General hints in studying

• Understand what you’ve done and why
  – There will be questions that test your understanding of the techniques implemented
    • why will/won’t this shortcut work?
    • what does the analysis say about the method?

• We’re mostly about learning meets computation here
  – there won’t be many “pure 601” questions
  – but 601 topics that are discussed in lecture are fair game for questions
General hints in studying

• Techniques covered in class but not assignments:
  – When/where/how to use them
  – That usually includes understanding the analytic results presented in class
  – Eg:
    • is the lazy regularization update an approximation or not? when does it help? when does it not help?
General hints in studying

• What about assignments you haven’t done?
  – You should **read through the assignments** and be familiar with the algorithms being implemented

• There won’t be questions about programming details that you could look up on line
  – but you should know how architectures like Hadoop work (eg, when and where they communicate, how they recover from failure, ...)
  – you should be able to sketch out simple map-reduce algorithms and answer questions about GuineaPig
  – you should be able to read programs in workflow operators and discuss how they’d be implemented and/or how to optimize them
General hints in studying

• There are not detailed questions on the guest speakers
• There might be high-level questions
General hints in exam taking

• You can bring in one 8 ½ by 11” sheet (front and back)
• Look over everything quickly and skip around
  – probably nobody will know everything on the test
• If you’re not sure what we’ve got in mind: state your assumptions clearly in your answer.
  – This is ok even on true/false
• If you look at a question and don’t know the answer:
  – we probably haven’t told you the answer
  – but we’ve told you enough to work it out

  – imagine arguing for some answer and see if you like it
Outline – major topics before midterm

• Hadoop
  – stream-and-sort is how I ease you into that, not really a topic on its own
• GuineaPig, Spark, ...
• Sample algorithms:
  – Phrase finding
• Parallelizing learners (perceptron, ...)
• Hash kernels and streaming SGD
• Distributed SGD for Matrix Factorization

• Some of these are easier to ask questions about than others.
MORE REVIEW SLIDES
What data gets lost if the job tracker is rebooted? If I have a 1Tb file and shard it 1000 ways will it take longer than sharding it 10 ways? Where should a combiner run?
$ hadoop fs -ls rcv1/small/sharded
Found 10 items
-rw-r--r--  3 ...  606405 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00000
-rw-r--r--  3 ... 1347611 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00001
-rw-r--r--  3 ...  939307 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00002
-rw-r--r--  3 ... 1284062 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00003
-rw-r--r--  3 ... 1009890 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00004
-rw-r--r--  3 ... 1206196 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00005
-rw-r--r--  3 ... 1384658 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00006
-rw-r--r--  3 ... 1299698 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00007
-rw-r--r--  3 ...  928752 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00008
-rw-r--r--  3 ...  806030 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00009

$ hadoop fs -tail rcv1/small/sharded/part-00005
weak as the arrival of arbitrated cargoes from the West has put the local market under pressure…
M14,M143,MCAT The Brent crude market on the Singapore International …

Where is this data? How many disks is it on?
If I set up a directory on /afs that looks the same will it work the same? what about a local disk?
Hadoop job_201301231150_0778 on hadoopjt

User: wcohen
Job Name: streamjob6055532903853567038.jar
Job Setup: Successful
Status: Failed
Started at: Wed Jan 30 11:46:47 EST 2013
Failed in: 41sec
Job Cleanup: Successful
Black-listed TaskTrackers: 2
Job Scheduling information: 5 running map tasks using 5 map slots, 0 running reduce tasks using 0 reduce slots.

<table>
<thead>
<tr>
<th>Kind</th>
<th>% Complete</th>
<th>Num Tasks</th>
<th>Pending</th>
<th>Running</th>
<th>Complete</th>
<th>Killed</th>
<th>Failed/Killed Task Attempts</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>100.00%</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>35 / 5</td>
</tr>
<tr>
<td>reduce</td>
<td>0%</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0 / 0</td>
</tr>
</tbody>
</table>

Job Counters

<table>
<thead>
<tr>
<th>Counter</th>
<th>Map</th>
<th>Reduce</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rack-local map tasks</td>
<td>0</td>
<td>0</td>
<td>38</td>
</tr>
<tr>
<td>Launched map tasks</td>
<td>0</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>Data-local map tasks</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Failed map tasks</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Map Completion Graph - close
Hadoop map task list for **job 201301231150 0778** on **hadoop**

### All Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Complete</th>
<th>Status</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>task 201301231150 0778 m 000000</td>
<td>0.00%</td>
<td></td>
<td>30-Jan-2013 11:47:01</td>
<td>30-Jan-2013 11:47:25 (24sec)</td>
<td>java.lang.RuntimeException: PipeMA&lt;br&gt;at org.apache.hadoop.streaming...</td>
</tr>
<tr>
<td>Task Attempts</td>
<td>Machine</td>
<td>Status</td>
<td>Progress</td>
<td>Start Time</td>
<td>Finish Time</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>----------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>attempt_201301231150_0778_m_000000_0</td>
<td>/default-rack/cloud3u12.opencloud</td>
<td>FAILED</td>
<td>0.00%</td>
<td>30-Jan-2013 11:47:01</td>
<td>30-Jan-2013 11:47:06 (4sec)</td>
</tr>
<tr>
<td>attempt_201301231150_0778_m_000000_1</td>
<td>/default-rack/cloud2u28.opencloud</td>
<td>FAILED</td>
<td>0.00%</td>
<td>30-Jan-2013 11:47:07</td>
<td>30-Jan-2013 11:47:11 (4sec)</td>
</tr>
</tbody>
</table>
Why do I see this same error over and over again?
Other systems: Spark

```scala
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.random(D) // current separating plane
for (i <- 1 to ITERATIONS) {
  val gradient = points.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x)))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w += gradient
}
println("Final separating plane: " + w)
```

Note that w gets shipped automatically to the cluster with every map call.

Faster or slower than Hadoop? Why?
Other systems: GPig

What needs to fit in memory here?
How would you avoid that if you had too?
What is the tradeoff if you did?

def loadDictView(view):
    result = {}
    for (key, val) in GPig.rowsOf(view):
        result[key] = val
    return result

class TFIDF(Planner):

    D = GPig.getParam()
    data = ReadLines(D.get('corpus', 'idcorpus.txt')) \
    | Map(by=lambda line: line.strip().split(), split=\n    | Map(by=lambda (docid, doc): (docid, doc.lower().split()), split=\n    | FlatMap(by=lambda (docid, words): map(lambda w: (docid, w), words))

    # compute document frequency and inverse doc freq
    docFreq = Distinct(data) \
    | Group(by=lambda (docid, term): term, retaining=lambda (docid, term): docid, reducingTo=ReduceToCount())

    ndoc = Map(data, by=lambda (docid, term): docid) \n    | Distinct() \n    | Group(by=lambda row: 'ndoc', reducingTo=ReduceToCount())

    inverseDocFreq = Augment(docFreq, sideview=ndoc, loadedBy=lambda v: GPig.onlyRowOf(v)) \
    | Map(by=lambda ((term, df), (dummy, ndoc)): (term, math.log(nndoc/df)))

    # compute unweighted document vectors
    udocvec = Augment(data, sideview=inverseDocFreq, loadedBy=loadDictView) \
    | Map(by=lambda ((docid, term), idfDict): (docid, term, idfDict[term]))

    # normalize
    norm = Group( udocvec, by=lambda (docid, term, weight): docid, 
                  retaining=lambda (docid, term, weight): weight*weight, 
                  reducingTo=ReduceToSum())

    docvec = Augment(udocvec, sideview=norm, loadedBy=loadDictView) \
    | Map(by=lambda ((docid, term, weight), normDict): (docid, term, weight/math.sqrt(normDict[docid])))
PARALLEL LEARNERS
Parallelizing perceptrons – take 2

\[ w \text{(previous)} \]

\[ \text{Instances/labels} \]

\[ \text{Split into example subsets} \]

\[ \text{Compute local } v_k \text{'s} \]

\[ w - 1 \]

\[ w - 2 \]

\[ w - 3 \]

\[ w \]

\[ \text{Combine by some sort of weighted averaging} \]

19
A theorem

**Theorem 3.** Assume a training set $\mathcal{T}$ is separable by margin $\gamma$. Let $k_{i,n}$ be the number of mistakes that occurred on shard $i$ during the $n$th epoch of training. For any $N$, when training the perceptron with iterative parameter mixing (Figure 3),

$$\sum_{n=1}^{N} \sum_{i=1}^{S} \mu_{i,n} k_{i,n} \leq \frac{R^2}{\gamma^2}$$

**Corollary:** if we weight the vectors uniformly, then the number of mistakes is still bounded.

I.e., this is “enough communication” to guarantee convergence.

I probably won’t ask about the proof, but I could definitely ask about the theorem.
What does the word “structured” mean here? Why is it important? Would the results be better or worse with a regular perceptron?
STREAMING SGD
Learning as optimization for regularized logistic regression

• Algorithm:

\[ w^j = w^j + \lambda (y - p) x^j - \lambda 2\mu w^j \]

1. Initialize a hashtable \( W \)

2. For \( t = 1, \ldots, T \)
   - For each example \( x_i, y_i \):
     - Compute the prediction for \( x_i \):
       \[ p_i = \frac{1}{1 + \exp(-\sum_{j:x_i^j > 0} x_i^j w^j)} \]
     - For each non-zero feature of \( x_i \) with index \( j \) and value \( x^j \):
       * If \( j \) is not in \( W \), set \( W[j] = 0 \).
       * Set \( W[j] = W[j] + \lambda (y - p) x^j - \lambda 2\mu w^j \)

3. Output the hash table \( W \).
Formalization of the “Hash Trick”:

First: Review of Kernels

What is it? how does it work? what aspects of performance does it help? What did we say about it formally?