# Phrase Finding; Stream-and-Sort vs "Request-and-Answer"

William W. Cohen

# Announcement

- Two-day extension for all on HW1B:
  - now due Thursday 1/29

# Correction…

# Using Large-vocabulary Naïve Bayes

- For each example *id, y,* $x_1,\ldots,x_d$ in ~~*test*~~: *train:* [For assignment]
- Sort the event-counter update "messages"
- Scan and add the sorted messages and output the final counter values

Model size: $O(|V|)$

- Initialize a HashSet NEEDED and a hashtable C
- For each example *id, y,* $x_1,\ldots,x_d$ in *test:*
  - Add $x_1,\ldots,x_d$ to NEEDED

Time: $O(n_2)$, *size of test*
Memory: same

- For each *event, C(event)* in the summed counters
  - If *event* involves a NEEDED term *x* read it into C

Time: $O(n_2)$
Memory: same

- For each example *id, y,* $x_1,\ldots,x_d$ in *test:*
  - For each $y'$ in $dom(Y)$:
    - Compute log $\Pr(y',x_1,\ldots,x_d) = \ldots$

Time: $O(n_2)$
Memory: same

# Review of NB algorithms

| HW | Train events | Test events | Suggested data |
|---|---|---|---|
| 1A | HashMap | HashMap | RCV1 |
| 1B | Msgs ➔ Disk | HashMap (for subset) | Wikipedia |
| --- | Msgs ➔ Disk | Msgs on Disk (coming….) | --- |

# Outline

- Even more on stream-and-sort and naïve Bayes
- Another problem: "meaningful" phrase finding
- Implementing phrase finding efficiently
- Some other phrase-related problems

# Last Week

- How to implement Naïve Bayes
  - Time is linear in size of data (one scan!)
  - We need to count, e.g. C( *X=word ^ Y=label)*
- How to implement Naïve Bayes with large vocabulary and small memory
  - General technique: "Stream and sort"
    - Very little seeking (only in merges in merge sort)
    - Memory-efficient

# Flaw: Large-vocabulary Naïve Bayes is Expensive to Use

- For each example *id*, *y*, $x_1,\ldots,x_d$ in *train:*
- Sort the event-counter update "messages"
- Scan and add the sorted messages and output the final counter values

Model size: max $O(n)$, $O(|V| \cdot |dom(Y)|)$

- For each example *id*, *y*, $x_1,\ldots,x_d$ in *test:*
  - For each $y'$ in $dom(Y)$:
    - Compute log $\Pr(y', x_1,\ldots,x_d)$ =

$$= \left( \sum_j \log \frac{C(X = x_j \wedge Y = y') + mq_x}{C(X = ANY \wedge Y = y') + m} \right) + \log \frac{C(Y = y') + mq_y}{C(Y = ANY) + m}$$

# The workaround I suggested

- For each example *id, y,* $x_1,\ldots,x_d$ in *train:*
- Sort the event-counter update "messages"
- Scan and add the sorted messages and output the final counter values

---

- Initialize a HashSet NEEDED and a hashtable C
- For each example *id, y,* $x_1,\ldots,x_d$ in *test:*
  - Add $x_1,\ldots,x_d$ to NEEDED
- For each *event, C(event)* in the summed counters
  - If *event* involves a NEEDED term *x* read it into C

---

- For each example *id, y,* $x_1,\ldots,x_d$ in *test:*
  - For each *y'* in *dom(Y):*
    - Compute log Pr(*y'*,$x_1,\ldots,x_d$) = ….

# Can we do better?

# Can we do better?

Test data

| |
|---|
| $id_1$ $w_{1,1}$ $w_{1,2}$ $w_{1,3}$ …. $w_{1,k1}$ |
| $id_2$ $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ …. |
| $id_3$ $w_{3,1}$ $w_{3,2}$ …. |
| $id_4$ $w_{4,1}$ $w_{4,2}$ … |
| $id_5$ $w_{5,1}$ $w_{5,2}$ …. |
| .. |

Event counts

| | |
|---|---|
| $X=w_1 \wedge Y=sports$ | 5245 |
| $X=w_1 \wedge Y=worldNews$ | 1054 |
| $X=..$ | 2120 |
| $X=w_2 \wedge Y=…$ | 37 |
| $X=…$ | 3 |
| … | … |

What we'd like

| | |
|---|---|
| $id_1$ $w_{1,1}$ $w_{1,2}$ $w_{1,3}$ …. $w_{1,k1}$ | $C[X=w_{1,1} \wedge Y=sports]=5245$, $C[X=w_{1,1} \wedge Y=..]$,$C[X=w_{1,2} \wedge …]$ |
| $id_2$ $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ …. | $C[X=w_{2,1} \wedge Y=….]=1054$,…, $C[X=w_{2,k2} \wedge …]$ |
| $id_3$ $w_{3,1}$ $w_{3,2}$ …. | $C[X=w_{3,1} \wedge Y=….]=…$ |
| $id_4$ $w_{4,1}$ $w_{4,2}$ … | … |
| | |

# Can we do better?

**Step 1**: group counters by word *w*

How:

- Stream and sort:
  - for each C[X=w^Y=y]=n
    - print "w  C[Y=y]=n"
  - sort and build a *list* of values associated with each key *w*

*Like an inverted index*

Event counts

| | |
|---|---|
| $X=w_1$^$Y=sports$ | *5245* |
| $X=w_1$^$Y=worldNews$ | *1054* |
| *X=..* | *2120* |
| $X=w_2$^$Y=…$ | *37* |
| *X=…* | *3* |
| … | … |

| w | Counts associated with W |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| agent | C[w^Y=sports]=1027,C[w^Y=worldNews]=564 |
| … | … |
| zynga | C[w^Y=sports]=21,C[w^Y=worldNews]=4464 |

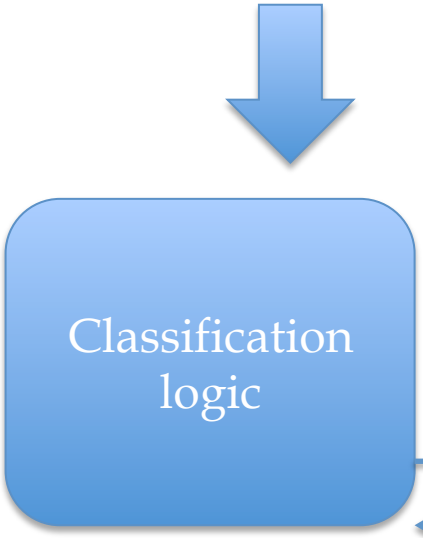
# If these records were in a key-value DB we would know what to do…

Test data

$id_1$ $w_{1,1}$ $w_{1,2}$ $w_{1,3}$ …. $w_{1,k1}$
$id_2$ $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ ….
$id_3$ $w_{3,1}$ $w_{3,2}$ ….
$id_4$ $w_{4,1}$ $w_{4,2}$ …
$id_5$ $w_{5,1}$ $w_{5,2}$ ….
..

Record of all event counts for each word

| w | Counts associated with W |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| agent | C[w^Y=sports]=1027,C[w^Y=worl |
| … | … |
| zynga | C[w^Y=sports]=21,C[w^Y=worldN |




Step 2: stream through and for each test case

$id_i$ $w_{i,1}$ $w_{i,2}$ $w_{i,3}$ …. $w_{i,ki}$

request the event counters needed to classify $id_i$ from the event-count DB, then classify using the answers

# Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

$id_1$ $w_{1,1}$ $w_{1,2}$ $w_{1,3}$ …. $w_{1,k1}$
$id_2$ $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ ….
$id_3$ $w_{3,1}$ $w_{3,2}$ ….
$id_4$ $w_{4,1}$ $w_{4,2}$ …
$id_5$ $w_{5,1}$ $w_{5,2}$ ….
..

Record of all event counts for each word

| w | Counts associated with W |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| agent | C[w^Y=sports]=1027,C[w^Y=worl |
| … | … |
| zynga | C[w^Y=sports]=21,C[w^Y=worldN |

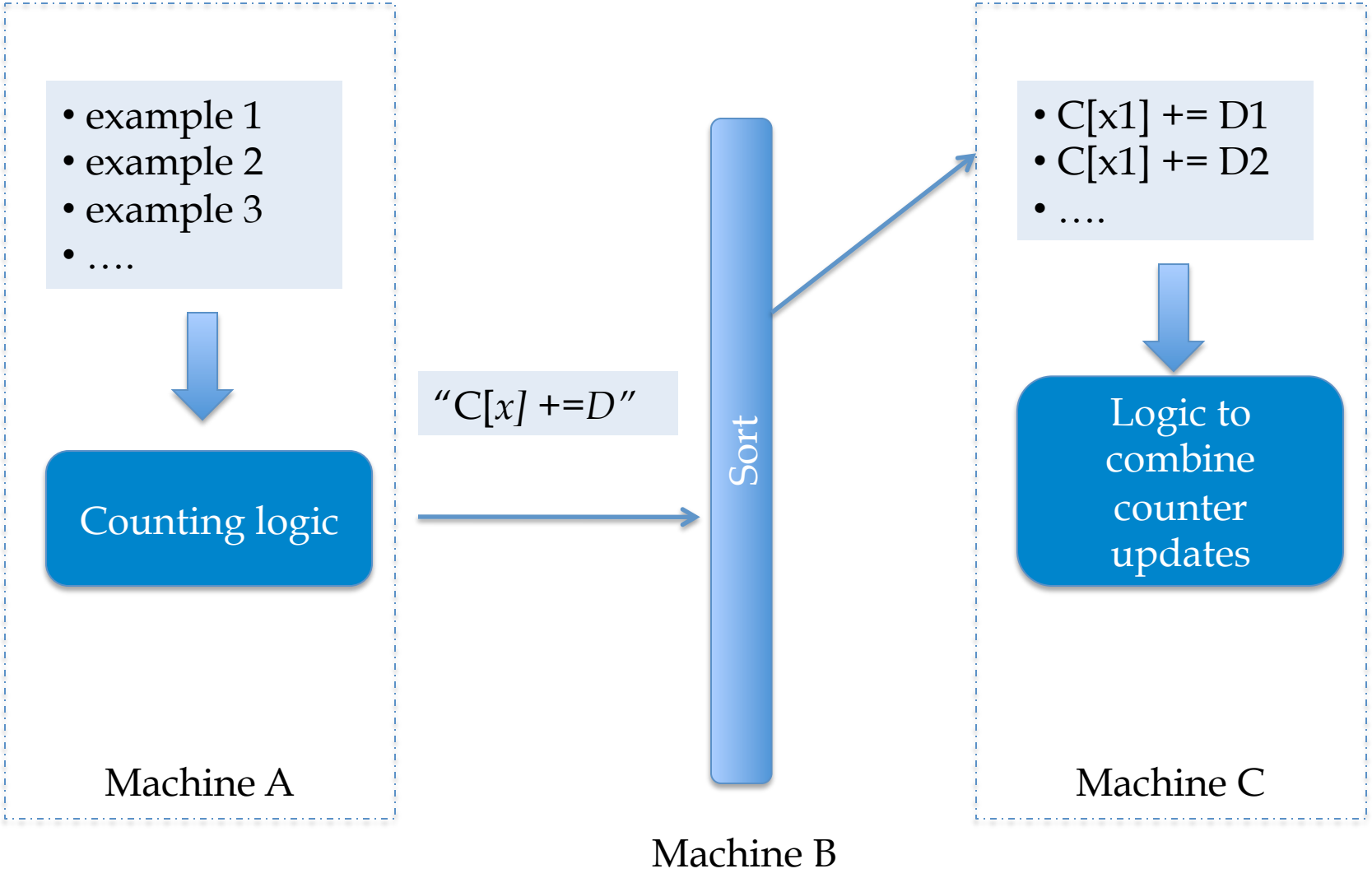
Classification logic

Step 2: stream through and for each test case

$id_i$ $w_{i,1}$ $w_{i,2}$ $w_{i,3}$ …. $w_{i,ki}$

**request** the event counters needed to classify $id_i$ from the event-count DB, then classify using the **answers**

# Recall: Stream and Sort Counting: sort messages so the recipient can stream through them



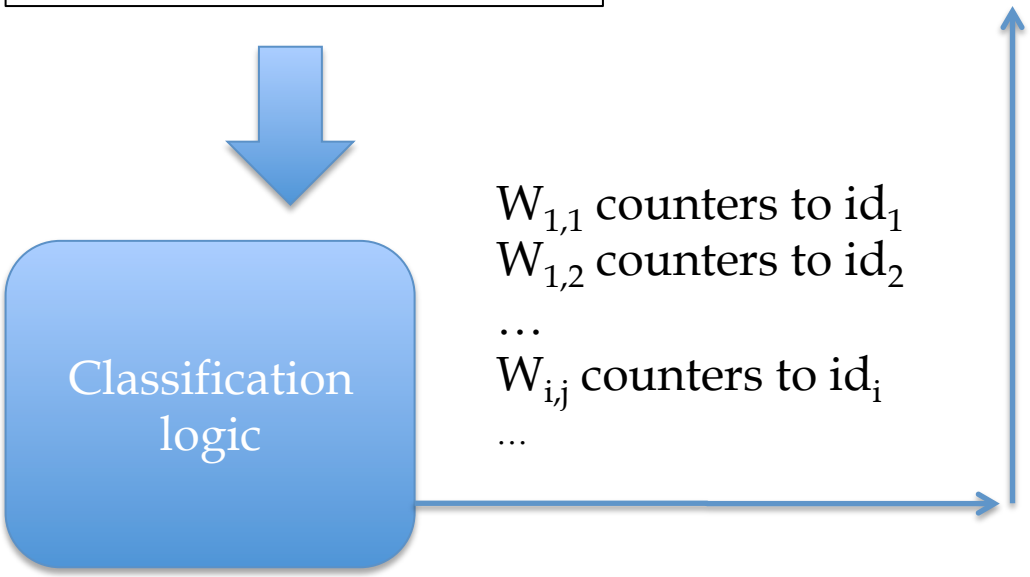
# Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

$id_1$ $w_{1,1}$ $w_{1,2}$ $w_{1,3}$ …. $w_{1,k1}$
$id_2$ $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ ….
$id_3$ $w_{3,1}$ $w_{3,2}$ ….
$id_4$ $w_{4,1}$ $w_{4,2}$ …
$id_5$ $w_{5,1}$ $w_{5,2}$ ….
..

Record of all event counts for each word

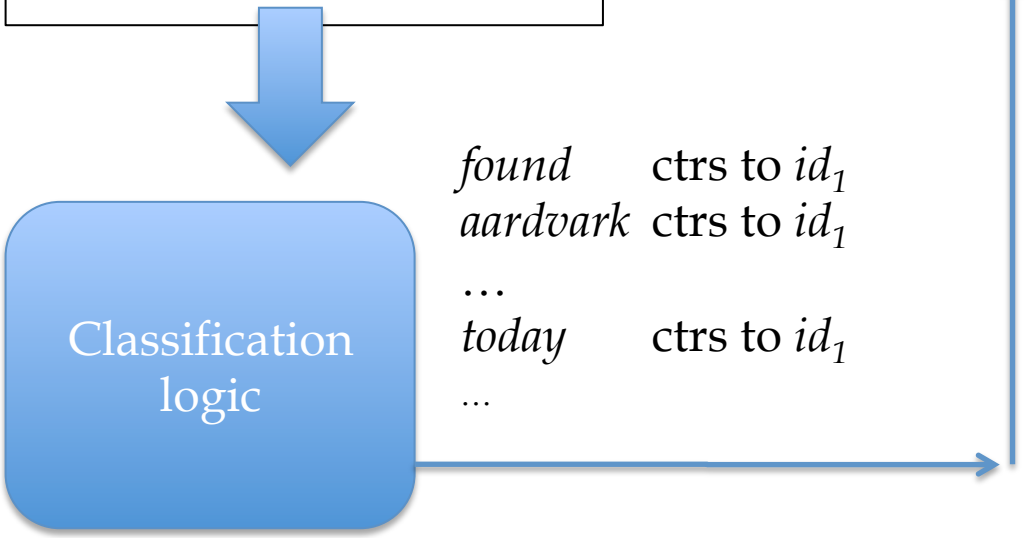
| w | Counts associated with W |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| agent | C[w^Y=sports]=1027,C[w^Y=worl |
| … | … |
| zynga | C[w^Y=sports]=21,C[w^Y=worldN |



# Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

*$id_1$ found an aarvark in zynga's farmville today!*
*$id_2$ …*
*$id_3$ ….*
*$id_4$ …*
*$id_5$ …*
..




Record of all event counts for each word

| w | Counts associated with W |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| agent | C[w^Y=sports]=1027,C[w^Y=worl |
| … | … |
| zynga | C[w^Y=sports]=21,C[w^Y=worldN |

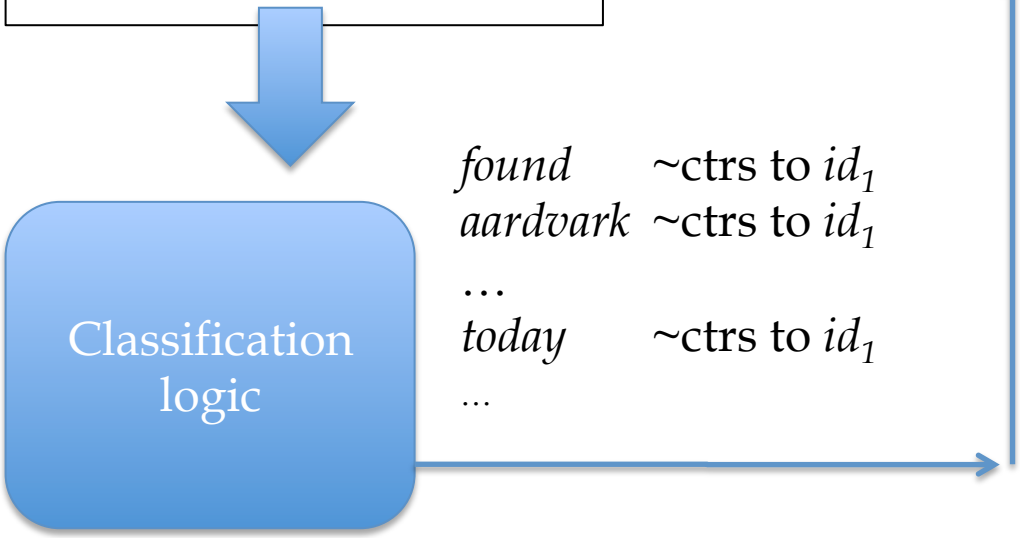
# Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

*$id_1$ found an aarvark in zynga's farmville today!*
*$id_2$ …*
*$id_3$ ….*
*$id_4$ …*
*$id_5$ …*
..

Record of all event counts for each word

| w | Counts associated with W |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| agent | C[w^Y=sports]=1027,C[w^Y=worl |
| … | … |
| zynga | C[w^Y=sports]=21,C[w^Y=worldN |




~ is the last ascii character

% export LC_COLLATE=C

means that it will sort *after* anything else with unix sort

# Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

*$id_1$ found an aardvark in zynga's farmville today!*
*$id_2$ …*
*$id_3$ ….*
*$id_4$ …*
*$id_5$ …*
..

Record of all event counts for each word

| w | Counts associated with W |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| agent | C[w^Y=sports]=1027,C[w^Y=worl |
| … | … |
| zynga | C[w^Y=sports]=21,C[w^Y=worldN |

Counter records

Classification logic

*found* ~ctr to *$id_1$*
*aardvark* ~ctr to $id_2$
…
*today* ~ctr to $id_i$
…

Combine and sort

requests

# A stream-and-sort analog of the request-and-answer pattern…

Record of all event counts for each word

| w | Counts |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| agent | … |
| … | |
| zynga | … |

Counter records

| w | Counts |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| aardvark | ~ctr to id1 |
| agent | C[w^Y=sports]=… |
| agent | ~ctr to id345 |
| agent | ~ctr to id9854 |
| … | ~ctr to id345 |
| agent | ~ctr to id34742 |
| … | |
| zynga | C[…] |
| zynga | ~ctr to id1 |

*found* ~ctr to $id_1$
*aardvark* ~ctr to $id_1$
…
*today* ~ctr to $id_1$
…

Combine and sort

requests

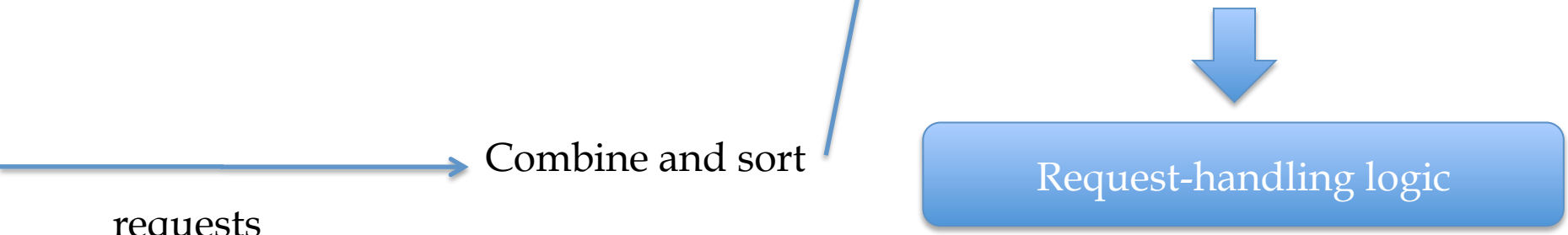
Request-handling logic

# A stream-and-sort analog of the request-and-answer pattern…

- previousKey = somethingImpossible
- For each (*key,val*) in input:
  - If *key*==previousKey
    - Answer(recordForPrevKey,*val*)
  - Else
    - previousKey = *key*
    - recordForPrevKey = *val*

define Answer(*record,request*):
- find *id* where "*request* = ~ctr to *id*"
- print "*id* ~ctr for *request* is *record*"

| w | Counts |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| aardvark | ~ctr to id1 |
| agent | C[w^Y=sports]=… |
| agent | ~ctr to id345 |
| agent | ~ctr to id9854 |
| … | ~ctr to id345 |
| agent | ~ctr to id34742 |
| … | |
| zynga | C[…] |
| zynga | ~ctr to id1 |



# A stream-and-sort analog of the request-and-answer pattern…

•previousKey = somethingImpossible
• For each (*key,val*) in input:
  • If *key*==previousKey
    • Answer(recordForPrevKey,*val*)
  • Else
    • previousKey = *key*
    • recordForPrevKey = *val*

define Answer*(record,request)*:
• find *id* where "*request* = ~ctr to *id*"
• print "*id* ~ctr for *request* is *record*"

Output:
*id1* ~ctr for *aardvark* is C[w^Y=sports]=2
…
*id1* ~ctr for *zynga* is ….
…

| w | Counts |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| aardvark | ~ctr to id1 |
| agent | C[w^Y=sports]=… |
| agent | ~ctr to id345 |
| agent | ~ctr to id9854 |
| … | ~ctr to id345 |
| agent | ~ctr to id34742 |
| … | |
| zynga | C[…] |
| zynga | ~ctr to id1 |

Combine and sort

requests

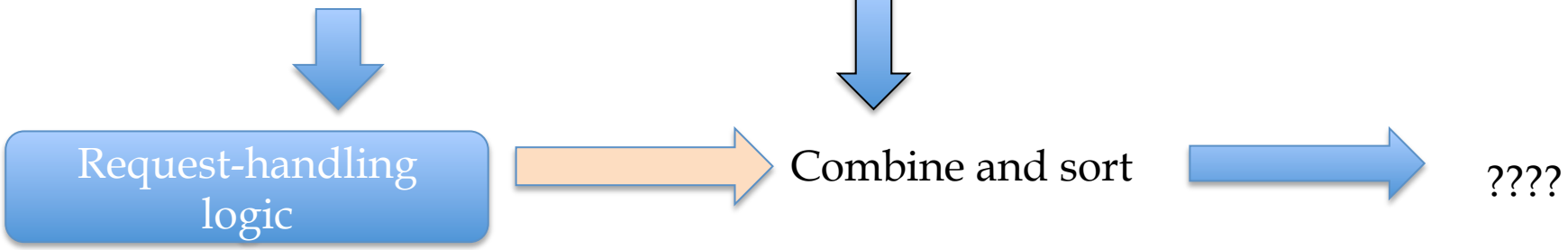
Request-handling logic

# A stream-and-sort analog of the request-and-answer pattern…

| w | Counts |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| aardvark | ~ctr to id1 |
| agent | C[w^Y=sports]=… |
| agent | ~ctr to id345 |
| agent | ~ctr to id9854 |
| … | ~ctr to id345 |
| agent | ~ctr to id34742 |
| … | |
| zynga | C[…] |
| zynga | ~ctr to id1 |

Output:
*id1* ~ctr for *aardvark* is C[w^Y=sports]=2
…
*id1* ~ctr for *zynga* is ….
…

$id_1$ *found an aardvark in zynga's farmville today!*
$id_2$ …
$id_3$ ….
$id_4$ …
$id_5$ …
..



What we'd wanted

| | |
|---|---|
| $id_1$ $w_{1,1}$ $w_{1,2}$ $w_{1,3}$ …. $w_{1,k1}$ | $C[X=w_{1,1}\text{^}Y=sports]=5245$, $C[X=w_{1,1}\text{^}Y=..]$,$C[X=w_{1,2}\text{^}…]$ |
| $id_2$ $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ …. | $C[X=w_{2,1}\text{^}Y=….]=1054$,…, $C[X=w_{2,k2}\text{^}…]$ |
| $id_3$ $w_{3,1}$ $w_{3,2}$ …. | $C[X=w_{3,1}\text{^}Y=….]$=… |
| $id_4$ $w_{4,1}$ $w_{4,2}$ … | … |
| | |

What we ended up with

| Key | Value |
|---|---|
| *id1* | *found aardvark zynga farmville today* |
| | ~ctr for *aardvark* is C[w^Y=sports]=2 |
| | ~ctr for *found* is C[w^Y=sports]=1027,C[w^Y=worldNews]=564 |
| | … |
| *id2* | $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ …. |
| | ~ctr for $w_{2,1}$ is … |
| … | … |

# Implementation summary

java CountForNB train.dat … > eventCounts.dat
java CountsByWord eventCounts.dat | sort
| java CollectRecords > words.dat

java requestWordCounts test.dat
| cat - words.dat | sort | java answerWordCountRequests
| cat - test.dat| sort | testNBUsingRequests

train.dat

$id_1$ $w_{1,1}$ $w_{1,2}$ $w_{1,3}$ …. $w_{1,k1}$
$id_2$ $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ ….
$id_3$ $w_{3,1}$ $w_{3,2}$ ….
$id_4$ $w_{4,1}$ $w_{4,2}$ …
$id_5$ $w_{5,1}$ $w_{5,2}$ ….
..

counts.dat

| | |
|---|---|
| *X=w1^Y=sports* | *5245* |
| *X=w1^Y=worldNews* | *1054* |
| *X=..* | *2120* |
| *X=w2^Y=…* | *37* |
| *X=…* | *3* |
| … | … |

# Implementation summary

java CountForNB train.dat … > eventCounts.dat
java CountsByWord eventCounts.dat | sort
| java CollectRecords > words.dat

java requestWordCounts test.dat
| cat - words.dat | sort | java answerWordCountRequests
| cat - test.dat| sort | testNBUsingRequests

words.dat

| w | Counts associated with W |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| agent | C[w^Y=sports]=1027,C[w^Y=worldNews]=564 |
| … | … |
| zynga | C[w^Y=sports]=21,C[w^Y=worldNews]=4464 |

# Implementation summary

java CountForNB train.dat … > eventCounts.dat
java CountsByWord eventCounts.dat | sort
| java CollectRecords > words.dat

java requestWordCounts test.dat
| cat - **words.dat** | sort | java answerWordCountRequests
| cat - test.dat| sort | testNBUsingRequests

**output looks like this**

**input looks like this**

*found* ~ctr to $id_1$
*aardvark* ~ctr to $id_2$
…
*today* ~ctr to $id_i$
…

**words.dat**

| w | Counts |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| agent | … |
| … | |
| zynga | … |

| w | Counts |
|---|---|
| aardvark | C[w^Y=sports]=2 |
| aardvark | ~ctr to id1 |
| agent | C[w^Y=sports]=… |
| agent | ~ctr to id345 |
| agent | ~ctr to id9854 |
| … | ~ctr to id345 |

# Implementation summary

```
java CountForNB train.dat … > eventCounts.dat
java CountsByWord eventCounts.dat | sort
 | java CollectRecords   > words.dat

java requestWordCounts  test.dat
 | cat - words.dat | sort | java answerWordCountRequests
 | cat - test.dat | sort | testNBUsingRequests
```

**Output looks like this**

Output:
*id1* ~ctr for *aardvark* is C[w^Y=sports]=2
…
*id1* ~ctr for *zynga* is ….
…

**test.dat**

$id_1$ *found an aardvark in zynga's farmville today!*
$id_2$ …
$id_3$ ….
$id_4$ …
$id_5$ …
..

# Implementation summary

java CountForNB train.dat … > eventCounts.dat
java CountsByWord eventCounts.dat | sort
| java CollectRecords > words.dat

java requestWordCounts test.dat
| cat - **words.dat** | sort | java answerWordCountRequests
| cat -**test.dat**| sort | testNBUsingRequests

**Input looks like this**

| Key | Value |
|---|---|
| id1 | found aardvark zynga farmville today |
| | ~ctr for aardvark is C[w^Y=sports]=2 |
| | ~ctr for found is C[w^Y=sports]=1027,C[w^Y=worldNews]=564 |
| | … |
| id2 | $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ …. |
| | ~ctr for $w_{2,1}$ is … |
| … | … |

# Outline

- Even more on stream-and-sort and naïve Bayes
- **Another problem: "meaningful" phrase finding**
- Implementing phrase finding efficiently
- Some other phrase-related problems

# A Language Model Approach to Keyphrase Extraction

**Takashi Tomokiyo and Matthew Hurst**
Applied Research Center
Intelliseek, Inc.
Pittsburgh, PA 15213
{ttomokiyo,mhurst}@intelliseek.com

# A Language Model Approach to Keyphrase Extraction

**Takashi Tomokiyo and Matthew Hurst**
Applied Research Center
Intelliseek, Inc.
Pittsburgh, PA 15213
{ttomokiyo,mhurst}@intelliseek.com

ACL Workshop 2003

| | | | |
|---|---|---|---|
| 1 | civic hybrid | 21 | mustang gt |
| 2 | honda civic hybrid | 22 | ford escape |
| 3 | toyota prius | 23 | steering wheel |
| 4 | electric motor | 24 | toyota prius today |
| 5 | honda civic | 25 | electric motors |
| 6 | fuel cell | 26 | gasoline engine |
| 7 | hybrid cars | 27 | internal combustion engine |
| 8 | honda insight | 28 | gas engine |
| 9 | battery pack | 29 | front wheels |
| 10 | sports car | 30 | key sense wire |
| 11 | civic si | 31 | civic type r |
| 12 | hybrid car | 32 | test drive |
| 13 | civic lx | 33 | street race |
| 14 | focus fcv | 34 | united states |
| 15 | fuel cells | 35 | hybrid powertrain |
| 16 | hybrid vehicles | 36 | rear bumper |
| 17 | tour de sol | 37 | ford focus |
| 18 | years ago | 38 | detroit auto show |
| 19 | daily driver | 39 | parking lot |
| 20 | jetta tdi | 40 | rear wheels |

Figure 1: Top 40 keyphrases automatically extracted from messages relevant to "*civic hybrid*" using our system

# Why phrase-finding?

- There are lots of phrases
- There's not supervised data
- It's hard to articulate
  - What makes a phrase a phrase, *vs* just an n-gram?
    - a phrase is independently meaningful ("test drive", "red meat") or not ("are interesting", "are lots")
  - What makes a phrase interesting?

# The breakdown: what makes a good phrase

- Two properties:
  - Phraseness: "the degree to which a given word sequence is considered to be a phrase"
    - Statistics: how often words co-occur together vs separately
  - Informativeness: "how well a phrase captures or illustrates the key ideas in a set of documents" – something novel and important **relative to a domain**
    - Background corpus and foreground corpus; how often phrases occur in each

# "Phraseness"$_1$ – based on BLRT

- Binomial Ratio Likelihood Test (BLRT):
  - Draw samples:
    - $n_1$ draws, $k_1$ successes
    - $n_2$ draws, $k_2$ successes
    - Are they from one binominal (i.e., $k_1/n_1$ and $k_2/n_2$ were different due to chance) or from two distinct binomials?
  - Define
    - $p_1=k_1/n_1$, $p_2=k_2/n_2$, $p=(k_1+k_2)/(n_1+n_2)$,
    - $L(p,k,n) = p^k(1-p)^{n-k}$

$$BLRT(n_1,k_1,n_2,k_2) = \frac{L(p_1,k_1,n_1)L(p_2,k_2,n_2)}{L(p,k_1,n_1)L(p,k_2,n_2)}$$

# "Phraseness"$_1$ – based on BLRT

- Binomial Ratio Likelihood Test (BLRT):
  - Draw samples:
    - $n_1$ draws, $k_1$ successes
    - $n_2$ draws, $k_2$ successes
    - Are they from one binominal (i.e., $k_1/n_1$ and $k_2/n_2$ were different due to chance) or from two distinct binomials?
  - Define
    - $p_i=k_i/n_i$, $p=(k_1+k_2)/(n_1+n_2)$,
    - $L(p,k,n) = p^k(1-p)^{n-k}$

$$BLRT(n_1,k_1,n_2,k_2) = 2\log\frac{L(p_1,k_1,n_1)L(p_2,k_2,n_2)}{L(p,k_1,n_1)L(p,k_2,n_2)}$$

# "Phraseness"$_1$ – based on BLRT

- Define
  - $p_i=k_i/n_i$, $p=(k_1+k_2)/(n_1+n_2)$,
  - $L(p,k,n) = p^k(1-p)^{n-k}$

Phrase *x y*: $W_1=x$ ^ $W_2=y$

$$\varphi_p(n_1,k_1,n_2,k_2) = 2\log\frac{L(p_1,k_1,n_1)L(p_2,k_2,n_2)}{L(p,k_1,n_1)L(p,k_2,n_2)}$$

| | | comment |
|---|---|---|
| $k_1$ | C($W_1$=x ^ $W_2$=y) | how often bigram *x y* occurs in corpus C |
| $n_1$ | C($W_1$=x) | how often word *x* occurs in corpus C |
| $k_2$ | C($W_1$≠x^$W_2$=y) | how often *y* occurs in C after a non-*x* |
| $n_2$ | C($W_1$≠x) | how often a non-*x* occurs in C |

Does *y* occur at the same frequency after *x* as in other positions?

# "Informativeness"$_1$ – based on BLRT

- Define
  - $p_i=k_i/n_i$, $p=(k_1+k_2)/(n_1+n_2)$,
  - $L(p,k,n) = p^k(1-p)^{n-k}$

Phrase *x y*: $W_1=x$ ^ $W_2=y$ and two corpora, C and B

$$\varphi_i(n_1,k_1,n_2,k_2) = 2\log\frac{L(p_1,k_1,n_1)L(p_2,k_2,n_2)}{L(p,k_1,n_1)L(p,k_2,n_2)}$$

| | | comment |
|---|---|---|
| $k_1$ | C($W_1$=x ^ $W_2$=y) | how often bigram *x y* occurs in corpus C |
| $n_1$ | C($W_1$=* ^ $W_2$=*) | how many bigrams in corpus C |
| $k_2$ | B($W_1$=x^$W_2$=y) | how often *x y* occurs in **background corpus** |
| $n_2$ | B($W_1$=* ^ $W_2$=*) | how many bigrams in background corpus |

Does x *y* occur at the same frequency in both corpora?

# The breakdown: what makes a good phrase

- "Phraseness" and "informativeness" are then combined with a tiny classifier, tuned on labeled data.

$$\varphi = \frac{1}{1 + \exp(-a\varphi_p - b\varphi_i + c)}$$

$$\left(\log \frac{p}{1-p} = s\right) \Leftrightarrow \left(p = \frac{1}{1+e^s}\right)$$

  - Background corpus: 20 newsgroups dataset (20k messages, 7.4M words)
  - Foreground corpus: rec.arts.movies.current-films June-Sep 2002 (4M words)
  - Results?

| | |
|---|---|
| 1 | message news |
| 2 | minority report |
| 3 | star wars |
| 4 | john harkness |
| 5 | derek janssen |
| 6 | robert frenchu |
| 7 | sean o'hara |
| 8 | box office |
| 9 | dawn taylor |
| 10 | anthony gaza |
| 11 | star trek |
| 12 | ancient race |
| 13 | scooby doo |
| 14 | austin powers |
| 15 | home.attbi.com hey |
| 16 | sixth sense |
| 17 | hey kids |
| 18 | gaza man |
| 19 | lee harrison |
| 20 | years ago |
| 21 | julia roberts |
| 22 | national guard |
| 23 | bourne identity |
| 24 | metrotoday www.zap2it.com |
| 25 | starweek magazine |
| 26 | eric chomko |
| 27 | wilner starweek |
| 28 | tim gueguen |
| 29 | jodie foster |
| 30 | johnnie kendricks |

# The breakdown: what makes a good phrase

- Two properties:
  - Phraseness: "the degree to which a given word sequence is considered to be a phrase"
    - Statistics: how often words co-occur together vs separately
  - Informativeness: "how well a phrase captures or illustrates the key ideas in a set of documents" – something novel and important relative to a domain
    - Background corpus and foreground corpus; how often phrases occur in each
  - Another intuition: our goal is to compare distributions and see how **different** they are:
    - Phraseness: estimate $x\ y$ with bigram model or unigram model
    - Informativeness: estimate with foreground vs background corpus

# The breakdown: what makes a good phrase

- Another intuition: our goal is to compare distributions and see how **different** they are:
  - Phraseness: estimate $x\ y$ with bigram model or unigram model
  - Informativeness: estimate with foreground vs background corpus
- To compare distributions, use KL-divergence

$$D(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

"Pointwise KL divergence"

$$\delta_{\mathbf{w}}(p \parallel q) \stackrel{\text{def}}{=} p(\mathbf{w}) \log \frac{p(\mathbf{w})}{q(\mathbf{w})}$$

# The breakdown: what makes a good phrase

– To compare distributions, use KL-divergence

$$D(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

"Pointwise KL divergence"

$$\delta_{\mathbf{w}}(p \parallel q) \stackrel{\text{def}}{=} p(\mathbf{w}) \log \frac{p(\mathbf{w})}{q(\mathbf{w})}$$

Phraseness: difference between bigram and unigram language model in foreground

$$\delta_{\mathbf{w}}(LM^N_{\text{fg}} \parallel LM^1_{\text{fg}})$$

Bigram model: P(x y)=P(x)P(y | x)

Unigram model: P(x y)=P(x)P(y)

# The breakdown: what makes a good phrase

– To compare distributions, use KL-divergence

$$D(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

"Pointwise KL divergence"

Informativeness: difference between foreground and background models

$$\delta_{\mathbf{w}}(p \parallel q) \stackrel{\text{def}}{=} p(\mathbf{w}) \log \frac{p(\mathbf{w})}{q(\mathbf{w})}$$

$\delta_{\mathbf{w}}(LM^N_{\text{fg}} \parallel LM^N_{\text{bg}})$, or

$\delta_{\mathbf{w}}(LM^1_{\text{fg}} \parallel LM^1_{\text{bg}})$

Bigram model: P(x y)=P(x)P(y | x)

Unigram model: P(x y)=P(x)P(y)

$\delta_{\mathbf{w}}(LM^N_{\text{fg}} \parallel LM^1_{\text{bg}})$

# The breakdown: what makes a good phrase

– To compare distributions, use KL-divergence

$$D(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

"Pointwise KL divergence"

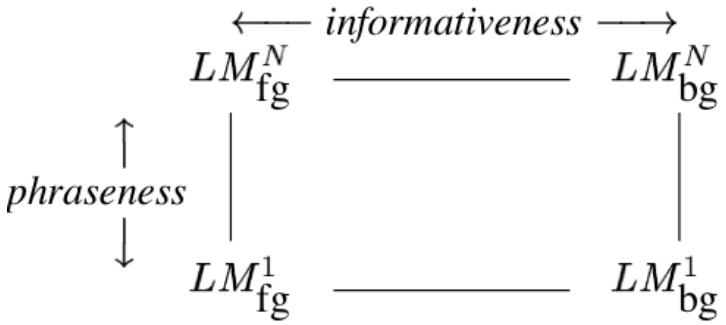
$$\delta_{\mathbf{w}}(p \parallel q) \stackrel{\text{def}}{=} p(\mathbf{w}) \log \frac{p(\mathbf{w})}{q(\mathbf{w})}$$

Combined: difference between foreground bigram model and background unigram model

$$\delta_{\mathbf{w}}(LM^N_{\text{fg}} \parallel LM^1_{\text{bg}})$$

Bigram model: P(x y)=P(x)P(y | x)

Unigram model: P(x y)=P(x)P(y)



# The breakdown: what makes a good phrase

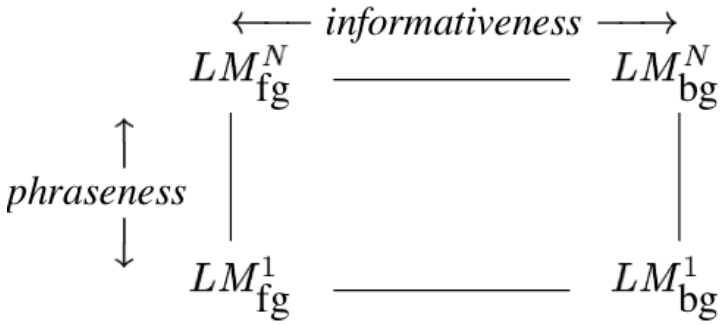
– To compare distributions, use KL-divergence

Subtle advantages:

- BLRT scores "more frequent in foreground" and "more frequent in background" symmetrically, pointwise KL does not.
- Phrasiness and informativeness scores are more comparable – straightforward combination w/o a classifier is reasonable.
- Language modeling is well-studied:
  - extensions to n-grams, smoothing methods, …
  - we can build on this work in a modular way

Combined: difference between foreground bigram model and background unigram model

$$\delta_{\mathbf{w}}(LM^{N}_{\text{fg}} \parallel LM^{1}_{\text{bg}})$$



# Pointwise KL, combined

| | | | |
|---|---|---|---|
| 1 | message news | 16 | hey kids |
| 2 | minority report | 17 | years ago |
| 3 | star wars | 18 | gaza man |
| 4 | john harkness | 19 | sixth sense |
| 5 | robert frenchu | 20 | lee harrison |
| 6 | derek janssen | 21 | julia roberts |
| 7 | box office | 22 | national guard |
| 8 | sean o'hara | 23 | bourne identity |
| 9 | dawn taylor | 24 | metrotoday www.zap2it.com |
| 10 | anthony gaza | 25 | starweek magazine |
| 11 | star trek | 26 | eric chomko |
| 12 | ancient race | 27 | wilner starweek |
| 13 | home.attbi.com hey | 28 | tim gueguen |
| 14 | scooby doo | 29 | jodie foster |
| 15 | austin powers | 30 | kevin filmnutboy |

# Why phrase-finding?

- Phrases are where the standard supervised "bag of words" representation starts to break.
- There's not supervised data, so it's hard to see what's "right" and why
- It's a nice example of using unsupervised signals to solve a task that could be formulated as supervised learning
- It's a nice level of complexity, if you want to do it in a scalable way.

# Implementation

- Request-and-answer pattern
  - Main data structure: tables of key-value pairs
    - *key* is a phrase *x y*
    - *value* is a mapping from a attribute names (like *phraseness*, *freq-in-B*, …) to numeric values.
  - Keys and values are just strings
  - We'll operate mostly by sending messages to this data structure and getting results back, or else streaming thru the whole table
  - For really big data: we'd also need tables where *key* is a word and *val* is set of attributes of the word *(freq-in-B, freq-in-C, …)*

| Key | Value |
|---|---|
| old man | freq(B)=10,freq(C)=13,informativeness=1.3,phrasiness=740 |
| bad service | freq(B)=8,freq(C)=25,informativeness=560,phrasiness=254 |
| ... | ... |

# Generating and scoring phrases: 1

- Stream through **foreground** corpus and count events "$W_1$=*x* ^ $W_2$=*y*" the same way we do in training naive Bayes: stream-and sort and accumulate deltas (a "sum-reduce")
  - Don't bother generating boring phrases (e.g., crossing a sentence, contain a stopword, …)
- Then stream through the output and convert to *phrase, attributes-of-phrase* records with one attribute: *freq-in-C=n*
- Stream through foreground corpus and count events "$W_1$=*x*" in a (memory-based) hashtable….
- This is enough* to compute phrasiness:
  - $\psi_p(x\ y) = f(\textit{freq-in-C}(x), \textit{freq-in-C}(y), \textit{freq-in-C}(x\ y))$
- …so you can do that with a scan through the phrase table that **adds** an extra attribute (holding word frequencies in memory).

* actually you also need total # words and total #phrases….

# Generating and scoring phrases: 2

- Stream through **background** corpus and count events "$W_1$=*x* ^ $W_2$=*y*" and convert to *phrase, attributes-of-phrase* records with one attribute: *freq-in-**B**=n*
- Sort the two phrase-tables: *freq-in-B* and *freq-in-C* and run the output through another "reducer" that
  - **appends** together all the attributes associated with the same key, so we now have elements like

| Key | Value |
|---|---|
| old man | freq(B)=10,freq(C)=13,phrasiness=740 |
| bad service | freq(B)=8,freq(C)=25,phrasiness=254 |
| ... | ... |

# Generating and scoring phrases: 3

- Scan the through the phrase table one more time and add the informativeness attribute and the overall quality attribute

| Key | Value |
|---|---|
| old man | freq(B)=10,freq(C)=13,informativeness=1.3,phrasiness=740 |
| bad service | freq(B)=8,freq(C)=25,informativeness=560,phrasiness=254 |
| ... | ... |

Summary, assuming word vocabulary $n_W$ is small:

- Scan foreground corpus C for phrases: $O(n_C)$ producing $m_C$ phrase records – of course $m_C << n_C$
- Compute phrasiness: $O(m_C)$ Assumes word counts fit in memory
- Scan background corpus B for phrases: $O(n_B)$ producing $m_B$
- Sort together and combine records: $O(m \log m)$, $m=m_B + m_C$
- Compute informativeness and combined quality: $O(m)$

# Ramping it up – keeping word counts out of memory

- Goal: records for *xy* with attributes *freq-in-B, freq-in-C, freq-of-x-in-C, freq-of-y-in-C, …*
- Assume I have built built phrase tables and word tables….how do I incorporate the word attributes into the phrase records?
- For each phrase *xy,* request necessary word frequencies:
  - Print "*x* ~request=freq-in-C,from=*xy*"
  - Print "*y* ~request=freq-in-C,from=*xy*"
- Sort all the word requests in with the word tables
- Scan through the result and generate the answers: for each word *w*, $a_1$=$n_1$,$a_2$=$n_2$,….
  - Print "*xy* ~request=freq-in-C,from=*w*"
- Sort the answers in with the *xy* records
- Scan through and augment the *xy* records appropriately

# Generating and scoring phrases: 3

Summary

1. Scan foreground corpus C for phrases, words: $O(n_C)$ producing $m_C$ phrase records, $v_C$ word records
2. Scan phrase records producing word-freq requests: $O(m_C)$ producing $2m_C$ requests
3. Sort requests with word records: $O((2m_C + v_C)\log(2m_C + v_C)) = O(m_C \log m_C)$ since $v_C < m_C$
4. Scan through and answer requests: $O(m_C)$
5. Sort answers with phrase records: $O(m_C \log m_C)$
6. Repeat 1-5 for background corpus: $O(n_B + m_B \log m_B)$
7. Combine the two phrase tables: $O(m \log m)$, $m = m_B + m_C$
8. Compute all the statistics: $O(m)$

# More cool work with phrases

- Turney: Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews.ACL '02.
- Task: review classification (65-85% accurate, depending on domain)
  - Identify candidate phrases (e.g., adj-noun bigrams, using POS tags)
  - Figure out the **semantic orientation** of each phrase using "pointwise mutual information" and aggregate

$$PMI(w_1, w_2) = log_2(p(w_1 \; and \; w_2)/p(w_1)p(w_2))$$

SO(phrase) = PMI(phrase,'excellent') − PMI(phrase,'poor')

$$SO(phrase) = log_2(\frac{hits(phrase \; NEAR \; 'excellent')hits('excellent')}{hits(phrase \; NEAR \; 'poor')hits('poor')})$$

Table 2. An example of the processing of a review that the author has classified as *recommended*.[6]

| Extracted Phrase | Part-of-Speech Tags | Semantic Orientation |
|---|---|---|
| online experience | JJ NN | 2.253 |
| low fees | JJ NNS | 0.333 |
| local branch | JJ NN | 0.421 |
| small part | JJ NN | 0.053 |
| online service | JJ NN | 2.780 |
| printable version | JJ NN | -0.705 |
| direct deposit | JJ NN | 1.288 |
| well other | RB JJ | 0.237 |
| inconveniently located | RB VBN | -1.541 |
| other bank | JJ NN | -0.850 |
| true service | JJ NN | -0.732 |
| Average Semantic Orientation | | 0.322 |

Table 3. An example of the processing of a review that the author has classified as *not recommended*.

| Extracted Phrase | Part-of-Speech Tags | Semantic Orientation |
|---|---|---|
| little difference | JJ NN | -1.615 |
| clever tricks | JJ NNS | -0.040 |
| programs such | NNS JJ | 0.117 |
| possible moment | JJ NN | -0.668 |
| unethical practices | JJ NNS | -8.484 |
| low funds | JJ NNS | -6.843 |
| old man | JJ NN | -2.566 |
| other problems | JJ NNS | -2.748 |
| probably wondering | RB VBG | -1.830 |
| virtual monopoly | JJ NN | -2.050 |
| other bank | JJ NN | -0.850 |
| extra day | JJ NN | -0.286 |
| direct deposits | JJ NNS | 5.771 |
| online web | JJ NN | 1.936 |
| cool thing | JJ NN | 0.395 |
| very handy | RB JJ | 1.349 |
| lesser evil | RBR JJ | -2.288 |
| Average Semantic Orientation | | -1.218 |

$$SO(phrase) = log_2(\frac{hits(phrase\ NEAR\ 'excellent')hits('excellent')}{hits(phrase\ NEAR\ 'poor')hits('poor')})$$

# "Answering Subcognitive Turing Test Questions: A Reply to French" - Turney

Robert French (1990, 2000) has argued that a disembodied computer cannot pass a Turing Test that includes *subcognitive* questions. He wrote (French, 2000):

> No computer that had not experienced the world as we humans had could pass a rigorously administered standard Turing Test. We show that the use of "subcognitive" questions allows the standard Turing Test to indirectly probe the human subcognitive associative concept network built up over a lifetime of experience with the world.

On a scale of 1 (awful) to 10 (excellent), please rate:

- How good is the name *Flugly* for a glamorous Hollywood actress?
- How good is the name *Flugly* for an accountant in a W.C. Fields movie?
- How good is the name *Flugly* for a child's teddy bear?

$$p(\text{Flu*} \mid \text{actress}) = \frac{\text{hits}((\text{actress NEAR Flu*}) \text{ AND glamorous})}{\text{hits}(\text{actress AND glamorous})}$$

LOW

$$p(\text{Flu*} \mid \text{accountant}) = \frac{\text{hits}((\text{accountant NEAR Flu*}) \text{ AND movie})}{\text{hits}(\text{accountant AND movie})}$$

HIGHER

$$p(\text{Flu*} \mid \text{bear}) = \frac{\text{hits}((\text{bear NEAR Flu*}) \text{ AND teddy})}{\text{hits}(\text{bear AND teddy})}$$

HIGHEST

On a scale of 1 (terrible) to 10 (excellent), please rate:

- banana peels as musical instruments
- coconut shells as musical instruments
- radios as musical instruments

Please rate the following smells (1 = very bad, 10 = very nice):

- Newly cut grass
- Freshly baked bread
- A wet bath towel
- The ocean
- A hospital corridor

On a scale of 1 (terrible) to 10 (excellent), please rate:

- banana peels as musical instruments
- coconut shells as musical instruments
- radios as musical instruments

Please rate the follo

- Newly cut grass
- Freshly baked b
- A wet bath towe
- The ocean
- A hospital corri

$p(\text{musical instruments} \mid \text{banana peels}) = 1 / 2{,}998 = 0.00033$

$p(\text{musical instruments} \mid \text{coconut shells}) = 5 / 1{,}880 = 0.0027$

$p(\text{musical instruments} \mid \text{radios}) = 1{,}253 / 1{,}006{,}207 = 0.0012$

$$p(\text{nice} \mid \text{newly cut grass}) = \frac{\text{hits}((\text{newly cut grass NEAR nice}) \text{ AND smell AND NOT } ((\text{newly cut grass OR nice}) \text{ NEAR "not"}))}{\text{hits}(\text{newly cut grass AND smell AND NOT } (\text{newly cut grass NEAR "not"}))}$$

Please

- N
- Fr
- A
- Th
- A

Table 2. Query results for questions about smell.

| | | |
|---|---|---|
| p(nice \| newly cut grass) | = 1 / 102 | = 0.0098 |
| p(bad \| newly cut grass) | = 0 / 102 | = 0.0 |
| p(nice \| freshly baked bread) | = 8 / 848 | = 0.0094 |
| p(bad \| freshly baked bread) | = 0 / 848 | = 0.0 |
| p(nice \| wet bath towel) | = 0 / 3 | = 0.0 |
| p(bad \| wet bath towel) | = 0 / 3 | = 0.0 |
| p(nice \| ocean) | = 270 / 45,360 | = 0.0060 |
| p(bad \| ocean) | = 107 / 45,360 | = 0.0024 |
| p(nice \| hospital corridor) | = 0 / 134 | = 0.0 |
| p(bad \| hospital corridor) | = 0 / 134 | = 0.0 |

On a scale of 1 (terrible) to 10 (excellent), please rate:

- banana peels as musical instruments
- coconut shells as musical instruments
- radios as musical instruments

Please rate the following smells (1 = very bad, 10 = very nice):

- Newly cut grass
- Freshly baked bread
- A wet bath towel
- The ocean
- A hospital corridor

- Newly cut grass = 10
- Freshly baked bread = 10
- A wet bath towel = 5
- The ocean = 7
- A hospital corridor = 5

# More cool work with phrases

- Locating Complex Named Entities in Web Text. Doug Downey, Matthew Broadhead, and Oren Etzioni, IJCAI 2007.
- Task: identify complex named entities like “Proctor and Gamble”, “War of 1812”, “Dumb and Dumber”, “Secretary of State William Cohen”, …
- Formulation: decide whether to or not to merge nearby sequences of capitalized words *axb,* using variant of

$$c_k(a,b) = \frac{p(ab)^k}{p(a)p(b)} \Rightarrow g_k(a,b,c) = \frac{p(abc)^k}{p(a)p(b)p(c)}$$

- For k=1, ck is PM (w/o the log). For k=2, ck is “Symmetric Conditional Probability”

# Downey et al results

| | F1 | Recall | Precision |
|---|---|---|---|
| SVMCMM | 0.42 | 0.48 | 0.37 |
| CRF | 0.35 | 0.42 | 0.31 |
| MAN | 0.18 | 0.22 | 0.16 |
| LEX | **0.63 (50%)** | **0.66** | **0.59** |

Table 2: **Performance on Difficult Cases** LEX's F1 score is 50% higher than the nearest competitor, SVMCMM.

| | F1 | Recall | Precision |
|---|---|---|---|
| SVMCMM | 0.96 | 0.96 | 0.96 |
| CRF | 0.94 | 0.94 | 0.95 |
| MAN | 0.97 | 0.96 | 0.98 |
| LEX | 0.97 | 0.97 | 0.97 |
| CAPS | **1.00 (3%)** | **1.00** | **1.00** |

Table 3: **Performance on Easy Cases** All methods perform comparably near the perfect performance of the CAPS baseline; CAPS outperforms LEX and MAN by 3%.

| | F1 | Recall | Precision |
|---|---|---|---|
| SVMCMM | 0.29 | 0.34 | 0.25 |
| CRF | 0.25 | 0.31 | 0.21 |
| MAN | 0.18 | 0.22 | 0.16 |
| LEX | **0.63 (117%)** | **0.66** | **0.60** |

Table 4: **Performance on Unseen Entity Classes (Difficult Cases)** LEX outperforms its nearest competitor (SVMCMM) by 117%.

| | F1 | Recall | Precision |
|---|---|---|---|
| SVMCMM | 0.93 | 0.92 | 0.94 |
| CRF | 0.94 | 0.93 | 0.95 |
| MAN | 0.97 | 0.96 | 0.98 |
| LEX | 0.95 | 0.95 | 0.95 |
| CAPS | **1.00 (3%)** | **1.00** | **1.00** |

Table 5: **Performance on Unseen Entity Classes (Easy Cases)** CAPS outperforms all methods by a small margin, performing 3% better than its nearest competitor (MAN).

| | F1 | Recall | Precision |
|---|---|---|---|
| LEX-PMI | 0.38 | 0.43 | 0.34 |
| LEX-SCP | **0.63 (66%)** | **0.66** | **0.59** |

Table 1: **Performance of LEX using collocation measures PMI and SCP.** SCP outperforms PMI by 66% in F1.

# Outline

- Even more on stream-and-sort and naïve Bayes
  - Request-answer pattern
- Another problem: "meaningful" phrase finding
  - Statistics for identifying phrases (or more generally correlations and differences)
  - Also using foreground and background corpora
- Implementing "phrase finding" efficiently
  - Using request-answer
- Some other phrase-related problems
  - Semantic orientation
  - Complex named entity recognition