>
<td>elections courses payments losses computers performances violations levels pictures</td>
</tr>
<tr>
<td>directors</td>
<td>professionals investigations materials competitors agreements papers transactions</td>
</tr>
<tr>
<td>goal</td>
<td>mood roof eye image tool song pool scene gap voice</td>
</tr>
<tr>
<td>japanese</td>
<td>chinese iraqi american western arab foreign european federal soviet indian</td>
</tr>
<tr>
<td>represent</td>
<td>reveal attend deliver reflect choose contain impose manage establish retain</td>
</tr>
<tr>
<td>think</td>
<td>believe wish know realize wonder assume feel say mean bet</td>
</tr>
<tr>
<td>york</td>
<td>angeles francisco sox rouge kong diego zone vegas inning layer</td>
</tr>
<tr>
<td>on</td>
<td>through in at over into with from for by across</td>
</tr>
<tr>
<td>must</td>
<td>might would could cannot will should can may does helps</td>
</tr>
<tr>
<td>they</td>
<td>we you i he she nobody who it everybody there</td>
</tr>
</tbody>
</table>
Dendrograms