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

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

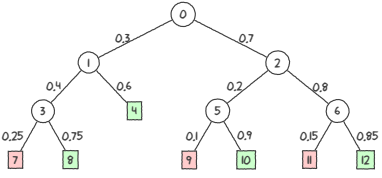
TU Dortmund University - Artifical Intelligence Group and Design Automation for Embedded Systems Group

July 7, 2019

Decision Trees and Computer Architectures

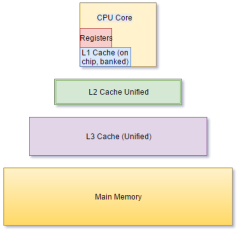
Recap Decision Trees and Random Forest

- 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 Computer architecture

- Memory-hierarchy (Caches) is used to hide slow memory
- Compulsory cache miss: Data is accessed for the first time
- Capacity cache miss: Complete Cache is full
- Conflict cache miss: Cannot access empty part of cache



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

```
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;
        }
    }
}
```

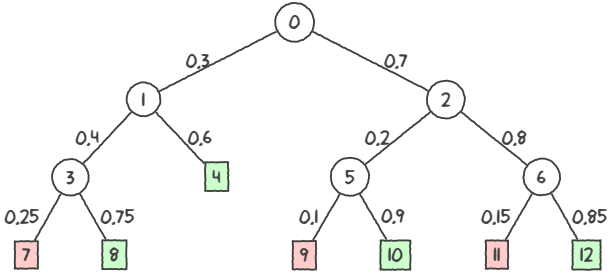
+ No indirect memory access
+ Compiler can optimize aggressivley
+ Only I-Cache required
- I-Cache usually small
- No ‘hot’-code

Probabilistic execution model of DTs

Observation Caches assume spatial-temporal locality of accesses.
But If data is not in the Cache, a miss occurs

Thus Minimize misses and maximize locality

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



Optimizations for Native Trees (1)

Compulsory cache misses

- Cache memory is not enough to hold complete array
- Leaf-nodes only store the prediction. Pointer to children not necessary

Solution Store prediction directly in 'parent' node

```
struct Node {
        bool isLeaf;
        unsigned int prediction;
        unsigned char feature;
        unsigned short split;
        unsigned char leftChild;
        unsigned char rightChild;
};
```

10 Byte

```
struct Node {
        unsigned char feature;
        unsigned short split;
        unsigned char leftChild;
        unsigned char rightChild;
        unsigned char indicator;
};
```

6 Byte

Optimizations for Native Trees (2)

Capacity and conflict cache misses

- Pre-fetching does not work, if nodes are discontinuously arranged
- Layout nodes in array so that they respect access pattern

Solution Greedily put nodes with highest probability in same cache set

- Put the root node into current working set C. Set $i = 0$
- If $|C| \leq \tau$: $C = C \cup \arg\max(p(i \to l(i)), p(i \to r(i)))$
- Continue until $|C| \geq \tau$
- Place nodes in C continuously in array

Optimizations for If-Else Trees (1)

Compulsory cache misses

- Cache memory is not enough to store all code
- Increase chance, that nodes with higher probabilities are in the cache

Solution Swap nodes if $p(i \to l(i)) \geq p(i \to r(i))$

```
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;
        }
    }
}
```

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

Optimizations for If-Else Trees (2)

Capacity and conflict cache misses

- Cache memory is not enough to store all code
- 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 exceeded

Note Estimate $s(\cdot)$ based on assembly analysis

Experimental setup

Approach Perform a total of 1800 experiments

- 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

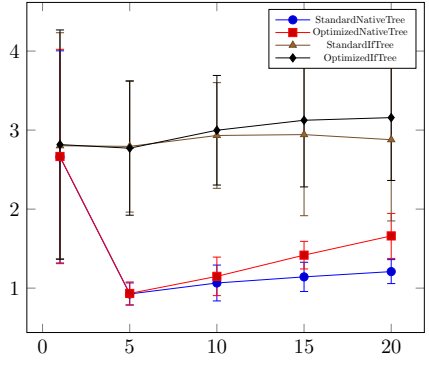
- **X86** Desktop PC with Intel i7-6700 with 16 GB RAM
- **ARM** Raspberry-Pi 2 with ARMv7 and 1GB RAM
- **PPC** NXP Reference Design Board with T4240 processors and 6GB RAM

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

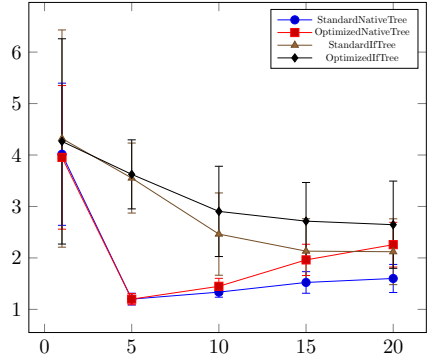
Interpretation

- 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

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

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

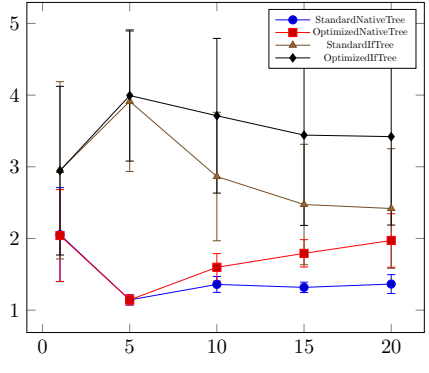
Interpretation

- 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 reccomandation

Results: NXP Reference Design Board with T4240 (PPC)

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



Results

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

Interpretation

- Small trees fit into small level 1 I-Cache (32 KiB)
- Larger trees rely on larger level 2 Cache (2MiB)
- Requires more instructions than CISC

Take-away On PPC CPUs, `if-else` trees should be favored