DIANA Fellowship: Improving BDTs in TMVA

By Andrew Carnes (University of Florida),
Sergei Gleyzer (University of Florida)

Introduction

- **DIANA Fellowship**
  - October, November, and December 2016
  - Improved **Boosted Decision Trees** (BDTs) in ROOT's TMVA

Presentation Outline

1) Debugged Regression Evaluation
2) Brief BDT algorithm overview
3) New Loss Functions
4) Parallelization

1. Evaluation

Evaluation Bug

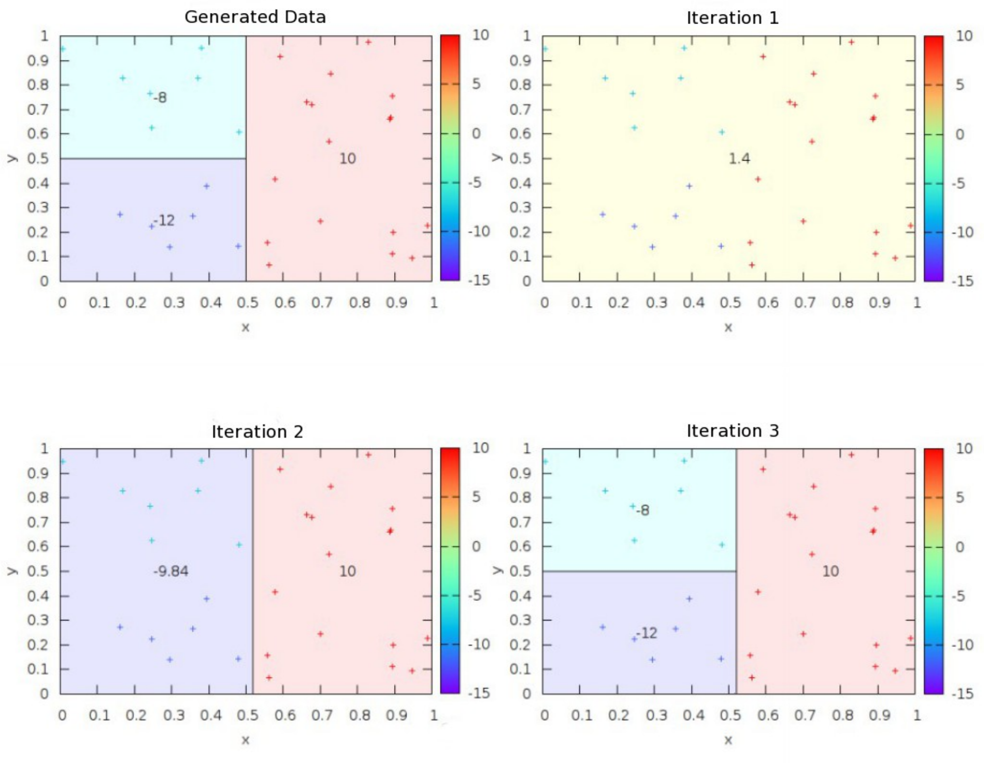
- After building a regression model you use it to predict other data/events
- **Evaluating predictions in regression was known to take too long**
    - Probably a bug somewhere
- 1 Million events over 10 trees with depth 4 **should take on the order of 1 second**
- **TMVA took about 15 minutes**
- Progress bar was drawn every event (1 Million Times in this case!)
- After fixing this bug the evaluation now takes 2 seconds
- **Evaluation time was reduced 460x**
    - **Bug fix applies for other regression algorithms**, but this has not been studied

General regression evaluation was fixed. Now ...

2. BDT Algorithm Overview

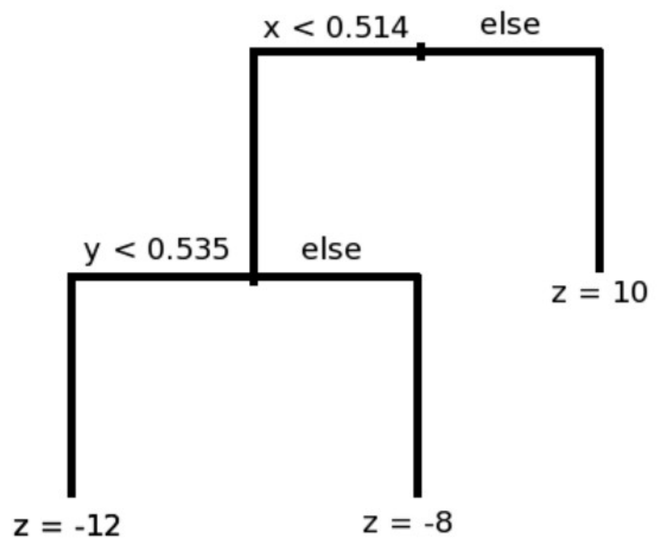
So that we can understand the other improvements

Decision Trees



- Supervised Machine Learning Algorithm
- **Models "true values" by discretizing space and fitting constants**

- **Recursively split subspaces into two regions**
- Search regions along all features for best split



Boosting

- Boosting uses a collection of trees to improve the accuracy of the algorithm
- Add trees to iteratively correct the predictions
- Each tree models optimum dy' that should be added at that stage

Boosting Iterations

0) $y' = T_0$

1) $y' = T_0 + T_1$

2) $y' = T_0 + T_1 + T_2$

...

N) $y' = T_0 + T_1 + T_2 + T_3 + \ldots T_N$

y' := predicted value

y := true value

- dy' models the negative gradient of the Loss Function so that the error heads towards a minimum

BDT Algorithm

- **Set Targets** for new tree (Tree models dy' = grad)
- **Build tree** to model targets
    - Search each feature for optimum split (split that minimizes loss)
    - Split region into two daughter regions
    - Fit appropriate constants in daughter regions
    - Repeat for daughter regions until stopping criteria (MAX_DEPTH)
- **Set optimal fits in terminal nodes,** now that these regions are finalized
    - Regions were built based upon the gradient
    - Set constant fit in each region to minimize the loss there
    - Scale by learning rate if desired
- **Repeat** until ntrees reaches MAX_TREES

3. Loss Functions

Loss Functions

- **Loss Functions tell the algorithm how to measure the error**
  - Build a tree to minimize the loss!
  - Different loss, different models!
  - Use the appropriate loss to focus the regression on important events

- TMVA had a hard coded Loss Function, Huber
- **Replaced hard coded loss function with an abstract class**
  - Implemented the abstract class with
    - **Least Squares** $\Sigma(y-y')^2$
    - **Absolute Deviation** $\Sigma|y-y'|$
    - **Huber** $\Sigma F(y-y')$, $F = 0.5(y-y')^2$ for $|y-y'| > a$ (quadratic core)

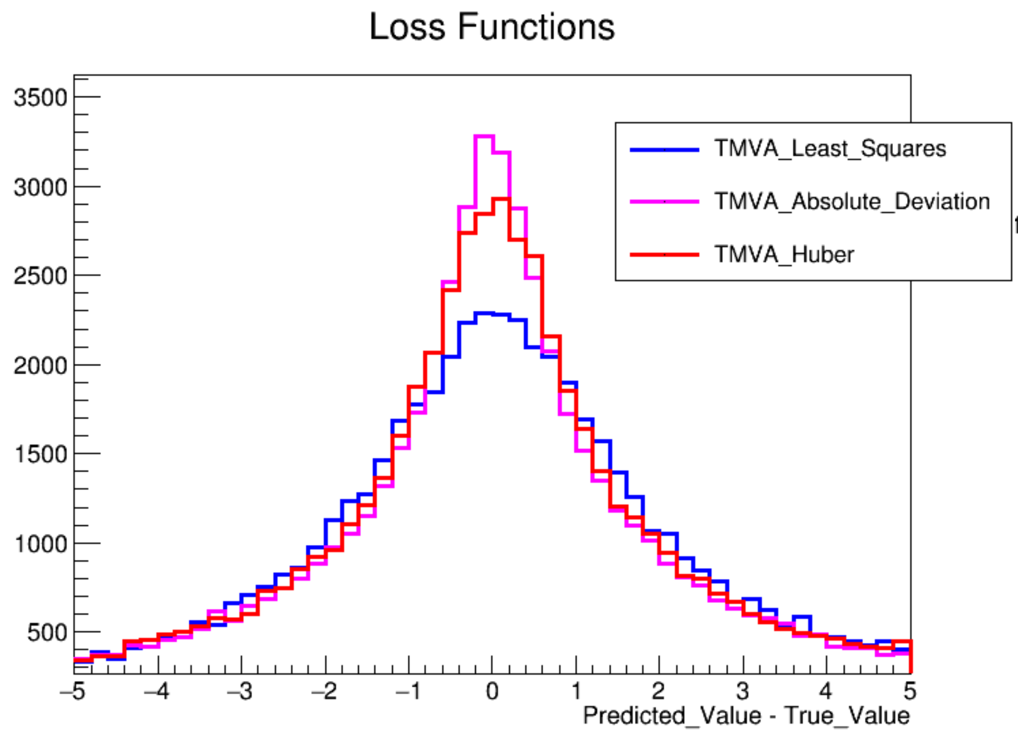
      $= a|y-y'| - 0.5a^2$ otherwise (linear tails)

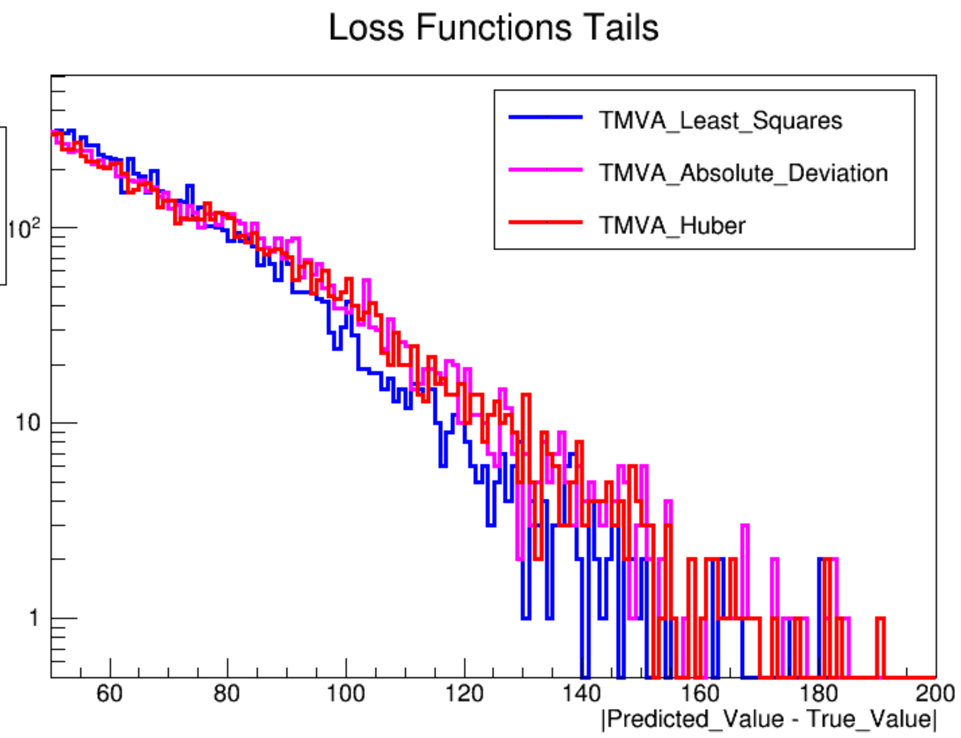
Loss Functions

- **Least Squares** ($\Sigma(y-y')^{2)}$ counts the tails (large |y-y'|) the most
  - Therefore should perform the best in the tails
  - Worst in the core
- **Absolute deviation** ($\Sigma|y-y'|$) counts the tails the least
  - Should perform the worst in the tails
  - Best in the core
- **Huber** (quadratic core, linear tails) is inbetween the two
  - Should be between both in the tails and the core

Loss Functions

- New **loss function capability tested on a standard regression sample**
- **Behaves as expected in the core**
  - Absolute Deviation is best
  - Followed by Huber
  - Followed by Least Squares
- **Behaves as expected in the tails**
  - Least Squares is best
  - Followed by Huber
  - Followed by Absolute Deviation





Loss Functions Conclusion

- **Three New Loss Functions** are implemented in TMVA BDTs and now available in ROOT
  - Since ROOT v6.0.8
  - User can target the events they care about the most
  - With the abstract class, the code is easily extended to add new loss functions in the future
- **The TMVA Users Guide has been updated** to document the new options
- **A Jupyter notebook is available**
  - Showcases the new Loss Functions
  - http://swan.web.cern.ch/content/machine-learning
  - Provides an example for regression, which was missing before

4. Parallelizing the Training

Parallelization

- **Parallelized the training of the algorithm**
  - Can't build multiple trees at once since each new tree depends on the results of the previous trees
  - So we targeted the computationally lengthy processes in the boosting and building of a single tree
  - Naturally **loops over the collection of training data are the longest processes**

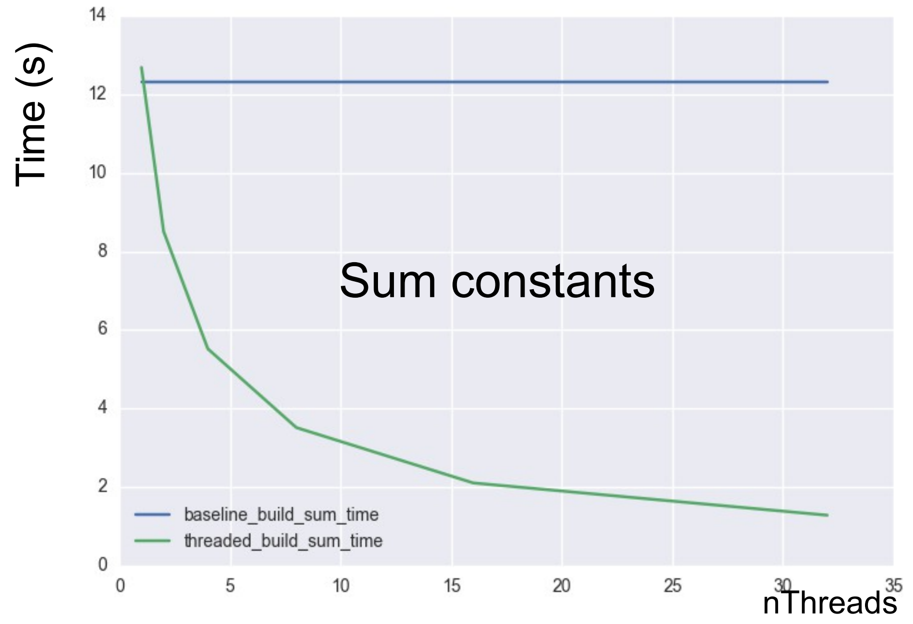
```
for(e in events)
    f(e)
```

```
for(e in group1)    for(e in group2)    ...    for(e in groupN)
    f(e)                f(e)                       f(e)
```

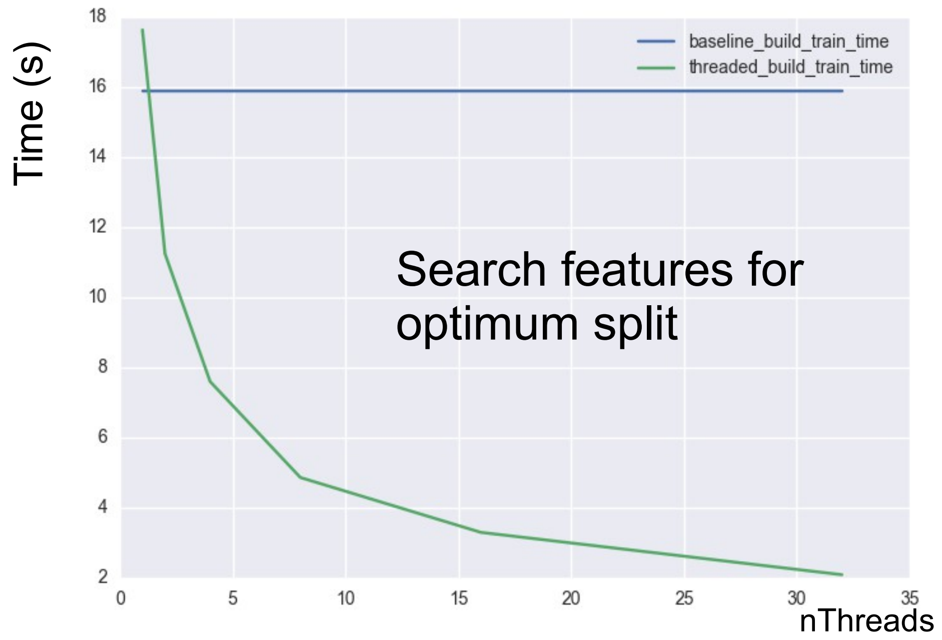
- **Broke up the loop into chunks to be run in parallel**
- Must loop over all the events to
  - **Update Targets** for the next tree → **Parallelized**
  - **Build the tree**
    - **Find the optimum split** along each feature → **Parallelized**
  - **Update Predictions** after building a tree → **Parallelized**

Build Tree Parallelization

Active time over 10 trees for 1 Million events



The loop over the events in order to calculate some constants needed for the best split calculation

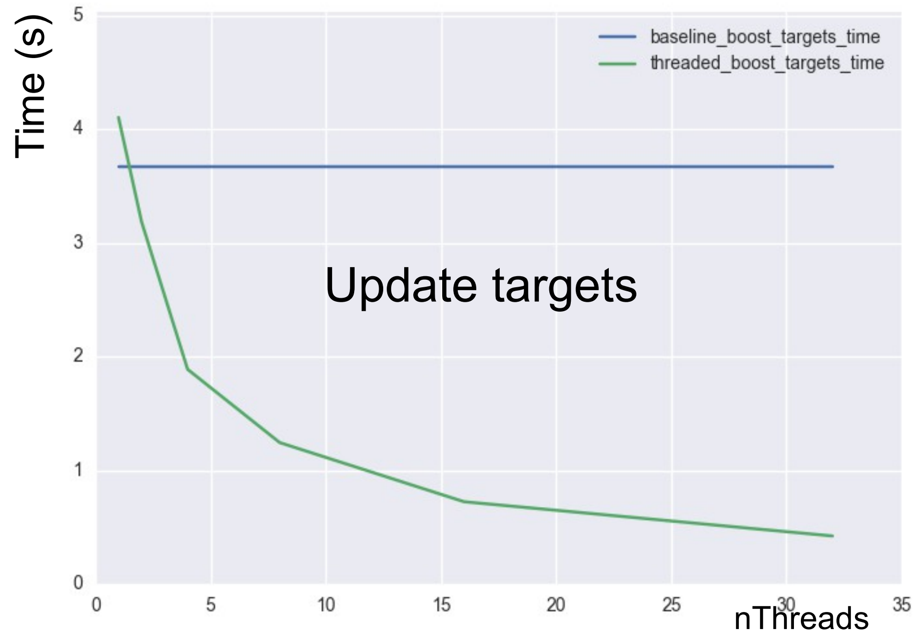


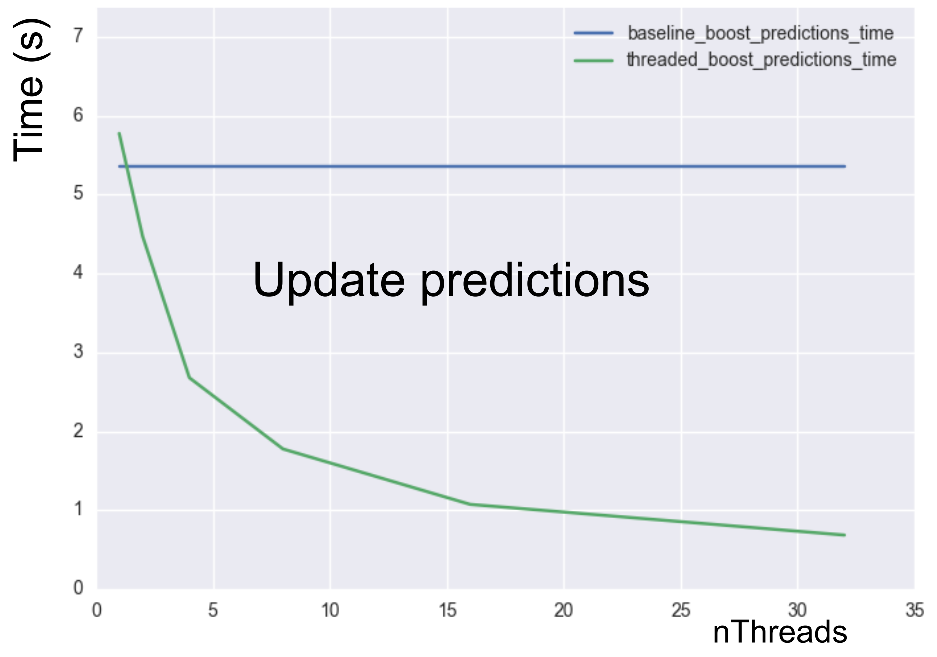
The loop over the events to get the best feature and split point

- There are three processes that loop over the events
  1) **Calculate some constants** necessary for the best split search
  2) **Search for the best split point** along each feature
  3) **Filter the events from the internal node to the daughter nodes**
     - This was **not parallelized** since it involves pushing back to a vector
- **1&2 were parallelized by chunking the event loop**
  - Used ROOT's TthreadExecutor.hxx - **multi-threading**
  - TThreadExecutor is built ontop of Intel's TBB library

Boosting Parallelization

Active time over 10 trees for 1 Million events

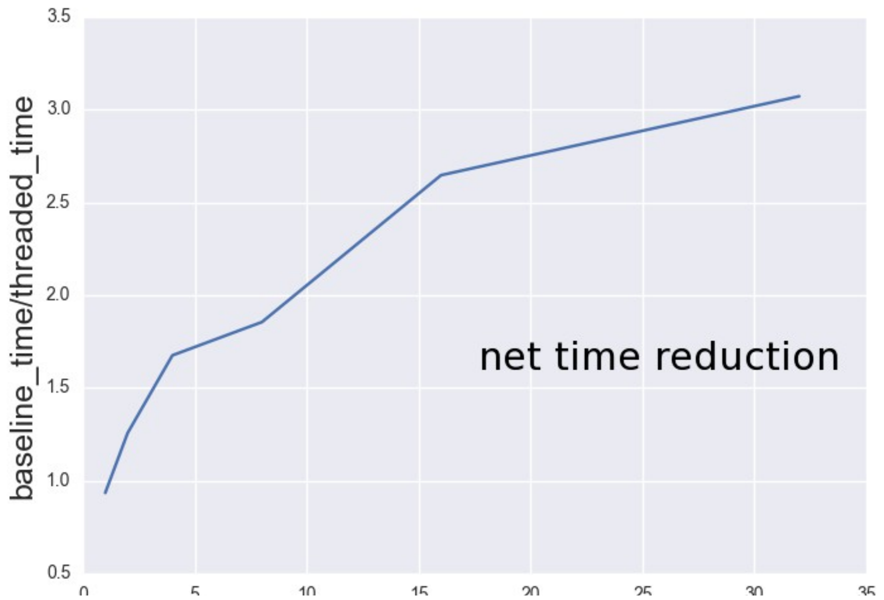


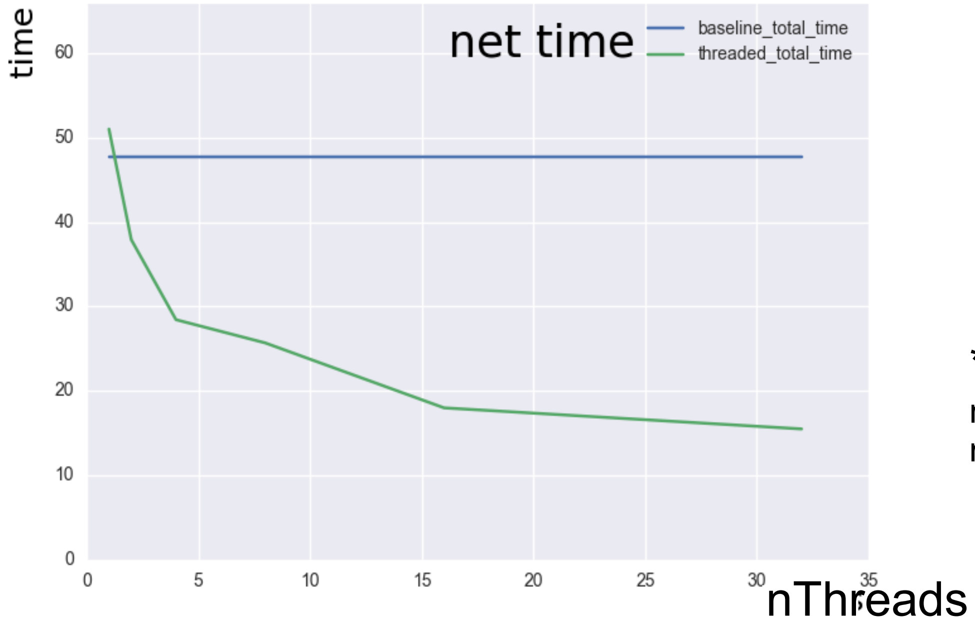


- Again, there are **three processes that loop over the events**
  1) **Update the targets** for the next tree for every event
  2) **Set optimum fits in terminal nodes**
     - Involves pushing the events into terminal node vectors, **couldn't parallelize this part**
  3) **Update the predictions** for every event based upon the previous tree
  - **1&3 were parallelized by chunking the event loop**

Parallelization Conclusions

For 1 Million events*





- **Reduction in time of about 1.6x for with 4 cores**
- **2.6x reduction for 16 cores**
- **Asymptotes at about 3x reduction**
  - Some of the intensive processes couldn't be parallelized
  - These required accessing the same changing iterator at the same time
- Clever schemes exist to parallelize the tricky push_back processes and further improve the speed?

* Used 10 trees to study timing, but the time is linear in the number of trees, so the net time reduction is the same regardless of the number of trees

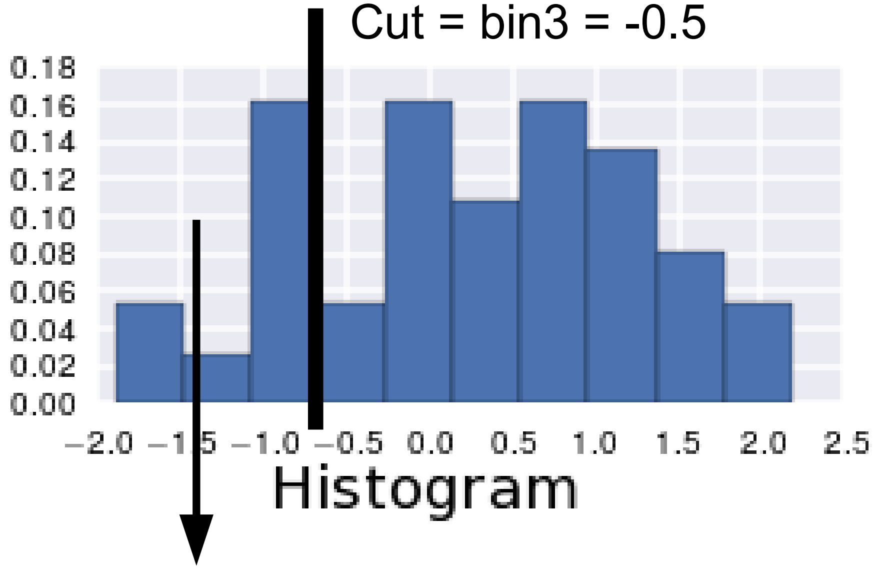
Conclusions

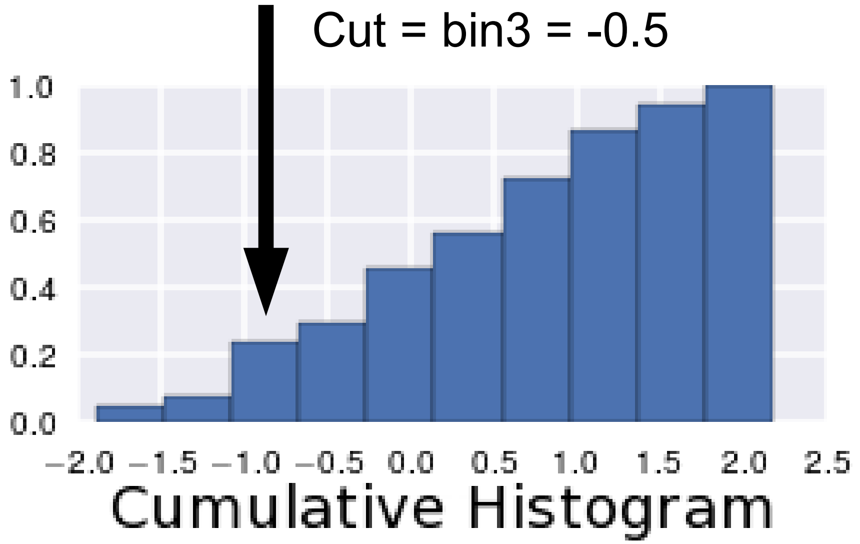
- Loss Function capabilities are abstracted, easy to add new ones in the future
- **Three available loss functions in the BDTs now**, compared to just one before
  - Least Squares
  - Absolute Deviation
  - Huber
- **Training has been parallelized with speed increases**
  - **1.6x for 4 CPUs**, 1 Million events
  - **2.6x for 16 CPUs**, 1 Million events
  - Some major processes remain unparallelized
    - Would require accessing a changing iterator simultaneously
    - Clever solutions may exist
    - There are possible further gains in speed if so
- **Evaluation time has been improved substantially for all regression methods**
  - **460x reduction in time for 1 million events**
  - Improvement by bug removal
- **Users Guide updated** documenting new features
- **Jupyter notebook** created as a regression example to illustrate new features interactively
- Loss Functions already available in ROOT (6.0.8), parallelization will be available in a later release

Backup

- Histogramming the optimum split search

Histograms





- Error reduction calculation for a bin requires sum of bins on left side, and the total sum
- Sum of bins on left side of the cut is the same as the value in bin = 3 of the cumulative sum histogram, net sum is the last bin
- Simply make the cumulative sum histogram and search through the bins to calculate the error reduction

- Need 3 histograms for the error reduction calculation
  - Target
  - Target squared
  - Number of events