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 “\(Y=y ^ X=x_j += 1\)”
- 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 \(1..d\):
    - \(C[Y=y ^ X=x_j] += 1\)
- 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

```
java MyTrainer train | sort | java MyCountAdder > model
```

“map”

“reduce”
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(“$Y=y \land X=x_j$”)

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();
    }
  }
}

5
Testing Large-vocab Naïve Bayes

• For each example id, y, x₁,…..,xₐ 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, x₁,…..,xₐ in test:
  – Add x₁,…..,xₐ 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, x₁,…..,xₐ in test:
  – For each y' in dom(Y):
    • Compute log Pr(y',x₁,…..,xₐ) = ....
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

- Example 1
- Example 2
- Example 3
- ...

Counting logic

Machines A1, ...

"C[x] += D"

Standardized message routing logic

Sort

Machines B1, ...

Machines C1, ...

C[x1] += D1
C[x1] += D2
...

Logic to combine counter updates

Trivial to parallelize!

Easy to parallelize!
Stream and Sort Counting → Distributed Counting

- example 1
- example 2
- example 3
- ...

Logic to combine counter updates

Machines A1, ...

"C[x] += D"

Counting logic

buf

Sort

Machines C1, ...

Spill 1

Spill 2

Spill 3

Merge Spill Files

Logic to combine counter updates

C[x1] += D1

C[x1] += D2

....
Stream and Sort Counting $\rightarrow$ Distributed Counting

- example 1
- example 2
- example 3
- ...

```
$C[x] += D$
```

**Map Machine**

**Counting logic**

```
$C[x1] += D1$
$C[x1] += D2$
.....
```

**Reduce Machine**

**Logic to combine counter updates**

Recap
Stream and Sort Counting → Distributed Counting

Map Machine 1
- Counting logic
  - example 1
  - example 2
  - example 3
  - ...
- Partition/Sort
  - Spill 1
  - Spill 2
  - Spill 3
  - ...
  - Spill n
- Logic to combine counter updates
Reduce Machine 1
- C[x1] += D1
- C[x1] += D2
- ...
- Merge Spill Files

Map Machine 2
- Counting logic
  - example 1
  - example 2
  - example 3
  - ...
- Partition/Sort
  - Spill 1
  - Spill 2
  - Spill 3
  - ...
Reduce Machine 2
- C[x1] += D1
- C[x1] += D2
- ...
- Merge Spill Files
- combine counter updates

Color of spill file based on hash code for events
MORE STREAM-AND-SORT EXAMPLES
Some other stream and sort tasks

• Coming up: classify Wikipedia pages
  – Features:
    • words on page: $src \ w_1 \ w_2 \ldots$
    • outlinks from page: $src \ dst_1 \ dst_2 \ldots$
    • how about **inlinks** to the page?
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

- `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 $w_1$ $w_2$ …. 
    • outlinks from page: src dst$_1$ dst$_2$ ...

an inverted index for the documents
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

- Later on: distributional clustering of words
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 “\( w_i \) 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

- \texttt{prevKey} = \texttt{Null}
- \texttt{sumForPrevKey} = 0
- For each (\texttt{event} += \texttt{delta}) in input:
  - If \texttt{event} == \texttt{prevKey}
    - \texttt{sumForPrevKey} += \texttt{delta}
  - Else
    - OutputPrevKey()
    - \texttt{prevKey} = \texttt{event}
    - \texttt{sumForPrevKey} = \texttt{delta}
    - OutputPrevKey()

\textbf{define OutputPrevKey():}
- If \texttt{PrevKey} != \texttt{Null}
  - print \texttt{PrevKey, sumForPrevKey}

\texttt{prevKey} = \texttt{Null}
\texttt{ctxOfPrevKey} = [ ]
- For each (\texttt{w c.=} \texttt{w}_1, \ldots, \texttt{w}_k) in input:
  - If \texttt{dst} == \texttt{prevKey}
    - ctxOfPrevKey.append( \texttt{w}_1, \ldots, \texttt{w}_k )
  - Else
    - OutputPrevKey()
    - \texttt{prevKey} = \texttt{w}
    - ctxOfPrevKey = [\texttt{w}_1, \ldots, \texttt{w}_k]
    - OutputPrevKey()

\textbf{define OutputPrevKey():}
- If \texttt{PrevKey} != \texttt{Null}
  - print \texttt{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
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

<table>
<thead>
<tr>
<th>HW</th>
<th>Train events</th>
<th>Test events</th>
<th>Parallel?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(Msgs → Disk)</td>
<td>HashMap (for subset)</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>(Msgs → Disk)</td>
<td>HashMap (for subset)</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>(Msgs → Disk)</td>
<td>Msgs on Disk (coming....)</td>
<td>yes</td>
</tr>
</tbody>
</table>
Testing Large-vocab Naïve Bayes

• 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 \( \text{dom}(Y) \):
    • Compute \( \log \Pr(y', x_1, \ldots, x_d) = \ldots \)

Model size: \( O(|V|) \)

Time: \( O(n^2) \), size of test
Memory: same

Time: \( O(n^2) \)
Memory: same

Time: \( O(n^2) \)
Memory: same
Can we do better?

Test data

<table>
<thead>
<tr>
<th>id</th>
<th>w_{1,1} w_{1,2} w_{1,3} .... w_{1,k_1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>w_{2,1} w_{2,2} w_{2,3} ....</td>
</tr>
<tr>
<td>id</td>
<td>w_{3,1} w_{3,2} ....</td>
</tr>
<tr>
<td>id</td>
<td>w_{4,1} w_{4,2} ...</td>
</tr>
<tr>
<td>id</td>
<td>w_{5,1} w_{5,2} ....</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Event counts

<table>
<thead>
<tr>
<th>X=_{w_1}^{Y=sports}</th>
<th>5245</th>
</tr>
</thead>
<tbody>
<tr>
<td>X=_{w_1}^{Y=worldNews}</td>
<td>1054</td>
</tr>
<tr>
<td>X=..</td>
<td>2120</td>
</tr>
<tr>
<td>X=_{w_2}^{Y=..}</td>
<td>37</td>
</tr>
<tr>
<td>X=..</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

What we’d like

<table>
<thead>
<tr>
<th>id</th>
<th>w_{1,1} w_{1,2} w_{1,3} .... w_{1,k_1}</th>
<th>C[X=<em>{w_1}^{Y=sports}=5245, C[X=</em>{w_1}^{Y=..}],C[X=_{w_2}^{Y=...}]</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>w_{2,1} w_{2,2} w_{2,3} ....</td>
<td>C[X=<em>{w_2}^{Y=...}]=1054,..., C[X=</em>{w_2}^{Y=...}]</td>
</tr>
<tr>
<td>id</td>
<td>w_{3,1} w_{3,2} ....</td>
<td>C[X=_{w_3}^{Y=...}]=...</td>
</tr>
<tr>
<td>id</td>
<td>w_{4,1} w_{4,2} ...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
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*

<table>
<thead>
<tr>
<th>( w )</th>
<th>Counts associated with ( W )</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>( C[w^Y=\text{sports}]=2 )</td>
</tr>
<tr>
<td>agent</td>
<td>( C[w^Y=\text{sports}]=1027, C[w^Y=\text{worldNews}]=564 )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zynga</td>
<td>( C[w^Y=\text{sports}]=21, C[w^Y=\text{worldNews}]=4464 )</td>
</tr>
</tbody>
</table>
If these records were in a key-value DB we would know what to do….

Test data

<table>
<thead>
<tr>
<th>id</th>
<th>w_{1,1}</th>
<th>w_{1,2}</th>
<th>w_{1,3}</th>
<th>\ldots</th>
<th>w_{1,k_1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>w_{2,1}</td>
<td>w_{2,2}</td>
<td>w_{2,3}</td>
<td>\ldots</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>w_{3,1}</td>
<td>w_{3,2}</td>
<td>\ldots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>w_{4,1}</td>
<td>w_{4,2}</td>
<td>\ldots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>w_{5,1}</td>
<td>w_{5,2}</td>
<td>\ldots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Record of all event counts for each word

<table>
<thead>
<tr>
<th>w</th>
<th>Counts associated with W</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^{Y}=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^{Y}=sports]=1027, C[w^{Y}=worldNews]=564</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>zynga</td>
<td>C[w^{Y}=sports]=21, C[w^{Y}=worldNews]=4464</td>
</tr>
</tbody>
</table>

Step 2: stream through and for each test case

\[ id_i \ w_{i,1} \ w_{i,2} \ w_{i,3} \ \ldots \ w_{i,k_i} \]

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

<table>
<thead>
<tr>
<th>id</th>
<th>w_1,1</th>
<th>w_1,2</th>
<th>w_1,3</th>
<th>...</th>
<th>w_1,k1</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>w_2,1</td>
<td>w_2,2</td>
<td>w_2,3</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>w_3,1</td>
<td>w_3,2</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>w_4,1</td>
<td>w_4,2</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>w_5,1</td>
<td>w_5,2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Record of all event counts for each word

<table>
<thead>
<tr>
<th>w</th>
<th>Counts associated with W</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=1027,C[w^Y=worldNews]=564</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zynga</td>
<td>C[w^Y=sports]=21,C[w^Y=worldNews]=4464</td>
</tr>
</tbody>
</table>

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

Machine A

- example 1
- example 2
- example 3
- ....

Counting logic

Sort

“C[x] += D”

Machine B

Machine C

- C[x1] += D1
- C[x1] += D2
- ....

Logic to combine counter updates
Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

<table>
<thead>
<tr>
<th>id</th>
<th>w_{1,1} w_{1,2} w_{1,3} .... w_{1,k1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_2</td>
<td>w_{2,1} w_{2,2} w_{2,3} ....</td>
</tr>
<tr>
<td>id_3</td>
<td>w_{3,1} w_{3,2} ....</td>
</tr>
<tr>
<td>id_4</td>
<td>w_{4,1} w_{4,2} ...</td>
</tr>
<tr>
<td>id_5</td>
<td>w_{5,1} w_{5,2} ....</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Record of all event counts for each word

<table>
<thead>
<tr>
<th>w</th>
<th>Counts associated with W</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=1027,C[w^Y=worldNews]=564</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zynga</td>
<td>C[w^Y=sports]=21,C[w^Y=worldNews]=4464</td>
</tr>
</tbody>
</table>

Classification logic

- $W_{1,1}$ 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 aardvark in zynga's farmville today!
id_2 ...
id_3 ....
id_4 ...
id_5 ...
```

Record of all event counts for each word

<table>
<thead>
<tr>
<th>w</th>
<th>Counts associated with W</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=1027, C[w^Y=worldNews]=564</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zynga</td>
<td>C[w^Y=sports]=21, C[w^Y=worldNews]=4464</td>
</tr>
</tbody>
</table>

Classification logic

```
found     ctrs to id_1
aardvark  ctrs to id_1
...       ctrs to id_1
today     ctrs to id_1
...       
```
Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

<table>
<thead>
<tr>
<th>id</th>
<th>Record of all event counts for each word</th>
</tr>
</thead>
<tbody>
<tr>
<td>id₁</td>
<td>found an aardvark in zynga’s farmville today!</td>
</tr>
<tr>
<td>id₂</td>
<td>...</td>
</tr>
<tr>
<td>id₃</td>
<td>...</td>
</tr>
<tr>
<td>id₄</td>
<td>...</td>
</tr>
<tr>
<td>id₅</td>
<td>...</td>
</tr>
</tbody>
</table>

Record of all event counts for each word

<table>
<thead>
<tr>
<th>w</th>
<th>Counts associated with W</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=1027,C[w^Y=worldNews]=564</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zynga</td>
<td>C[w^Y=sports]=21,C[w^Y=worldNews]=4464</td>
</tr>
</tbody>
</table>

Classification logic

- % 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

<table>
<thead>
<tr>
<th>w</th>
<th>Counts associated with W</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=1027, C[w^Y=worldNews]=564</td>
</tr>
</tbody>
</table>

Counter records

\[
\text{found} \sim \text{ctr to } id_1 \\
\text{aardvark} \sim \text{ctr to } id_2 \\
\ldots \\
\text{today} \sim \text{ctr to } id_i \\
\ldots
\]

Combine and sort requests
A stream-and-sort analog of the request-and-answer pattern...

Record of all event counts for each word

<table>
<thead>
<tr>
<th>w</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id1</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id9854</td>
</tr>
<tr>
<td>...</td>
<td>~ctr to id34742</td>
</tr>
<tr>
<td>zynga</td>
<td>~ctr to id1</td>
</tr>
</tbody>
</table>

Counter records

found ~ctr to id₁
aardvark ~ctr to id₁
... ~ctr to id₁
today ~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`"

### Request-handling logic
- `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`"

### Counts

<table>
<thead>
<tr>
<th>w</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=...</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id9854</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id34742</td>
</tr>
<tr>
<td>zynga</td>
<td>C[...]</td>
</tr>
<tr>
<td>zynga</td>
<td>~ctr to id1</td>
</tr>
</tbody>
</table>

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*”

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

<table>
<thead>
<tr>
<th>w</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>aardvark</td>
<td>~ctr to id1</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=...</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id9854</td>
</tr>
<tr>
<td>...</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id34742</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>zynga</td>
<td>C[...]</td>
</tr>
<tr>
<td>zynga</td>
<td>~ctr to id1</td>
</tr>
</tbody>
</table>

Combine and sort requests

Request-handling logic
A stream-and-sort analog of the request-and-answer pattern...

<table>
<thead>
<tr>
<th>w</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>aardvark</td>
<td>~ctr to id1</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=...</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id9854</td>
</tr>
<tr>
<td>...</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id34742</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>zynga</td>
<td>C[...]</td>
</tr>
<tr>
<td>zynga</td>
<td>~ctr to id1</td>
</tr>
</tbody>
</table>

Output:

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

id1 found an aardvark in zynga's farmville today!
id2 ....
id3 ....
id4 ...
id5 ...
..
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td><em>found aardvark zynga farmville today</em></td>
</tr>
<tr>
<td></td>
<td>~ctr for <em>aardvark</em> is $C[w^Y=sports]=2$</td>
</tr>
<tr>
<td></td>
<td>~ctr for <em>found</em> is $C[w^Y=sports]=1027, C[w^Y=worldNews]=564$</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>id2</td>
<td>$w_{2,1} w_{2,2} w_{2,3} ....$</td>
</tr>
<tr>
<td></td>
<td>~ctr for $w_{2,1}$ is ...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>train.dat</th>
<th>counts.dat</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1 w1,1 w1,2 w1,3 …. w1,k1</td>
<td></td>
</tr>
<tr>
<td>id2 w2,1 w2,2 w2,3 ….</td>
<td></td>
</tr>
<tr>
<td>id3 w3,1 w3,2 ….</td>
<td></td>
</tr>
<tr>
<td>id4 w4,1 w4,2 ….</td>
<td></td>
</tr>
<tr>
<td>id5 w5,1 w5,2 ….</td>
<td></td>
</tr>
<tr>
<td>..</td>
<td></td>
</tr>
<tr>
<td>X=w1^Y=sports 5245</td>
<td></td>
</tr>
<tr>
<td>X=w1^Y=worldNews 1054</td>
<td></td>
</tr>
<tr>
<td>X=.. 2120</td>
<td></td>
</tr>
<tr>
<td>X=w2^Y=… 37</td>
<td></td>
</tr>
<tr>
<td>X=… 3</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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
```

<table>
<thead>
<tr>
<th>w</th>
<th>Counts associated with W</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=1027, C[w^Y=worldNews]=564</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zynga</td>
<td>C[w^Y=sports]=21, C[w^Y=worldNews]=4464</td>
</tr>
</tbody>
</table>
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

words.dat

<table>
<thead>
<tr>
<th>w</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>...</td>
</tr>
</tbody>
</table>
... | ...
| zynga | ...

<table>
<thead>
<tr>
<th>w</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id1</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=...</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id9854</td>
</tr>
</tbody>
</table>
... | ~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:

```
id1 ~ctr for aardvark is C[w^Y=sports]=2
...
id1  ~ctr for zynga is ....
...
```

Output looks like this

test.dat

```
id1  found an aardvark in zynga’s farmville today!
id2 ...
id3 ....
id4 ...
id5 ...
..```
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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>found aardvark zynga farmville today</td>
</tr>
<tr>
<td>~ctr for aardvark is C[w^Y=sports]=2</td>
<td></td>
</tr>
<tr>
<td>~ctr for found is C[w^Y=sports]=1027,C[w^Y=worldNews]=564</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>id2</td>
<td>w_{2,1} w_{2,2} w_{2,3} ....</td>
</tr>
<tr>
<td>~ctr for w_{2,1} is ...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>