On-line Random Forests

Amir Saffari, Christian Leistner, Jakob Santner
Martin Godec, Horst Bischof

Institute for Computer Graphics and Vision
Graz University of Technology, Austria

October 3, 2009
Motivations

- Random Forest (RF) is an ensemble of random trees.
Motivations

- Random Forest (RF) is an ensemble of random trees.
- RFs achieve state-of-the-art performance in many applications.
Motivations

- Random Forest (RF) is an ensemble of random trees.
- RFs achieve state-of-the-art performance in many applications.
- It is fast both during the training and testing phase.
Motivations

- Random Forest (RF) is an ensemble of random trees.
- RFs achieve state-of-the-art performance in many applications.
- It is fast both during the training and testing phase.
- It is easy to implement them in a distributed computing environment or on multi-core CPUs/GPUs.
Motivations

- Random Forest (RF) is an ensemble of random trees.
- RFs achieve state-of-the-art performance in many applications.
- It is fast both during the training and testing phase.
- It is easy to implement them in a distributed computing environment or on multi-core CPUs/GPUs.
- RFs are inherently multi-class classifiers.
Motivations

- Random Forest (RF) is an ensemble of random trees.
- RFs achieve state-of-the-art performance in many applications.
- It is fast both during the training and testing phase.
- It is easy to implement them in a distributed computing environment or on multi-core CPUs/GPUs.
- RFs are inherently multi-class classifiers.
- On-line learning is needed for many applications where the size of the data is huge or the data is available from a stream.
Decision Trees

\[ \mathcal{X} = \{(x_1, y_1), \ldots, (x_N, y_N)\}, \quad x_i = [x_i^1, \ldots, x_i^D]^T, \quad y_i \in \{1, \ldots, K\} \]

\[ p = [p_1, \ldots, p_K]^T \]

① Split Node
② Leaf Node
Decision Trees

\[ X = \{(x_1, y_1), \ldots, (x_N, y_N)\}, \ x_i = [x_i^1, \ldots, x_i^D]^T, \ y_i \in \{1, \ldots, K\} \]

Test: \( g_p(x) > \theta_p, \ g(x) \in \mathcal{G} \)

Gain: \( \Delta L = I_{j\bar{j}} - \frac{|\bar{j}|}{|j|} I_{j\bar{j}r} - \frac{|\bar{j}|}{|j|} I_{\bar{j}\bar{j}} \)

Gini index: \( L = \sum_{k=1}^{K} p_k(1 - p_k) \)

Entropy: \( L = -\sum_{k=1}^{K} p_k \log(p_k) \)

Feature Test: \( \mathcal{G} = \{x^1, \ldots, x^D\} \)
\[ \mathcal{X} = \{ (x_1, y_1), \ldots, (x_N, y_N) \}, \quad x_i = [x_i^1, \ldots, x_i^D]^T, \quad y_i \in \{1, \ldots, K\} \]

Test: \( (g_p(x), \theta_p) : x^p > \theta_p \)

Test: \( (g_r(x), \theta_r) : x^r > \theta_r \)
Decision Trees

Test sample: \( x \)

\[
p(y = k | x) = p_k
\]
Decision Trees

- Decision tree is a greedy method which uses a local optimization.
Decision Trees

- Decision tree is a greedy method which uses a local optimization.
- The class of tests could be limited since for finding the best split an optimization step is required.
Decision Trees

- Decision tree is a greedy method which uses a local optimization.
- The class of tests could be limited since for finding the best split an optimization step is required.
- Decision tree is very sensitive to data noise.
Ensemble of Bagged Trees

L. Breiman (1996)

Subsample with replacement: $\mathcal{X} \rightarrow \mathcal{X}_i \cup \mathcal{X}_o$

Train with in-bag-samples: $\mathcal{X}_i$  Evaluate with out-of-bag-samples: $\mathcal{X}_o$

Out-of-bag-error

Refinement
Ensemble of Bagged Trees

Test sample: $x$

$$p(y = k | x) = \frac{1}{T} \sum_{t=1}^{T} p_t(y = k | x)$$
Random Forests

L. Breiman (2001)

\[ \mathcal{X} = \{(x_1, y_1), \ldots, (x_N, y_N)\}, \quad x_i = [x_i^1, \ldots, x_i^D]^T, \quad y_i \in \{1, \ldots, K\} \]

Set of Tests \( S = \{(g_1(x), \theta_1), \ldots, (g_M(x), \theta_M)\} \)

Gain \( \Delta L = L_j - \frac{|j_r|}{|j|} L_{j_r} - \frac{|j_l|}{|j|} L_{j_l} \)

Feature Test \( \mathcal{G} = \{x^1, \ldots, x^D\} \)

Hyperplane Test \( \mathcal{G} = \{g_w(x) = w^T x | w \in \mathbb{R}^D\} \)
Elements of On-line Learning

Sample \((x, y)\) is arriving sequentially from a stream.
Sample \((x, y)\) is arriving sequentially from a stream.

- On-line bagging.
- On-line random tree growing mechanism.
On-line Bagging

Oza and Russell (2001):

- Draw a random integer: \( k \sim \text{Poisson}(\lambda) \)
On-line Bagging

Oza and Russell (2001):

- Draw a random integer: $k \sim \text{Poisson}(\lambda)$
- If $k > 0$:
  - Train the model (tree) on $(x, y)$ $k$ times.
- else:
  - Use $(x, y)$ to compute the out-of-bag-error and refinement.
Optimizing the structure of a tree on-line is difficult.

\[(x_i, y_i)\]
On-line Random Tree

$(x_i, y_i)$

Set of Tests: $\mathcal{S} = \{(g_1(x), \theta_1), \cdots, (g_M(x), \theta_M)\}$

$g_1(x) > \theta_1$

$g_2(x) > \theta_2$

$\cdots$

$g_M(x) > \theta_M$
On-line Random Tree

\[(x_i, y_i)\]

Node size: \(|j| > \alpha\)

Gain: \(\Delta L_m > \beta\)

Set of Tests: \(S = \{(g_1(x), \theta_1), \ldots, (g_M(x), \theta_M)\}\)

\[g_1(x) > \theta_1\]

\[g_2(x) > \theta_2\]

\[\cdots\]

\[g_M(x) > \theta_M\]
On-line Random Tree

\((x_i, y_i)\)

\(g_m(x) > \theta_m\)
On-line Random Tree

\[(x_i, y_i)\]

\[S\]

Left

\[g_1(x) > \theta_1\]

\[g_2(x) > \theta_2\]

\[\ldots\]

\[g_M(x) > \theta_M\]

Right

\[g_1(x) > \theta_1\]

\[g_2(x) > \theta_2\]

\[\ldots\]

\[g_M(x) > \theta_M\]
Temporal Knowledge Weighting

- In some applications, the distribution of the data is changing over time.
In some applications, the distribution of the data is changing over time.

- Select a tree randomly from \( \{ t | t \in \{1, \ldots, T\}, a_t > 1/\gamma \} \).
- If \( OOB_{E_t} > \text{rand}() \)
  - Discard the \( t \)-th tree
  - \( f_t = \text{newTree()} \)
We set: $T = 200$, $\alpha = 0.1 \times N_{\text{train}}$, $\beta = 0.1$.

For on-line boosting models, we use 50 selectors with 10 decision stumps in each selector and for multi-class datasets we use a 1-vs-all strategy.

Code is available at: 
www.ymer.org/amir/software/online-random-forests

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Train</th>
<th># Test</th>
<th># Class</th>
<th>#Feat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mushrooms</td>
<td>6000x20</td>
<td>2124</td>
<td>2</td>
<td>112</td>
</tr>
<tr>
<td>DNA</td>
<td>1400x20</td>
<td>1186</td>
<td>3</td>
<td>180</td>
</tr>
<tr>
<td>SatImage</td>
<td>3104x20</td>
<td>2000</td>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>USPS</td>
<td>7291x20</td>
<td>2007</td>
<td>10</td>
<td>256</td>
</tr>
<tr>
<td>Letter</td>
<td>15000x20</td>
<td>5000</td>
<td>26</td>
<td>16</td>
</tr>
</tbody>
</table>
# Machine Learning Datasets - Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Off-line RF</th>
<th>On-line RF</th>
<th>On-line Ada</th>
<th>On-line Logit</th>
<th>On-line Savage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mushrooms</td>
<td>0.010</td>
<td>0.012</td>
<td>0.013</td>
<td>0.012</td>
<td>0.013</td>
</tr>
<tr>
<td>DNA</td>
<td>0.109</td>
<td>0.112</td>
<td>0.173</td>
<td>0.117</td>
<td><strong>0.097</strong></td>
</tr>
<tr>
<td>SatImage</td>
<td>0.113</td>
<td><strong>0.118</strong></td>
<td>0.257</td>
<td><strong>0.152</strong></td>
<td>0.156</td>
</tr>
<tr>
<td>USPS</td>
<td>0.078</td>
<td>0.086</td>
<td>0.224</td>
<td><strong>0.134</strong></td>
<td>0.139</td>
</tr>
<tr>
<td>Letter</td>
<td>0.097</td>
<td><strong>0.104</strong></td>
<td>0.263</td>
<td><strong>0.223</strong></td>
<td>0.241</td>
</tr>
</tbody>
</table>
Machine Learning Datasets - Results
Tracking

- We only use simple Haar-features, without implementing any rotation and scale search and avoid any other engineering methods.
- We use 100 trees, $\alpha = 100$, and $\beta = 0.1$.
- For the on-line boosting, we use 50 selectors with each 150 features.
- We evaluate over public datasets: *Occluded Face, David Indoor, Sylvester, Rotating Girl*.
- An implementation of the on-line RF on a common NVidia GPU allows an additional 10-times speed up.
We only use simple Haar-features, without implementing any rotation and scale search and avoid any other engineering methods.

We use 100 trees, $\alpha = 100$, and $\beta = 0.1$.

For the on-line boosting, we use 50 selectors with each 150 features.

We evaluate over public datasets: *Occluded Face, David Indoor, Sylvester, Rotating Girl*.

An implementation of the on-line RF on a common NVidia GPU allows an additional 10-times speed up.

Video
Interactive Segmentation

- We use the interactive segmentation algorithm of Santner et al. (BMVC 2009).
- It uses the off-line RF to learn a foreground model, which then is used as a prior for a weighted Total Variation based segmentation algorithm.
- We replace the off-line RF with our on-line version.
- Both the on-line RF and the segmentation are implemented on a GPU.
Interactive Segmentation

Saffari et al.
On-line Random Forests
Comparison to On-line Boosting

- Robustness to label noise.
Comparison to On-line Boosting

- Robustness to label noise.
- Proper plasticity/elastcity trade-off.
Comparison to On-line Boosting

- Robustness to label noise.
- Proper plasticity/elacticity trade-off.
- Shrinkage factor effect.
Comparison to On-line Boosting

- Robustness to label noise.
- Proper plasticity/elasticity trade-off.
- Shrinkage factor effect.
- Inherently multi-class.
Comparison to On-line Boosting

- Robustness to label noise.
- Proper plasticity/elacticity trade-off.
- Shrinkage factor effect.
- Inherently multi-class.
- Suitable for GPU/multi-core/distributed computing.
Thank you!
Code available at:
www.ymer.org/amir/software/online-random-forests