Asynchronous Peer-to-Peer Data Mining

Róbert Ormándi, István Hegedűs, and Márk Jelasity
Motivation

- For P2P networks the only viable communication model is the **chaotic** model
  - Eventual update assumption, but
  - No reliability assumptions for individual messages
- In P2P networks one needs to build global models of distributed data for
  - Recommendations, spam filtering, optimizations of performance
- Does not hurt to have privacy preservation
P2P chaotic algorithms

• Examples of algorithms for which no provably chaotic version is known (not clear if possible at all)
  – Peer sampling
  – Average calculation

• Example of algorithms that do work in this model
  – Chaotic iteration (Lubachevsky and Mitra, 1986)

• We illustrate the problems through averaging
Illustration of averaging
Illustration of averaging

request

12

6

2

3

7

8
Illustration of averaging

reply

12

3

7

6

2

8
Illustration of averaging

(12+6)/2=9
Problems

- **Message delay: overlapping updates compromise the system state**
  - Can be solved by blocking (deadlock can be avoided) or maintaining state for each link (Mehyar et al, 2007)
  - Can be also solved by the push-sum algorithm (this is the more elegant way) (Kempe et al, 2003)

- **Message drop failures**
  - No known solutions!
  - TCP is of course good enough in practice, but we do not want to rely on it in the chaotic model
Classification problem in machine learning

- We are given a set of \((x_i, y_i)\) examples, where \(y_i\) is the class of \(x_i\) (\(y_i\) is eg. -1 or 1)
- We want a model \(f()\), such that for all \(i\), \(f(x_i) = y\)
- \(f()\) is very often a parameterized function \(f_w()\), and the classification problem becomes an error minimization problem in \(w\).
  - Neural net weights, linear model parameters, etc
- The error is often defined as a sum of errors over the examples
Illustration of classification with a linear model
Classification vs averaging

- **Classification is more general**
  - $f_w()$ can be in any form, and an optimal $w$ can be any global function of the examples

- **Classification is still easier**
  - We do not always want a specific $w$, but instead we want one that classifies the examples well
  - Even if unique, an optimal $w$ can depend on a small subset of examples only (borderline cases)
P2P classification

- So the problem is to solve the error minimization problem to find w (ie, train the model)
- For different model families (neural nets, support vector machines, etc) different sophisticated optimization algorithms are used most of which are very difficult to parallelize, let alone asynchronously
- Except stochastic gradient descent
  - Not always the most efficient but
  - A very generic method, very simple, and ideal for parallelization in our model
Stochastic gradient descent

- Assume the error is defined as
\[ \text{Err}(w) = \sum_{i=1}^{n} \text{Err}(w, x_i) \]
- Then the gradient is
\[ \frac{\partial \text{Err}(w)}{\partial w} = \sum_{i=1}^{n} \frac{\partial \text{Err}(w, x_i)}{\partial w} \]
- So the full gradient method looks like
\[ w(t+1) = w(t) - \alpha(t) \sum_{i=1}^{n} \frac{\partial \text{Err}(w, x_i)}{\partial w} \]
- But one can take only one example at a time iterating in random order over examples
\[ w(t+1) = w(t) - \alpha(t) \frac{\partial \text{Err}(w, x_i)}{\partial w} \]
P2P stochastic gradient

1: initModel()
2: loop
3: wait(Δ)
4: p ← selectPeer()
5: m ← selectModel()
6: send model(m) to p
7: end loop
8: procedure ONRECEIVEMODEL(m)
9: m ← updateModel(m)
10: currentModel ← m
11: modelQueue.add(m)
12: end procedure

- SelectPeer is implemented by a peer sampling service for uniform random samples
- Depending on the implementation of updateModel we get different ML methods
- What can go wrong due to failures and delays is only the sampling distribution
P2P stochastic gradient

- Local prediction is based on locally available model(s)
- One can use the current model
- Or a combination of available models via eg voting

1: procedure PREDICT($x$)
2:     $w \leftarrow$ currentModel
3:     return sign($\langle w, x \rangle$)
4: end procedure

5: procedure VOTEDPREDICT($x$)
6:     pRatio $\leftarrow$ 0
7: for $m \in$ modelQueue do
8:     if sign($\langle m.w, x \rangle$) $\geq$ 0 then
9:         pRatio $\leftarrow$ pRatio + 1
10:     end if
11: end for
12: return sign(pRatio/modelQueue.size() $-$ 0.5)
13: end procedure
Experiments

• We implemented a support vector machine with stochastic gradient (Pegasos alg.)

• We used several benchmark data sets for evaluations
  – Data is fully distributed: one data point per node

• We used extreme scenarios
  – 50% message drop rate
  – 1-10 cycles of message delay
  – Churn modeled after the FileList.org trace from Delft
## The data sets

<table>
<thead>
<tr>
<th></th>
<th>Iris1</th>
<th>Iris2</th>
<th>Iris3</th>
<th>Reuters</th>
<th>SpamBase</th>
<th>Malicious10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training set size</strong></td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>2000</td>
<td>4140</td>
<td>2155622</td>
</tr>
<tr>
<td><strong>Test set size</strong></td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>600</td>
<td>461</td>
<td>240508</td>
</tr>
<tr>
<td><strong>Number of features</strong></td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>9947</td>
<td>57</td>
<td>10</td>
</tr>
<tr>
<td><strong>Classlabel ratio</strong></td>
<td>50/50</td>
<td>50/50</td>
<td>50/50</td>
<td>1300/1300</td>
<td>1813/2788</td>
<td>792145/1603985</td>
</tr>
<tr>
<td><strong>Pegasos 20000 iter.</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.025</td>
<td>0.111</td>
<td>0.080 (0.081)</td>
</tr>
<tr>
<td><strong>Pegasos 1000 iter.</strong></td>
<td>0</td>
<td>0</td>
<td>0.4</td>
<td>0.057</td>
<td>0.137</td>
<td>0.095 (0.060)</td>
</tr>
<tr>
<td><strong>SVMLight</strong></td>
<td>0</td>
<td>0</td>
<td>0.1</td>
<td>0.027</td>
<td>0.074</td>
<td>0.056 (→)</td>
</tr>
</tbody>
</table>

- Properties of data sets in the experiments
- Prediction error of baseline algorithms
Remarks regarding the chaotic model

- If uniformity of random walk is guaranteed, then all the models converge to the true model eventually, irrespective of all failures.

- If no uniformity can be guaranteed, but the local data is statistically independent of the visiting probability, then again convergence to the true model is guaranteed in general for all models.

- If no uniformity and no independence could be guaranteed, convergence to a good model is still ensured provided that the data is separable, and all misclassified examples are visited “often enough.”
Work in progress!

- Privacy preservation
  - Promising candidate, because the own value is revealed only in the first step at the moment

- Boosting efficiency
  - Models can be based on an exponential number of examples if we also average models at nodes (and not only update them); this seems to work...

- Dynamic fine-tuning
  - Based on local information the voting procedure, starting off new models, and deleting old ones could be controlled smartly