Distributed forests for MapReduce-based machine learning

Ryoji Wakayama, Ryuei Murata, Akisato Kimura, Takayoshi Yamashita, Yuji Yamauchi, Hironobu Fujiyoshi
(1) Dept. Robotics Science & Technology, Chubu University.
(2) NTT Communication Science Laboratories.
1-page summary

“Distributed Forests” : Training random forests in parallel, suitable for MapReduce clusters.

- Accelerate the training of random forests
- while maintaining test performances
- Naïve parallelizations easily overfit
The era of big data
Parallel processing with MapReduce

Data

Sub-task
(solved in parallel)

Intermediate results

Outputs

Cats

Dogs

Birds

Fishes
Parallel processing with MapReduce

- **Map Stage** (on worker nodes)
  
- **Intermediate results**
  
- **Reduce stage** (to a master node)
  
- **Outputs**
  - Cats
  - Dogs
  - Birds
  - Fishes
Parallel processing with MapReduce

Data

Outputs

Cats
Dogs
Birds
Fishes

Intermediate results

Take advantage of the locality of data
Random forests

Random forests are suitable for parallel processing.

- Composed of multiple decision trees
- Can be independently trained

Previous studies [Li+ PAKDD12][Assuncao+ BigData13] have already reported parallel implementations of random forest training. → great improvements of computational costs.
What’s the problem?

Easily overfit the training data $\Rightarrow$ poor test performance

Training

Biased samples

Overfitting

Testing
What’s the problem?

Easily overfit the training data → poor test performance
Distributed Forests - Contributions

• Accelerate the training of random forests
• while maintaining the test performances.

(1) “Share forests” are built in advance and shared with all the worker nodes

(2) Avoid overfitting by introducing transfer learning

(3) Trees responsible for poor performances are eliminated.
Distributed Forests - Contributions

• Accelerate the training of random forests
• while maintaining the test performances.

(1) “Share forests” are built in advance and shared with all the worker nodes

(2) Avoid overfitting by introducing transfer learning

(3) Trees responsible for poor performances are eliminated.
Map stage

Consider a scenario for distributed file systems

- All the training data have already split into local training data and distributed to worker nodes.

Each worker node has access to only its local training data.
→ Resorts the locality of data.
Map stage

- **Shared data** $D_s$
- **Shared forest** $T_{ws}$
- Local training data $D_{w1}$, $D_{w2}$, $D_{wN}$
- Worker nodes

The shared forest can be built in advance **only just once**. → Can reuse shared data/forest repeatedly for various datasets.

Shared data and forest are shared with all the worker nodes.
Map stage

**Auxiliary domain** (many samples)

Exploit Transfer Forest (PS3-35) as a transfer learning method.

**Target domain** (much less samples)
Reduce stage

**Computational cost in testing**
The number of trees should be as small as possible.

**Classification performance**
Blindly removing trees sacrifices the performance.

**Purpose**
Eliminating undesirable trees from the candidate forests.

Diagram:
- Local training data $D_{w2}$
- Worker nodes
- Candidate forests
- Master node
- Final forests $T_w$
Reduce stage

Find undesirable trees based on posteriors

\[ p(\star |s; t_1) = 0.9 \]
\[ p(\square |s; t_1) = 0.1 \]
\[ p(\star |s; t_2) = 0.2 \]
\[ p(\square |s; t_2) = 0.8 \]
Reduce stage

Remove irrelevant trees based on posteriors

**Score for a tree** (lower $\rightarrow$ removed)

$$Score(t) = -\sum_s \max_{c \neq c(s)} p(c|s; t)$$

- $p(\star|s; t_1) = 0.9$
- $p(\square|s; t_1) = 0.1$
- $p(\star|s; t_2) = 0.2$
- $p(\square|s; t_2) = 0.8$
Experiments

[Dataset]
• Letter Recognition (20,000 samples, 16 dims, 26 classes)
• MNIST (70,000 samples, 784 dims, 10 classes)

[Settings]
• Shared data: 4,000 (Letter), 10,000 (MNIST).
• Test data: 6,666 (Letter), 10,000 (MNIST).
• The rests are all for local training data.
• 50 trees constructed in total in the Map stage.

[Implementations]
• Intel Xeon E5-2637 v2 / Windows Server / C++
Classification performances
(for heavily biased local training data)

<table>
<thead>
<tr>
<th># Workers</th>
<th>RF (Letter)</th>
<th>RF (MNIST)</th>
<th>DF (Letter)</th>
<th>DF (MNIST)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.18</td>
<td>5.31</td>
<td>5.14</td>
<td>5.31</td>
</tr>
<tr>
<td>2</td>
<td>9.92</td>
<td>11.2</td>
<td>5.71</td>
<td>6.59</td>
</tr>
<tr>
<td>5</td>
<td>26.4</td>
<td>32.1</td>
<td>8.28</td>
<td>7.03</td>
</tr>
<tr>
<td>10</td>
<td>67.7</td>
<td>88.4</td>
<td>9.53</td>
<td>7.48</td>
</tr>
</tbody>
</table>
Classification performances
(for heavily biased local training data)

<table>
<thead>
<tr>
<th># Workers</th>
<th>RF (Letter)</th>
<th>RF (MNIST)</th>
<th>DF (Letter)</th>
<th>DF (MNIST)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.18</td>
<td>5.31</td>
<td>5.14</td>
<td>5.31</td>
</tr>
<tr>
<td>2</td>
<td>9.92</td>
<td>11.2</td>
<td>5.71</td>
<td>6.59</td>
</tr>
<tr>
<td>5</td>
<td>26.4</td>
<td>32.1</td>
<td>8.28</td>
<td>7.03</td>
</tr>
<tr>
<td>10</td>
<td>67.7</td>
<td>88.4</td>
<td>9.53</td>
<td>7.48</td>
</tr>
</tbody>
</table>

A naïve implementation considerably sacrificed test performance.

Our proposed method achieved better test performance.
Speed vs. Accuracy

(for unbiased local training data)
The proposed method accelerated the training process as the number of worker nodes increased, while maintaining classification performances.
We intentionally injected mislabeled samples into local training data, so that irrelevant trees were often generated on those worker nodes.
Effect of Reduce stage

Our propose method effectively removed irrelevant trees while maintaining classification performances, compared with random eliminations.

Classifcation Error [%] vs Ratio of Mislabelling [%]

- Random (30 trees)
- Proposed (30 trees)
- No deletion (50 trees)
Conclusion

“Distributed Forests” : Training random forests in parallel, suitable for MapReduce clusters.

(1) Share forests

(2) Transfer Forests

(3) Irrelevant tree elimination

Future work

• More sophisticated implementation on real MapReduce architectures.
• Experiments with much larger datasets, extensions to other types of ML tasks.