Using Large-Vocabulary Classifiers

William W. Cohen

Announcements

Guest lecture next Tuesday: Manik Varma, MSR India

Topic: Extreme Classification

Abstract: The objective in extreme multi-label classification is to learn a classifier that can automatically tag a data point with the most relevant subset of labels from a large label set. Extreme multi-label classification is an important research problem since not only does it enable the tackling of applications with many labels but it also allows the reformulation of ranking and recommendation problems with certain advantages over existing formulations.

Recap - Tuesday

Recap: Naïve Bayes training

- For each example *id, y,* $x_1, \ldots, x_d$ in *train:*
  - Print "$Y=y$ += 1"
  - For j in $1..d$:
    - Print "$Y=y \wedge X=x_j$ += 1"
- Sort the event-counter update "messages"
- Scan and add the sorted messages and output the final counter values

- Initialize hashtable C
- For each example *id, y,* $x_1, \ldots, x_d$ in *train:*
  - C[$Y=y$] += 1
  - For j in $1..d$:
    - C[$Y=y \wedge X=x_j$] += 1
  - If memory is getting full: *output all values from C as messages and re-initialize C*
- Sort the event-counter update "messages"
- Scan and add the sorted messages

"map" "reduce"

```
java MyTrainertrain | sort | java MyCountAdder > model
```

Recap: Naïve Bayes coding suggestion

- Create a hashtable C
- For each example *id, y,* $x_1, \ldots, x_d$ in *train:*
  - C.inc("$Y=y$")
  - For j in $1..d$:
    - C.inc("$Y=y \wedge X=x_j$")

```
class EventCounter {
    void inc(String event) {
        // increment the right hashtable slot
        if (hashtable.size()>BUFFER_SIZE) {
            for (e,n) in hashtable.entries : print e + "\t" + n
            hashtable.clear();
        }
    }
}
```

Testing Large-vocab Naïve Bayes

[For assignment]

- 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: $O(|V|)$

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

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

Time: $O(n_2)$
Memory: same

Alternative Large-Vocabulary Naïve Bayes

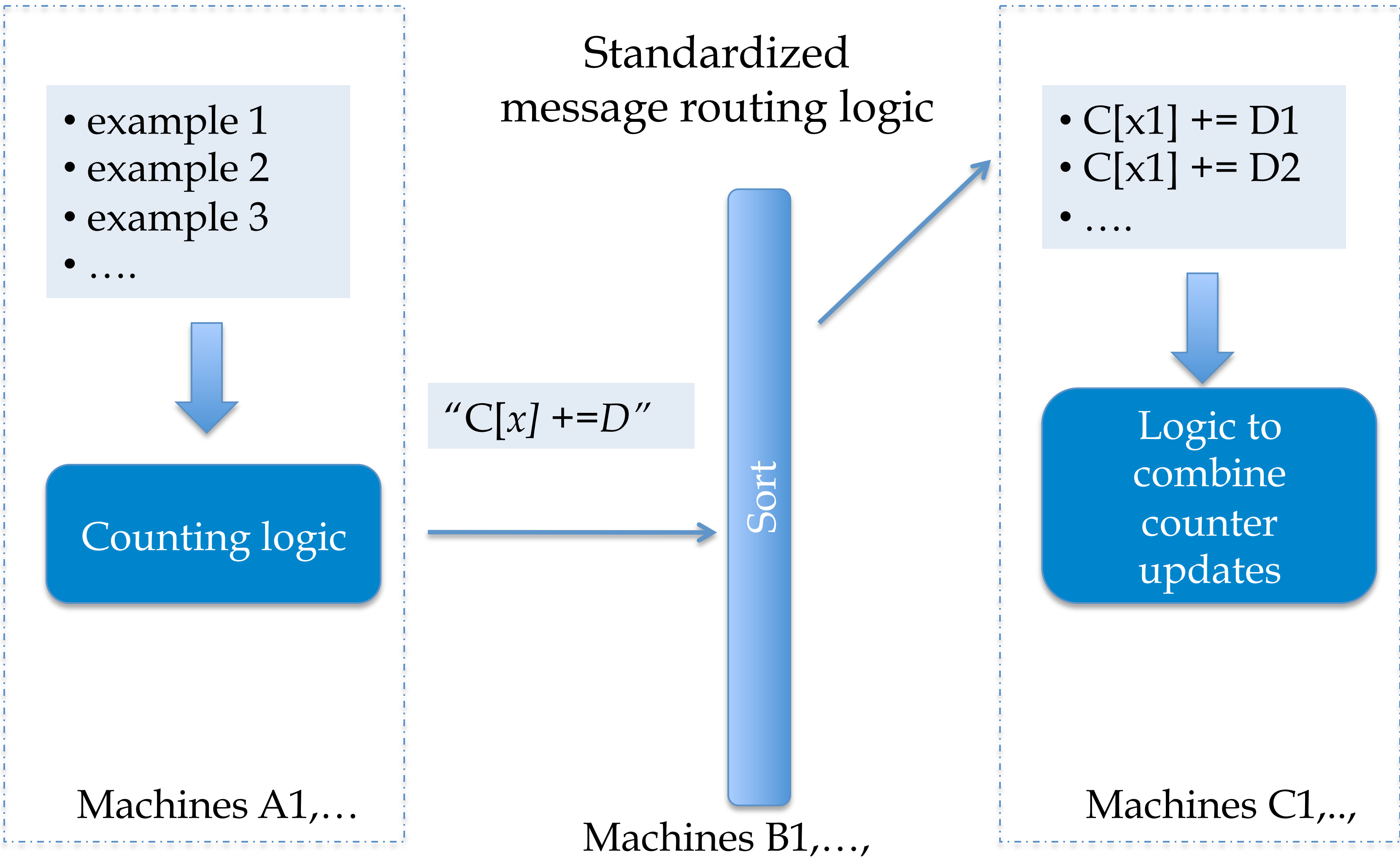
Learning/Counting

- Counts on disk with a key-value store
- Counts as messages to a set of distributed processes
- Repeated scans to build up partial counts
- Counts as messages in a stream-and-sort system
- **Assignment: Counts as messages but buffered in memory**

Using Counts

- **Assignment:**
  - **Scan through counts to find those needed for test set**
  - **Classify with counts in memory**
- Put counts in a database
- …?

Stream and Sort Counting → Distributed Counting

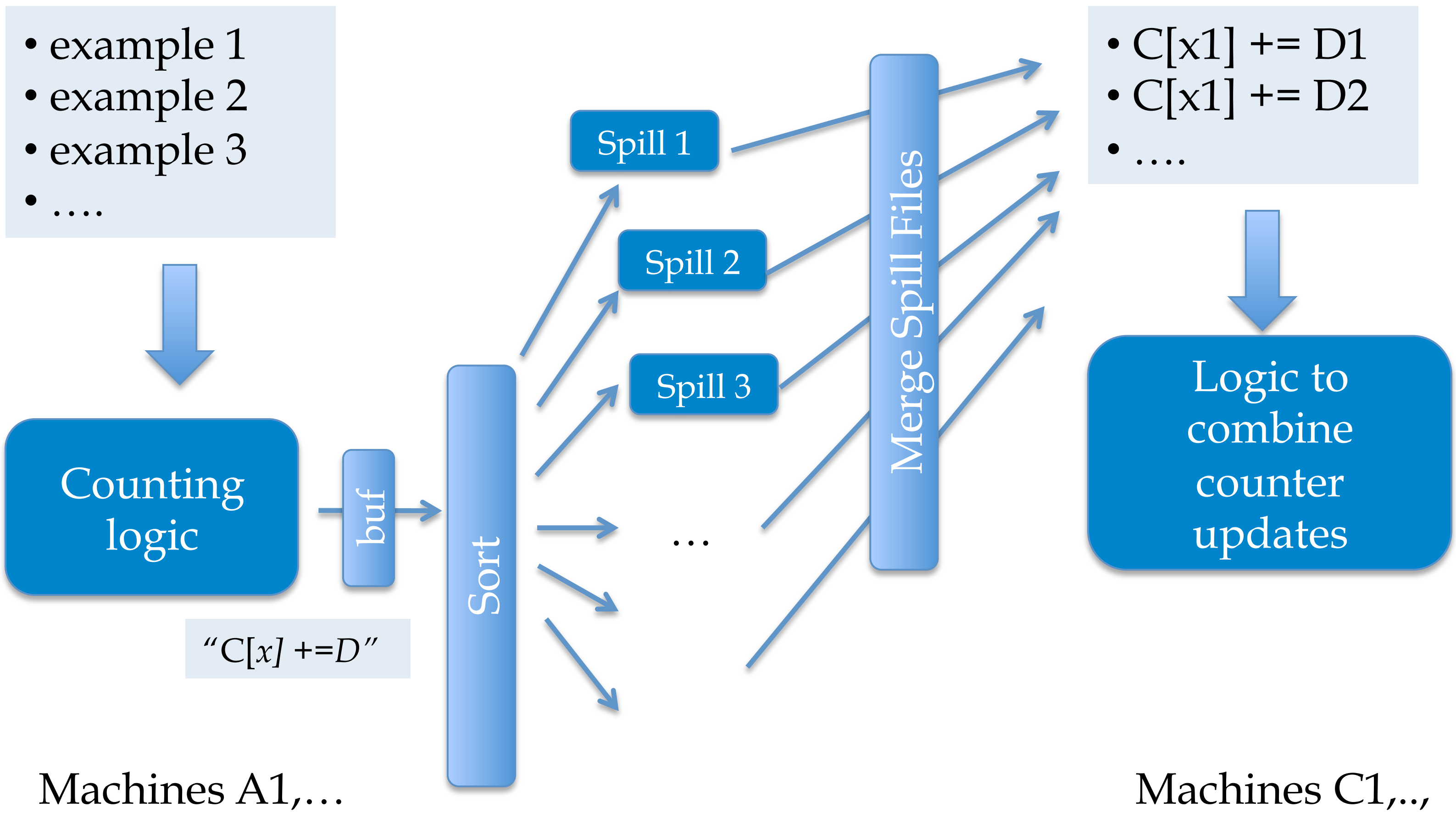


Trivial to parallelize!

Easy to parallelize!

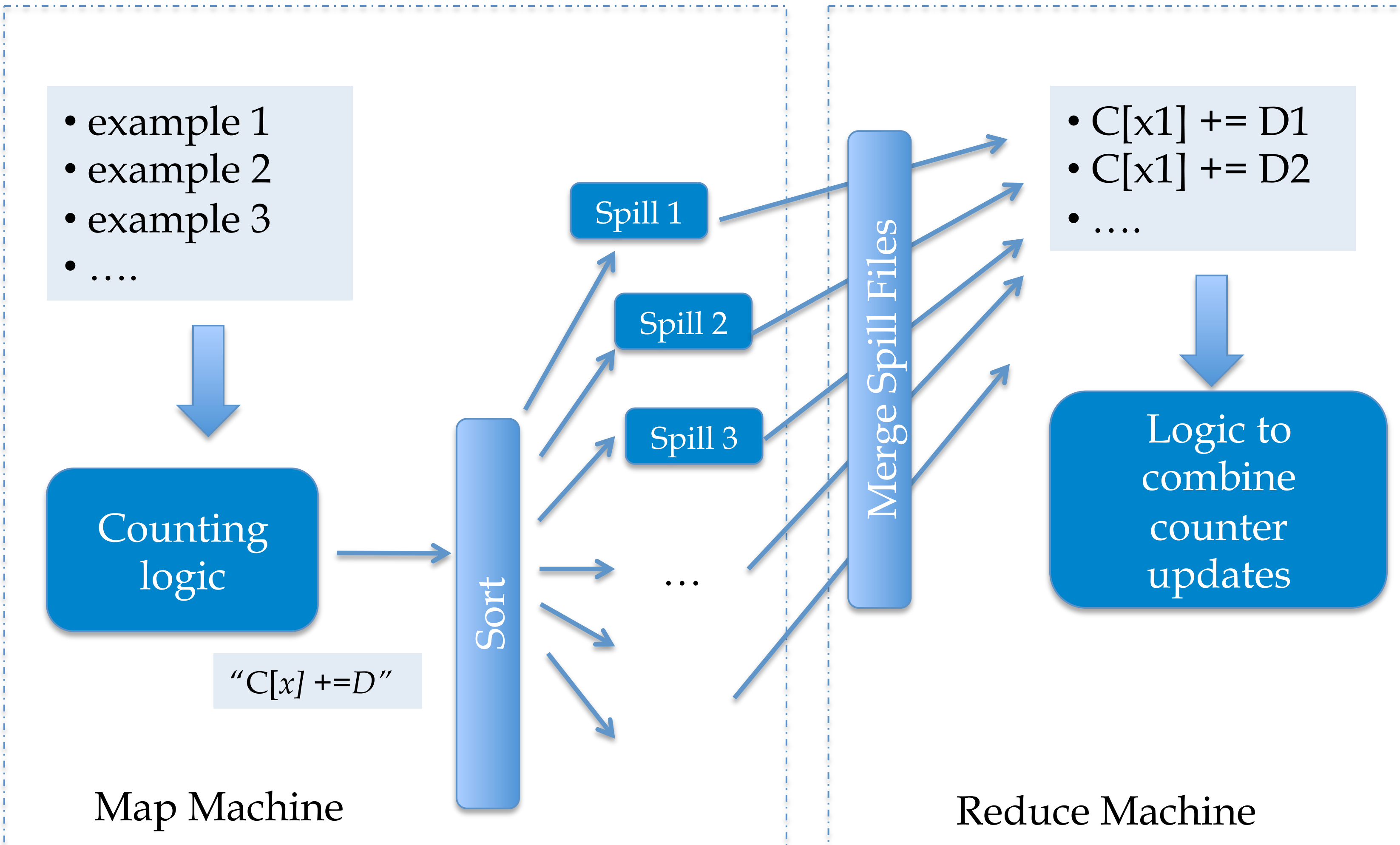
Recap

Stream and Sort Counting → Distributed Counting



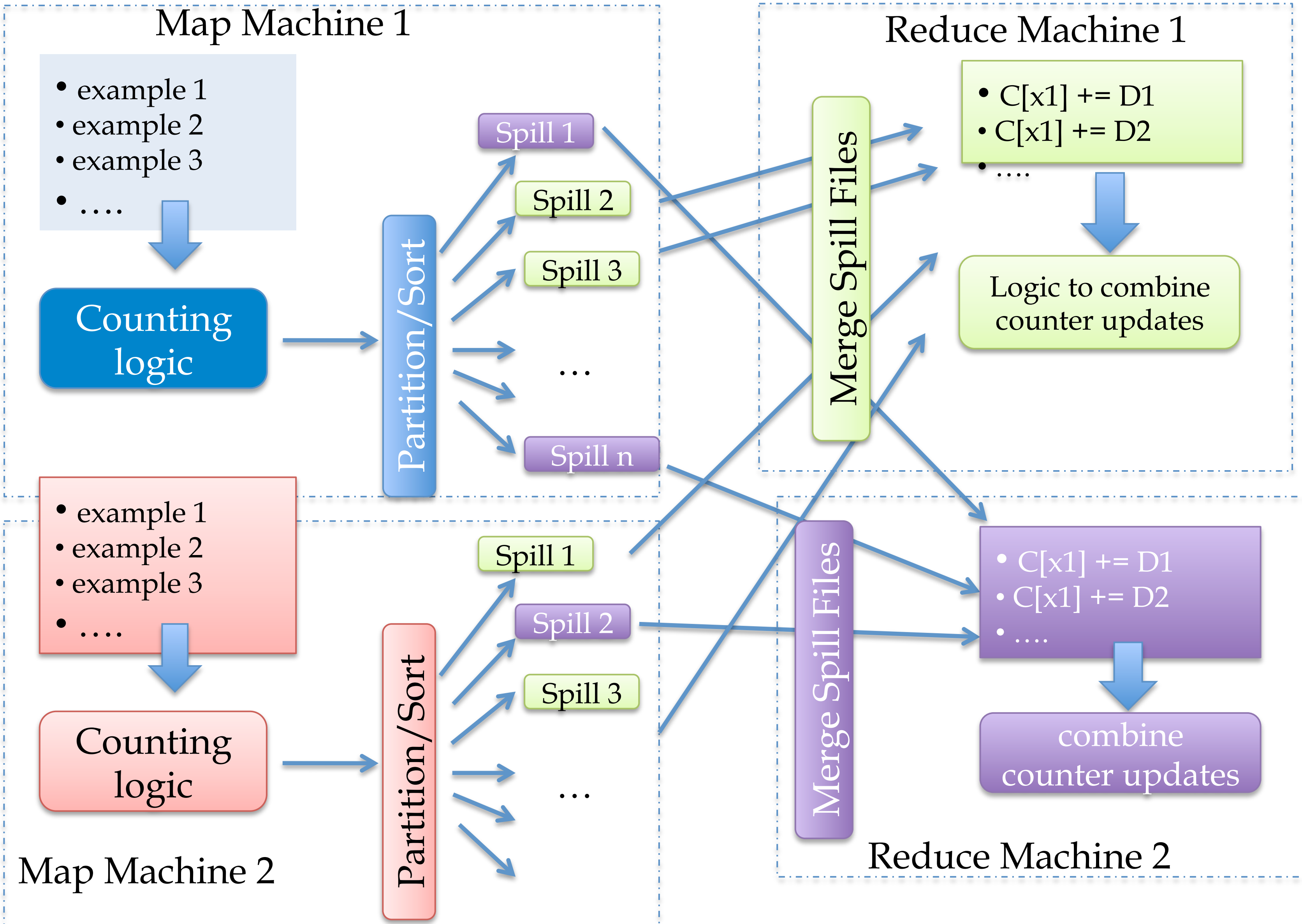
Recap

Stream and Sort Counting → Distributed Counting



color of spill file based on hash code for events

Stream and Sort Counting → Distributed Counting



MORE STREAM-AND-SORT EXAMPLES

Some other stream and sort tasks

- Coming up: classify Wikipedia pages
  - Features:
    - words on page: *src* w_1 w_2 ….
    - outlinks from page: *src* dst_1 dst_2 …
    - how about **inlinks** to the page?

Some other stream and sort tasks

- outlinks from page: *src* dst_1 dst_2 …

– Algorithm:

- For each input line *src* dst_1 dst_2 … dst_n print out
  - dst_1 inlinks.= *src*
  - dst_2 inlinks.= *src*
  - …
  - dst_n inlinks.= *src*
- Sort this output
- Collect the messages and group to get
  - *dst* src_1 src_2 … src_n

Some other stream and sort tasks

- prevKey = Null
- sumForPrevKey = 0
- For each (*event* += *delta*) in input:
    - If *event*==prevKey
        - sumForPrevKey += *delta*
    - Else
        - OutputPrevKey()
        - prevKey = *event*
        - sumForPrevKey = *delta*
- OutputPrevKey()

define OutputPrevKey():

- If PrevKey!=Null
    - print PrevKey,sumForPrevKey

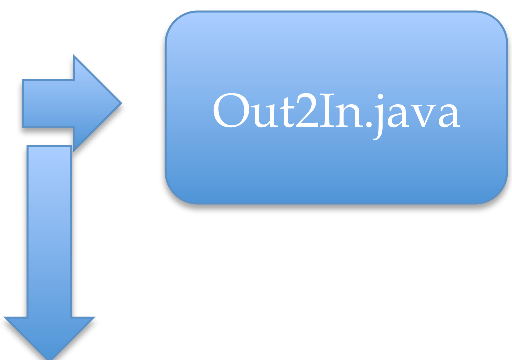
- prevKey = Null
- linksToPrevKey = [ ]
- For each (*dst* inlinks.= *src*) in input:
    - If *dst*==prevKey
        - linksPrevKey.append(*src*)
    - Else
        - OutputPrevKey()
        - prevKey = *dst*
        - linksToPrevKey=[*src*]
- OutputPrevKey()

define OutputPrevKey():

- If PrevKey!=Null
    - print PrevKey, linksToPrevKey

Some other stream and sort tasks

- What if we run this same program on the words on a page?
  - Features:
    - words on page: $src\ w_1\ w_2\ \ldots.$
    - outlinks from page: $src\ dst_1\ dst_2\ \ldots$



$w_1\ src_{1,1}\ src_{1,2}\ src_{1,3}\ \ldots.$
$w_2\ src_{2,1}\ \ldots$
…

an *inverted index* for the documents

Some other stream and sort tasks

- outlinks from page: $src\ dst_1\ dst_2 \ldots$
- Algorithm:
  - For each input line $src\ dst_1\ dst_2 \ldots dst_n$ print out
    - dst_1 inlinks.= src
    - dst_2 inlinks.= src
    - …
    - dst_n inlinks.= src
  - Sort this output
  - Collect the messages and group to get
    - $dst\ src_1\ src_2 \ldots src_n$

Some other stream and sort tasks

- Later on: distributional clustering of words

```
duty
|___responsibility 0.21 0.21
|   |___role 0.12 0.11
|   |   |___action 0.11 0.10
|   |       |___change 0.24 0.08
|   |       |   |___rule 0.16 0.08
|   |       |       |___restriction 0.27 0.08
|   |       |       |   |___ban 0.30 0.08
|   |       |       |   |___sanction 0.19 0.08
|   |       |       |___schedule 0.11 0.07
|   |       |       |___regulation 0.37 0.07
|   |       |___challenge 0.13 0.07
|   |       |   |___issue 0.13 0.07
|   |       |       |___reason 0.14 0.07
|   |       |       |___matter 0.28 0.07
|   |       |___measure 0.22 0.07
|   |___obligation 0.12 0.10
|   |___power 0.17 0.08
|   |   |___jurisdiction 0.13 0.08
|   |   |___right 0.12 0.07
|   |   |___control 0.20 0.07
|   |   |___ground 0.08 0.07
|   |___accountability 0.14 0.08
|   |___experience 0.12 0.07
|   post 0.14 0.14
```

```
|___post 0.14 0.14
|   |___job 0.17 0.10
|   |   |___work 0.17 0.10
|   |       |___training 0.11 0.07
|   |___position 0.25 0.10
|___task 0.10 0.10
|   |___chore 0.11 0.07
|___operation 0.10 0.10
|   |___function 0.10 0.08
|   |___mission 0.12 0.07
|   |   |___patrol 0.07 0.07
|   |___staff 0.10 0.07
|___penalty 0.09 0.09
|   |___fee 0.17 0.08
|       |___tariff 0.13 0.08
|       |___tax 0.19 0.07
|___reservist 0.07 0.07
```

Some other stream and sort tasks

- Later on: distributional clustering of words

Algorithm:

- For each word *w* in a corpus print *w* and the words in a window around it
  - Print "*wi* context .= ($w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k}$)"
- Sort the messages and collect all contexts for each *w* – thus creating an instance associated with *w*
- Cluster the dataset
  - Or train a classifier and classify it

Some other stream and sort tasks

- prevKey = Null
- sumForPrevKey = 0
- For each (*event* += *delta*) in input:
  - If *event*==prevKey
    - sumForPrevKey += *delta*
  - Else
    - OutputPrevKey()
    - prevKey = *event*
    - sumForPrevKey = *delta*
- OutputPrevKey()

define OutputPrevKey():

- If PrevKey!=Null
  - print PrevKey,sumForPrevKey

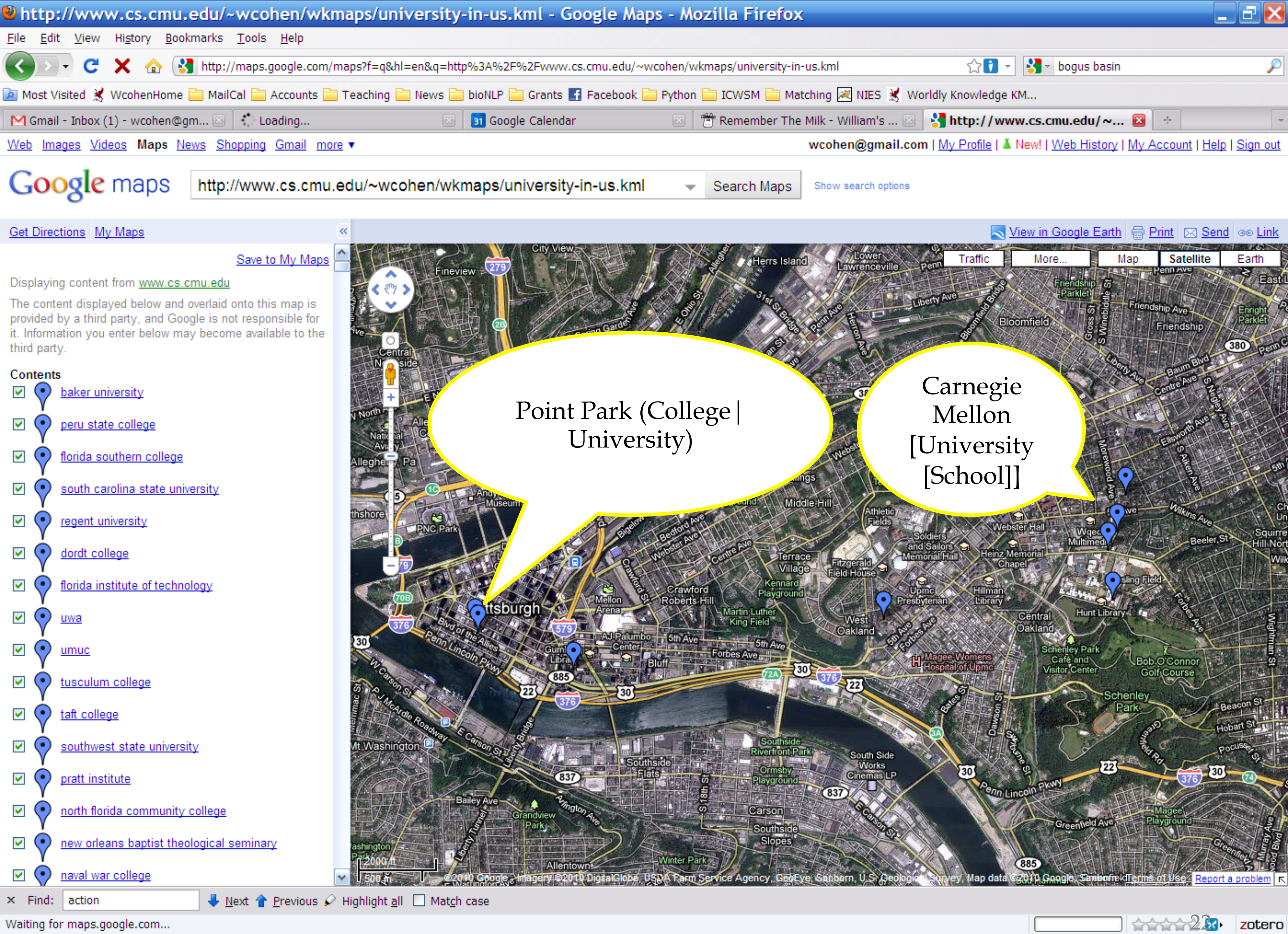
- prevKey = Null
- ctxOfPrevKey = [ ]
- For each (w c.= $w_1,\ldots,w_k$) in input:
  - If *dst*==prevKey
    - ctxOfPrevKey.append($w_1,\ldots,w_k$)
  - Else
    - OutputPrevKey()
    - prevKey = w
    - ctxOfPrevKey=[$w_1,..,w_k$]
- OutputPrevKey()

define OutputPrevKey():

- If PrevKey!=Null
  - print PrevKey, ctxOfPrevKey

Some other stream and sort tasks

- Finding unambiguous geographical names
- GeoNames.org: for each place in its database, stores
  - Several alternative names
  - Latitude/Longitude
  - …
- Lets you put places on a map (e.g., Google Maps)
- Problem: many names are ambiguous, especially if you allow an approximate match
  - Paris, London, … even Carnegie Mellon


http://www.cs.cmu.edu/~wcohen/wkmaps/university-in-us.kml - Google Maps - Mozilla Firefox
Google maps
http://www.cs.cmu.edu/~wcohen/wkmaps/university-in-us.kml
Search Maps
Show search options
Get Directions
My Maps
Save to My Maps
Displaying content from www.cs.cmu.edu
The content displayed below and overlaid onto this map is provided by a third party, and Google is not responsible for it. Information you enter below may become available to the third party.
Contents
baker university
peru state college
florida southern college
south carolina state university
regent university
dordt college
florida institute of technology
uwa
umuc
tusculum college
taft college
southwest state university
pratt institute
north florida community college
new orleans baptist theological seminary
naval war college
Point Park (College | University)
Carnegie Mellon [University [School]]

Some other stream and sort tasks

- Finding almost unambiguous geographical names
- GeoNames.org: for each place in the database
  - print all plausible soft-match substrings in each alternative name, paired with the lat/long, e.g.
    - Carnegie Mellon University at lat1,lon1
    - Carnegie Mellon at lat1,lon1
    - Mellon University at lat1,lon1
    - Carnegie Mellon School at lat2,lon2
    - Carnegie Mellon at lat2,lon2
    - Mellon School at lat2,lon2
    - …
  - Sort and collect… and filter

Scalable out-of-core classification (of large test sets)

can we do better
that the current approach?

Review of NB algorithms

HW	Train events	Test events	Parallel?
1	Msgs → Disk	HashMap (for subset)	no
2	Msgs → Disk	HashMap (for subset)	yes
3	Msgs → Disk	Msgs on Disk (coming….)	yes

Testing Large-vocab Naïve Bayes

[For assignment]

- 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: $O(|V|)$

- 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

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

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

What we'd like

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

Can we do better?

Event counts

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

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

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

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

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

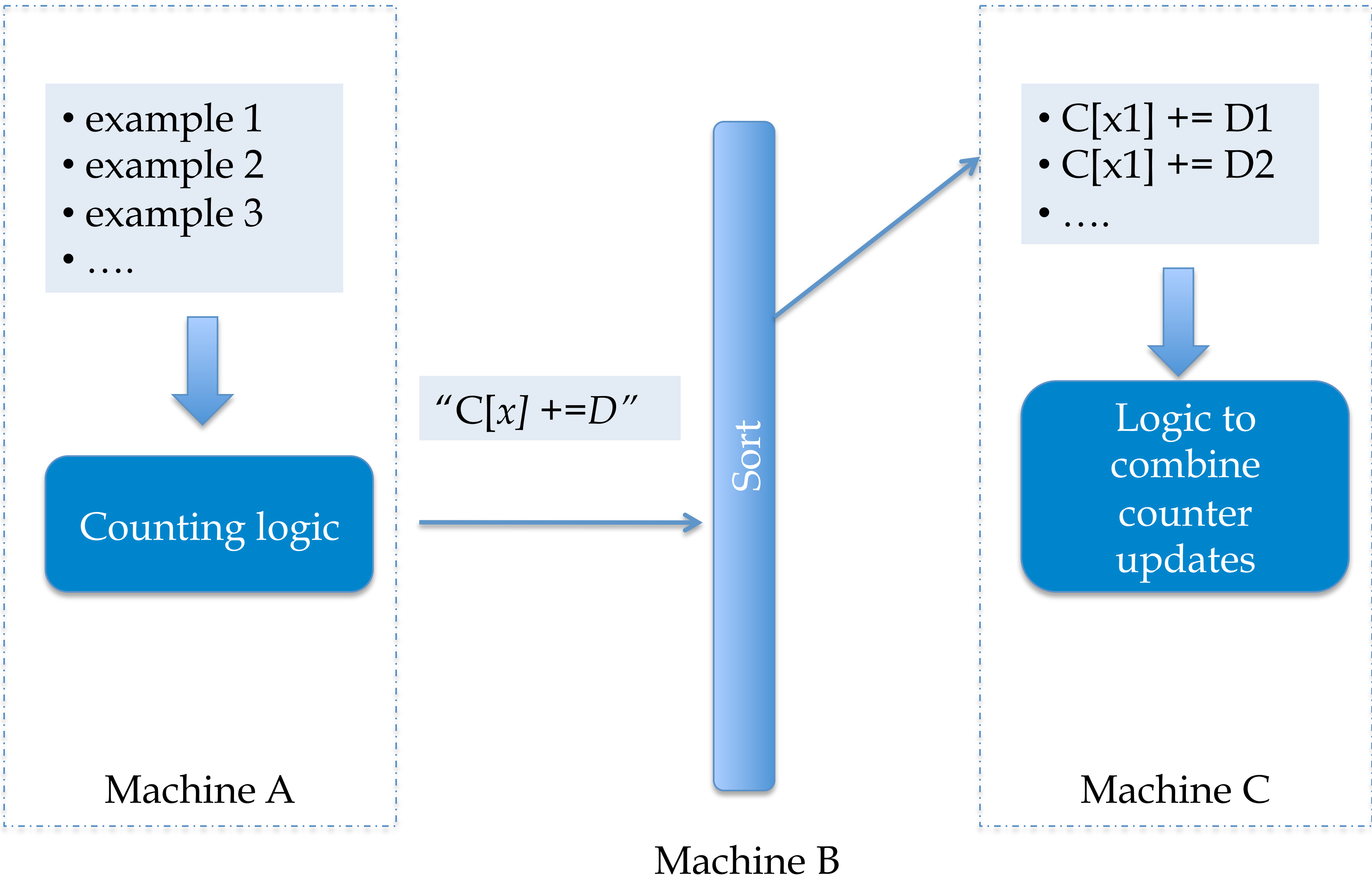
Classification logic

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



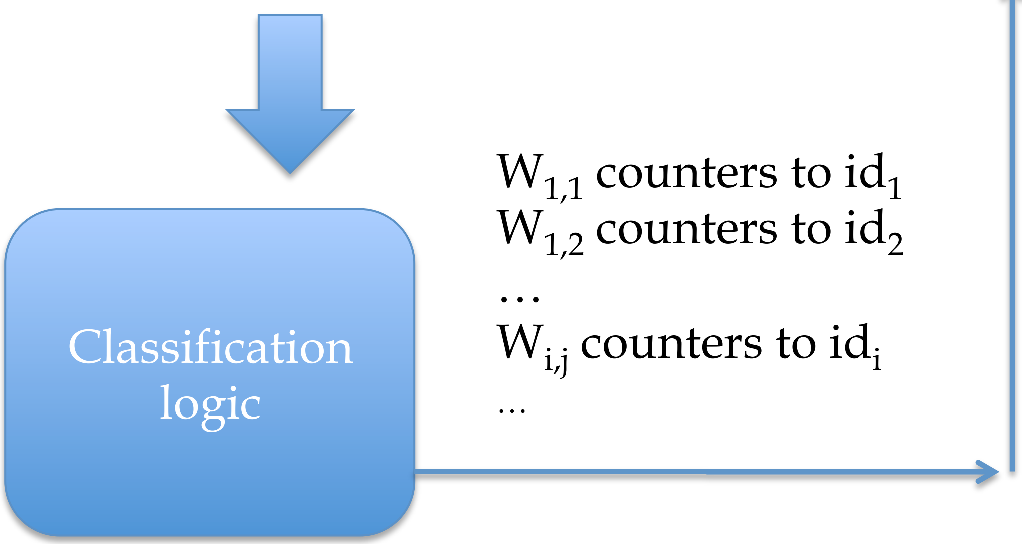
Is there a stream-and-sort analog of this request-and-answer pattern?

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



$W_{1,1}$ counters to id_1
$W_{1,2}$ counters to id_2
…
$W_{i,j}$ counters to id_i
…

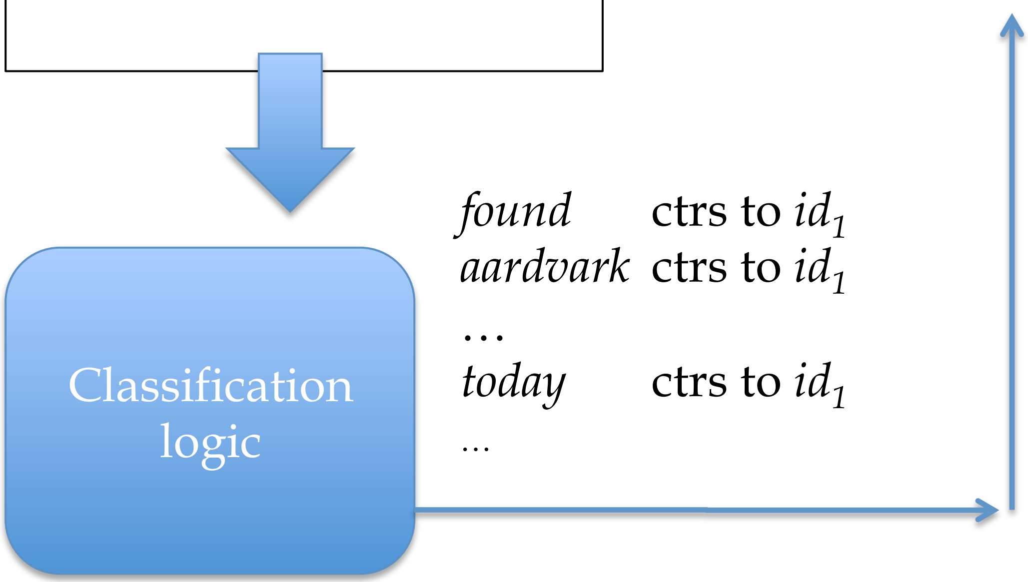
Is there a stream-and-sort analog of this request-and-answer pattern?

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

Classification logic

found ~ctrs to *id_1*
aardvark ~ctrs to *id_1*
…
today ~ctrs to *id_1*
…

~ 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

Classification logic

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

requests

Counter records

Combine and sort

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

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

requests

Combine and sort

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

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

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

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

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 …
..

Request-handling logic → Combine and sort → ????

What we'd wanted

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

What we ended up with … and it's good enough!

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

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	…

input looks like this

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 …
…	…