Decision Tree and Random Forest Implementations for Fast Filtering of Sensor Data

[sebastian.buschjaeger|katharina.morik]@tu-dortmund.de

technische universität dortmund

Fakultät für Informatik
Lehrstuhl für
Künstliche Intelligenz

FACT Telescope

Filtering Sensor Data

Produces: Roughly 180 MB/s of data

But: Only 1 in 10.000 measurements is interesting

Idea: Use a Random Forest to filter measurements before processing

Question: Which system can keep up with 180 MB/s of data?

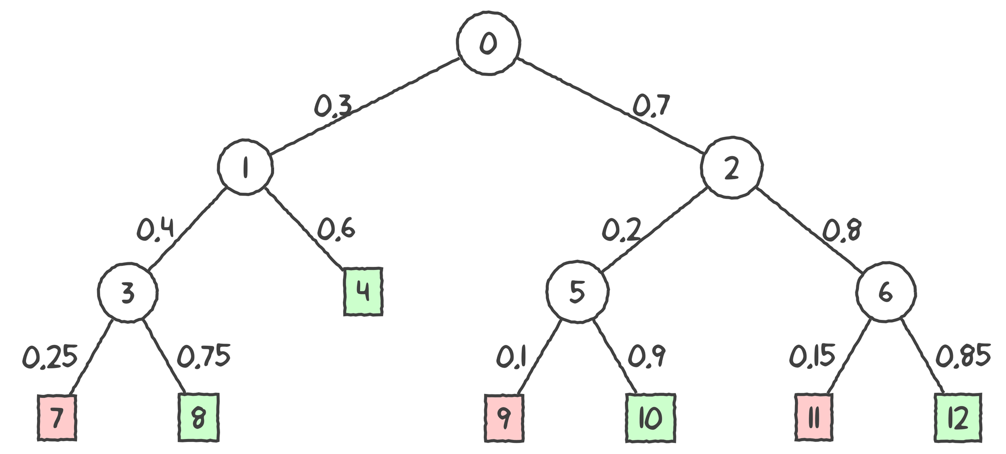
Challenge: Runtime of Decision Tree depends on structure of tree

Probabilistic analysis based on Bernoulli Experiments

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 number of comparisons $\mathbb{E}[L] = \sum_\pi p(\pi) \cdot |\pi|$

But: Runtime also depends on the implementation

Example Tree



NativeTree

```
bool predict(short const * x){
    unsigned int i = 0;
    while(!tree[i].isLeaf) {
        if (x[tree[i].f] <= tree[i].split) {
            i = tree[i].left;
        } else {
            i = tree[i].right;
        }
    }
    return tree[i].prediction;
}
```

Clock cycles:

$$c = 9 \cdot E[L] + 3$$

+ Small code size
+ Hot-code for I-Cache
- Indirect memory access
- Depends on D-Cache

If-Else-Tree

```
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;
        }
    }
}
```

Clock cycles:

$$c = 4 \cdot E[L] + 1$$

+ No indirect memory access
+ D-Cache not used
- I-Cache usually small
- Binary becomes large

VectTree

```
bool predict(short const * x){
    unsigned int i = 0;
    unsigned int mask;
    void * tmp;
    while(!tree[i].isLeaf) {
        load_vectorized(tree[i],tmp);
        mask = compare_vectorized(tmp, x);
        i = mask_to_index(mask);
    }
    return tree[i].prediction;
}
```

Clock cycles:

$$c = 10 \cdot \frac{\mathbb{E}[L]}{min(v, E_{SIMD})}$$

+ Small code size
+ Hot-code for I-Cache
+ Less indirect memory access
- Not always available

Realization and Results

In theory: If-Else seems to be the fastest
In practice: I-Cache is small and thus Native-Tree might be faster
Solution: Implement a code-generator which generates architectural and tree specific code. Include optimizations in future work.
Result: A small microcontroller with less than 16MHz is already enough to filter 12% of data without losing important events
Bonus: Different backends can be used to generate code for FPGAs

Funding

Part of this work has been supported by DFG within the Collaborative Research Center SFB 876, project A1