Announcements

- Working AWS codes will be out soon
- 10-805 project deadlines now posted
- Waitlist is at 28 + 10
  - You need approval if you are an MS student on the 805 waitlist
- William has no offer hours next week

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?

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

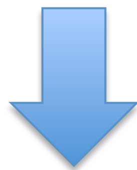
If these records were in a key-value DB we would know what to do….

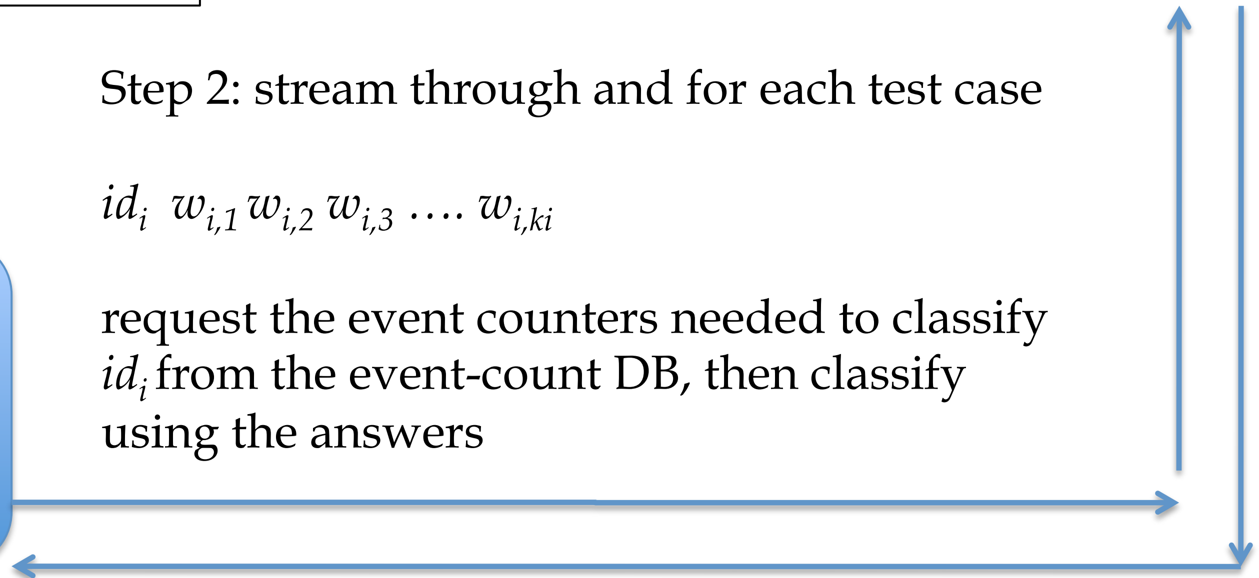
Test data

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

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





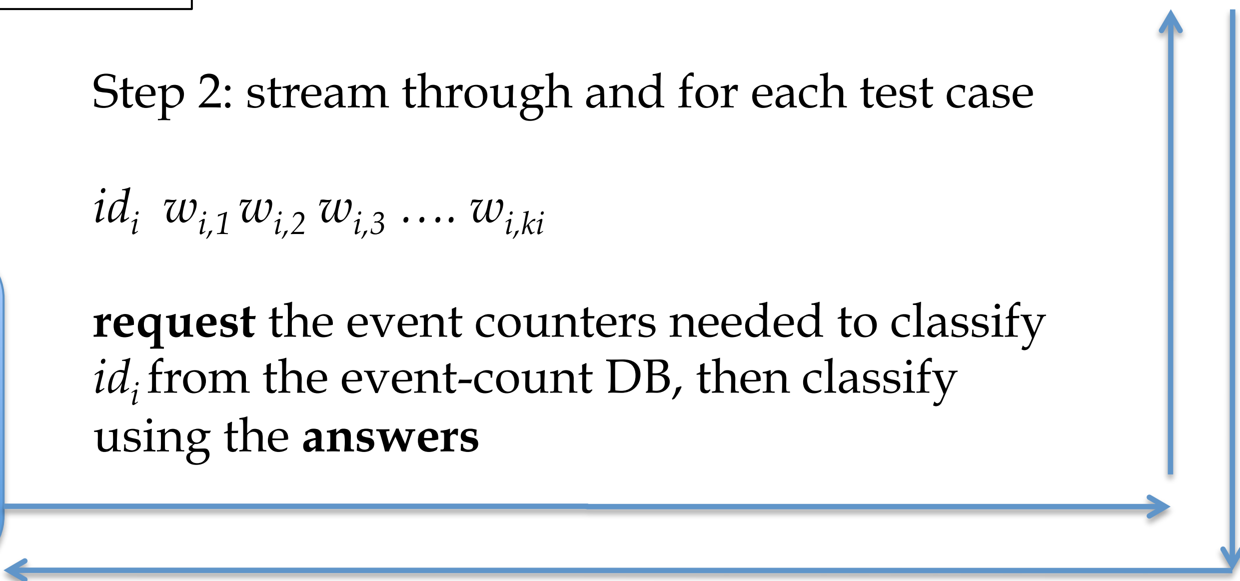
Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

id_1 $w_{1,1}\, w_{1,2}\, w_{1,3}$ …. $w_{1,k1}$
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

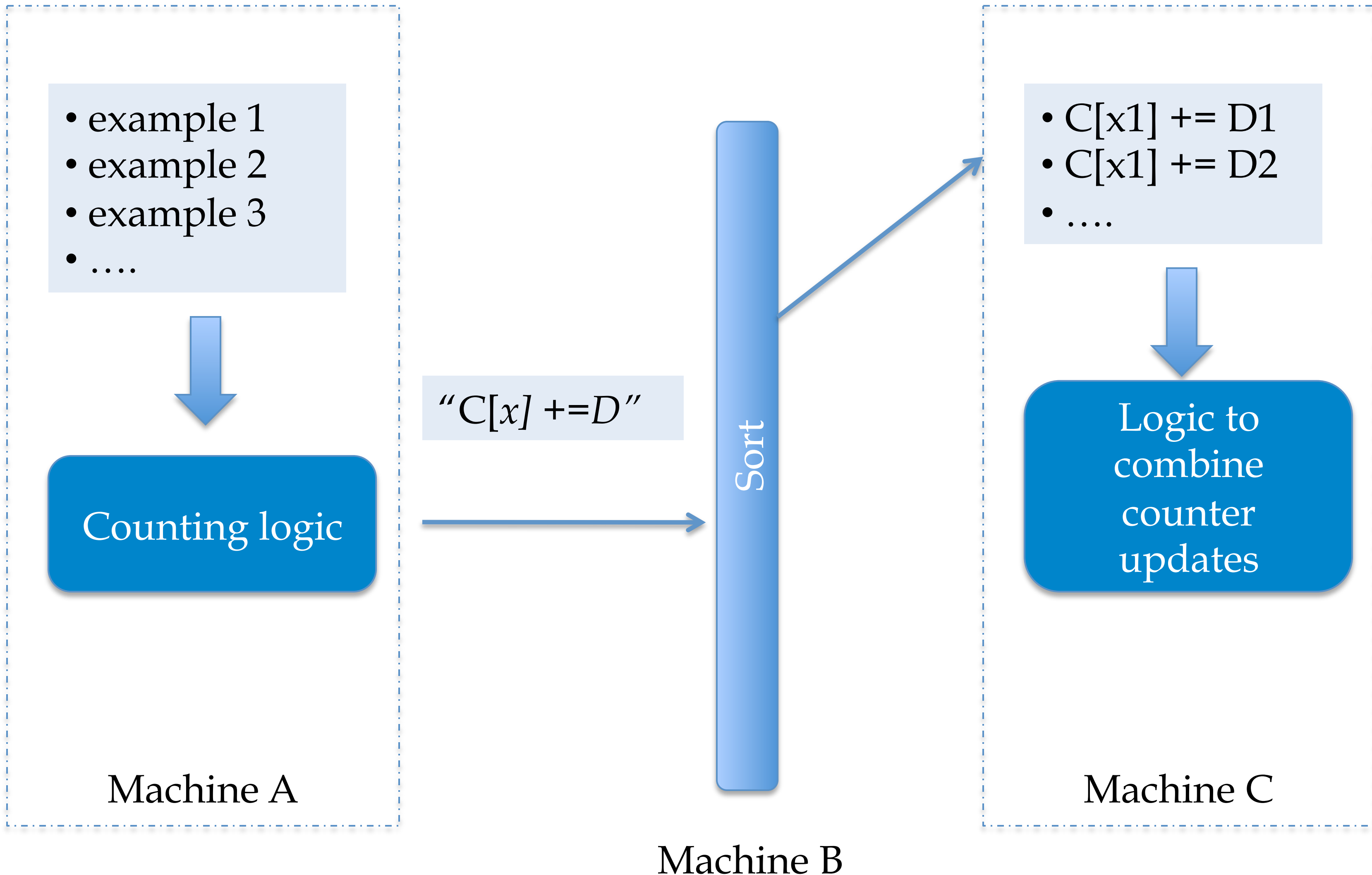


Step 2: stream through and for each test case

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**

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



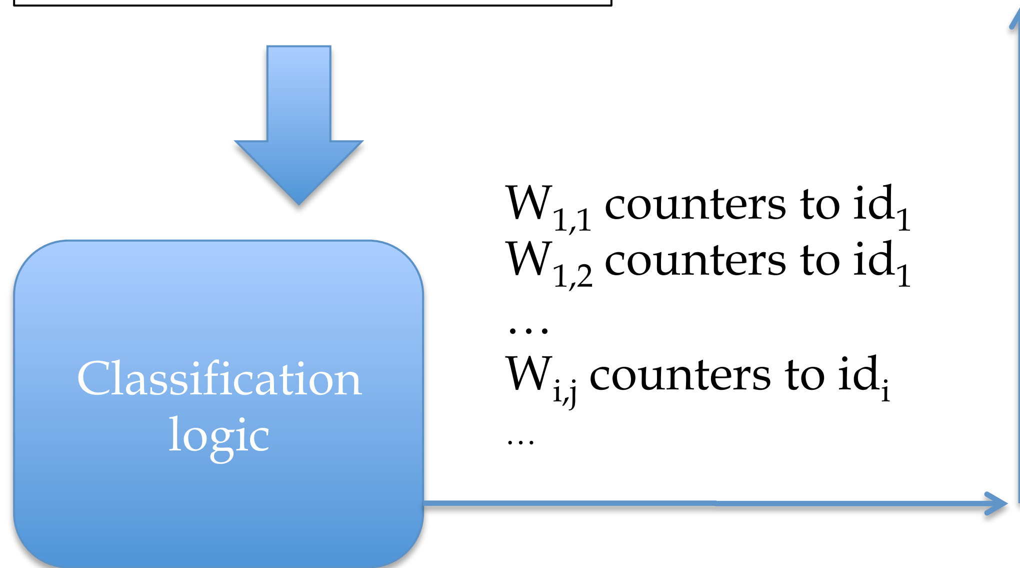
Is there a stream-and-sort analog of this request-and-answer pattern?

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

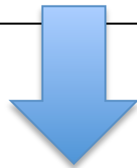
w	Counts associated with W
aardvark	C[w^Y=sports]=2
agent	C[w^Y=sports]=1027,C[w^Y=worl
…	…
zynga	C[w^Y=sports]=21,C[w^Y=worldN



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 …
..



found ctrs to id_1
aardvark ctrs to id_1
…
today ctrs to id_1
…

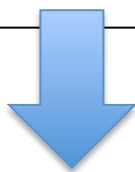
Record of all event counts for each word

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

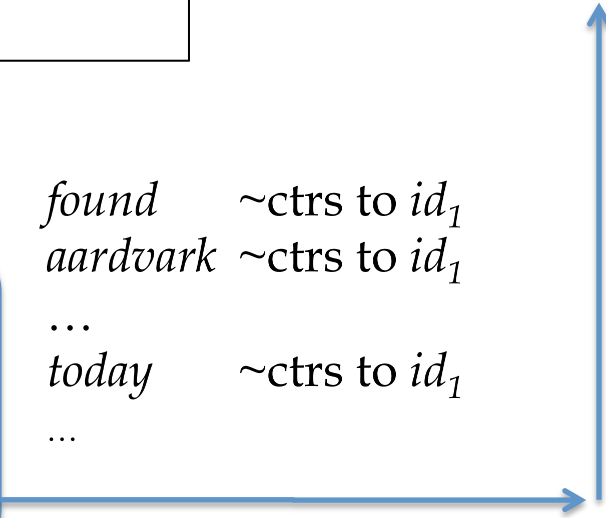
Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

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



Classification logic



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

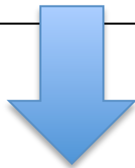
Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

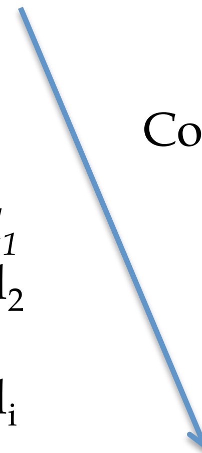
id_1 *found an aardvark in zynga's farmville today!*
id_2 …
id_3 ….
id_4 …
id_5 …
..

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



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



Counter records

requests

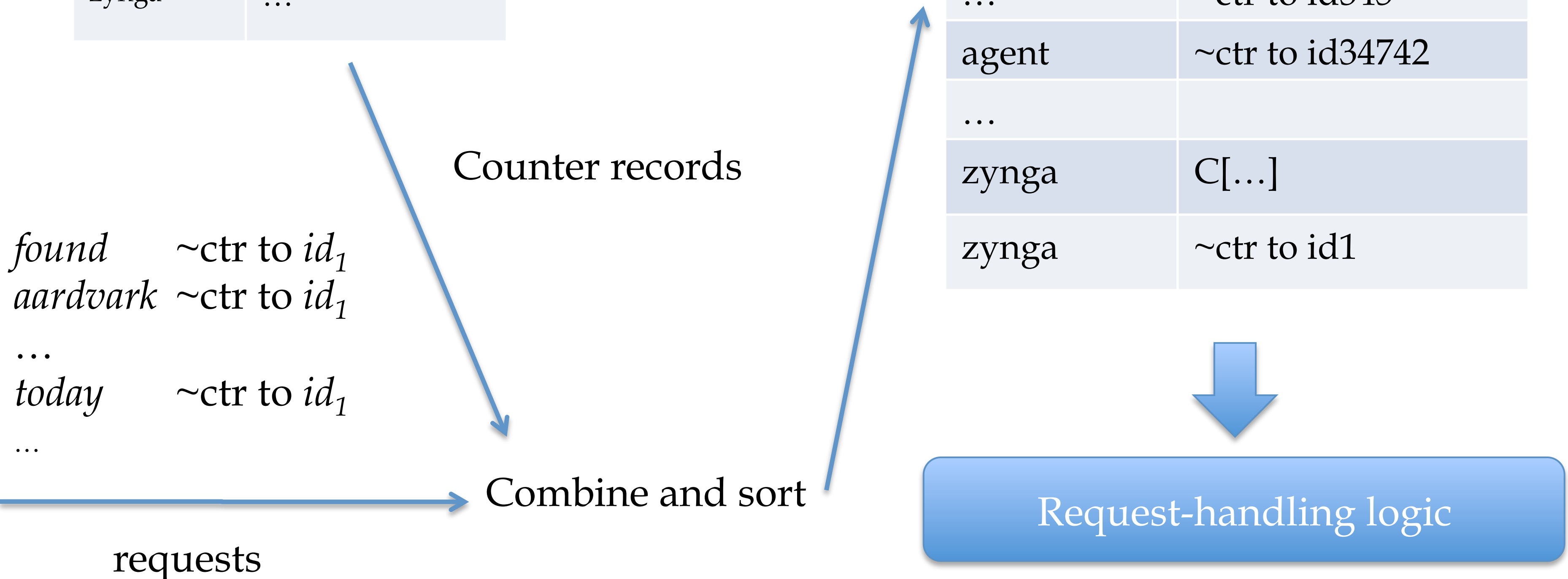
Combine and sort →

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	…

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

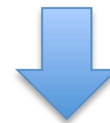


A stream-and-sort analog of the request-and-answer pattern…

- 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*"

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

Combine and sort

requests

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*

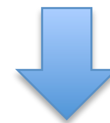
define Answer*(record,request):*
- find *id* where "*request* = ~ctr to *id*"
- print "*id* ~ctr for *request* is *record*"

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

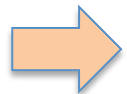
Output:
id1 ~ctr for *aardvark* is C[w^Y=sports]=2
…
id1 ~ctr for *zynga* is ….
…

Combine and sort

requests



Request-handling logic

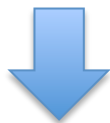


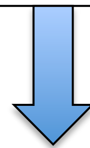
A stream-and-sort analog of the request-and-answer pattern…

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

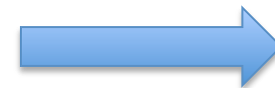
Output:
id1 ~ctr for *aardvark* is C[w^Y=sports]=2
…
id1 ~ctr for *zynga* is ….
…

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 …
…	…

Summary

- We've filled in the missing piece
  - Training (event-counting)
    - hashtable in-memory
    - on-disk, low-memory
  - Testing (classification)
    - hashtable in-memory
    - on-disk, low-memory

Rocchio's Algorithm

Rocchio's algorithm

Many variants of these formulae

$$DF(w) = \#\text{different docs } w \text{ occurs in}$$

$$TF(w,d) = \#\text{different times } w \text{ occurs in doc } d$$

$$IDF(w) = \frac{|D|}{DF(w)}$$

$$u(w,d) = \log(TF(w,d)+1)\cdot\log(IDF(w))$$

...as long as $u(w,d)=0$ for words not in d!

$$\mathbf{u}(d) = \langle u(w_1,d),....,u(w_{|V|},d)\rangle$$

Store only non-zeros in $\mathbf{u}(d)$, so size is $O(|d|)$

$$\mathbf{u}(y) = \alpha\frac{1}{|C_y|}\sum_{d\in C_y}\frac{\mathbf{u}(d)}{\|\mathbf{u}(d)\|_2} - \beta\frac{1}{|D-C_y|}\sum_{d'\in D-C_y}\frac{\mathbf{u}(d')}{\|\mathbf{u}(d')\|_2}$$

But size of $\mathbf{u}(y)$ is $O(|n_V|)$

$$f(d) = \arg\max_y \frac{\mathbf{u}(d)}{\|\mathbf{u}(d)\|_2}\cdot\frac{\mathbf{u}(y)}{\|\mathbf{u}(y)\|_2}$$

$$\|\mathbf{u}\|_2 = \sqrt{\sum_i u_i^2}$$

Rocchio's algorithm

Given a table mapping w to $DF(w)$, we can compute $\mathbf{v}(d)$ from the words in d… and the rest of the learning algorithm is just adding…

$$DF(w) = \#\text{ different docs } w \text{ occurs in}$$

$$TF(w,d) = \#\text{ different times } w \text{ occurs in doc } d$$

$$IDF(w) = \frac{|D|}{DF(w)}$$

$$u(w,d) = \log(TF(w,d)+1) \cdot \log(IDF(w))$$

$$\mathbf{u}(d) = \langle u(w_1,d),\ldots.,u(w_{|V|},d)\rangle, \qquad \mathbf{v}(d) = \frac{\mathbf{u}(d)}{\|\mathbf{u}(d)\|_2} = \langle v(w_1,d),\ldots.\rangle$$

$$\mathbf{u}(y) = \alpha \frac{1}{|C_y|} \sum_{d \in C_y} \mathbf{v}(d) - \beta \frac{1}{|D - C_y|} \sum_{d' \in D - C_y} \mathbf{v}(d), \qquad \mathbf{v}(y) = \frac{\mathbf{u}(y)}{\|\mathbf{u}(y)\|_2}$$

$$f(d) = \text{argmax}_y\, \mathbf{v}(d) \cdot \mathbf{v}(y)$$

Rocchio v Bayes

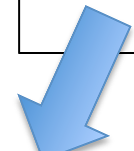
Imagine a similar process
but for labeled documents…

Train data

id_1	$y1$	$w_{1,1}\, w_{1,2}\, w_{1,3}$ …. $w_{1,k1}$
id_2	$y2$	$w_{2,1}\, w_{2,2}\, w_{2,3}$ ….
id_3	$y3$	$w_{3,1}\, w_{3,2}$ ….
id_4	$y4$	$w_{4,1}\, w_{4,2}$ …
id_5	$y5$	$w_{5,1}\, w_{5,2}$ ….
..		

Event counts

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



Recall Naïve Bayes test process?

id_1 $y1$ $w_{1,1}\, w_{1,2}\, w_{1,3}$ …. $w_{1,k1}$	$C[X=w_{1,1}$^Y=*sports*]=5245, $C[X=w_{1,1}$^Y=..],$C[X=w_{1,2}$^…]
id_2 $y2$ $w_{2,1}\, w_{2,2}\, w_{2,3}$ ….	$C[X=w_{2,1}$^Y=….]=1054,…, $C[X=w_{2,k2}$^…]
id_3 $y3$ $w_{3,1}\, w_{3,2}$ ….	$C[X=w_{3,1}$^Y=….]=…
id_4 $y4$ $w_{4,1}\, w_{4,2}$ …	…

Rocchio….

Train data

id_1 $y1$ $w_{1,1}$ $w_{1,2}$ $w_{1,3}$ …. $w_{1,k1}$
id_2 $y2$ $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ ….
id_3 $y3$ $w_{3,1}$ $w_{3,2}$ ….
id_4 $y4$ $w_{4,1}$ $w_{4,2}$ …
id_5 $y5$ $w_{5,1}$ $w_{5,2}$ ….
..

Rocchio: DF counts

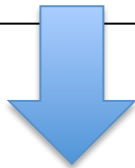
aardvark	12
agent	1054
…	2120
	37
	3
	…

id_1 $y1$ $w_{1,1}$ $w_{1,2}$ $w_{1,3}$ …. $w_{1,k1}$	$v(w_{1,1},id1)$, $v(w_{1,2},id1)$…$v(w_{1,k1},id1)$
id_2 $y2$ $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ ….	$v(w_{2,1},id2)$, $v(w_{2,2},id2)$…
id_3 $y3$ $w_{3,1}$ $w_{3,2}$ ….	…
id_4 $y4$ $w_{4,1}$ $w_{4,2}$ …	…

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

Test data

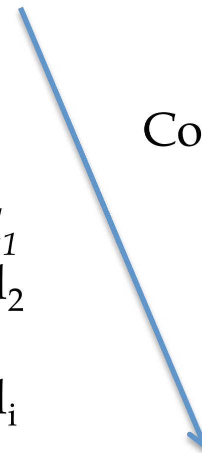
id_1 found an aardvark in zynga's farmville today!
id_2 …
id_3 ….
id_4 …
id_5 …
..



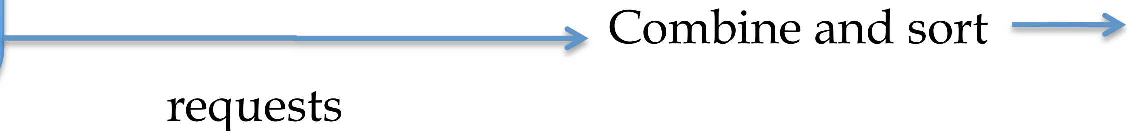
Record of all event counts for each word

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

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



Counter records



Combine and sort

requests

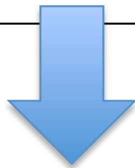
recap: Is there a stream-and-sort analog of this request-and-answer pattern?

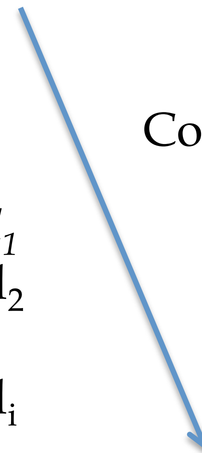
Test data

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

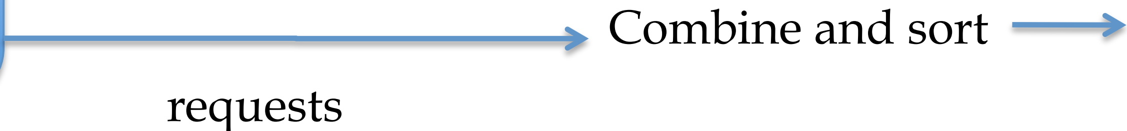
w	DF(w)
aardvark	4
agent	67
…	…
zynga	43





Counter records

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

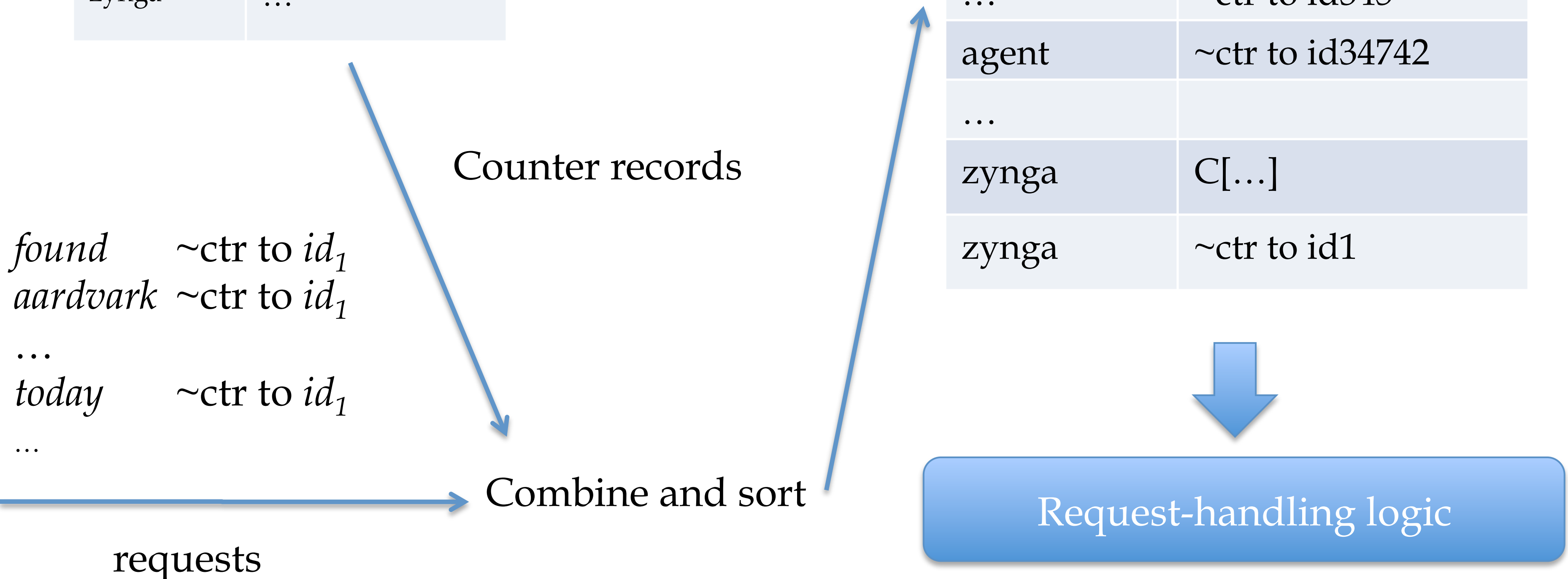


recap: A stream-and-sort analog of the request-and-answer pattern…

Record of all event counts for each word

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

w	Counts
aardvark	4
aardvark	~ctr to id1
agent	
agent	~ctr to id345
agent	~ctr to id9854
…	~ctr to id345
agent	~ctr to id34742
…	
zynga	C[…]
zynga	~ctr to id1

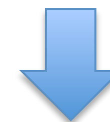


recap: A stream-and-sort analog of the request-and-answer pattern…

- 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	4
aardvark	~ctr to id1
agent	
agent	~ctr to id345
agent	~ctr to id9854
…	~ctr to id345
agent	~ctr to id34742
…	
zynga	C[…]
zynga	~ctr to id1



Request-handling logic

Combine and sort

requests

Rocchio….

$DF(w) = \#$ different docs w occurs in

$TF(w,d) = \#$ different times w occurs in doc d

$$IDF(w) = \frac{|D|}{DF(w)}$$

$$u(w,d) = \log(TF(w,d)+1) \cdot \log(IDF(w))$$

$$\mathbf{u}(d) = \langle u(w_1,d),....,u(w_{|V|},d) \rangle, \qquad \mathbf{v}(d) = \frac{\mathbf{u}(d)}{\|\mathbf{u}(d)\|_2}$$

Train data

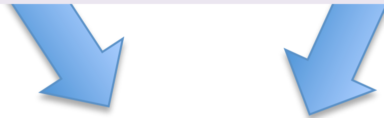
$id_1\ y1\ w_{1,1}\ w_{1,2}\ w_{1,3}\ ....\ w_{1,k1}$
$id_2\ y2\ w_{2,1}\ w_{2,2}\ w_{2,3}\ ....$
$id_3\ y3\ w_{3,1}\ w_{3,2}\ ....$
$id_4\ y4\ w_{4,1}\ w_{4,2}\ ...$
$id_5\ y5\ w_{5,1}\ w_{5,2}\ ....$
..

…

$id_1\ y1\ w_{1,1}\ w_{1,2}\ w_{1,3}\ ....\ w_{1,k1}$	$\mathbf{v}(id_1)$
$id_2\ y2\ w_{2,1}\ w_{2,2}\ w_{2,3}\ ....$	$\mathbf{v}(id_2)$
$id_3\ y3\ w_{3,1}\ w_{3,2}\ ....$	…
$id_4\ y4\ w_{4,1}\ w_{4,2}\ ...$	…

$$\mathbf{u}(y) = \alpha \frac{1}{|C_y|} \sum_{d \in C_y} \mathbf{v}(d) - \beta \frac{1}{|D - C_y|} \sum_{d' \in D - C_y} \mathbf{v}(d), \quad \mathbf{v}(y) = \frac{\mathbf{u}(y)}{\| \mathbf{u}(y) \|_2}$$

$$f(d) = \text{argmax}_y \ \mathbf{v}(d) \cdot \mathbf{v}(y)$$



id_1 $y1$ $w_{1,1}$ $w_{1,2}$ $w_{1,3}$ …. $w_{1,k1}$	$v(w_{1,1}\ w_{1,2}\ w_{1,3}\ \ldots\ w_{1,k1})$, the document vector for id_1
id_2 $y2$ $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ ….	$v(w_{2,1}\ w_{2,2}\ w_{2,3}\ \ldots) = v(w_{2,1},d), v(w_{2,2},d), \ldots$
id_3 $y3$ $w_{3,1}$ $w_{3,2}$ ….	…
id_4 $y4$ $w_{4,1}$ $w_{4,2}$ …	…

For each (y, **v**), go through the non-zero values in **v** …one for each w in the document d…and increment a counter for that dimension of **v**(y)

Message: increment **v**($y1$)'s weight for $w_{1,1}$ by $\alpha\, v(w_{1,1},d) \,/\, |C_y|$

Message: increment **v**($y1$)'s weight for $w_{1,2}$ by $\alpha\, v(w_{1,2},d) \,/\, |C_y|$

$$f(d) = \text{argmax}_y\ \mathbf{v}(d) \cdot \mathbf{v}(y) = \text{argmax}_y \sum_w v(w,d) v(w,y)$$

Train data

$id_1\ y1\ w_{1,1}\ w_{1,2}\ w_{1,3}\ \ldots.\ w_{1,k1}$ $id_2\ y2\ w_{2,1}\ w_{2,2}\ w_{2,3}\ \ldots.$ $id_3\ y3\ w_{3,1}\ w_{3,2}\ \ldots.$ $id_4\ y4\ w_{4,1}\ w_{4,2}\ \ldots$ $id_5\ y5\ w_{5,1}\ w_{5,2}\ \ldots.$ ..

Rocchio: DF counts

aardvark	$v(y1,w)=0.0012$
agent	$v(y1,w)=0.013,\ v(y2,w)=\ldots$
…	….
	…

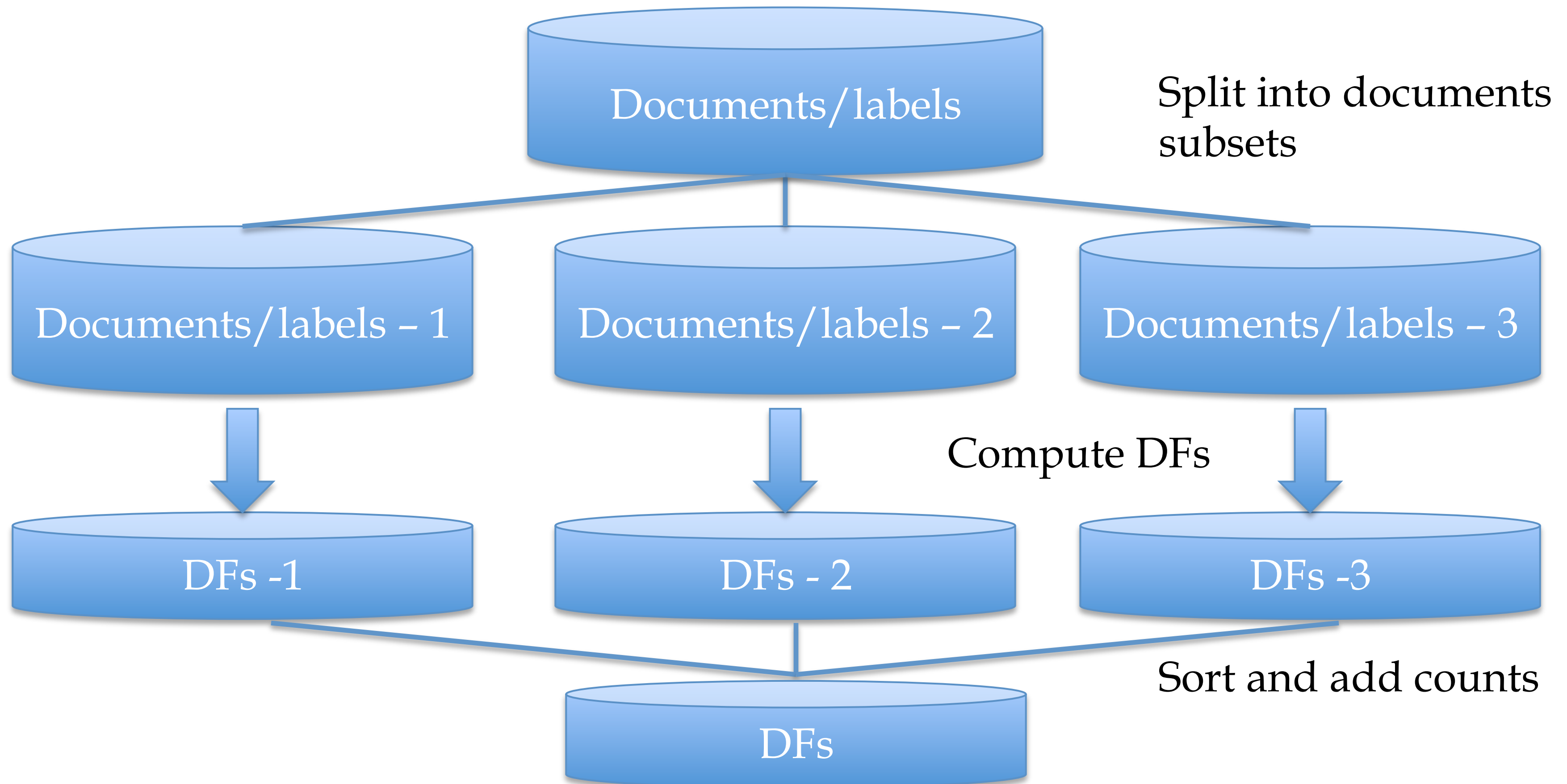
$id_1\ y1\ w_{1,1}\ w_{1,2}\ w_{1,3}\ \ldots.\ w_{1,k1}$	$\mathbf{v}(id_1),\ v(w_{1,1},y1), v(w_{1,1},y1), \ldots., v(w_{1,k1},yk), \ldots, v(w_{1,k1},yk)$
$id_2\ y2\ w_{2,1}\ w_{2,2}\ w_{2,3}\ \ldots.$	$\mathbf{v}(id_2),\ v(w_{2,1},y1), v(w_{2,1},y1), \ldots.$
$id_3\ y3\ w_{3,1}\ w_{3,2}\ \ldots.$	…
$id_4\ y4\ w_{4,1}\ w_{4,2}\ \ldots$	…

Rocchio Summary

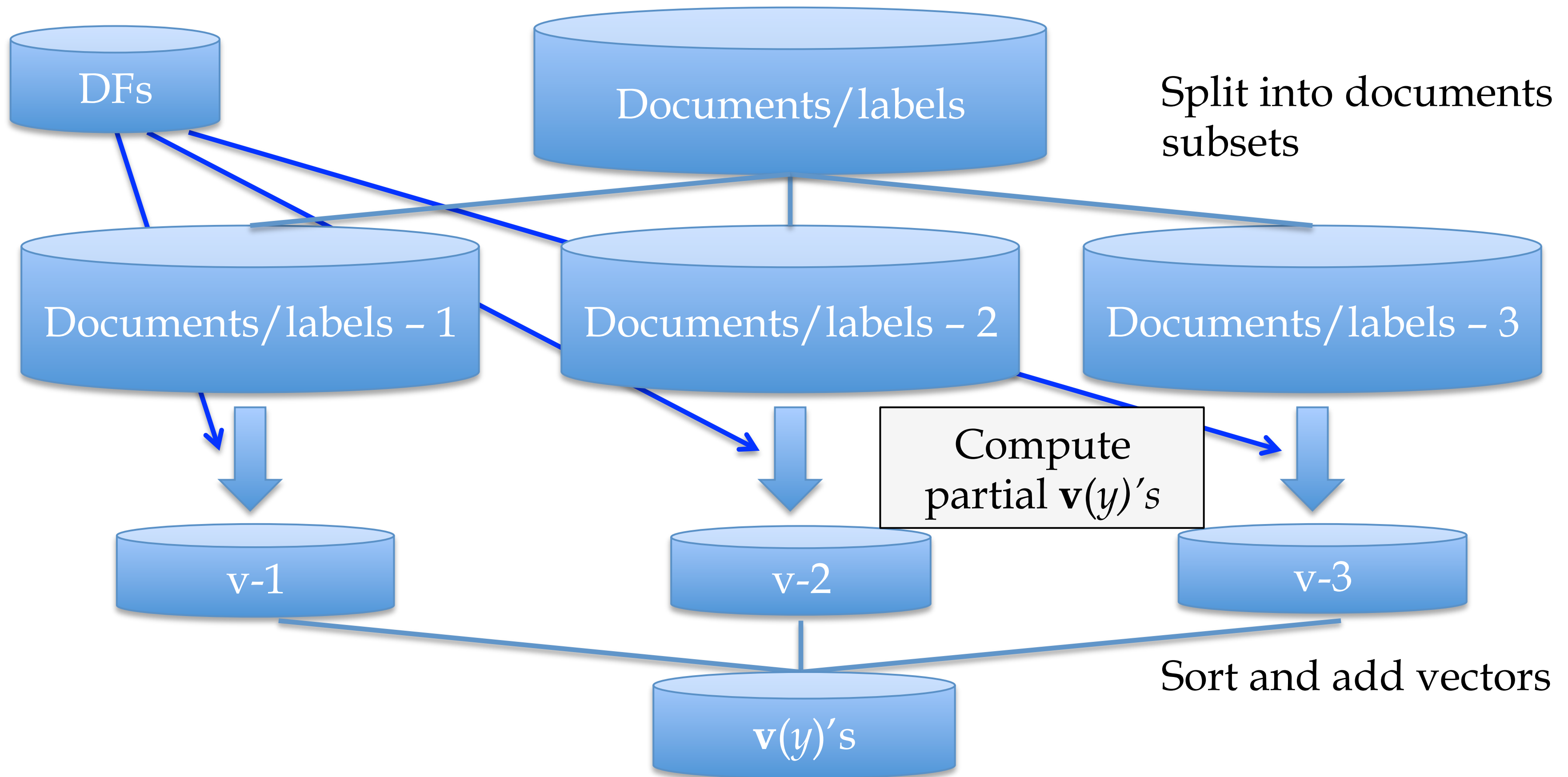
- Compute DF
  - one scan thru docs
- Compute **v**(id_i) for each document
  - output size O(n)
- Add up vectors to get **v**(y)
- Classification ~= disk NB

- time: O(n), n=corpus size
  - like NB event-counts
- time: O(n)
  - one scan, if DF fits in memory
  - like first part of NB test procedure otherwise
- time: O(n)
  - one scan if **v**(y)'s fit in memory
  - like NB training otherwise

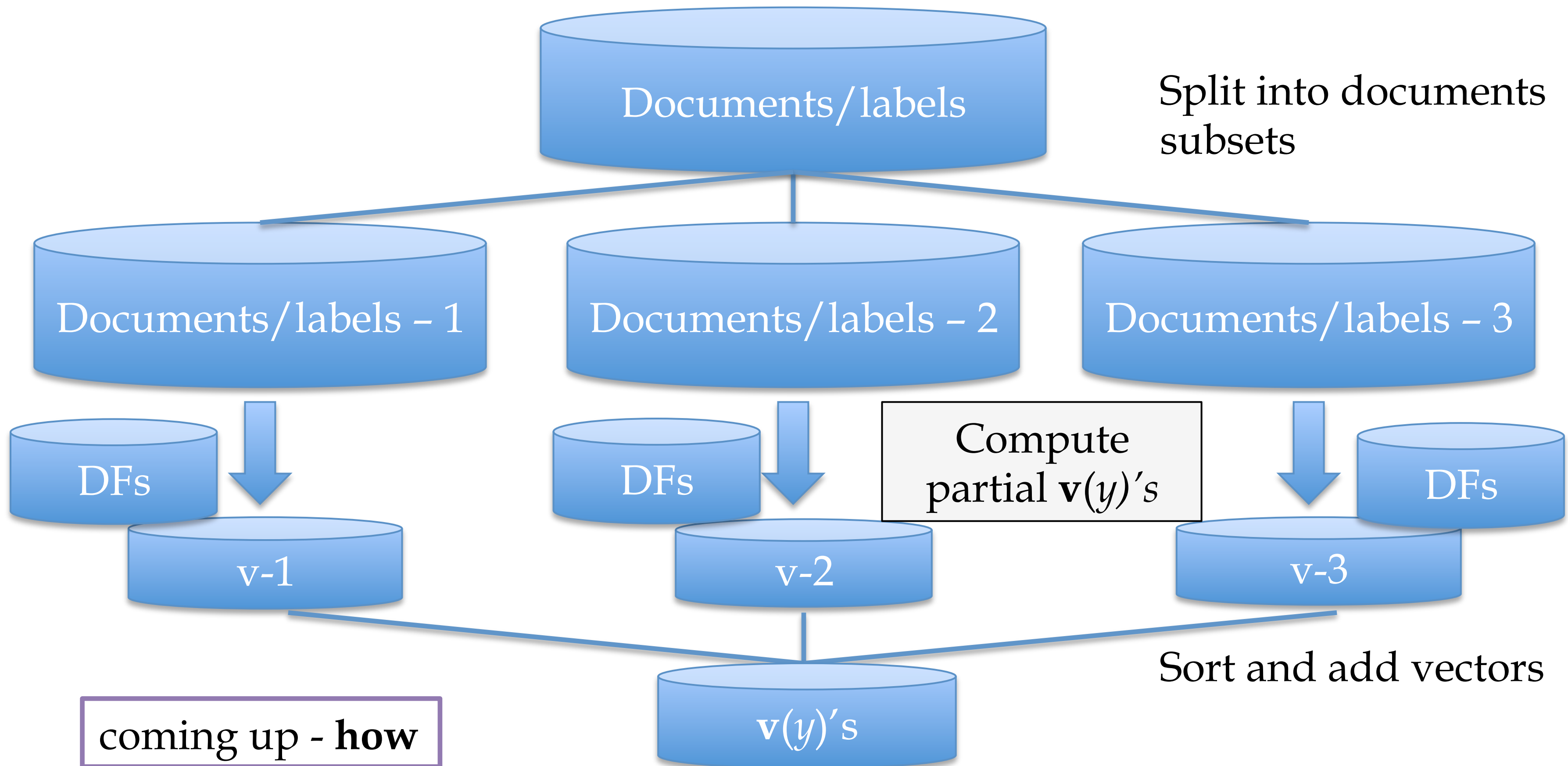
One? more Rocchio observation



One?? more Rocchio observation



O(1) more Rocchio observation



We have shared access to the DFs, but only shared *read* access – we don't need to share *write* access. So we only need to copy the information across the different processes.

ABSTRACTIONS FOR STREAM AND SORT AND MAP-REDUCE

Abstractions On Top Of Map-Reduce

- We've decomposed some algorithms into a map-reduce "workflow" (series of map-reduce steps)
  - naive Bayes training
  - naïve Bayes testing
  - …
- How else can we express these sorts of computations? Are there some common special cases of map-reduce steps we can parameterize and reuse?

Abstractions On Top Of Map-Reduce

- Some obvious streaming processes:
  - for each row in a table
    - Transform it and output the result
    - Decide if you want to keep it with some boolean test, and copy out only the ones that pass the test

Example: stem words in a stream of word-count pairs:

("aardvarks",1) ➔ ("aardvark",1)

Proposed syntax: *f(row)* ➔ *row'*

$table2 = \text{MAP } table1 \text{ TO } \lambda\, row : f(row))$

Example: apply stop words

("aardvark",1) ➔ ("aardvark",1)
("the",1) ➔ *deleted*

Proposed syntax: *f(row)* ➔ *{true,false}*

$table2 = \text{FILTER } table1 \text{ BY } \lambda\, row : f(row))$

Abstractions On Top Of Map-Reduce

- A non-obvious? streaming processes:
  - for each row in a table
    - Transform it to a list of items
    - Splice all the lists together to get the output table (**flatten**)

Proposed syntax: *f(row)* → *list of rows*

$table2 = \text{FLATMAP } table1 \text{ TO } \lambda\, row : f(row))$

Example: tokenizing a line

"I found an aardvark" → ["i", "found","an","aardvark"]
"We love zymurgy" → ["we","love","zymurgy"]

..but final table is one word per row

"i"
"found"
"an"
"aardvark"
"we"
"love"
...

Abstractions On Top Of Map-Reduce

- An example: the Naïve Bayes test program…

NB Test Step

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

NB Test Step

The general case:

We're taking rows from a table

- In a particular format *(event,count)*

Applying a function to get a new value

- The *word* for the event

And *grouping* the rows of the table by this new value

➔ **Grouping operation**

Special case of a map-reduce

Proposed syntax: *f(row)*➔*field*

GROUP *table* BY λ *row* : *f*(*row*)

Could define *f* via: a function, a field of a defined *record* structure, ...

Event counts

$X=w_1$^$Y=sports$	*5245*
$X=w_1$^$Y=worldNews$	*1054*
X=..	*2120*
$X=w_2$^$Y=\ldots$	*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

NB Test Step

The general case:
We're taking rows from a table
- In a particular format *(event,count)*

Applying a function to get a new value
- The *word* for the event

And *grouping* the rows of the table by this new value

➔ **Grouping operation**
Special case of a map-reduce

Proposed syntax: *f(row)* ➔ *field*

GROUP *table* BY λ *row* : *f*(*row*)

Could define *f* via: a function, a field of a defined *record* structure, ...

Aside: you guys know how to implement this, right?

1. Output pairs (f(row),row) with a map/streaming process
2. Sort pairs by key – which is f(row)
3. Reduce and aggregate by *appending together* all the values associated with the same key

Abstractions On Top Of Map-Reduce

- And another example from the Naïve Bayes test program…

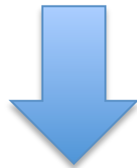
Request-and-answer

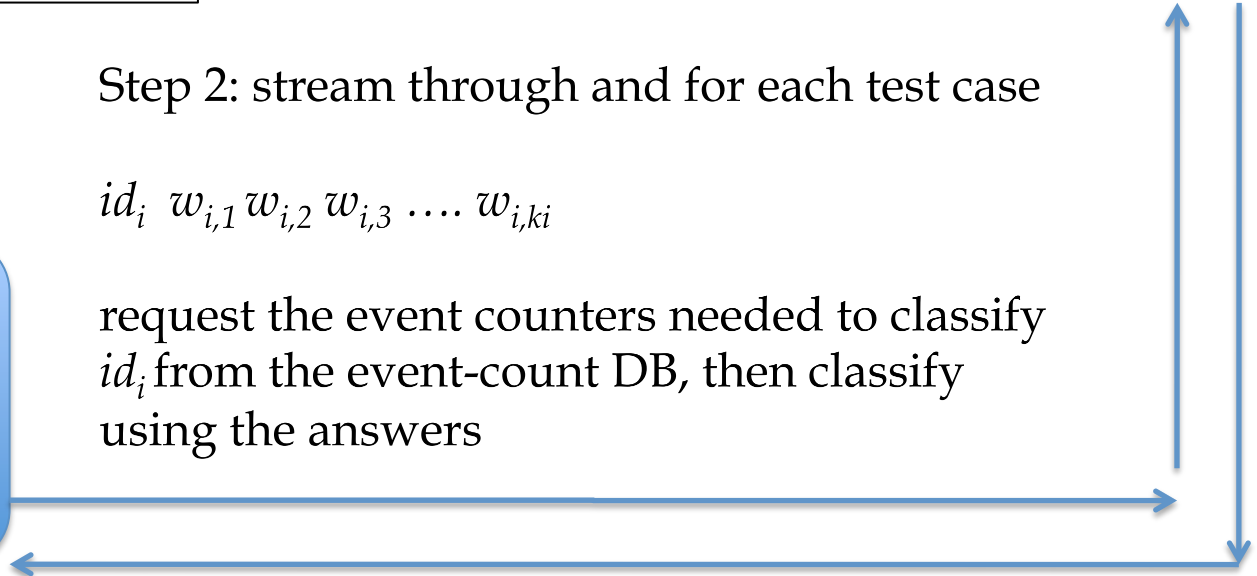
Test data

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

Request-and-answer

- Break down into stages
  - Generate the data being requested (indexed by key, here a word)
    - Eg with group … by
  - Generate the requests as (key, requestor) pairs
    - Eg with flatmap … to
  - **Join** these two tables by key
    - Join defined as (1) cross-product and (2) filter out pairs with different values for keys
    - This replaces the step of concatenating two different tables of key-value pairs, and reducing them together
  - Postprocess the joined result

w	Request
found	~ctr to id1
aardvark	~ctr to id1
…	
zynga	~ctr to id1
…	~ctr to id2

w	Counters
aardvark	C[w^Y=sports]=2
agent	C[w^Y=sports]=1027,C[w^Y=worldNews
…	…
zynga	C[w^Y=sports]=21,C[w^Y=worldNews]=

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

w	Request
found	id1
aardvark	id1
…	
zynga	id1
…	id2

w	Counters
aardvark	C[w^Y=sports]=2
agent	C[w^Y=sports]=1027,C[w^Y=worldNews
…	…
zynga	C[w^Y=sports]=21,C[w^Y=worldNews]=

Examples:

JOIN *wordInDoc* BY *word, wordCounters* BY *word* --- *if word(row) defined correctly*

JOIN wordInDoc BY lambda (word,docid):word,
wordCounters BY lambda (word,counters):word – *using python syntax for functions*

	C[w^Y=sports]=…	id345
	C[w^Y=sports]=…	id34742
	C[…]	id1
	C[…]	…

Proposed syntax:

JOIN *table1* BY $\lambda\ row : f(row)$,
table2 BY $\lambda\ row : g(row)$

Abstract Implementation: [TF]IDF

1/2

data = *pairs (docid, term) where term is a word appears in document with id docid*
opera
• DIST
• GRO ... *ce ste*

key	value
found	~~(d123,found),(d134,found),~~… 2456
aardvark	~~(d123,aardvark),~~… 7

docId	term
d123	found
d123	aardvark

docFreq = DISTINCT data
| GROUP BY λ (docid,term):term REDUCING T

key	value
1	12451

docIds = MAP DATA BY= λ (docid,term):docid | DISTINCT
numDocs = GROUP docIds BY λ docid:1 REDUCING TO count /* *(1,numDocs)* */

question – how many reducers should I use here?

dataPlusDF =
JOIN data BY λ (docid, term):term, docFreq BY λ (term, df):term
| MAP λ ((docid,term),(term,df)):(docId,term,df) /* *(docId,term,document-freq)* */

unnormalizedDocVecs = JOIN dataPlusDF by λ row:1, numDocs by λ row:1
| MAP λ ((docId,term,df),(dummy,numDocs)): (docId,term,log(numDocs/df))
/* *(docId, term, weight-before-normalizing) :* ***u*** */

Abstract Implementation: [TF]IDF

1/2

data = *pairs (docid, term) where term is a word appears in document with id docid*
opera
• DIS
• GRO
ce ste

key	value
found	~~(d123,found),(d134,found),~~… 2456
aardvark	~~(d123,aardvark),~~… 7

docId	term
d123	found
d123	aardvark

docFreq = DISTINCT data
| GROUP BY λ (docid,term):term REDUCING T

key	value
1	12451

docIds = MAP DATA BY= λ (docid,term):docid | DISTINCT
numDocs = GROUP docIds BY λ docid:1 REDUCING TO count /* *(1,numDocs)* */

question – how many reducers should I use here?

dataPlusDF =
JOIN data BY λ (docid, term):term, docFreq BY λ (term, df):term
| MAP λ ((docid,term),(term,df)):(docId,term,df) /* *(docId,term,document-freq)* */

unnormalizedDocVecs = JOIN dataPlusDF by λ row:1, numDocs by λ row:1
| MAP λ ((docId,term,df),(dummy,numDocs)): (docId,term,log(numDocs/df))
/* *(docId, term, weight-before-normalizing) : u* */

question – how many reducers should I use here?

Abstract Implementation: TFIDF

2/2

```
normalizers =
  GROUP unnormalizedDocVecs BY λ (docId,term,w):docid
  RETAINING λ (docId,term,w): w²
  REDUCING TO sum /* (docid,sum-of-square-weights) */
```

key		
d1234	(d1234,found,1.542), (d1234,aardvark,13.23),…	37.234
d3214	….	29.654

```
docVec = JOIN unnormalizedDocVecs BY λ (docId,term,w):docid,
              normalizers BY λ (docId,norm):docid
         | MAP λ ((docId,term,w), (docId,norm)): (docId,term,w/sqrt(norm))
         /* (docId, term, weight) */
```

docId	term	w
d1234	found	1.542
d1234	aardvark	13.23

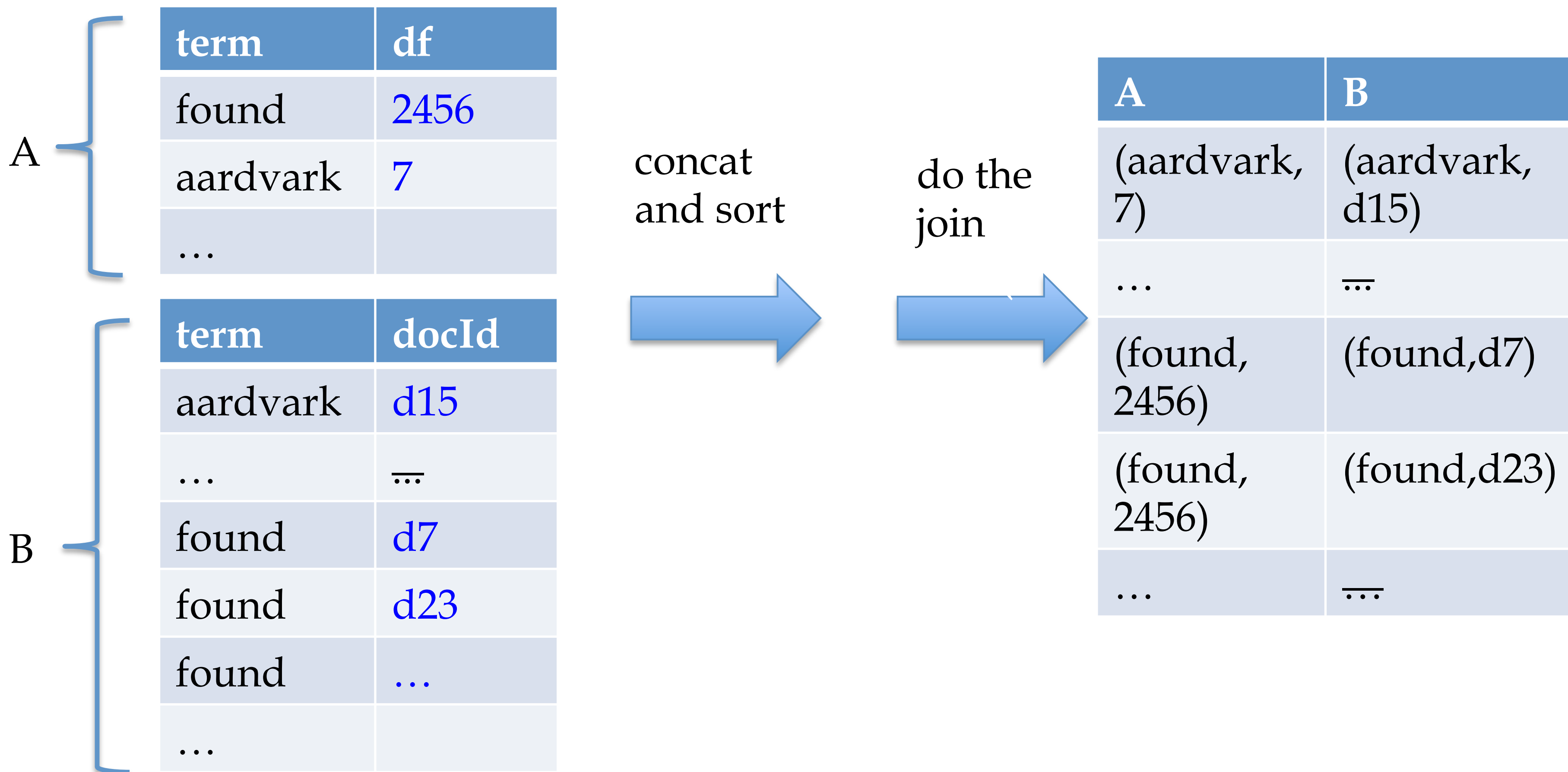
docId	w
d1234	37.234
d1234	37.234

Two ways to join

- Reduce-side join
- Map-side join

Two ways to join

- Reduce-side join for A,B



Two ways to join

- Reduce-side join for A,B

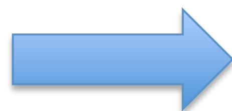
tricky bit: need **sort** by **first two** values (aardvark, *AB*) – we want the DF's to come first

but all tuples with key "aardvark" should go to **same worker**

	df
	2456
ark	7

	docId
ark	d15
	…
	d7
	d23
	…

concat and sort



term		df
found	A	2456
aardvark	A	7
…		

term		docId
aardvark	B	d15
…		…
found	B	d7
found	B	d23
found	B	…
…		

do the join

A	B
(aardvark, 7)	(aardvark, d15)
…	…
(found, 2456)	(found,d7)
(found, 2456)	(found,d23)
…	…

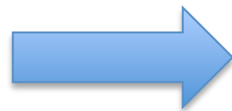
Two ways to join

- Reduce-side join for A,B

	df
	2456
ark	7

	docId
ark	d15
	...
	d7
	d23
	…

concat and sort



term		df
found	A	2456
aardvark	A	7
…		

term		docId
aardvark	B	d15
…		...
found	B	d7
found	B	d23
found	B	…
…		

tricky bit: need **sort** by **first two** values (aardvark, *AB*) – we want the DF's to come first

but all tuples with key "aardvark" should go to **same worker**

custom sort (secondary sort key): Writeable with your own Comparator

custom Partitioner (specified for job like the Mapper, Reducer, ..)

Two ways to join

- Map-side join
  - write the smaller relation out to disk
  - send it to each Map worker
    - DistributedCache
  - when you **initialize** each Mapper, load in the small relation
    - Configure(…) is called at initialization time
  - map through the larger relation and do the join
  - faster but requires one relation to go in memory