A Dynamic Adaptation of AD-trees for Efficient Machine Learning on Large Data Sets

Paul Komarek and Andrew Moore
komarek@cmu.edu
awm@cs.cmu.edu

Auton Lab, Carnegie Mellon University
http://www.autonlab.org
Daily Pittsburgh Weather Data

- Three attributes: “Temperature”, “Humidity” and “Storming”.

- Each attribute has only two discrete values in this example. The humidity, for instance, is either high (100%), or low (99%).

- Each horizontal row of the data set represents one day’s weather.

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Humidity</th>
<th>Storming</th>
</tr>
</thead>
<tbody>
<tr>
<td>105 F</td>
<td>100%</td>
<td>yes</td>
</tr>
<tr>
<td>105 F</td>
<td>99%</td>
<td>no</td>
</tr>
<tr>
<td>5 F</td>
<td>99%</td>
<td>no</td>
</tr>
<tr>
<td>5 F</td>
<td>99%</td>
<td>yes</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
</tbody>
</table>
Datacube representation of data set

Storming?
- Yes
- No

Humidity
- 100%
- 99%

Temperature
- 5F
- 105F
Datacube with marginalizations

- Storming?
- Humidity: 100%, 99%
- Temperature: 5F, 105F

Either Yes, No

Datacube with marginalizations
Static AD–tree
Dynamic AD–tree
How Dynamic AD-trees Are Grown

Query 1: How many rows have Humidity = 100%?

Query 2: How many rows have Humidity = 100% and Storming = Yes?
How Dynamic AD-trees Are Grown (slide 2)

Query 3: How many rows have Humidity = 99% and Storming = Yes?
Additional Dynamic AD-tree Benefit

- Amortized updates of AD-nodes to accommodate row additions to the data set.

- Accomplished with a tiny bit of extra state in each AD-node.

- Can be done with Static AD-trees, but more hassle.
Tests with the e29. fds Data Set

The e29. fds data set is a relatively small dataset. It comes from the Sloan Digital Sky Survey, and the attributes represent information about galaxies.

- 29 attributes
- Each attribute has low arity, i.e. between 2 and 6 possible values.
- 3079792 rows
Rule Learner:

- chooses a set of attributes for the left-hand-side of the rule
- each new set of attributes is an incremental change from previous sets considered
- for various assignments of values to these attributes, it evaluates the accuracy of the resulting rule

As a result, successive queries are very similar to one another at all times. Therefore we expect the amortized performance of Dynamic AD-trees to be similar to that of Static AD-trees, if enough memory is allocated to the row caches and skinny AD-nodes. “One-off” performance of Dynamic AD-trees should of course be superior given enough memory.
Rule Learner, unlimited memory, max time 1200s, on e29.fds

- Empty bullets are non-amortized times
- Filled bullets are amortized times

(numatts, the max number of attributes in LHS of the rule)

23 MB
Rule Learner, unlimited memory, max time 1200s, on e29.fds

- Empty bullets are non-amortized times.
- Filled bullets are amortized times.

- (static, single) 55 MB
- (non-AD-tree, single) 55 MB, 23 MB
- (static, multi) 55 MB

numatts, the max number of attributes in LHS of the rule

time in seconds
Rule Learner, unlimited memory, max time 1200s, on e29.fds

- Empty bullets are non-amortized times
- Filled bullets are amortized times

- 55 MB (static, single)
- 137 MB (dynamic, single)
- 137 MB (dynamic, multi)
- 55 MB (static, multi)
- 23 MB (non-AD-tree, single)
- 84 MB

numatts, the max number of attributes in LHS of the rule

time in seconds
Limited Memory Model gives 60% / 10% / 30% split to row caches, skinny AD-nodes, and the AD-tree itself:

- 32 MB model implies 19 MB / 3 MB / 10 MB split
- 64 MB model implies 38 MB / 6 MB / 20 MB split
- 128 MB model implies 77 MB / 13 MB / 38 MB split
Rule Learner, limited memory, multi, on e29.fds

- 32 MB (dynamic, multi, 64)
- 64 MB (dynamic, multi, 128)
- 128 MB (dynamic, multi, 32)
- 134 MB (dynamic, multi, unltd)
- 55 MB (static, multi)

All runs are multi
Partial bullets for ltd mem
Filled bullets for unltd mem

numatts, the max number of attributes in LHS of the rule

time in seconds
Rule Learner, limited memory, single, max time 1200s, on e29.fds

- (dynamic, single, 32)
- (dynamic, single, 64)
- (dynamic, single, 128)

- 55 MB (static, single)
- 137 MB (dynamic, single, unltd)
- 84 MB

All runs are single
Partial bullets for ltd mem
Empty bullets for unltd mem
Rule Learner, growing e29.fds in 500,000-row chunks

- (static, single, grow)
- (dynamic, single, grow)
- (dynamic, multi, grow)
The decision tree learner:

- given an existing node in the decision tree, it creates a child
- it chooses the “best” attribute to associate with the child
- to score potential attributes, queries are made which are identical to previously made queries, but with the addition of the attribute being scored

Again, successive queries are similar to one another. However, when changing which decision tree node is creating a child, a “jump” is made which reduces the applicability of the current row caches and skinny AD-nodes.
Decision Tree Learner, multi runs, on e29.fds

All runs are multi
Empty bullets for ID3 run
Partial bullets for limited memory
Filled bullets for unlimited memory

(time in seconds)

size, the max number of nodes allowed in the decision tree
Decision Tree Learner, single runs, on e29.fds

All runs are single
Empty bullets for ID3 and (static, single)
Partial bullets for limited memory
Filled bullets for unlimited memory

(time in seconds)

(size, the max number of nodes allowed in the decision tree)
The Bayes net structure finder uses a stochastic hill-climbing algorithm to find a Bayes net structure which approximately “explains” the data set. This stochastic element induces a query pattern which is seldom localized for more than a few successive queries.
Bayes Net Structure Finder, unlimited memory, on e29.fds

Empty bullets are non-amortized times
Filled bullets are amortized times

- 55 MB (static, single)
- 715 MB (dynamic, single)
- 23 MB (non-AD-tree, single)
- 55 MB (static, multi)
- 707 MB (dynamic, multi)
- 100 MB for (dynamic, single, iters=50)
Bayes Net Structure Finder, limited memory, on e29.fds

- (dynamic, multi, 32)
- (dynamic, multi, 64)
- (dynamic, multi, 128)
- 55 MB (static, multi)
- 707 MB (dynamic, multi, unlt)

The graph shows the time in seconds on the y-axis and the number of iterations for the stochastic optimizer on the x-axis. The data points are spread across different memory configurations and iteration counts.
Summary of Static and Dynamic AD-trees

- AD-trees pre-cache counting queries in an efficient manner.

- The time to answer a query is independent of the number of data set rows, once the Static AD-tree is built. Static AD-trees thrive in amortized contexts.

- Dynamic AD-trees only grow the AD-tree where needed, when needed. Row caches and skinny AD-nodes are used to maintain performance, and hence are sensitive to query patterns.

- Dynamic AD-trees are flexible and efficient in both amortized and “one-off” applications. They can serve as generic, high-performance replacements for more specialized structures.
Future Work

- Better management of row cache and skinny AD-node priority queues.

- Dynamic reordering and addition of data set attributes. Simple with Dynamic AD-trees.

- Localized variation of parameters within a Dynamic AD-tree. For instance, adding or hiding some attributes to/from an AD-node. More interesting is dynamic attribute resolution.

- The above items would allow dynamic, variable discretization of real-valued attributes.

Paul Komarek and Andrew Moore
komarek@cmu.edu
awm@cs.cmu.edu

Auton Lab, Carnegie Mellon University http://www.autonlab.org