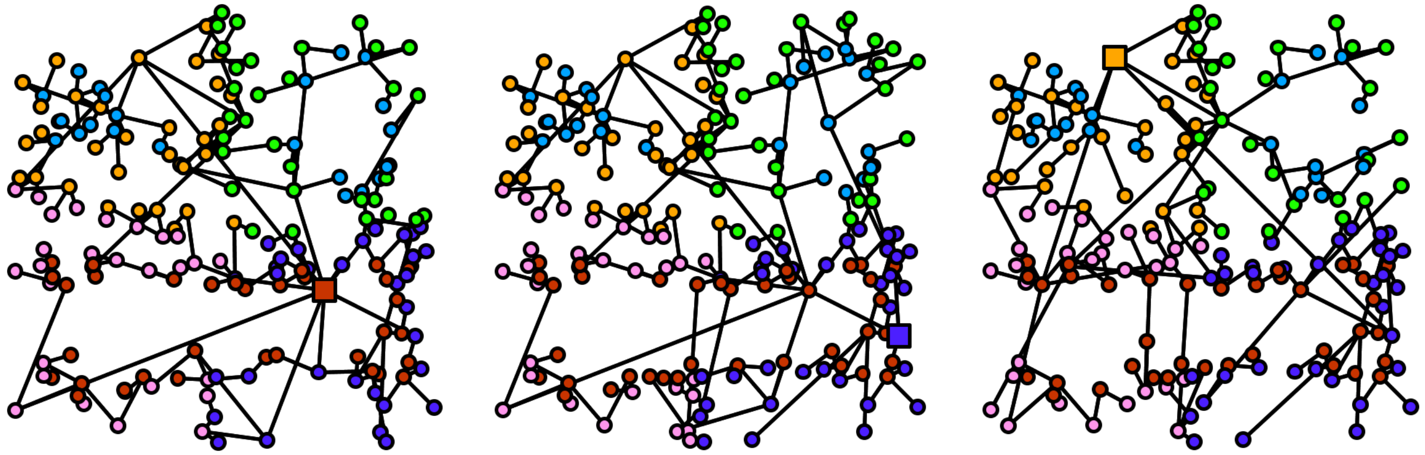
The Boundary Forest Algorithm



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

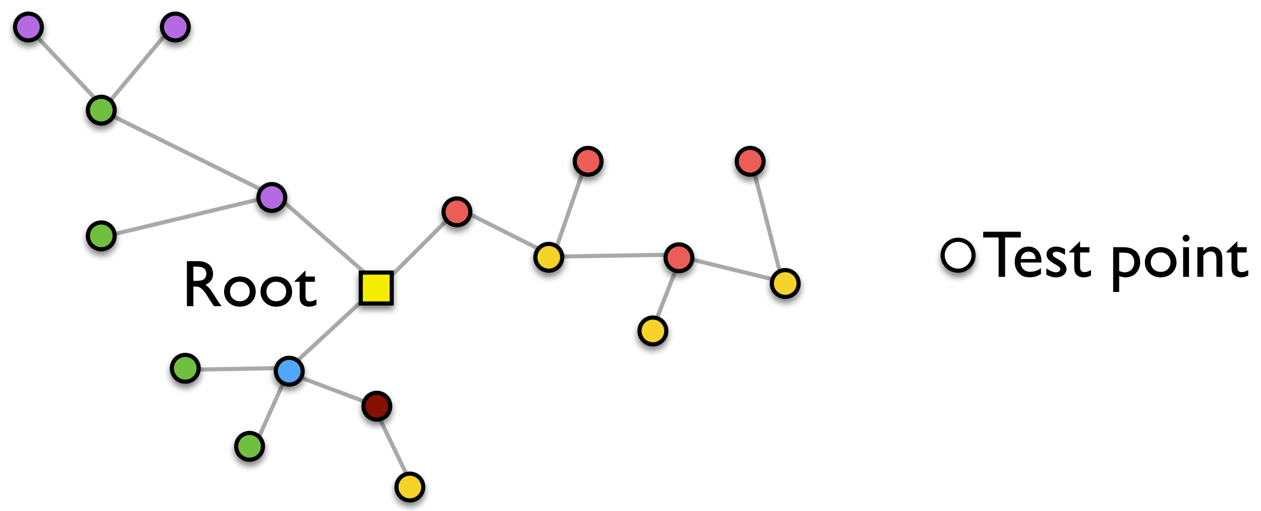
- Low error on previously unseen queries (good generalization)
- Very fast processing of training examples and test queries
- On-line learning with no error on most recent training example
- Can easily achieve zero error on training set in offline setting
- Able to absorb and learn from *unlimited* training 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

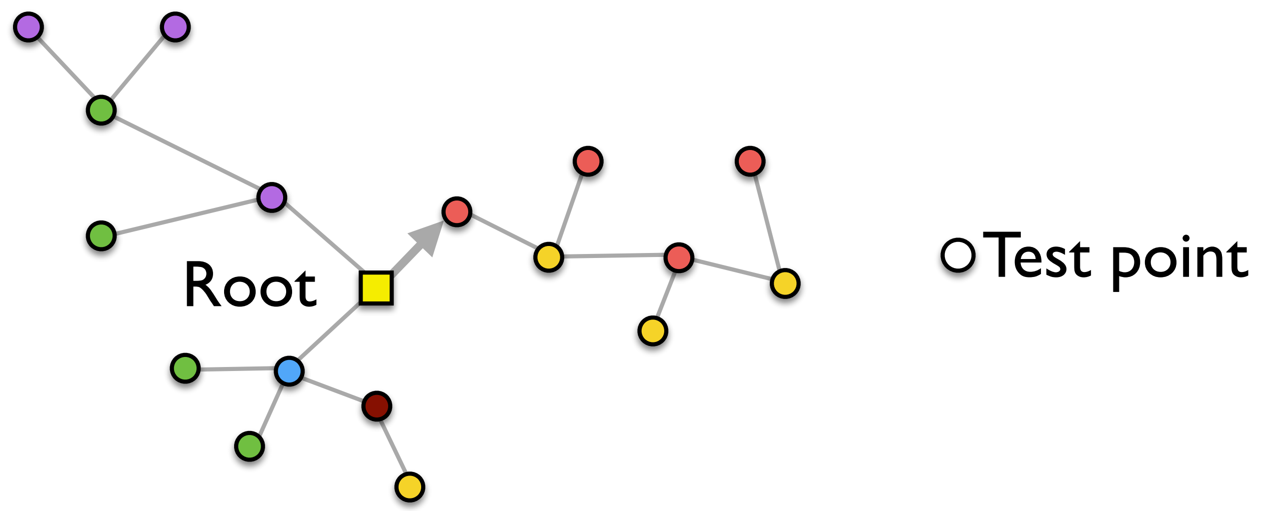
Testing and training

- 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



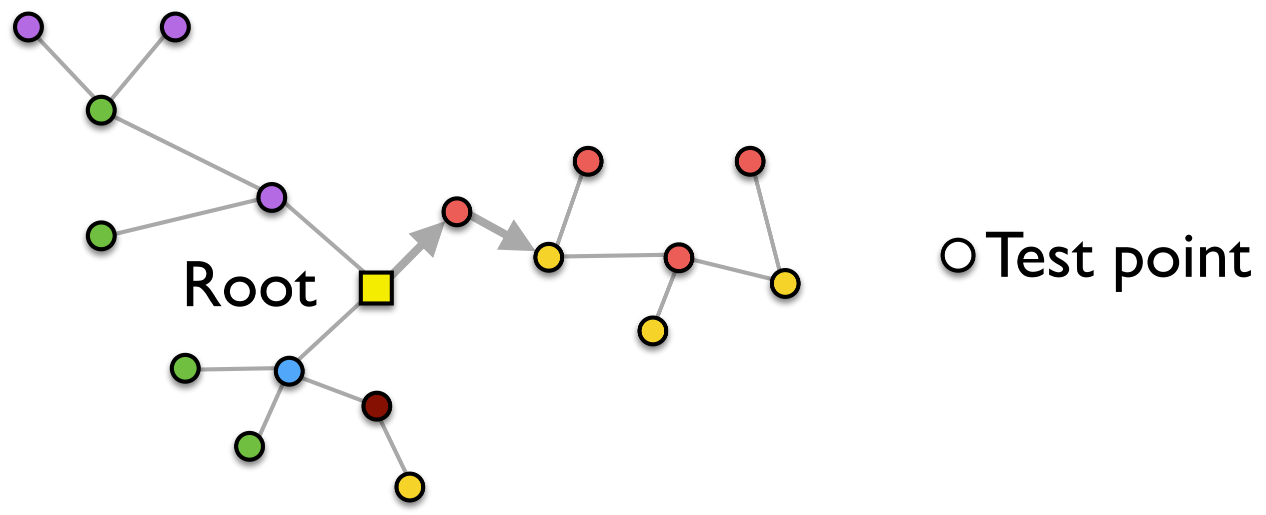
Testing and training

- 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



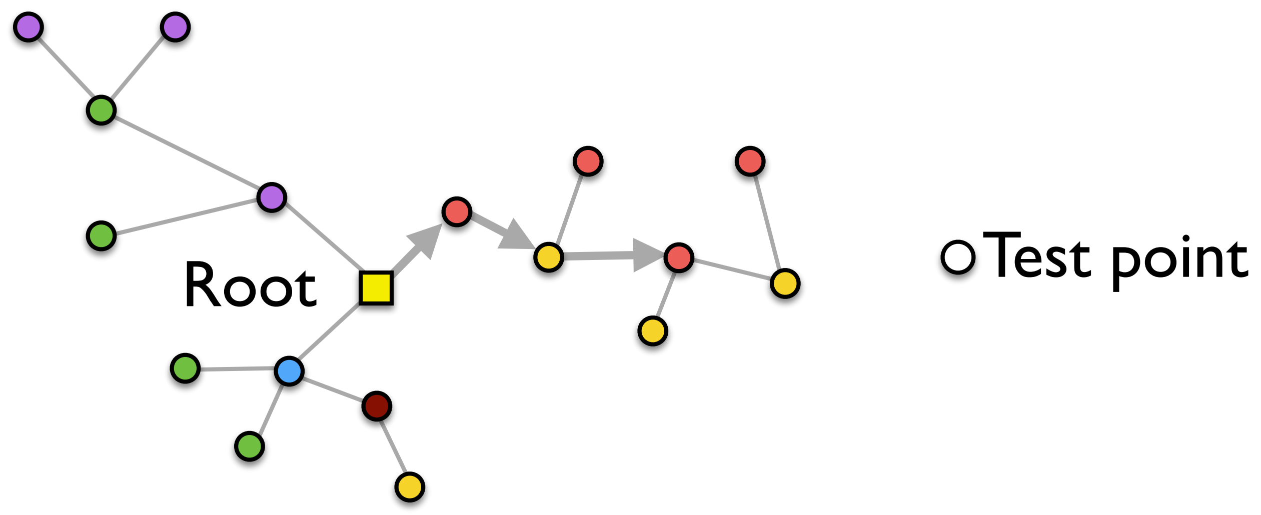
Testing and training

- 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



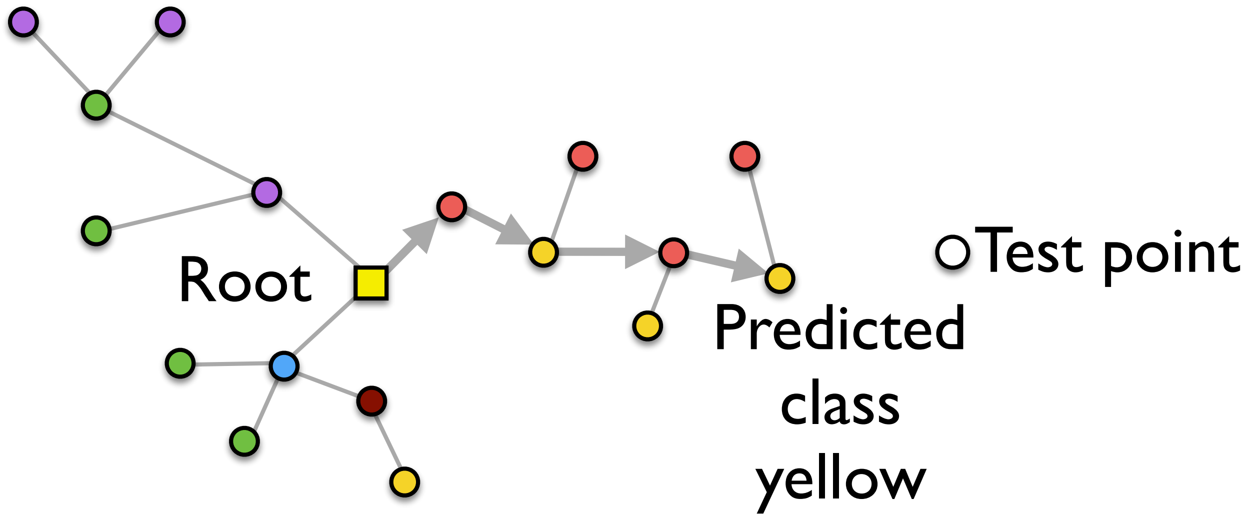
Testing and training

- 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



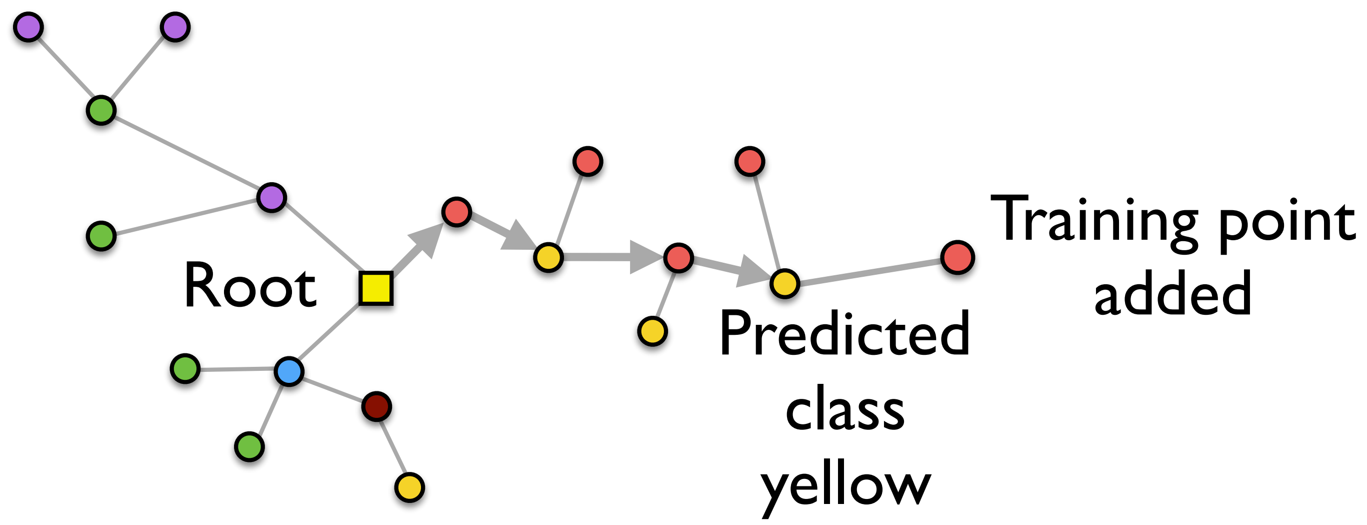
Testing and training

- 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



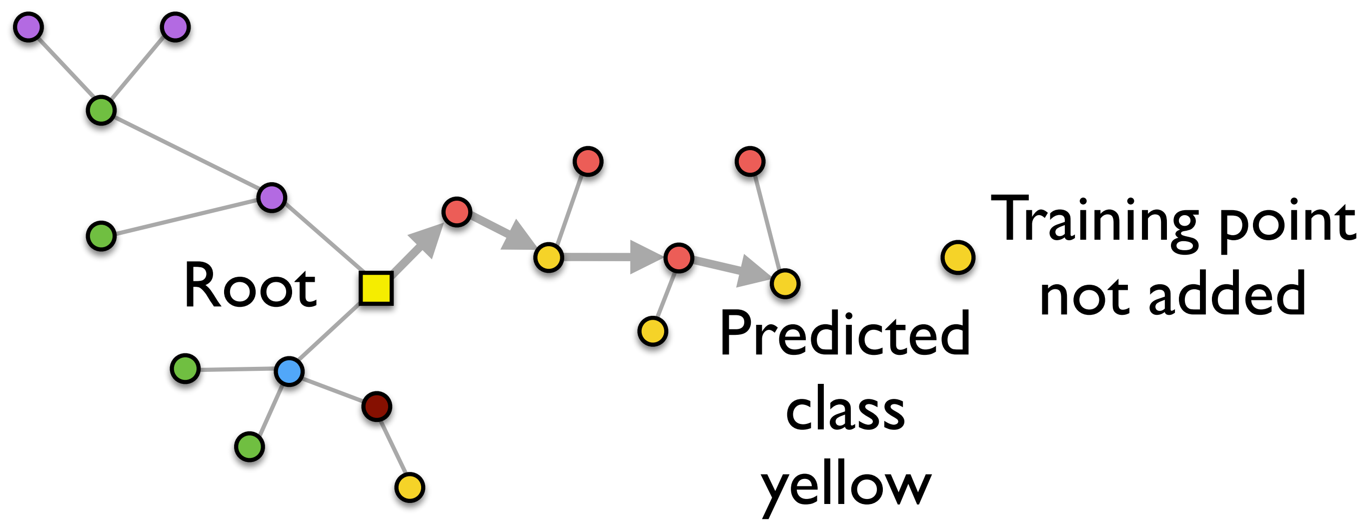
Testing and training

- 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



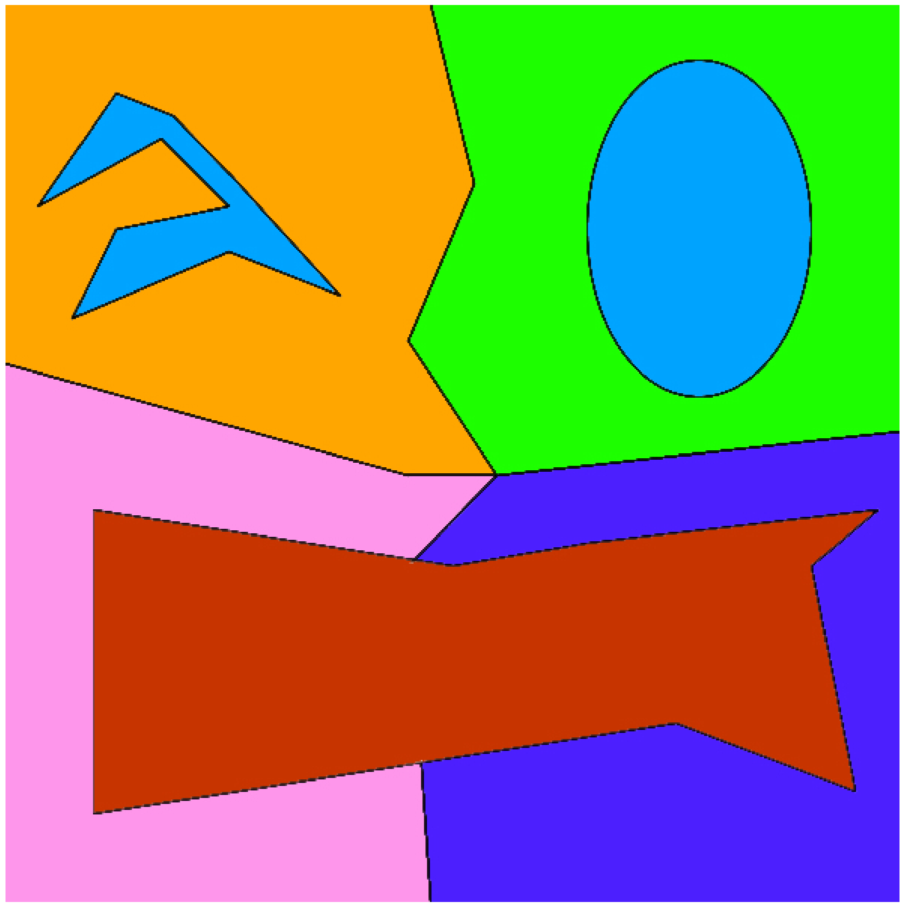
Testing and training

- 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



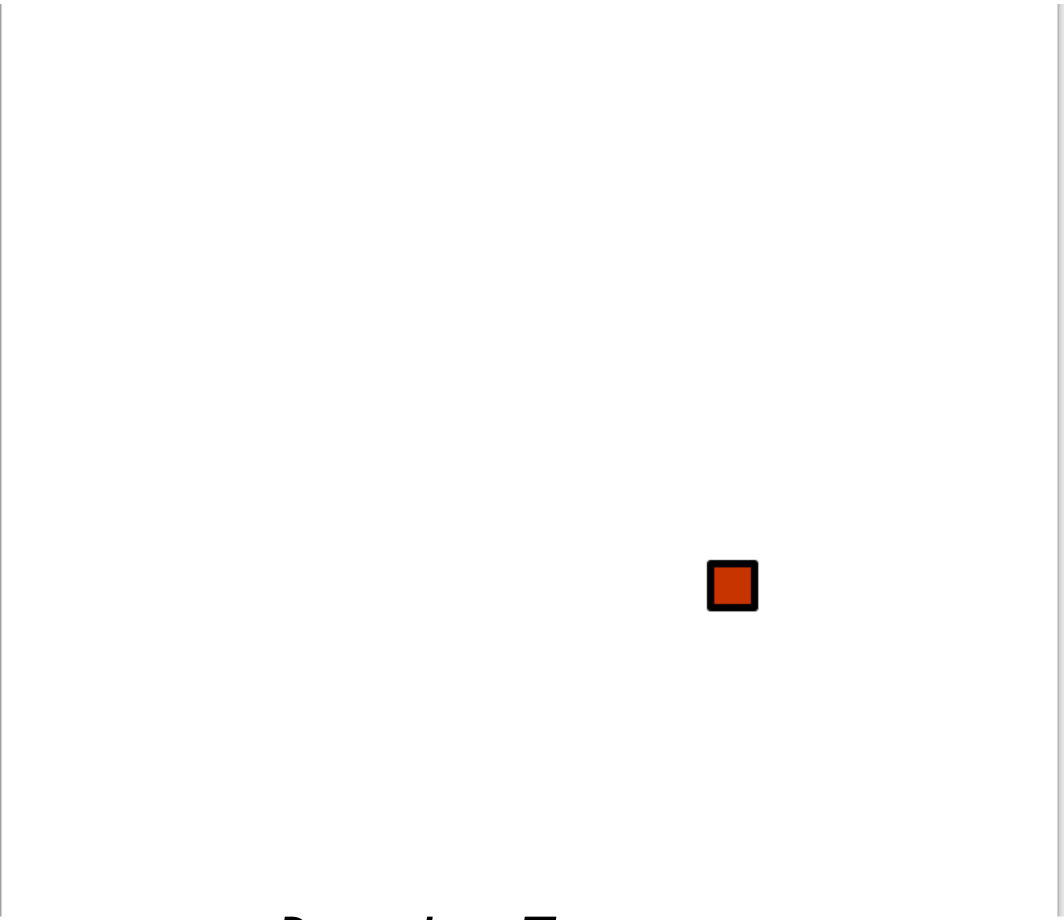
Toy world

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

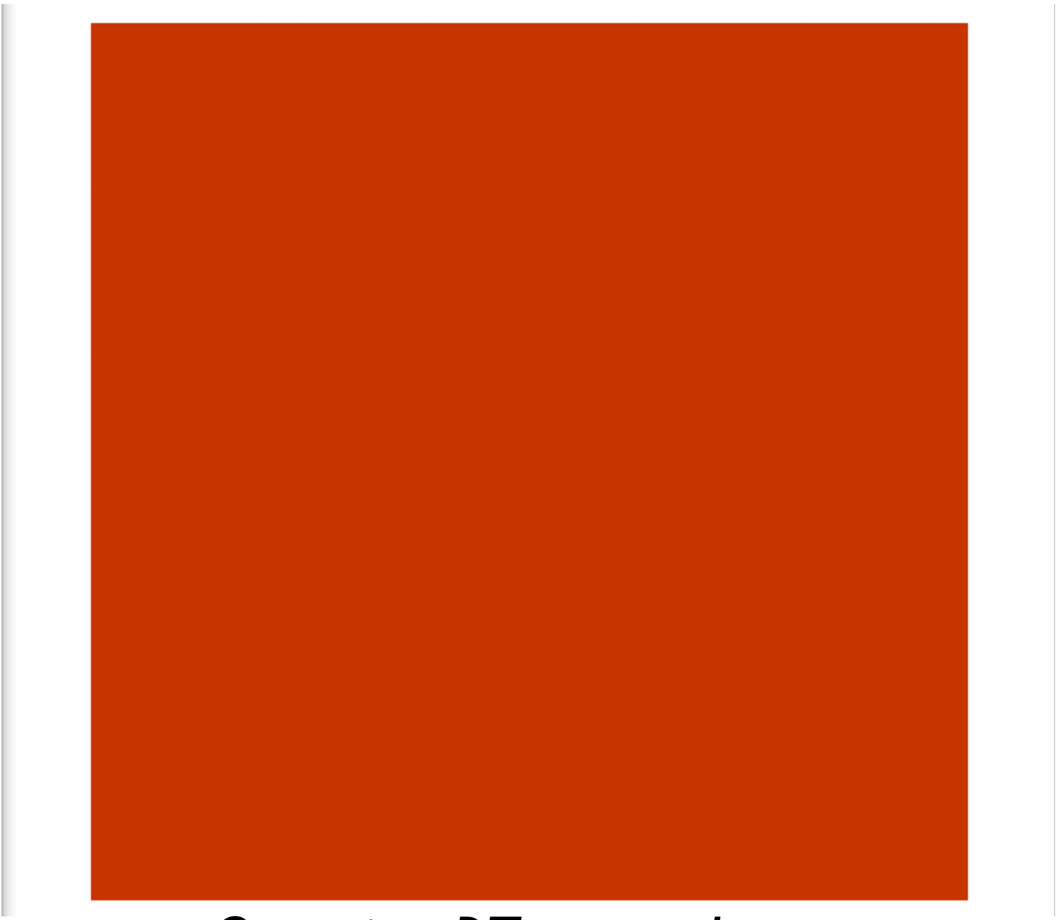


Toy world

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



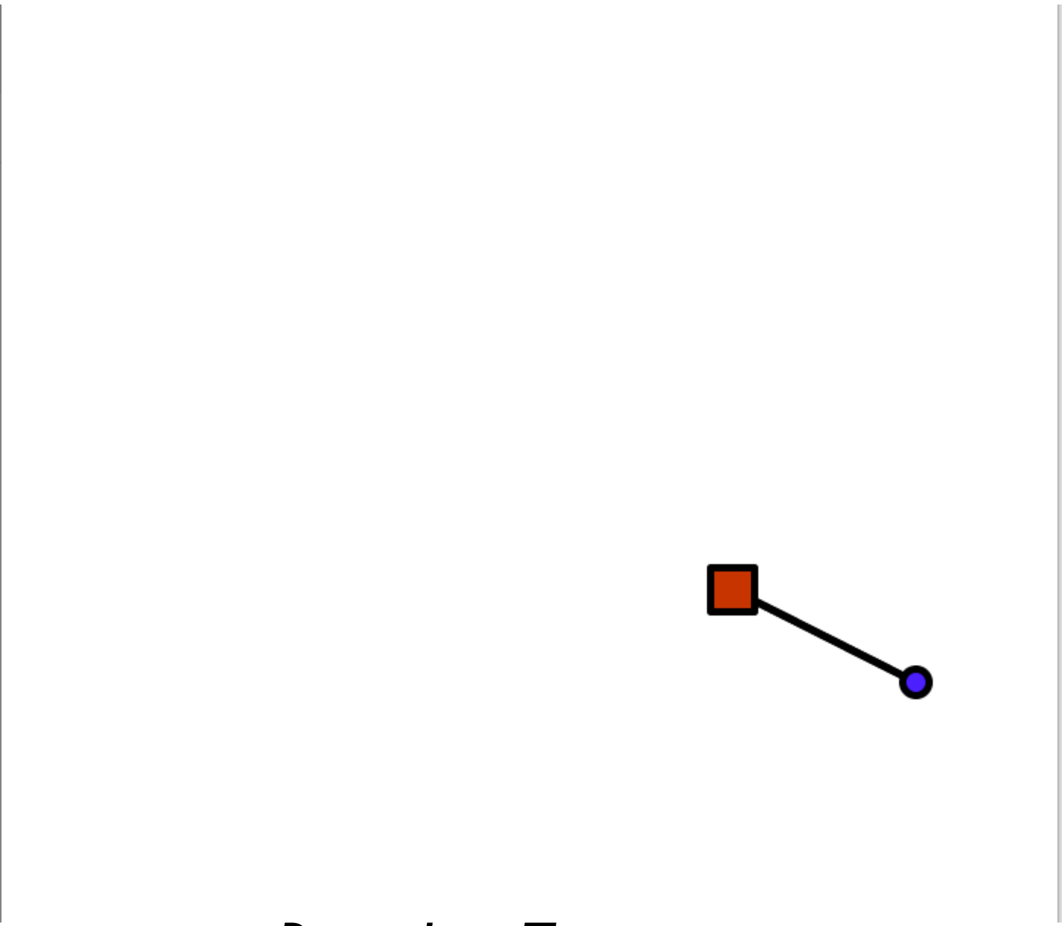
Boundary Tree

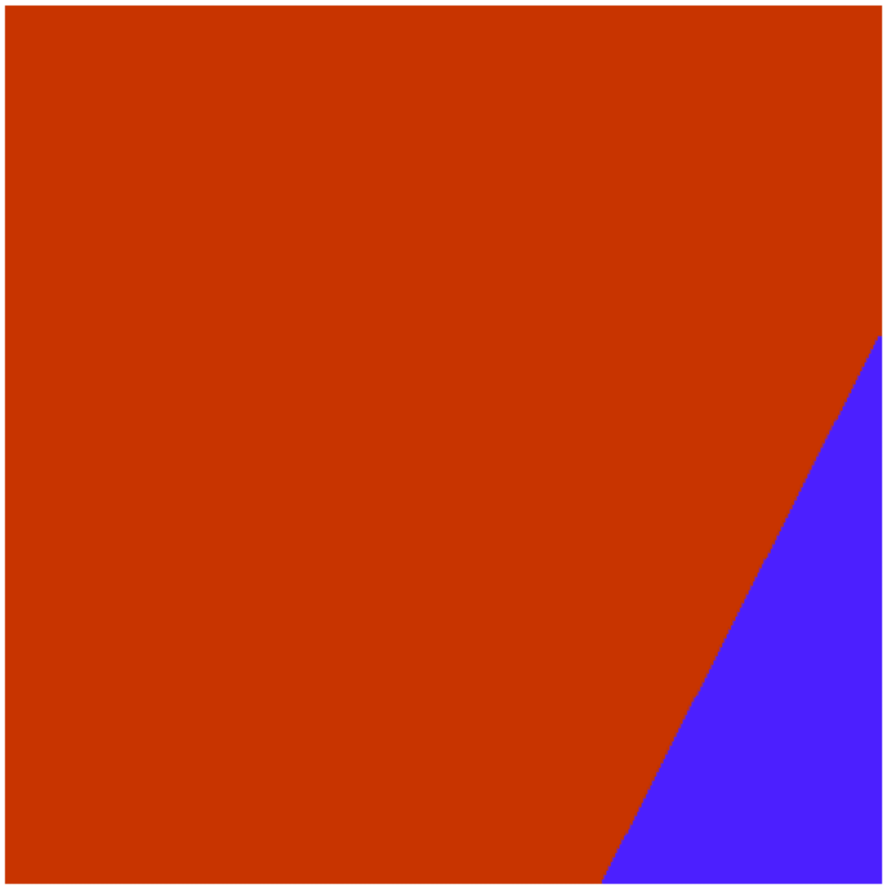


Querying BT everywhere

Toy world

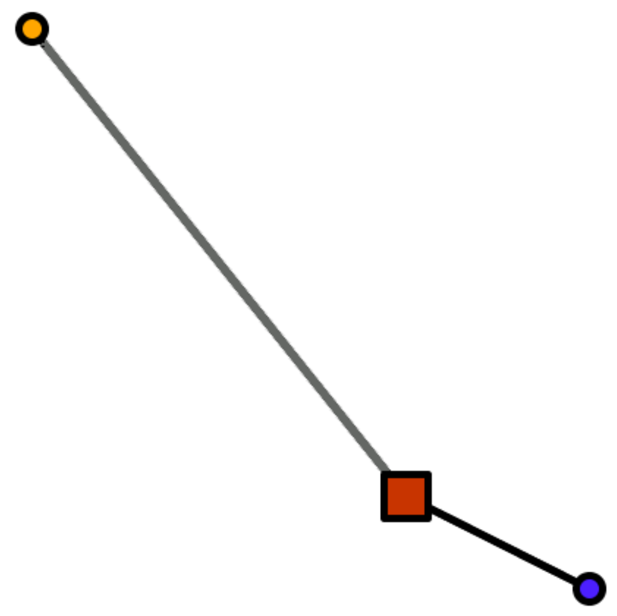
- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added

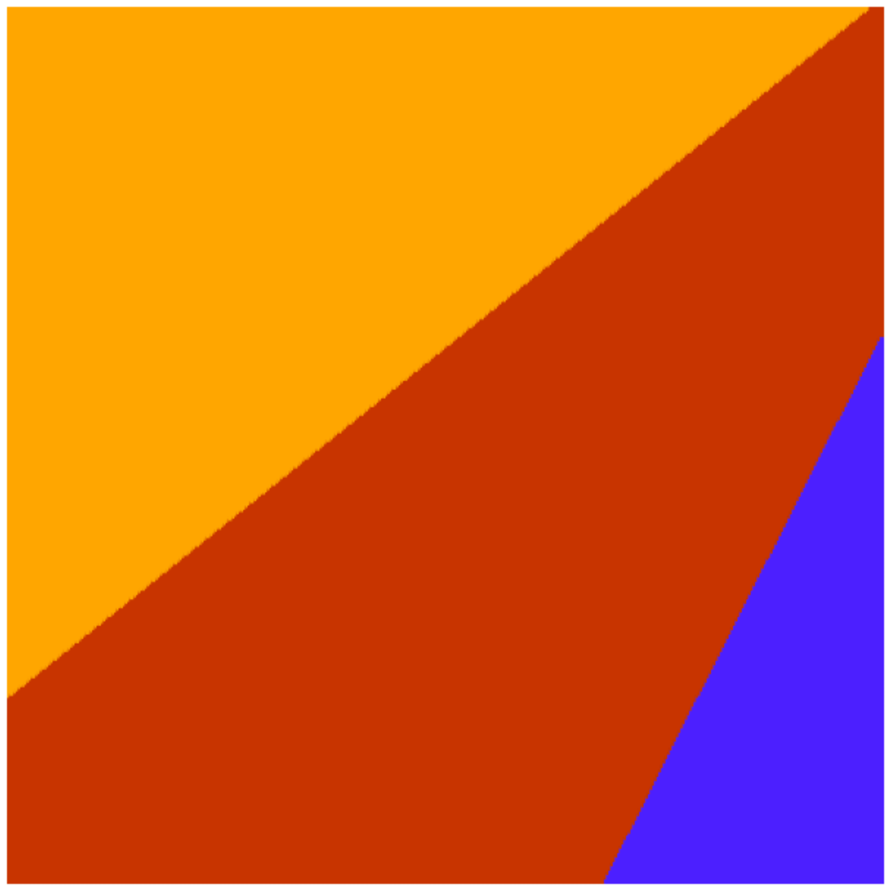

Boundary Tree


Querying BT everywhere

Toy world

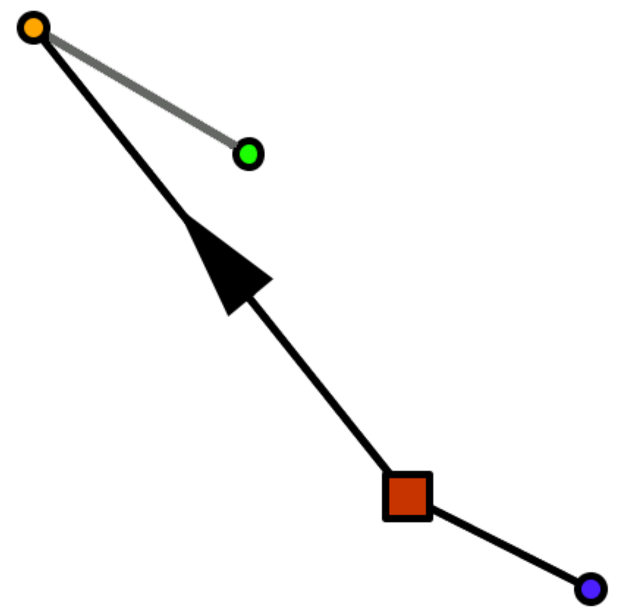
- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added


Boundary Tree

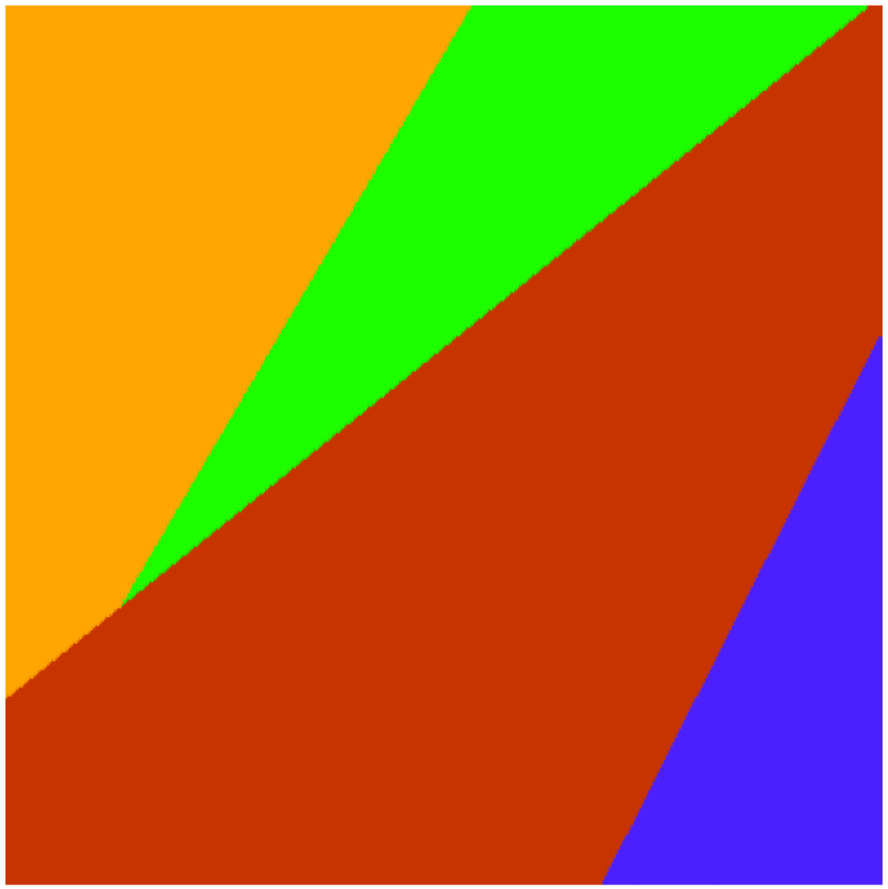

Querying BT everywhere

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



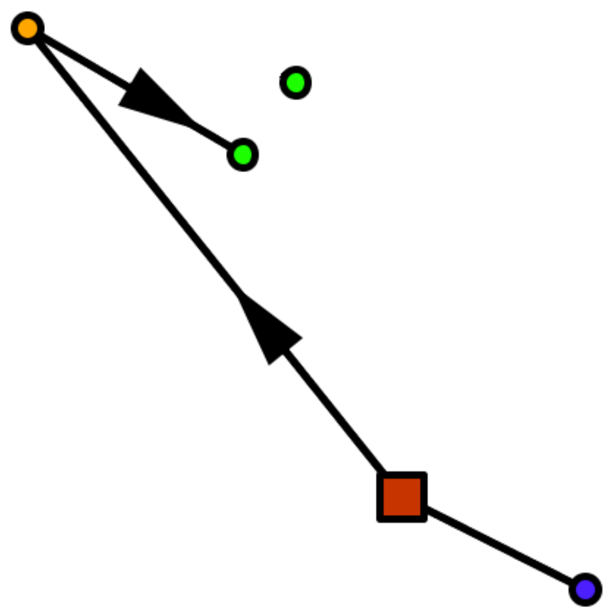
Boundary Tree

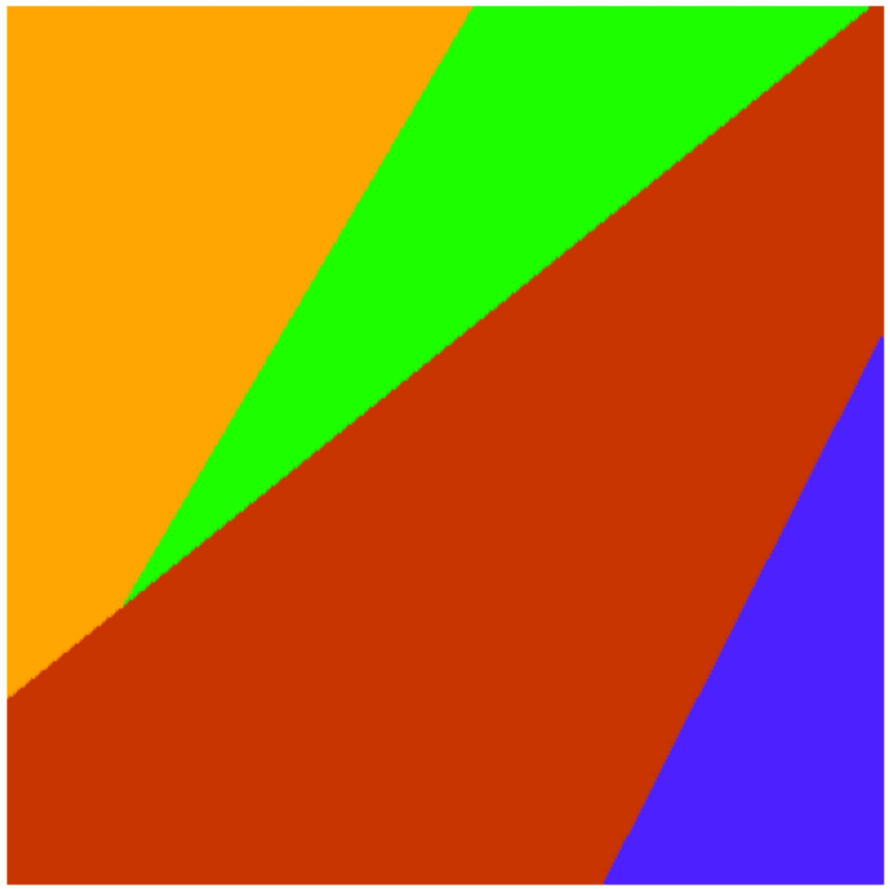


Querying BT everywhere

Toy world

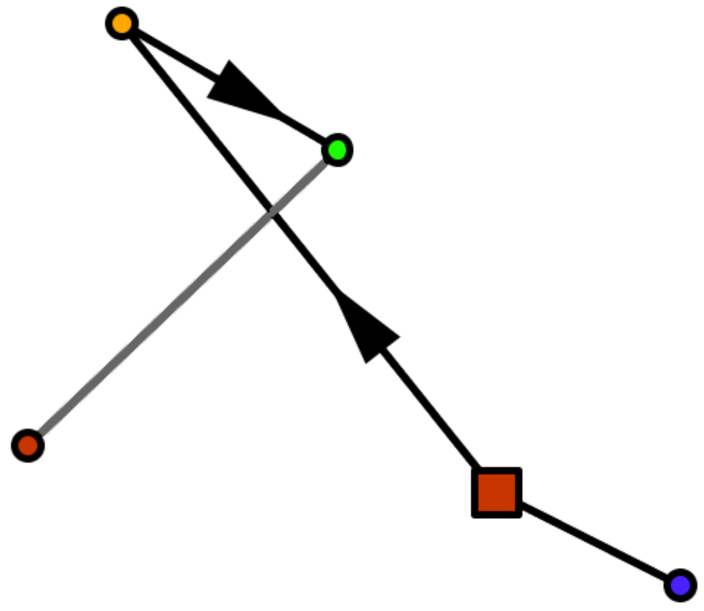
- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added


Boundary Tree

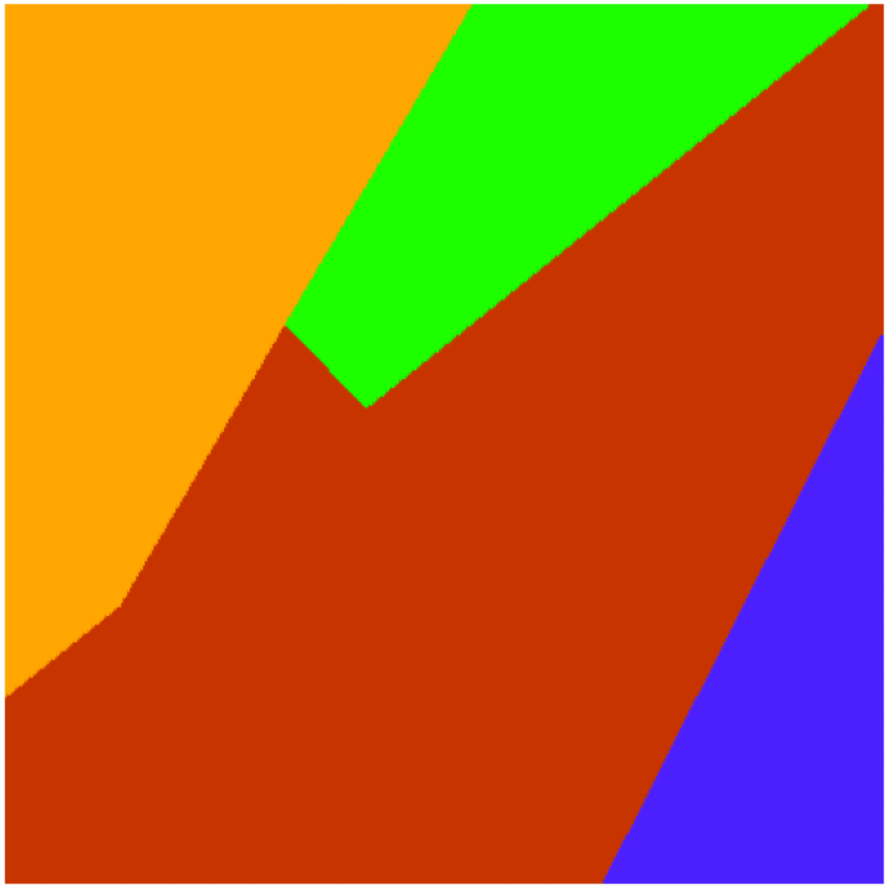

Querying BT everywhere

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



Boundary Tree



Querying BT everywhere

Toy world

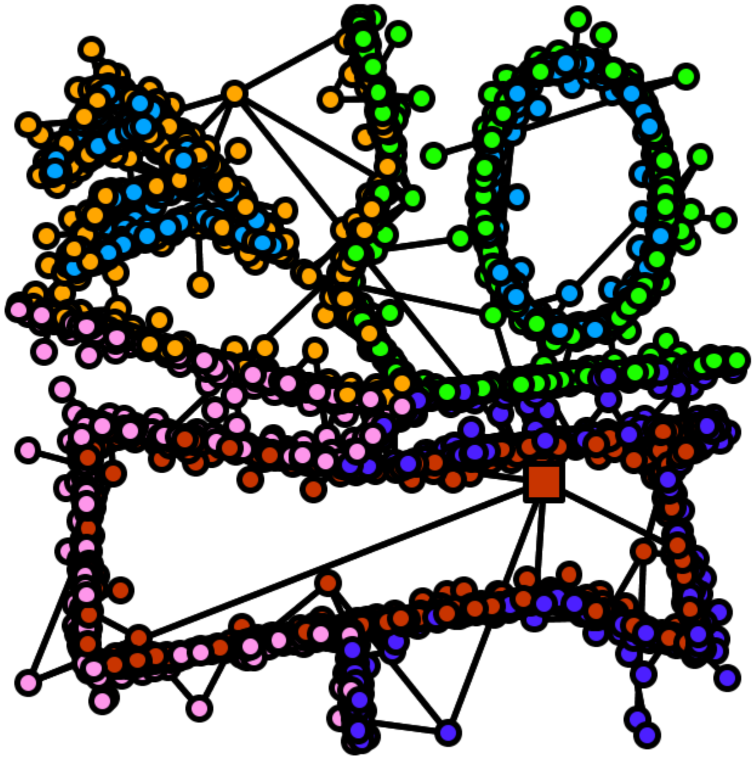
- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added
- After showing 10,000 points, 680 points stored

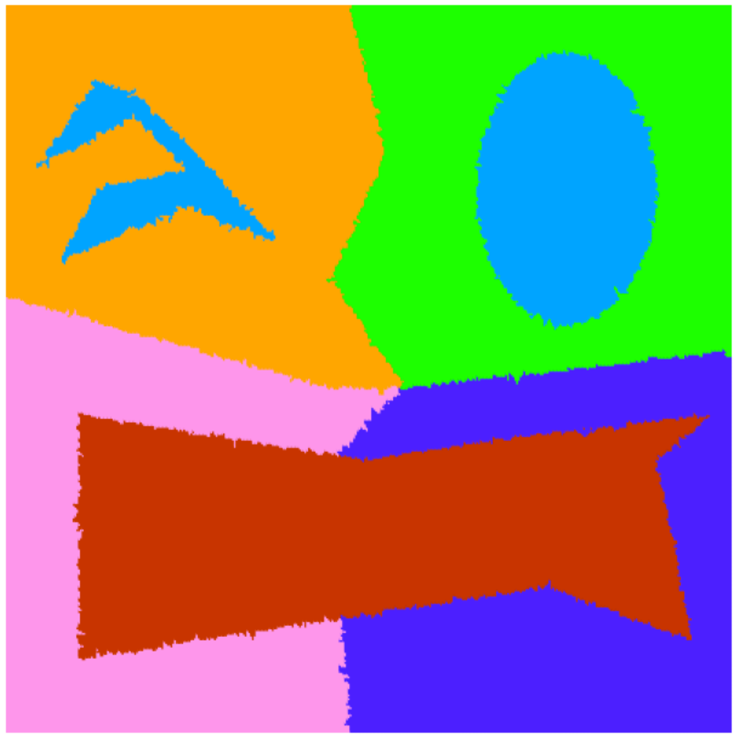
Boundary Tree

Querying BT everywhere

Toy world

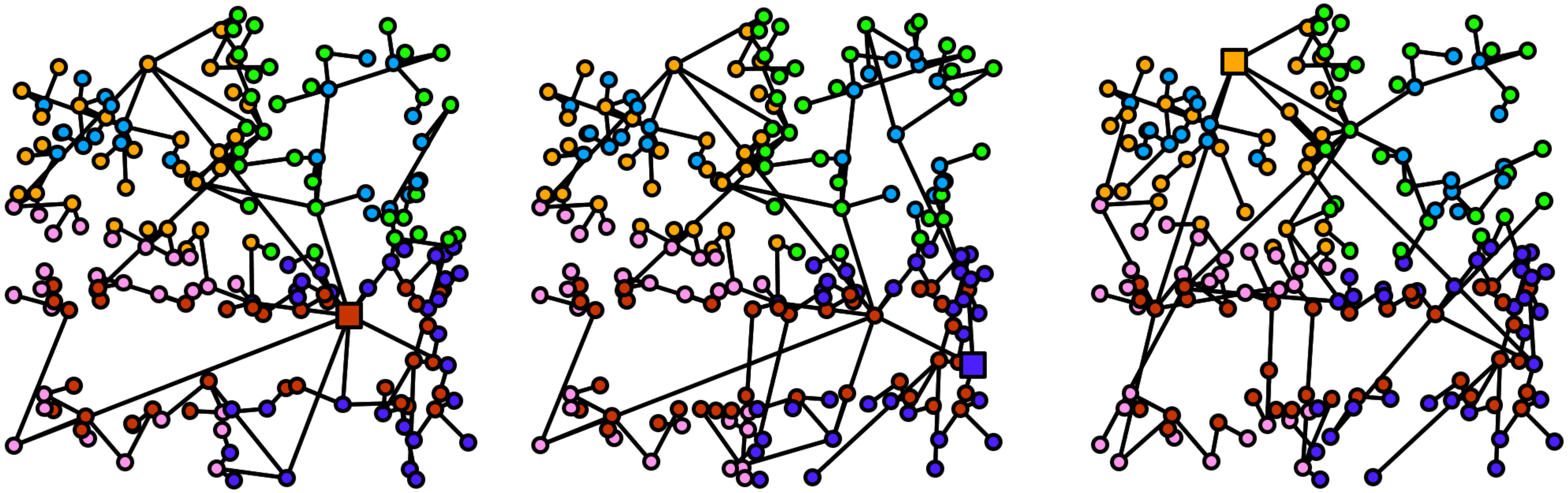
- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added
- After showing 100,000 points, 2220 points stored


Boundary Tree


Querying BT everywhere

Toy world

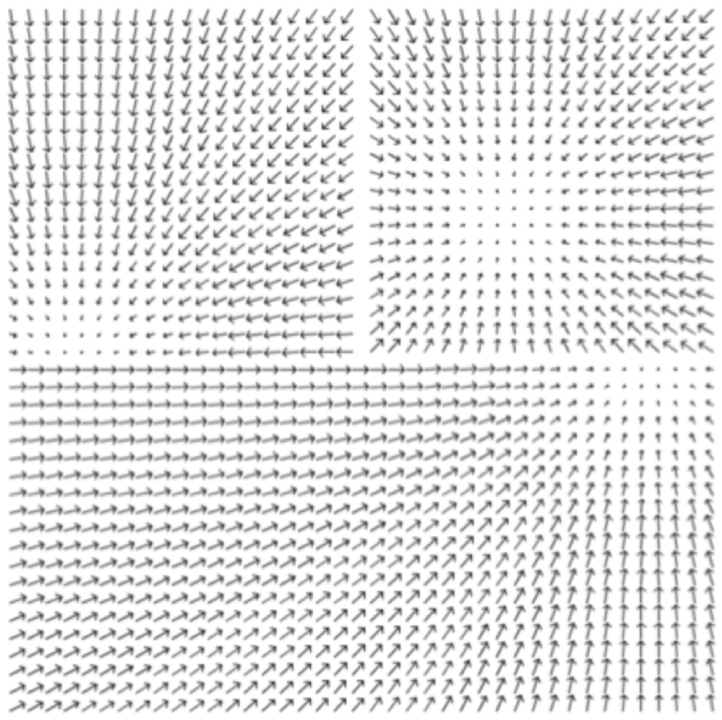
- Points come in one at a time with associated class
- First point becomes root node
- Greedy traversal of tree, class of result is prediction
- If prediction is incorrect, training point and edge is added
- Forest of trees - differently seeded - better generalization



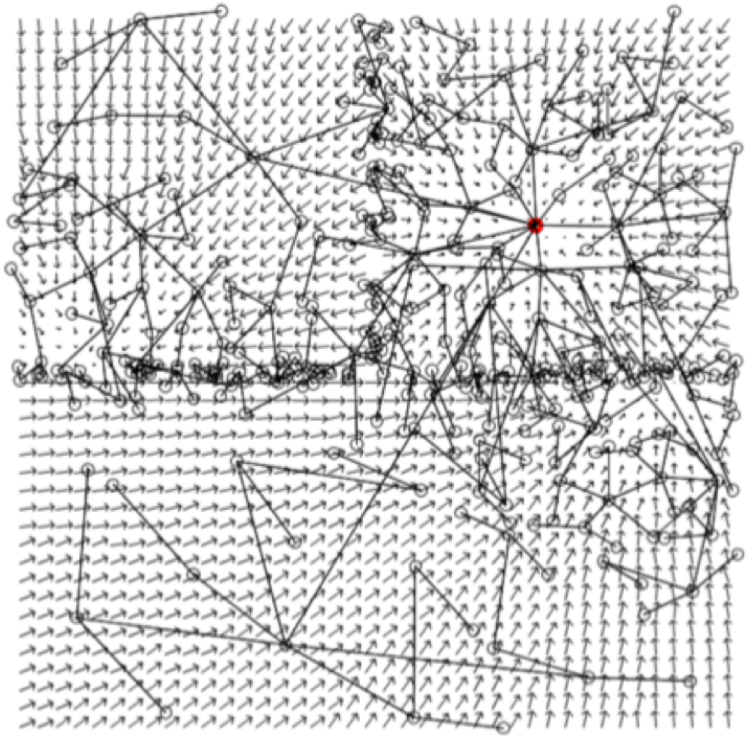
Shown 10,000 points

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

Mathy, C.; Derbinsky, N.; Bento. J; Rosenthal, J.; Yedidia, J.S., "The Boundary Forest Algorithm for Online Supervised and Unsupervised Learning," AAAI 2015.

(a)

Data	BF	BF-4	R-kd	CT	RF
dna	0.34	0.15	0.042	0.32	3.64
letter	1.16	0.80	0.12	1.37	7.5
mnist	103.9	37.1	5.67	168.4	310
pendigits	0.34	0.42	0.059	0.004	4.7
protein	35.47	13.81	0.90	44.4	191
seismic	48.59	16.30	1.86	176.1	2830

(b)

BF	BF-4	1-NN	3-NN	R-kd	CT	RF
0.34	0.15	3.75	4.23	0.050	0.25	0.025
1.16	0.80	5.5	6.4	1.67	0.91	0.11
23.9	8.7	2900	3200	89.2	417.6	0.3
0.34	0.42	2.1	2.4	0.75	0.022	0.03
35.47	13.8	380	404	11.5	51.4	625
16.20	5.2	433	485	65.7	172.5	1.32

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).

Data	BF	OBF	1-NN	3-NN	RF	R-kd	CT
dna	14.3	13.1	25.0	23.9	5.7	22.5	25.55
letter	5.4	5.5	5.5	5.4	7.6	5.5	5.6
mnist	2.24	2.6	3.08	2.8	3.2	3.08	2.99
pendigits	2.62	2.60	2.26	2.2	5.2	2.26	2.8
protein	44.2	41.7	52.7	50.7	32.8	53.6	52.0
seismic	40.6	39.6	34.6	30.7	23.7	30.8	38.9

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

Zoran, D.; Lakshminarayanan, B.; Blundell, C. "Learning Deep Nearest Neighbor Representations Using Differentiable Boundary Trees," ArXiv 2017

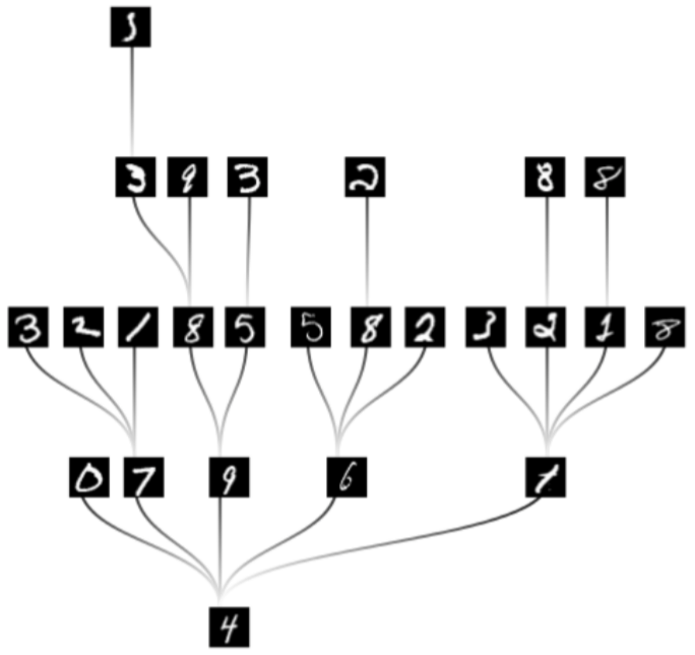


Figure 5. A tree built using the trained representation over MNIST digits using 1000 samples. Samples are presented here in the original pixel space, but the learned representation is used to construct the tree. Note the simple, interpretable structure — nodes are either prototypical examples or boundary cases. Remarkably, this tree still achieves less than 2% error on the test set. See text for details.

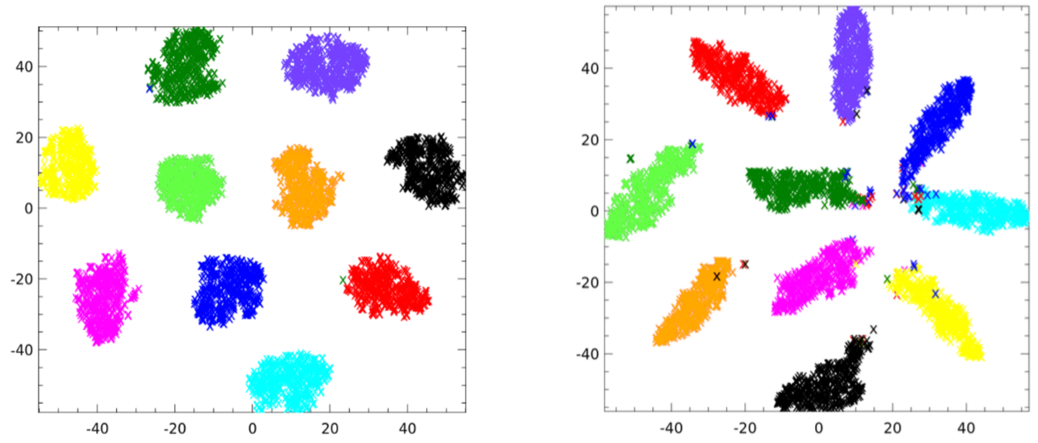


Figure 7. *t*-SNE visualisation of representations. On the left are samples projected using our trained representation (from the training set). As can be seen, the transform separates the classes nicely, allowing the tree to have a very small number of nodes in order to achieve its task. On the right is the t-SNE plot made with neural net classifier outputs. Though classes are separated nicely, it's not as clean as our result.

See also Wu, H.; Wang, C.; Yin, J.; Lu, K.; Zhu, L. "Interpreting Shared Deep Learning Models via Explicable Boundary Trees," ArXiv 2017

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.