Realization of Random Forest for Real-Time Evaluation through Tree Framing

Sebastian Buschjäger, Kuan-Hsun Chen, Jian-Jia Chen and Katharina Morik

TU Dortmund University - Artifical Intelligence Group and Design Automation for Embedded Systems Group

November 18, 2018

Motivation

FACT First G-APD Cherenkov Telescope continously monitors the sky for gamma rays
Goal Have a small, cheap telescope which can be deployed everywhere on earth

Motivation

FACT First G-APD Cherenkov Telescope continously monitors the sky for gamma rays
Goal Have a small, cheap telescope which can be deployed everywhere on earth

- It produces roughly 180 MB/s of data
- Only 1 in 10.000 measurements is interesting
- Bandwidth to transmit measurements is limited

Motivation

FACT First G-APD Cherenkov Telescope continously monitors the sky for gamma rays
Goal Have a small, cheap telescope which can be deployed everywhere on earth

- It produces roughly 180 MB/s of data
- Only 1 in 10.000 measurements is interesting
- Bandwidth to transmit measurements is limited

Idea Use a Random Forest to filter measurements before further processing

- Pre-train forest on simulated data, then apply it in the real world
- Physicist know Random Forests
- Very good black-box learner, no hyperparameter tuning necessary

Motivation

FACT First G-APD Cherenkov Telescope continously monitors the sky for gamma rays
Goal Have a small, cheap telescope which can be deployed everywhere on earth

- It produces roughly 180 MB/s of data
- Only 1 in 10.000 measurements is interesting
- Bandwidth to transmit measurements is limited

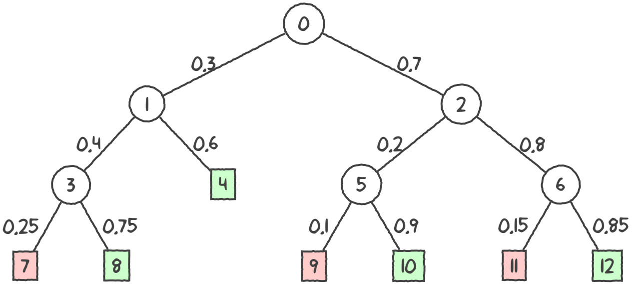
Idea Use a Random Forest to filter measurements before further processing

- Pre-train forest on simulated data, then apply it in the real world
- Physicist know Random Forests
- Very good black-box learner, no hyperparameter tuning necessary

Goal Execute Random Forest in real-time and keep-up with 180 MB/s of data

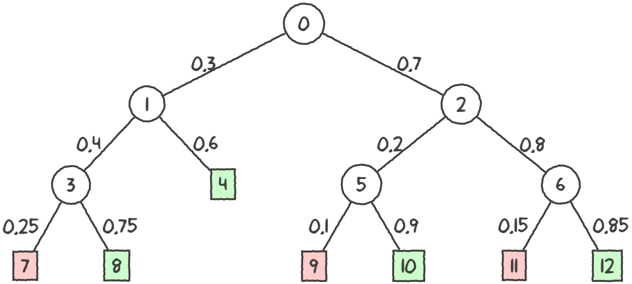
Constraint Size and energy available is limited → Model must run on embedded system

Recap Decision Trees and Random Forest



- DTs split the data in regions until each region is “pure”
- Splits are binary decisions if x belongs to certain region
- Leaf nodes contain actual prediction for a given region
- RFs built multiple DTs on subsets of the data/features

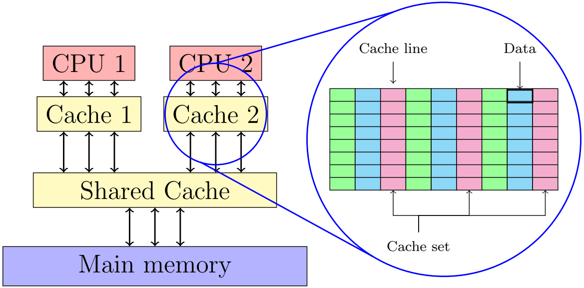
Recap Decision Trees and Random Forest



- DTs split the data in regions until each region is "pure"
- Splits are binary decisions if *x* belongs to certain region
- Leaf nodes contain actual prediction for a given region
- RFs built multiple DTs on subsets of the data/features

Question How to implement a Decision Tree / Random Forest?

Recap Computer architecture



- CPU computations are much faster than memory access
- Memory-Hierarchy (Caches) is used to hide slow memory
- Caches assume spatial-temporal locality of accesses

Question How to implement a Decision Tree / Random Forest?

Implementing Decision Trees (1)

Fact There are at-least two ways to implement DTs in modern programming languages

Native-Tree Store nodes in array and iterate it in a loop

Implementing Decision Trees (1)

Fact There are at-least two ways to implement DTs in modern programming languages

Native-Tree Store nodes in array and iterate it in a loop

```
Node t[] = {/* ... */};
bool predict(short const * x){
    unsigned int i = 0;
    while(!t[i].isLeaf) {
        if (x[t[i].f] <= t[i].s) {
            i = t[i].l;
        } else {
            i = t[i].r;
        }
    }
    return t[i].pred;
}
```

+ Simple to implement
+ Small 'hot'-code
- Requires D-Cache (array)
- Requires I-Cache (code)
- Requires indirect memory access

Implementing Decision Trees (2)

Fact There are at-least two ways to implement DTs in modern programming languages

If-Else-Tree Unroll tree into `if-else` instructions

Implementing Decision Trees (2)

Fact There are at-least two ways to implement DTs in modern programming languages

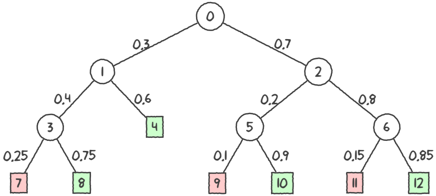
If-Else-Tree Unroll tree into `if-else` instructions

```
bool predict(short const * x){
    if(x[0] <= 8191){
        if(x[1] <= 2048){
            return true;
        } else {
            return false;
        }
    } else {
        if(x[2] <= 512){
            return true;
        } else {
            return false;
        }
    }
}
```

+ No indirect memory access
+ Compiler can optimize aggressively
+ Only I-Cache required
- I-Cache usually small
- No ‘hot’-code

Probabilistic execution model of DTs

Basic idea Analyse the structure of trained tree and keep most important paths in Cache



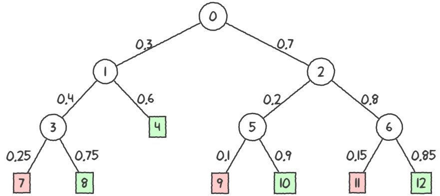
Branch-probability $p_{i \to j}$

Path-probability $p(\pi) = p_{\pi_0 \to \pi_1} \cdot \ldots \cdot p_{\pi_{L-1} \to \pi_L}$

Expected path length $\mathbb{E}[L] = \sum_\pi p(\pi) \cdot |\pi|$

Probabilistic execution model of DTs

Basic idea Analyse the structure of trained tree and keep most important paths in Cache



Branch-probability $p_{i \to j}$

Path-probability $p(\pi) = p_{\pi_0 \to \pi_1} \cdot \ldots \cdot p_{\pi_{L-1} \to \pi_L}$

Expected path length $\mathbb{E}[L] = \sum_{\pi} p(\pi) \cdot |\pi|$

Example

$$p((0,1,3)) = 0.3 \cdot 0.4 \cdot 0.25 = 0.03$$
$$p((0,2,6)) = 0.7 \cdot 0.8 \cdot 0.85 = 0.476$$

Probabilistic optimizations for DTs

Capacity misses Cache memory is not enough to store all code

But Computation kernel of tree might fit into cache

Probabilistic optimizations for DTs

Capacity misses Cache memory is not enough to store all code

But Computation kernel of tree might fit into cache

Solution Compute computation kernel for budget β

$$\mathcal{K} = \arg\max \left\{ p(T) \middle| T \subseteq \mathcal{T} \text{ s.t. } \sum_{i \in T} s(i) \leq \beta \right\}$$

Probabilistic optimizations for DTs

Capacity misses Cache memory is not enough to store all code

But Computation kernel of tree might fit into cache

Solution Compute computation kernel for budget β

$$\mathcal{K} = \arg\max \left\{ p(T) \middle| T \subseteq \mathcal{T} \text{s.t.} \sum_{i \in T} s(i) \leq \beta \right\}$$

- Start with the root node
- Greedily add nodes until budget is exceeded

Note

- Estimate $s(\cdot)$ based on assembly analysis
- Choose β based on the properties of specific CPU model

Probabilistic optimizations for DTs (2)

Further optimizations

- Reduce memory consumption of nodes for `native` trees with clever implementation
- Increase cache-hit rate for `if-else` trees by swapping nodes with higher probability

Probabilistic optimizations for DTs (2)

Further optimizations

- Reduce memory consumption of nodes for `native` trees with clever implementation
- Increase cache-hit rate for `if-else` trees by swapping nodes with higher probability

In total Compare 1 baseline method and 4 different implementations

Probabilistic optimizations for DTs (2)

Further optimizations

- Reduce memory consumption of nodes for `native` trees with clever implementation
- Increase cache-hit rate for `if-else` trees by swapping nodes with higher probability

In total Compare 1 baseline method and 4 different implementations

Questions

- What is the performance-gain of these optimizations?
- How do these optimizations perform on different CPU architectures?
- How do these optimizations perform with different forest configurations?

Experimental Setup

Approach

- Use a Code-Generator to compile `sklearn` forests (DTs,RF,ET) of varying size to `C-Code`
- Test resulting code + optimizations on 12 datatest on 3 different CPU architectures

Experimental Setup

Approach

- Use a Code-Generator to compile `sklearn` forests (DTs,RF,ET) of varying size to `C-Code`
- Test resulting code + optimizations on 12 datatest on 3 different CPU architectures

Hardware

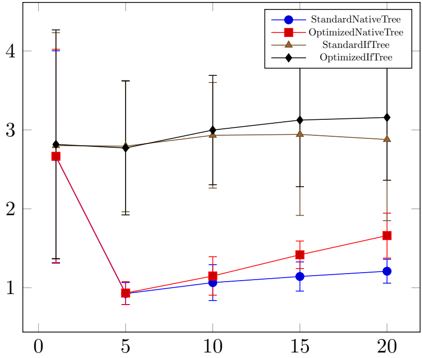
- **X86** Desktop PC with Intel i7-6700 with 16 GB RAM
- **ARM** Raspberry-Pi 2 with ARMv7 and 1GB RAM
- **PPC** NXP Reference Design Board with T4240 processors and 6GB RAM

Experimental Setup (2)

Dataset	# Examples	# Features	Accuracy
adult	8141	64	0.76 - 0.86
bank	10297	59	0.86 - 0.90
covertype	145253	54	0.51 - 0.88
fact	369450	16	0.81 - 0.87
imdb	25000	10000	0.54 - 0.80
letter	5000	16	0.06 - 0.95
magic	4755	10	0.64 - 0.87
mnist	10000	784	0.17 - 0.96
satlog	2000	36	0.40 - 0.90
sensorless	14628	48	0.10 - 0.99
wearable	41409	17	0.57 - 0.99
wine-quality	1625	11	0.49 - 0.68

Results: Desktop PC with Intel (X86)

Note Behaviour similar for DTs, RF and ET → Focus in RF here



Results

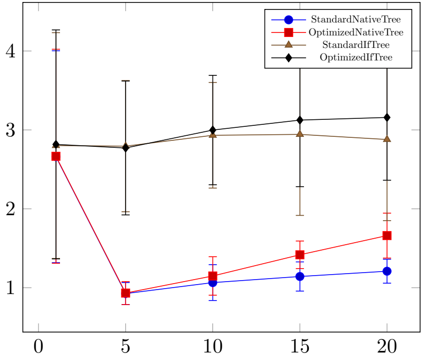
- Optimizations improve performance
- `if-else` trees are clear winner

Interpretation

- Large I-Cache (256 KiB) favors `if-else`
- Compiler can utilize CISC architecture for `if-else`
- `Native` trees do not benefit from I-Cache and CISC

Results: Desktop PC with Intel (X86)

Note Behaviour similar for DTs, RF and ET → Focus in RF here



Results

- Optimizations improve performance
- `if-else` trees are clear winner

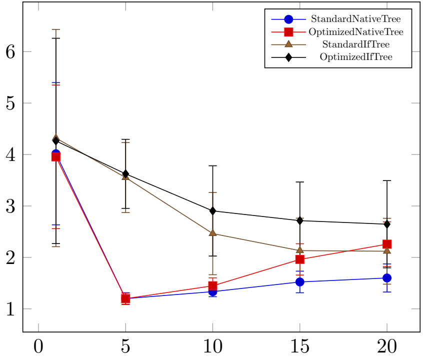
Interpretation

- Large I-Cache (256 KiB) favors `if-else`
- Compiler can utilize CISC architecture for `if-else`
- `Native` trees do not benefit from I-Cache and CISC

Take-away On X86 CPUs, `if-else` trees should be favoured

Results: Raspberry Pi with ARMv7 (ARM)

Note Behaviour similar for DTs, RF and ET → Focus in RF here



Results

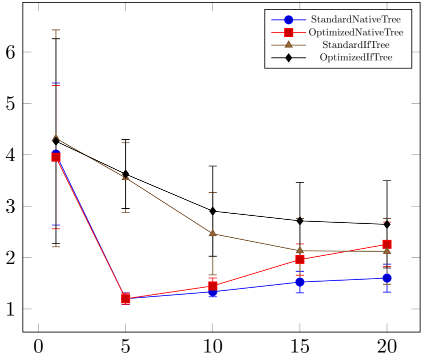
- Optimizations improve performance
- No clear winner for larger trees

Interpretation

- Smaller I-Cache (32 KiB) only fits small trees
- Smaller D-Cache (512 KiB) only fits small trees
- Requires more instructions than CISC

Results: Raspberry Pi with ARMv7 (ARM)

Note Behaviour similar for DTs, RF and ET → Focus in RF here



Results

- Optimizations improve performance
- No clear winner for larger trees

Interpretation

- Smaller I-Cache (32 KiB) only fits small trees
- Smaller D-Cache (512 KiB) only fits small trees
- Requires more instructions than CISC

Take-away Use `if-else` version for small trees. For larger ones there is no clear recommendation

Summary and Take-Aways

Modern physics experiments generate huge amounts of data

But We can use ML to filter-out unwanted measurements before further processing

Summary and Take-Aways

Modern physics experiments generate huge amounts of data

But We can use ML to filter-out unwanted measurements before further processing

Our approach Use a code-generator to generate optimized RF code

- There are at-least two ways to implement Decision Trees in modern languages
- `Native` trees mostly rely on the data cache
- `If-else` trees mostly rely on the instruction cache
- Careful cache management can increase performance by 2 – 6 (1500 compared to `sklearn`)
- Optimizations & implementations behave differently on different CPU architectures

Summary and Take-Aways

Modern physics experiments generate huge amounts of data

But We can use ML to filter-out unwanted measurements before further processing

Our approach Use a code-generator to generate optimized RF code

- There are at-least two ways to implement Decision Trees in modern languages
- `Native` trees mostly rely on the data cache
- `If-else` trees mostly rely on the instruction cache
- Careful cache management can increase performance by 2 – 6 (1500 compared to `sklearn`)
- Optimizations & implementations behave differently on different CPU architectures

Now Physicist can generate optimized C code for each new experiment

And you as well!

`https://bitbucket.org/sbuschjaeger/arch-forest`