The Boundary Forest Algorithm

Jonathan Yedidia
Director of AI Research, Analog Devices
Director of the Algorithmic Systems Group, Analog Garage

With Charles Mathy, Nate Derbinsky, José Bento and Jonathan Rosenthal
Three types of learning problems:

- Classification: given a query, return a label
- Regression: given a query, return a number or a vector
- Retrieval: given a query, return a similar example

Two settings:

- Off-line: entire training set available before any queries
- On-line: training and querying are inter-mixed
Desirable Features for Learning Algorithms

• Low error on previously unseen queries (good generalization)
• Very fast processing of training examples and test queries
• On-line learning with no error on most recent training example
• Can easily achieve zero error on training set in offline setting
• Able to absorb and learn from unlimited training examples: query time and memory only grow slowly with more training
• Can learn and represent complex functions—no intrinsic limits in the kind of functions it can learn
• Easy to understand how and why it works
The Boundary Forest (BF)

- Satisfies all these properties with the caveat that you must provide a distance function between examples.
- Uses a collection of trees called “Boundary Trees,” whose nodes are previously seen points \( x \) with label or vector \( C(x) \).
- When queried with a point \( y \), each tree quickly outputs an approximate nearest neighbor to \( y \) from the training examples.
- Set of ANNs can be used for classification or regression (Shepard weighting) or retrieval (closest ones).
- Built online, one point at a time.
- Gives similar or better error rate compared to other Approximate Nearest Neighbor methods, while being online and fast to train and query.
Testing and training

- Nodes have associated class (color)
- Start at root, look for child closest to test point
- Go to that child, recurse until you find locally closest node
- For queries, class of locally closest node is the prediction
- For training, add node and edge if prediction is incorrect
Testing and training

- Nodes have associated class (color)
- Start at root, look for child closest to test point
- Go to that child, recurse until you find locally closest node
- For queries, class of locally closest node is the prediction
- For training, add node and edge if prediction is incorrect
Testing and training

• Nodes have associated class (color)
• Start at root, look for child closest to test point
• Go to that child, recurse until you find locally closest node
• For queries, class of locally closest node is the prediction
• For training, add node and edge if prediction is incorrect
Testing and training

- Nodes have associated class (color)
- Start at root, look for child closest to test point
- Go to that child, recurse until you find locally closest node
- For queries, class of locally closest node is the prediction
- For training, add node and edge if prediction is incorrect
Testing and training

- Nodes have associated class (color)
- Start at root, look for child closest to test point
- Go to that child, recurse until you find locally closest node
- For queries, class of locally closest node is the prediction
- For training, add node and edge if prediction is incorrect
Testing and training

- Nodes have associated class (color)
- Start at root, look for child closest to test point
- Go to that child, recurse until you find locally closest node
- For queries, class of locally closest node is the prediction
- For training, add node and edge if prediction is incorrect
Testing and training

- Nodes have associated class (color)
- Start at root, look for child closest to test point
- Go to that child, recurse until you find locally closest node
- For queries, class of locally closest node is the prediction
- For training, add node and edge if prediction is incorrect
Toy world

- Points come in one at a time with associated class (color)
Toy world

- Points come in one at a time with associated class (color)
- First point becomes root node
Toy world

- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added
Toy world

- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added

Boundary Tree                                   Querying BT everywhere
Toy world

• Points come in one at a time with associated class
• First point becomes root node
• Greedy traversal of tree, class of result is prediction
• If prediction is incorrect, training point and edge is added
Toy world

- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added
Toy world

- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added
Toy world

- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added
- After showing 10,000 points, 680 points stored
**Toy world**

- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added
- After showing 100,000 points, 2220 points stored
Toy world

- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added
- Forest of trees - differently seeded - better generalization

Shown 10,000 points
Other applications

- Regression

Ground truth

BT: 395 nodes after seeing 10,000 samples

- Retrieval: add all points
MNIST

- MNIST: 60,000 labeled handwritten digits for training, 10,000 for testing (784 pixels)
- K Nearest Neighbor with Euclidean L2 distance: 97.2% accuracy (3-NN)
- Forest of 50 Boundary Trees: 97.8%
- Training on full MNIST with 4 cores in 37 seconds in Java
- Testing on 10,000 samples in 9 seconds. 3-NN takes an hour.
- Better metric: HOG (Histogram of Gradients). Gets 98.9% accuracy. 3-NN: 98.6%
General Performance

• Gives good performance on wide variety of data sets (see our paper for comparison with a variety of other algorithms on a variety of benchmark problems).

• Querying or training a single example only takes log N time, where N is the amount of stored (!) data. (This requires a simple modification to what was presented so far: maximum number of children per node k)

• Has the ability to learn, in an online fashion, from a huge number of training examples.

• Responds to queries correctly immediately after training on them, and it’s easy to drive training error to zero.
References


Table 1: (a) Total training time and (b) total testing time, in seconds, for classification benchmarks, single core. In (b) the datasets are in the same order as in (a). BF has $n_T = 50$, $k = 50$. For $1 - NN$, $3 - NN$ and $RF$ we use the Weka(Hall et al. 2009) implementation. RF has 50 trees, 50 features per node. BF-4 is the same BF with 4 cores. Rkd has 4 kd trees and can visit at most 10% of previously seen examples, and points are added online. CT has $\epsilon = 10$ and $b = 1.3$, and uses online insertion. See Appendix for 10-NN, RF with 100 trees and $\sqrt{D}$ features per node (recommended in (Breiman 2001)), and CT with $b = 1.1$, $\epsilon = 0.1$. The datasets are from the LIBSVM repository(Chang and Lin 2008).

Table 2: Error rate for classification benchmarks. The values represent percentages. The offline BF algorithm does only marginally better on average than the BF. The random kd trees and cover tree are about the same accuracy as $1 - NN$. 
References


Summary

• The Boundary Forest is a new algorithm for online classification, regression and retrieval

• Can be thought of as fast online nearest neighbor algorithm

• Fast both at training and testing time

• Memory and computation time grow slowly, so can deal with huge numbers of training examples

• Easy to implement and performs better than other approximate nearest neighbor online algorithms

• Biggest issue is need for a good metric on examples.