Introduction to Natural Language Processing I
[Statistické metody zpracování přirozených jazyků I]
(NPFL067)
http://ufal.mff.cuni.cz/courses/npfl067

prof. RNDr. Jan Hajič, Dr. / doc. RNDr. Pavel Pecina, Ph.D.
ÚFAL MFF UK
{hajic,pecina}@ufal.mff.cuni.cz
http://ufal.mff.cuni.cz/jan-hajic
Word Classes:
Programming Tips & Tricks
The Algorithm (review)

• Define merge(r,k,l) = (r',C') such that
  • C' = C - \{k,l\} ∪ \{m \text{ (a new class)}\}
  • r'(w) = r(w) except for k,l member words for which it is m.

• 1. Start with each word in its own class (C = V), r = id.
• 2. Merge two classes k,l into one, m, such that
  \[(k,l) = \arg\max_{k,,l} I_{\text{merge}(r,k,l)}(D,E).\]
• 3. Set new (r,C) = merge(r,k,l).
• 4. Repeat 2 and 3 until |C| reaches a predetermined size.
Complexity Issues

• Still too complex:
  – $|V|$ iterations of the steps 2 and 3.
  – $|V|^2$ steps to maximize $\text{argmax}_{k,l}$ (selecting $k,l$ freely from $|C|$, which is in the order of $|V|^2$)
  – $|V|^2$ steps to compute $I(D,E)$ (sum within sum, all classes, also: includes log)
  – $\Rightarrow$ total: $|V|^5$
  – i.e., for $|V| = 100$, about $10^{10}$ steps; ~ several hours!
  – but $|V| \sim 50,000$ or more
Trick #1: Recomputing The MI the Smart Way: Subtracting...

• Bigram count table:

<table>
<thead>
<tr>
<th>l \ r</th>
<th>c₁</th>
<th>c₂</th>
<th>c₃</th>
<th>c₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>c₁</td>
<td>10</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c₂</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>c₃</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>c₄</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• Test-merging c₂ and c₄: recompute only rows/cols 2 & 4:
  – subtract column/row (2 & 4) from the MI sum (intersect.!)  
  – add sums of merged counts (row & column)
...and Adding

- Add the merged counts:

```
<table>
<thead>
<tr>
<th>l \ r</th>
<th>c_1</th>
<th>c_2'</th>
<th>c_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_1</td>
<td>10</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>c_2'</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>c_3</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>
```

- Be careful at intersections:
  - (don’t forget to add this:)

```
<table>
<thead>
<tr>
<th>l \ r</th>
<th>c_2</th>
<th>c_3</th>
<th>c_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_2</td>
<td>0</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>c_3</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>c_4</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Trick #2: Precompute the Counts-to-be-Subtracted

- Summing loop goes through i,j
- ...but the single row/column sums do not depend on the (resulting sums after the) merge
- \( \Rightarrow \) can be precomputed
  - only 2k logs to compute at each algorithm iteration, instead of \( k^2 \)
- Then for each “merge-to-be” compute only add-on sums, plus “intersection adjustment”
Formulas for Tricks #1 and #2

- Let’s have \( k \) classes at a certain iteration. Define:
  \[
  q_k(l,r) = p_k(l,r) \log(p_k(l,r) / (p_{kl}(l) p_{kr}(r)))
  \]
  now the same, but using counts:
  \[
  q_k(l,r) = c_k(l,r)/N \log(N c_k(l,r)/(c_{kl}(l) c_{kr}(r)))
  \]
- Define further (row+column i sum):
  \[
  s_k(a) = \sum_{l=1}^{k} q_k(l,a) + \sum_{r=1}^{k} q_k(a,r) - q_k(a,a)
  \]
- Then, the subtraction part of Trick #1 amounts to
  \[
  \text{sub}_k(a,b) = s_k(a) + s_k(b) - q_k(a,b) - q_k(b,a)
  \]
Formulas - cont.

- After-merge add-on:
  \[
  add_k(a,b) = \sum_{l=1..k, l \neq a, b} q_k(l, a+b) + \sum_{r=1..k, r \neq a, b} q_k(a+b, r) + q_k(a+b, a+b)
  \]

- What is it \(a+b\)? Answer: the new (merged) class.

- Hint: use the definition of \(q_k\) as a “macro”, and then
  \[
  p_k(a+b, r) = p_k(a, r) + p_k(b, r)
  \]
  (same for other sums, equivalent)

- The above sums cannot be precomputed

- After-merge Mutual Information (\(I_k\) is the “old” MI, kept from previous iteration of the algorithm):
  \[
  I_k(a,b) \text{ (MI after merge of cl. a,b)} = I_k - sub_k(a,b) + add_k(a,b)
  \]
Trick #3: Ignore Zero Counts

- Many bigrams are 0
  - (see the paper: Canadian Hansards, < .1 % of bigrams are non-zero)
- Create linked lists of non-zero counts in columns and rows (similar effect: use perl’s hashes)
- Update links after merge (after step 3)
Trick #4: Use Updated Loss of MI

- We are now down to $|V|^4$: $|V|$ merges, each merge takes $|V|^2$ “test-merges”, each test-merge involves order-of-$|V|$ operations ($add_k(i,j)$ term, foil #8)
- **Observation**: many numbers ($s_k$, $q_k$) needed to compute the mutual information loss due to a merge of $i+j$ **do not change**: namely, those which are not in the vicinity of neither $i$ nor $j$.
- **Idea**: keep the MI loss matrix for all pairs of classes, and (after a merge) update only those cells which have been influenced by the merge.
Formulas for Trick #4 (s_{k-1}, L_{k-1})

- Keep a matrix of “losses” $L_k(d,e)$.\(^1\)
- Init: $L_k(d,e) = \text{sub}_k(d,e) - \text{add}_k(d,e)$ [then $I_k(d,e) = I_k - L_k(d,e)$]
- Suppose $a, b$ are now the two classes merged into $a$:
- Update $(k-1$: index used for the next iteration; $i, j \neq a, b$):
  
  $- s_{k-1}(i) = s_k(i) - q_k(i, a) - q_k(a, i) - q_k(i, b) - q_k(b, i) + q_{k-1}(a, i) + q_{k-1}(i, a)$
  $- 2L_{k-1}(i, j) = L_k(i, j) - s_k(i) + s_{k-1}(i) - s_k(j) + s_{k-1}(j) +$
  $+ q_k(i+j, a) + q_k(a, i+j) + q_k(i+j, b) + q_k(b, i+j) -$
  $- q_{k-1}(i+j, a) - q_{k-1}(a, i+j)$ \([NB: may substitute even for s_k, s_{k-1}]\)

NB \(^1\) $L_k$ is symmetrical $L_k(d,e) = L_k(e,d)$ ($q_k$ is something different!)

\(^2\) The update formula $L_{k-1}(l,m)$ is wrong in the Brown et. al paper
Completing Trick #4

- $s_{k-1}(a)$ must be computed using the “Init” sum.
- $L_{k-1}(a,i) = L_{k-1}(i,a)$ must be computed in a similar way, for all $i \neq a, b$.
- $s_{k-1}(b), L_{k-1}(b,i), L_{k-1}(i,b)$ are not needed anymore (keep track of such data, i.e. mark every class already merged into some other class and do not use it anymore).
- Keep track of the minimal loss during the $L_k(i,j)$ update process (so that the next merge to be taken is obvious immediately after finishing the update step).
Efficient Implementation

- **Data Structures:** (N - # of bigrams in data [fixed])
  - Hist(k)       history of merges
    - Hist(k) = (a,b) merged when the remaining number of classes was k
  - $c_k(i,j)$    bigram class counts [updated]
  - $c_{kl}(i), c_{kr}(i)$ unigram (marginal) counts [updated]
  - $L_k(a,b)$   table of losses; upper-right triangle [updated]
  - $s_k(a)$     “subtraction” subterms [optionally updated]
  - $q_k(i,j)$   subterms involving a log [opt. updated]

- The optionally updated data structures will give linear improvement only in the subsequent steps, but at least $s_k(i)$ is necessary in the initialization phase (1st iteration)
Implementation: the Initialization Phase

- 1 Read data in, init counts $c_k(l,r)$; then $\forall l,r,a,b; a < b$:
- 2 Init unigram counts:
  
  \[
  c_{kl}(l) = \sum_{r=1..k} c_k(l,r), \quad c_{kr}(r) = \sum_{l=1..k} c_k(l,r)
  \]
  
  – complicated? remember, must take care of start & end of data!
- 3 Init $q_k(l,r)$: use the 2\textsuperscript{nd} formula (count-based) on foil 7,
  
  \[
  q_k(l,r) = \frac{c_k(l,r)}{N} \log(\frac{c_k(l,r)}{(c_{kl}(l) c_{kr}(r))})
  \]
- 4 Init $s_k(a) = \sum_{l=1..k} q_k(l,a) + \sum_{r=1..k} q_k(a,r) - q_k(a,a)$
- 5 Init $L_k(a,b) = s_k(a) + s_k(b) - q_k(a,b) - q_k(b,a) - q_k(a+b,a+b) + $
  
  - $\sum_{l=1..k, l\neq a,b} q_k(l,a+b) - \sum_{r=1..k, r\neq a,b} q_k(a+b,r)$
Implementation: Select & Update

• 6 Select the best pair \((a,b)\) to merge into \(a\) (watch the candidates when computing \(L_k(a,b)\)); save to Hist(k)
• 7 Optionally, update \(q_k(i,j)\) for all \(i,j \neq b\), get \(q_{k-1}(i,j)\)
  – remember those \(q_k(i,j)\) values needed for the updates below
• 8 Optionally, update \(s_k(i)\) for all \(i \neq b\), to get \(s_{k-1}(i)\)
  – again, remember the \(s_k(i)\) values for the “loss table” update
• 9 Update the loss table, \(L_k(i,j)\), to \(L_{k-1}(i,j)\), using the tabulated \(q_k\), \(q_{k-1}\), \(s_k\) and \(s_{k-1}\) values, or compute the needed \(q_k(i,j)\) and \(q_{k-1}(i,j)\) values dynamically from the counts:
  \[c_k(i+j,b) = c_k(i,b) + c_k(j,b); \quad c_{k-1}(a,i) = c_k(a+b,i)\]
Towards the Next Iteration

- 10 During the $L_k(i,j)$ update, keep track of the minimal loss of MI, and the two classes which caused it.
- 11 Remember such best merge in Hist(k).
- 12 Get rid of all $s_k, q_k, L_k$ values.
- 13 Set $k = k - 1$; stop if $k == 1$.
- 14 Start the next iteration
  - either by the optional updates (steps 7 and 8), or
  - directly updating $L_k(i,j)$ again (step 9).
Moving Words Around

• Improving Mutual Information
  – take a word from one class, move it to another (i.e., two classes change: the moved-from and the moved-to), compute $I_{\text{new}}(D,E)$; keep change permanent if $I_{\text{new}}(D,E) > I(D,E)$
  – keep moving words until no move improves $I(D,E)$
• Do it at every iteration, or at every $m$ iterations
• Use similar “smart” methods as for merging
Using the Hierarchy

- **Natural Form of Classes**
  - follows from the sequence of merges:

```
  4
 / \
2   2
 / \
1   3
 / \
evaluation assessment analysis understanding opinion
```
Numbering the Classes (within the Hierarchy)

- Binary branching
- Assign 0/1 to the left/right branch at every node:

- prefix determines class:
  00 ~ \{evaluation, assessment\}

```
000  001  010  100  110
```

evaluation assessment analysis understanding opinion [padding: 0]