Motivation

• Data is accumulated in data centers
• Costly storage and processing
  – Maintenance, Infrastructure, Privacy
• Limited access
  – For researchers as well
• But, data was produced by us
Motivation – ML Applications

• Personalized Queries
  – Web search history
• Recommender Systems
  – Clicks on items and movies
• Document Clustering
• Spam Filtering
• Image Segmentation
  – We annotate pictures
Motivation – ML Applications

- Devices have internet connection
- Can communicate each other
- Store data as well
- Build models on the data collaboratively
Example: compute min value

5 3 7 9 2 4 8
Example: compute min value

5 3 7 9 2 4 8
Example: compute min value
Example: compute min value

3 5 3 7 9 2 4 8
Example: compute min value

3 5 3 7 9 2 4 8
Example: compute min value

\[ 2 \ 5 \ 3 \ 7 \ 9 \ 2 \ 4 \ 8 \]
Example: compute min value

2 5 3 7 9 2 4 8
Example: compute min value
Example: compute min value

Centralized Algorithm
- Upload data
- Can not use the model without communication
Example: compute min value

2 5 3 7 9 2 4 8
Example: compute min value

2 5 3 7 9 2 4 8
Example: compute min value
Example: compute min value
Example: compute min value

2 5 3 7 9 2 4 8
Example: compute min value

2 5 3 7 9 2 4 8

Distributed Algorithm
• More communication
• Can use the model without communication
Classification

- Binary classification
  - Given a set of training samples: \((x_1, y_1), \ldots, (x_n, y_n)\)
  where \(x_i \in \mathbb{R}^d\)
Classification

- Binary classification

  - Given a set of training samples: \((x_1, y_1), \ldots, (x_n, y_n)\)

where \(x_i \in \mathbb{R}^d\) and \(y_i \in \{-1, 1\}\)
Classification

• Binary classification
  – Given \((x_1, y_1), \ldots, (x_n, y_n)\) training samples, where \(x_i \in \mathbb{R}^d\) and \(y_i \in \{-1, 1\}\)
  – Task: looking for a model \(f : \mathbb{R}^d \rightarrow \{-1, 1\}\) that correctly separates the samples from different classes (minimizes the number of misclassifications)

\[
\min_f \sum_{i=1}^{n} (f(x_i) - y_i)^2 \quad i = 1, \ldots, n
\]
Classification

- Binary classification
  - Given \((x_1, y_1), \ldots, (x_n, y_n)\) training samples, where \(x_i \in \mathbb{R}^d\) and \(y_i \in \{-1, 1\}\)
  - Task: looking for a model \(f: \mathbb{R}^d \to \{-1, 1\}\) that minimizes the error
    \[
    \min_{f} \sum_{i=1}^{n} (f(x_i) - y_i)^2
    \]
  - In linear case the model is a hyper-plane \((\mathbf{w})\)
Classification

- Binary classification
  - Given \((x_1, y_1), \ldots, (x_n, y_n)\) training samples, where \(x_i \in \mathbb{R}^d\) and \(y_i \in \{-1, 1\}\)
  - Task: looking for a model \(f: \mathbb{R}^d \rightarrow \{-1, 1\}\) minimizes the error
    \[
    \min_f \sum_i (f(x_i) - y_i)^2 \quad i = 1, \ldots, n
    \]
  - In linear case the model is a hyper-plane (\(\mathbf{w}\))
  - The label of a new instance can be predicted
Gossip Learning

- ML is often an optimization problem
- Local data is not enough
Gossip Learning

- ML is often an optimization problem
- Local data is not enough
- Models are sent and updated on nodes
Gossip Learning

- ML is often an optimization problem
- Local data is not enough
- Models are sent and updated on nodes
  - Taking random walks
  - Updated instance-by-instance
  - Data is never sent
Gossip Learning

- ML is often an optimization problem
- Local data is not enough
- Models are sent and updated on nodes
  - Taking random walks
  - Updated instance-by-instance
  - Data is never sent
- Stochastic Gradient Descent (SGD)
SGD

- Objective function

\[ w = \arg \min_w J(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_w(x_i), y_i) + \frac{\lambda}{2} \|w\|^2 \]
SGD

- Objective function

\[ w = \arg \min_w J(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_w(x_i), y_i) + \frac{\lambda}{2} \|w\|^2 \]

- Gradient method

\[ w_{t+1} = w_t - \eta_t \left( \frac{\partial J}{\partial w} \right) = w_t - \eta_t (\lambda w + \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(f_w(x_i), y_i)) \]
SGD

- Objective function
  
  $$w = \arg \min_w J(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_w(x_i), y_i) + \frac{\lambda}{2} \|w\|^2$$

- Gradient method

  $$w_{t+1} = w_t - \eta_t \left( \frac{\partial J}{\partial w} \right)$$

  $$= w_t - \eta_t (\lambda w + \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(f_w(x_i), y_i))$$

- SGD, data can be processed online (instance by instance)
SGD

- **Objective function**
  \[ w = \arg\min_w J(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_w(x_i), y_i) + \frac{\lambda}{2} \|w\|^2 \]

- **Gradient method**
  \[ w_{t+1} = w_t - \eta_t \frac{\partial J}{\partial w} \]
  \[ = w_t - \eta_t (\lambda w + \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(f_w(x_i), y_i)) \]

- **SGD, data can be processed online (instance by instance)**

- **Gossip Learning**
GoLF (Gossip Learning Framework)

Algorithm 1 Skeleton of original GoLF learning protocol

1: currentModel ← initModel()
2: loop
3: wait(Δ)
4: p ← selectPeer()
5: sendModel(p, currentModel)
6: procedure onReceiveModel(m)
7: m.updateModel(x, y)
8: currentModel ← m
GoLF (Gossip Learning Framework)

Algorithm 1 Skeleton of original GoLF learning protocol

1: currentModel ← initModel()
2: loop
3:   wait(Δ)
4:   p ← selectPeer()
5:   sendModel(p, currentModel)
6:   procedure onReceiveModel(m)
7:     m.updateModel(x, y)
8:     currentModel ← m

wait (Δ)
GoLF (Gossip Learning Framework)

Algorithm 1 Skeleton of original GoLF learning protocol

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td><code>currentModel ← initModel()</code></td>
</tr>
<tr>
<td>2:</td>
<td><code>loop</code></td>
</tr>
<tr>
<td>3:</td>
<td><code>wait(Δ)</code></td>
</tr>
<tr>
<td>4:</td>
<td><code>p ← selectPeer()</code></td>
</tr>
<tr>
<td>5:</td>
<td><code>sendModel(p, currentModel)</code></td>
</tr>
<tr>
<td>6:</td>
<td><code>procedure onReceiveModel(m)</code></td>
</tr>
<tr>
<td>7:</td>
<td><code>m.updateModel(x, y)</code></td>
</tr>
<tr>
<td>8:</td>
<td><code>currentModel ← m</code></td>
</tr>
</tbody>
</table>

![Diagram of computer network with select node]
GoLF (Gossip Learning Framework)

Algorithm 1: Skeleton of original GoLF learning protocol

1: currentModel ← initModel()
2: loop
3:   wait(Δ)
4:   p ← selectPeer()
5:   sendModel(p, currentModel)
6: procedure onReceiveModel(m)
7:   m.updateModel(x, y)
8:   currentModel ← m
**GoLF (Gossip Learning Framework)**

**Algorithm 1** Skeleton of original GoLF learning protocol

1: `currentModel ← initModel()`
2: **loop**
3: \[\text{wait}(\Delta)\]
4: \[p ← \text{selectPeer()}\]
5: \[\text{sendModel}(p, \text{currentModel})\]
6: **procedure** `onReceiveModel(m)`
7: \[m.\text{updateModel}(x, y)\]
8: \[\text{currentModel} ← m\]
GoLF (Gossip Learning Framework)

Algorithm 1 Skeleton of original GoLF learning protocol

1: \( \text{currentModel} \leftarrow \text{initModel()} \)
2: loop
3: \( \text{wait}(\Delta) \)
4: \( p \leftarrow \text{selectPeer()} \)
5: \( \text{sendModel}(p, \text{currentModel}) \)
6: procedure \( \text{onReceiveModel}(m) \)
7: \( m.\text{updateModel}(x, y) \)
8: \( \text{currentModel} \leftarrow m \)
Gossip-Based Learning

- SGD-based machine learning algorithms can be applied, e.g.
  - Logistic Regression
  - Support Vector Machines
  - Perceptron
  - Artificial Neural Networks

- Training data never leave the nodes

- Models can be used locally additional communication is not required
GoLF (Gossip Learning Framework)

Algorithm 1 Skeleton of original GoLF learning protocol

1: currentModel ← initModel()
2: loop
3: wait(Δ)
4: p ← selectPeer()
5: sendModel(p, currentModel)
6: procedure onReceiveModel(m)
7: m.updateModel(x, y)
8: currentModel ← m
GoLF (Gossip Learning Framework)

Algorithm 1 Skeleton of original GoLF learning protocol

1: currentModel ← initModel()
2: loop
3:  wait(Δ)
4:  p ← selectPeer()
5:  sendModel(p, currentModel)
6:  procedure onReceiveModel(m)
7:    m.updateModel(x, y)
8:    currentModel ← m
GoLF (Gossip Learning Framework)

Algorithm 1 Skeleton of original GoLF learning protocol

1: \( \textit{currentModel} \leftarrow \textit{initModel()} \)
2: loop
3: \( \text{wait}(\Delta) \)
4: \( p \leftarrow \textit{selectPeer}() \)
5: \( \text{sendModel}(p, \textit{currentModel}) \)
6: \textit{procedure onReceiveModel}(m)
7: \( m.\text{updateModel}(x, y) \)
8: \( \textit{currentModel} \leftarrow m \)
Algorithm 1 Skeleton of original GoLF learning protocol

1: \textit{currentModel} ← initModel()
2: \textbf{loop}
3: \hspace{1em} wait(\Delta)
4: \hspace{1em} \textit{p} ← selectPeer()
5: \hspace{1em} sendModel(\textit{p, currentModel})

6: \textbf{procedure} onReceiveModel(\textit{m})
7: \hspace{1em} \textit{m}.updateModel(\textit{x}, \textit{y})
8: \hspace{1em} \textit{currentModel} ← \textit{m}
GoLF (Gossip Learning Framework)

Algorithm 1 Skeleton of original GoLF learning protocol

1: \( currentModel \leftarrow initModel() \)
2: \( \text{loop} \)
3: \( \text{wait}(\Delta) \)
4: \( p \leftarrow \text{selectPeer}() \)
5: \( \text{sendModel}(p, currentModel) \)
6: \( \text{procedure onReceiveModel}(m) \)
7: \( m.\text{updateModel}(x, y) \)
8: \( currentModel \leftarrow m \)
Low-rank Matrix Decomposition

- Given a matrix $A$
- Looking for matrices $X$, $Y$ that $XY^T \sim A$
- $k << \min(n,m)$
- Data compression
Low-rank Matrix Decomposition

- Given a matrix $A$
- Looking for matrices $X$, $Y$ that $XY^T \sim A$
- $k \ll \min(n,m)$
- Data compression

$$X_{m \times k}$$

$$A_{m \times n}$$

$$x_i$$

$$a_{ij}$$

$$Y^T_{k \times n}$$

$$y_j^T$$
Low-rank Matrix Decomposition

\[ J(X, Y) = \frac{1}{2} \| A - X Y^T \|_F^2 \]

\[ = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} (a_{ij} - \sum_{l=1}^{k} x_{il} y_{jl})^2 \]

\[ \frac{\partial J}{\partial X} = (X Y^T - A) Y, \quad \frac{\partial J}{\partial Y} = (Y X^T - A^T) X \]

SGD: \[ w_{t+1} = w_t - \eta_t \nabla_w E_w \]
Low-rank Matrix Decomposition

- User related data: a row of $X$ and $A$
- Random walk: $Y$

\[
J(X, Y) = \frac{1}{2} \|A - XY^T\|_F^2
\]

\[
= \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} (a_{ij} - \sum_{l=1}^{k} x_{il}y_{jl})^2
\]

\[
\frac{\partial J}{\partial X} = (XY^T - A)Y, \quad \frac{\partial J}{\partial Y} = (YX^T - A^T)X
\]

SGD: $w_{t+1} = w_t - \eta_t \nabla_w E_w$
Low-rank Matrix Decomposition

- User related data: a row of $X$ and $A$
- Random walk: $Y$

Algorithm 1 P2P low–rank factorization at node $i$

1: $a_i$ \hspace{1cm} \triangleright row $i$ of $A$
2: initialize $Y$
3: initialize $x_i$ \hspace{1cm} \triangleright row $i$ of $X$
4: loop
5: \hspace{0.5cm} wait($\Delta$)
6: \hspace{0.5cm} $p \leftarrow$ selectPeer()
7: \hspace{0.5cm} send $Y$ to $p$
8: end loop
9: procedure ON_RECEIVE $Y(\tilde{Y})$
10: \hspace{0.5cm} $Y \leftarrow \tilde{Y}$
11: \hspace{0.5cm} $(Y, x_i) \leftarrow$ update($Y, x_i, a_i$)
12: end procedure
Low-rank Matrix Decomposition

Algorithm 1 P2P low-rank factorization at node $i$

1: $a_i$  \hspace{1cm} \triangleright \text{row } i \text{ of } A
2: initialize $Y$
3: initialize $x_i$  \hspace{1cm} \triangleright \text{row } i \text{ of } X
4: loop
5: \hspace{1cm} wait($\Delta$)
6: \hspace{2cm} $p \leftarrow \text{selectPeer}()$
7: \hspace{2cm} send $Y$ to $p$
8: end loop
9: \text{procedure ONRECEIVEY($\hat{Y}$)}
10: \hspace{1cm} $Y \leftarrow \hat{Y}$
11: \hspace{2cm} $(Y, x_i) \leftarrow \text{update}(Y, x_i, a_i)$
12: end procedure

Algorithm 2 rank-$k$ update at node $i$

1: $\eta$  \hspace{1cm} \triangleright \text{learning rate}
2: \text{procedure UPDATE($Y$, $x_i$, $a_i$)}
3: \hspace{1cm} err $\leftarrow a_i - x_i Y^T$
4: \hspace{2cm} $x'_i \leftarrow x_i + \eta \cdot \text{err} \cdot Y$
5: \hspace{2cm} $Y' \leftarrow Y + \eta \cdot \text{err}^T \cdot x_i$
6: \text{return} $(Y', x'_i)$
7: end procedure

\[
\frac{\partial J}{\partial X} = (XY^T - A)Y, \quad \frac{\partial J}{\partial Y} = (YX^T - A^T)X
\]
Singular Value Decomposition

- Columns of $X$ and $Y$ should have the same directions as singular vectors.
- We use:
  \[ A = U \Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T \]
  \[ X^* = U_k \Sigma_U, \quad Y^* = V_k \Sigma_V \]
- Compute rank-1 approximations
  \[ X_1 Y_1^T = X^* Y^*^T = \sigma_1 u_1 v_1^T \]
  \[ A - X_1 Y_1^T = \sum_{i=2}^{r} \sigma_i u_i v_i^T \]
sequentially
Singular Value Decomposition

- Columns of X and Y should have the same directions as singular vectors.
- We use:

\[ A = U \Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T \]

**Algorithm 3** rank-\(k\) SVD update at node \(i\)

1: \(\eta\)  \hspace{1cm} \triangleright \text{learning rate}
2: **procedure** UPDATE\((Y, x_i, a_i)\)
3: \(a'_i \leftarrow a_i\)
4: **for** \(\ell = 1\) to \(k\) **do**  \hspace{1cm} \triangleright \text{column } \ell \text{ of } Y
5: \(\text{err} \leftarrow a'_i - x_{i\ell} \cdot y_{\ell}^T\)
6: \(x_{i\ell}' \leftarrow x_{i\ell} + \eta \cdot \text{err} \cdot y_{\ell}\)
7: \(y_{\ell}' \leftarrow y_{\ell} + \eta \cdot \text{err}^T \cdot x_{i\ell}\)
8: \(a'_i = a'_i - x_{i\ell} \cdot y_{\ell}'\)
9: **end for**
10: **return** \((Y', x_i')\)
11: **end procedure**
SVD in Practice

- Dimensionality reduction (PCA)

\[ \mathbf{X}_{m \times k} \quad A_{m \times n} \]

original data space

PCA

component space

Gene 3

PC 2

PC 1

Gene 2

Gene 1
SVD in Practice

- Topic modeling (LSI)

\[ X_{m \times k} \]

\[ A_{m \times n} \]
SVD in Practice

- Multidimensional Scaling (MDS)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOS</td>
<td>BOST</td>
<td>0</td>
<td>206</td>
<td>429</td>
<td>1504</td>
<td>963</td>
<td>2976</td>
<td>3095</td>
<td>2979</td>
</tr>
<tr>
<td>NY</td>
<td>206</td>
<td>0</td>
<td>233</td>
<td>1308</td>
<td>802</td>
<td>2815</td>
<td>2934</td>
<td>2786</td>
<td>1771</td>
</tr>
<tr>
<td>DC</td>
<td>429</td>
<td>233</td>
<td>0</td>
<td>1075</td>
<td>671</td>
<td>2684</td>
<td>2799</td>
<td>2631</td>
<td>1616</td>
</tr>
<tr>
<td>MIAMI</td>
<td>1504</td>
<td>1308</td>
<td>1075</td>
<td>0</td>
<td>1329</td>
<td>3273</td>
<td>3053</td>
<td>2687</td>
<td>2037</td>
</tr>
<tr>
<td>CHICAGO</td>
<td>963</td>
<td>802</td>
<td>671</td>
<td>1329</td>
<td>0</td>
<td>2013</td>
<td>2142</td>
<td>2054</td>
<td>996</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>2976</td>
<td>2815</td>
<td>2684</td>
<td>3273</td>
<td>2013</td>
<td>0</td>
<td>808</td>
<td>1131</td>
<td>1307</td>
</tr>
<tr>
<td>SF</td>
<td>3095</td>
<td>2934</td>
<td>2799</td>
<td>3053</td>
<td>2142</td>
<td>808</td>
<td>0</td>
<td>379</td>
<td>1235</td>
</tr>
<tr>
<td>LA</td>
<td>2979</td>
<td>2786</td>
<td>2631</td>
<td>2687</td>
<td>2054</td>
<td>1131</td>
<td>379</td>
<td>0</td>
<td>1058</td>
</tr>
<tr>
<td>DENVER</td>
<td>1949</td>
<td>1771</td>
<td>1616</td>
<td>2037</td>
<td>996</td>
<td>1307</td>
<td>1235</td>
<td>1059</td>
<td>0</td>
</tr>
</tbody>
</table>
SVD in Practice

- Graph Spectral Clustering

\[ X_{m \times k} \]
\[ A_{m \times n} \]
\[ Y_{k \times n} \]

FIG. 18  Basic principle of the spectral algorithm by Capocci et al. (Capocci et al., 2005). The bottom diagram shows the values of the components of the second eigenvector of the right stochastic matrix for the graph drawn on the top. The three plateaus of the eigenvector components correspond to the three evident communities of the graph. Reprinted figures with permission from Ref. (Capocci et al., 2005). ©2005 by Elsevier.
Conclusion

- Machine learning without collecting data
- Prediction without extra communication

- Federated Learning
  https://research.googleblog.com/2017/04/federated-learning-collaborative.html