Three DIKU Open-Source Machine Learning Tools

Fabian Gieseke, Oswin Krause, Christian Igel
Department of Computer Science
In this episode . . .

Highly efficient C++ machine learning library

Ultrafast exact nearest neighbor computation using GPUs – processing millions of data points in seconds

Building random forest with $10^8$ data points in less than ten minutes on desktop computers
Shark

• Object-oriented software library for machine learning and optimization
• Methods for supervised and unsupervised learning and a wide range of standard methods for classification and regression.
• Toolbox for direct and gradient-based single- and multi-objective optimization
• New version 3.0: Almost complete rewrite
• New linear algebra engine Remora

## Exemplary methods

<table>
<thead>
<tr>
<th>Category</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel methods</td>
<td>support vector machines, kernel regression, kernel selection algorithms</td>
</tr>
<tr>
<td>neural networks</td>
<td>feed-forward neural networks, auto-encoders, restricted Boltzmann machines, dropout training</td>
</tr>
<tr>
<td>standard techniques</td>
<td>K-means clustering, LASSO regression, classification and regression trees, random forests</td>
</tr>
<tr>
<td>optimization</td>
<td>L-BFGS, CG, trust-region Newton, steepest descent, CMA-ES, MO-CMA-ES</td>
</tr>
</tbody>
</table>
Code snippets

```cpp
ClassificationDataset trainingSet;
importSparseData(trainingSet, "dataset");

FFNet<RectifierNeuron, LinearNeuron> model;
CrossEntropy loss;
ErrorFunction objFunc(trainingSet, &model, &loss);

GradientDescent optimizer;
optimizer.init(objFunc);
for(std::size_t i = 0; i != 1000; ++i)
    optimizer.step(objFunc);

GaussianRbfKernel<> kernel(gamma);
KernelClassifier<RealVector> svm;
CSvmTrainer<RealVector> trainer(&kernel, C);
trainer.trainingSet(svm, trainingSet);
```
Developed in DABAI: UP-MO-CMA-ES

Multi-objective optimization with unbounded solution sets using UP-MO-CMA-ES:
- Derivative-free optimization algorithm
- Pareto front approximated by multivariate local search distributions
- All non-dominated points are kept
- Selection based on hypervolume
- Promising competition results
  - 1st BBComp 3-objectives
  - 2nd BBOB and BBComp 2-objectives

Krause et al. Unbounded population MO-CMA-ES for the bi-objective BBOB test suite. GECCO BBOB, ACM, 2016
True multi-class SVMs

- There is no canonical way to extend SVMs to multiple classes
- One-vs-all often works, but has conceptual problems
- We developed
  - a theoretical framework
  - efficient solvers in Shark
for true “all-in-one” multi-class SVMs

All-in-one multi-class SVMs

There are many different all-in-one SVMs in Shark, e.g.:

- The equivalent multi-class SVMs by Weston & Watkins (WW), Vapnik, and Bredensteiner & Bennett
  

- Crammer and Singer’s popular variant (CS)
  

- Lee, Lin and Wahba’s consistent multi-class SVM (LLW)
  

- Multi-class maximum margin regression (MMR)
  

- ...
How fast is the Shark? SVMs

Left: Gaussian kernel SVM, cod-rna data, $\ell = 59535$, $d = 8$, $\varepsilon = 0.001$, $C = 0.01, \ldots, 1$, $\gamma = 1$, kernel cache 256MB

Right: linear SVM, rcv1.binary data, $\ell = 20,242$, $d = 47236$, $\varepsilon = 0.001$, $C = 1, \ldots, 100000$

How fast is the Shark? Other examples . . .

Random forest on cod-rna data, linear/ridge regression on BlogFeedback ($\ell = 60,021, d = 281$)
Nearest neighbor queries

- Determining nearest neighbors is a fundamental task in machine learning, e.g., for regression, classification, outlier detection, density estimation,

- Applying (exact) nearest neighbor queries to huge data sets is a challenge

Idea: Combine \( k \)-d trees and massively-parallel programming
K-d trees

Big Speed-Up!

But can still be too slow . . .
Buffer K-d trees

Buffer $k$-d Trees (Sketch)

1. **Top tree**: First levels of a standard $k$-d tree, pointer-less memory layout

2. **Leaf structure**: Training patterns, sorted *in-place* during top tree construction

3. **Buffers**: One buffer for each leaf of the top tree storing query indices

4. **Queues** $\text{input & reinsert}$: FIFO queues
Buffer K-d trees speed

Evaluation on real-world astronomy task:

<table>
<thead>
<tr>
<th>$d$</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>12</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>57</td>
<td>527</td>
<td>4616</td>
<td>16394</td>
<td>–</td>
</tr>
<tr>
<td>GPU</td>
<td>12</td>
<td>36</td>
<td>210</td>
<td>478</td>
<td>1717</td>
</tr>
<tr>
<td></td>
<td>×5</td>
<td>×15</td>
<td>×22</td>
<td>×34</td>
<td>–</td>
</tr>
</tbody>
</table>

CPU: $k$-d tree
GPU: buffer $k$-d tree

Time in s, Intel i7@3.40GHz (4 cores), GeForce GTX 770 (1536 cores, 4GB RAM), $2 \cdot 10^6$ training and $10^7$ test patterns


Woody: Large-scale random forests

- Random forests are among the most powerful machine learning techniques in practice – how do we apply them to millions of data points without expensive compute resources?
- We would like to grow full trees to tackle class imbalance.
- **Solution:** Built top-tree(s) that lead to more balanced tree(s)!
$d = 18$, full trees, checking all features per split, 4 trees, Intel(R) Xeon(R) CPU E3-1220 v3 @ 3.10GHz (4 cores), 32GB RAM
Similar classification accuracies; H2/XGBoost optimized for shallow trees (max_depth=1000/max_depth=100)
$d = 28$, full trees, checking all features per split, 4 trees, Intel(R) Xeon(R) CPU E3-1220 v3 @ 3.10GHz (4 cores), 32GB RAM

Similar classification accuracies; H2/XGBoost optimized for shallow trees (max_depth=1000/max_depth=100)
### Big trees on small machines

#### Training Output

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:51:20,588</td>
<td>Number of training patterns: 113212922</td>
<td></td>
</tr>
<tr>
<td>13:51:20,588</td>
<td>Dimensionality of the data: 11</td>
<td></td>
</tr>
<tr>
<td>13:51:20,588</td>
<td>Fitting forest ...</td>
<td></td>
</tr>
<tr>
<td>13:51:20,596</td>
<td>Setting n_top to 532007.</td>
<td></td>
</tr>
<tr>
<td>13:51:20,596</td>
<td>Setting n_patterns_leaf to 106401.</td>
<td></td>
</tr>
<tr>
<td>13:51:20,596</td>
<td>(I) Retrieving random subsets for top trees ...</td>
<td></td>
</tr>
<tr>
<td>13:51:20,596</td>
<td>Retrieving random subsets for all estimators...</td>
<td></td>
</tr>
<tr>
<td>13:52:10,882</td>
<td>Storing subsets for all estimators ...</td>
<td></td>
</tr>
<tr>
<td>13:52:10,977</td>
<td>(II) Fitting all top trees ...</td>
<td></td>
</tr>
<tr>
<td>13:52:10,977</td>
<td>Fitting top tree for estimator 0 ...</td>
<td></td>
</tr>
<tr>
<td>13:52:13,629</td>
<td>Saving top tree for estimator 0 ...</td>
<td></td>
</tr>
<tr>
<td>13:52:13,629</td>
<td>(III) Distributing all patterns to leaves ...</td>
<td></td>
</tr>
<tr>
<td>13:59:52,777</td>
<td>[2054/2054] Fitting bottom subforest 4018 for 56551 patterns ...</td>
<td></td>
</tr>
<tr>
<td>13:59:52,908</td>
<td>Fitting Statistics</td>
<td></td>
</tr>
<tr>
<td>13:59:52,908</td>
<td>(I) Retrieving subsets:</td>
<td>53.034 (s)</td>
</tr>
<tr>
<td>13:59:52,909</td>
<td>(II) Top tree constructions:</td>
<td>53.034 (s)</td>
</tr>
<tr>
<td>13:59:52,909</td>
<td>(III) Distributing to top tree leaves:</td>
<td>240.208 (s)</td>
</tr>
<tr>
<td>13:59:52,909</td>
<td>(IV) Bottom trees constructions:</td>
<td>219.071 (s)</td>
</tr>
<tr>
<td>13:59:52,909</td>
<td></td>
<td>512.312 (s)</td>
</tr>
</tbody>
</table>
Shark

image.diku.dk/shark


Gold Prize at Open Source Software World Challenge 2011
**Buffer k-d trees**

http://bufferkdtree.readthedocs.org


# Woody (Python package)

## Repository

<table>
<thead>
<tr>
<th>Name</th>
<th>Last commit</th>
<th>Last Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>gendata</td>
<td>4 weeks ago</td>
</tr>
<tr>
<td>examples</td>
<td>update</td>
<td>4 weeks ago</td>
</tr>
<tr>
<td>experiments</td>
<td>update exps</td>
<td>about 6 hours ago</td>
</tr>
<tr>
<td>literature</td>
<td>update</td>
<td>2 days ago</td>
</tr>
<tr>
<td>papers</td>
<td>update</td>
<td>2 days ago</td>
</tr>
<tr>
<td>woody</td>
<td>update</td>
<td>about 16 hours ago</td>
</tr>
<tr>
<td>.gitignore</td>
<td>data update</td>
<td></td>
</tr>
<tr>
<td>CONTRIBUTING.md</td>
<td>initial stuff</td>
<td></td>
</tr>
<tr>
<td>LICENSE</td>
<td>initial stuff</td>
<td></td>
</tr>
<tr>
<td>MANIFEST.in</td>
<td>update</td>
<td></td>
</tr>
<tr>
<td>README.md</td>
<td>readme</td>
<td></td>
</tr>
<tr>
<td>README.rst</td>
<td>readme</td>
<td></td>
</tr>
</tbody>
</table>
Take me home

- We are further developing the Shark library – have a look if you need highly efficient machine learning on commodity hardware!

Recent highlights:
- Highly efficient linear algebra
- True multi-class SVMs
- New multi-objective algorithms

- Try buffer $k$-d trees for large-scale nearest neighbor queries using GPUs!

- Stay tuned: Woody, random forests (w/ large trees) with millions of training data points on your desktop!

- There is great, professional open source machine learning software maintained next door!