Gossip Learning

Márk Jelasity
Motivation

- Explosive growth of smart phone platforms, and
- Availability of sensor and other contextual data
- Makes collaborative data mining possible
  - Health care: following and predicting epidemics, personal diagnostics
  - Smart city: traffic optimization, accident forecasting
  - (predicting earthquakes, financial applications, etc)
- P2P networks, grid, etc, are also relevant platforms
P2P system model

- Large number (millions or more) computers (nodes)
- Packet switched communication
  - Every node has an address
  - Any node can send a message to any given address
    - Not actually true: NATs, firewalls
- Messages can be delayed or lost, nodes can crash (unreliable asynchronous communication)
Fully distributed data

- Horizontal data distribution
- Every node has very few records, we assume they have only one
- We do not allow for moving data, only local processing (privacy preservation)
- We require that the models are cheaply available for all the nodes
P2P or Servers (cloud)?

- The cloud is flexible and scalable but it is not free and not public: business model needed
  - It is cheap but with LOTS of data and communication it will get expensive
- Privacy is a concern
- P2P is more limited in what it can do
  - But not as much as it seems at first!
- Smart phones (unlike motes) are increasingly powerful devices
- P2P and cloud hybrids possible (the network can act as a sensor!)
Illustration: averaging
Illustration: averaging

kérés
Illustration: averaging

válasz
Illustration: averaging

\[(12+6)/2=9\]
The Point

- Asynchronous distributed numeric algorithms exist that
  - Are very fast
  - Are very simple to implement (but not always simple to analyze)
  - Provide almost exact, or often exact results in the face of chaotic communication

- Eg power method, gossip based aggregation

- Our ambition is to achieve this for data mining algorithms (we focus on classification now)
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
**Stochastic gradient descent**

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

- So the problem is to find an optimization method that fits into our system and data model.
- Most distributed methods build local models and then combine these through ensemble learning: but we don't have enough local data.
- Online algorithms
  - Need only one data record at a time
  - They update the model using this record
- The stochastic gradient method is a popular online learning algorithm (we apply it to the primal form of the SVM error function).
Gossip learning

Algorithm 1 Gossip Learning Scheme

1: initModel()
2: loop
3: \( \text{wait}(\Delta) \)
4: \( p \leftarrow \text{selectPeer()} \)
5: \( \text{currentModel} \leftarrow \text{createModel()} \)
6: \( \text{send currentModel to } p \)
7: end loop
8:
9: procedure ONRECEIVEMODEL\((m)\)
10: \( \text{modelQueue}.\text{add}(m) \)
11: end procedure

1: procedure CREATEMODELRW
2: \( m \leftarrow \text{modelQueue}.\text{first()} \)
3: \( \text{update}(m) \)
4: \( \text{return } m \)
5: end procedure
6:
7: procedure CREATEMODELMU
8: \( m_1 \leftarrow \text{modelQueue}.\text{first()} \)
9: \( m_2 \leftarrow \text{modelQueue}.\text{second()} \)
10: \( m \leftarrow \text{merge}(m_1, m_2) \)
11: \( \text{update}(m) \)
12: \( \text{return } m \)
13: end procedure
The merge function

- Let $z = \text{merge}(x, y) = (x + y)/2$ ($x$ and $y$ are linear models)
- In the case of the Adaline perceptron
  - Updating $z$ using an example has the same effect as updating $x$ and $y$ with the same example and then averaging these two updated models
  - Making predictions using $z$ is the same as calculating the weighted average of the predictions of $x$ and $y$
- This means we effectively propagate an exponential number of models, and the voting of these is our prediction
- For the linear SVM algorithm this is only a heuristic argument
Local prediction

- We use only local models
  - The current model
  - Or voting over a number of recent models

```latex
1: \textbf{procedure} PREDICT(x) \\
2: \quad w \leftarrow \text{currentModel} \\
3: \quad \text{return} \ \text{sign}(\langle w, x \rangle) \\
4: \text{end procedure}

5: \textbf{procedure} VOTEDPREDICT(x) \\
6: \quad \text{pRatio} \leftarrow 0 \\
7: \quad \text{for} \ m \in \text{modelQueue} \ \text{do} \\
8: \quad \quad \text{if} \ \text{sign}(\langle m.w, x \rangle) \geq 0 \ \text{then} \\
9: \quad \quad \quad \text{pRatio} \leftarrow \text{pRatio} + 1 \\
10: \quad \quad \text{end if} \\
11: \quad \text{end for} \\
12: \quad \text{return} \ \text{sign}(\text{pRatio}/\text{modelQueue.size()-0.5}) \\
13: \text{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
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>Training set size</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>2000</td>
<td>4140</td>
<td>2155622</td>
</tr>
<tr>
<td>Test set size</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>600</td>
<td>461</td>
<td>240508</td>
</tr>
<tr>
<td>Number of features</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>9947</td>
<td>57</td>
<td>10</td>
</tr>
<tr>
<td>Classlabel ratio</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>Pegasos 20000 iter.</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>Pegasos 1000 iter.</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>SVMLight</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>

- Statistics of data sets
- The performance of some known algorithms
Without merge
Additional results

- We implemented multiclass boosting in the gossip framework
- We developed techniques for dealing with concept drift
  - The algorithm is running continuously
  - We keep the age distribution of models fixed
  - At any point in time we have good models
Adaptivity
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 we will have no bias, but variance will increase (effectively we work with fewer samples).

- 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”
Publications

