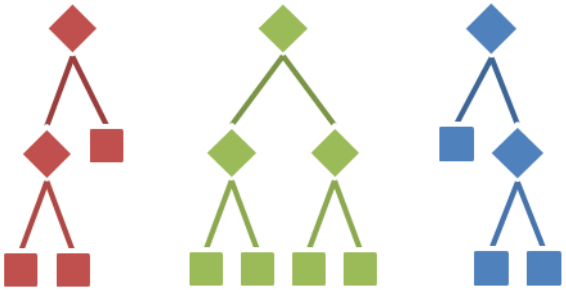
Scaling Up
Decision Tree Ensembles

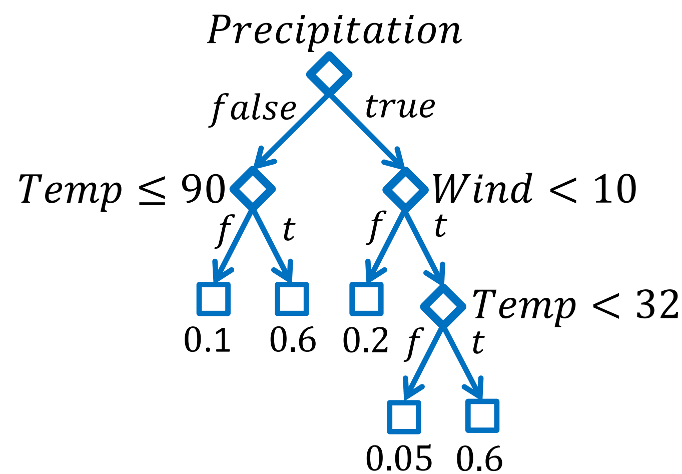


Misha Bilenko (Microsoft)
with Ron Bekkerman (LinkedIn) and John Langford (Yahoo!)

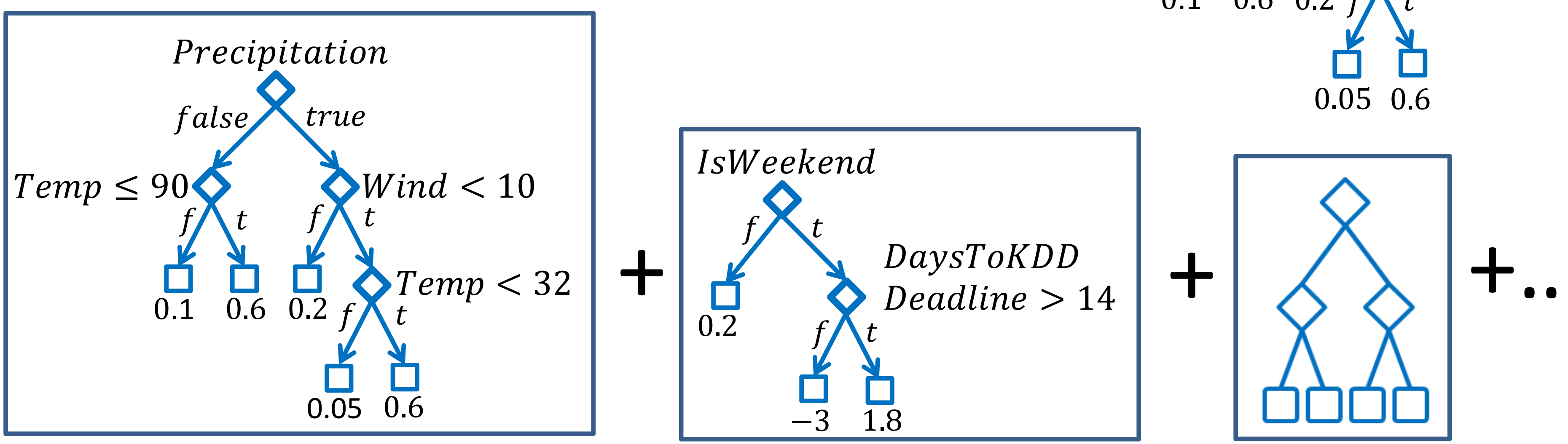
http://hunch.net/~large_scale_survey

Tree Ensembles: Basics

- Rule-based prediction is natural and powerful (non-linear)
  - *Play outside: if no rain and not hot, or if snowing but not windy.*
- Trees hierarchically encode rule-based prediction
  - Nodes test features to branch
  - Leaves produce predictions
  - Regression trees: numeric outputs



- Ensembles combine tree predictions



Tree Ensemble Zoo

- Different models can define different types of:
  - Combiner function: voting vs. weighting
  - Leaf prediction models: constant vs. regression
  - Split conditions: single vs. multiple features

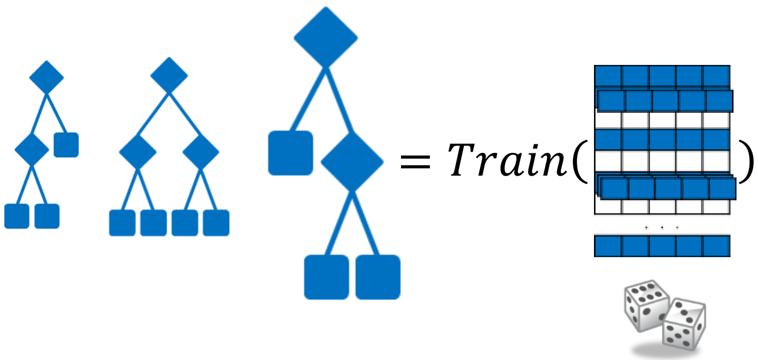
- Examples (small biased sample, some are not tree-specific)
  - Boosting: AdaBoost [FS97], LogitBoost [F+00], GBM/MART [F01b], BrownBoost [F01a], Transform Regression [SUML11-Ch9]
  - Random Forests [B01]: Random Subspaces [H98], Bagging [B98], Additive Groves [S+07], BagBoo [P+10]
  - Beyond regression and binary classification: RankBoost [F+03], abc-mart [L09], GBRank [Z+08], LambdaMART [SUML11-Ch8]

Tree Ensembles Are Rightfully Popular

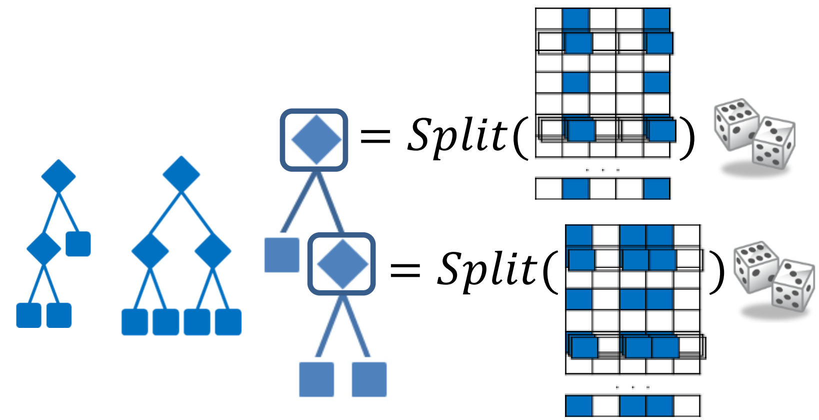
- **State-of-the-art accuracy**: web, vision, CRM, bio, …
- **Efficient at prediction time**
  - Multithread evaluation of individual trees; optimize/short-circuit
- **Principled**: extensively studied in statistics and learning theory
- **Practical**
  - Naturally handle mixed, missing, (un)transformed data
  - Feature selection embedded in algorithm
  - Well-understood parameter sweeps
  - Scalable to extremely large datasets: rest of this section

Naturally Parallel Tree Ensembles

- No interaction when learning individual trees
  - **Bagging**: each tree trained on a bootstrap sample of data
  - **Random forests**: bootstrap plus subsample features at each split
  - For large datasets, local data replaces bootstrap -> **embarrassingly parallel**



Bagging tree construction

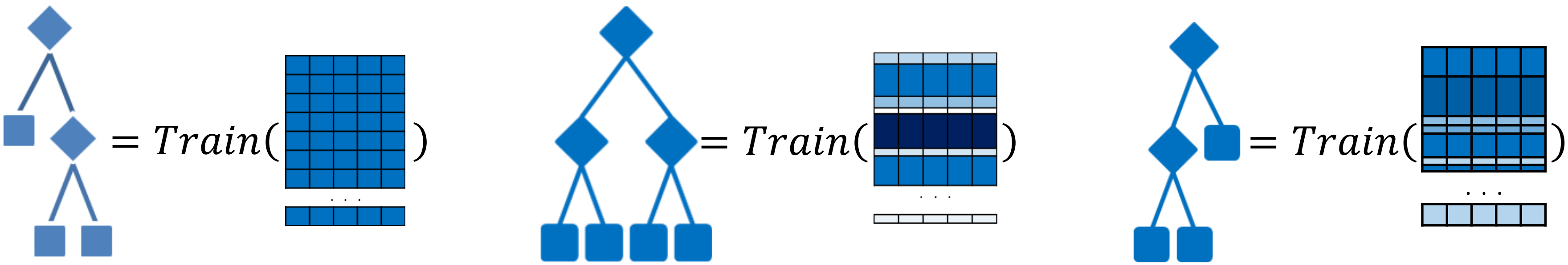


Random forest tree construction

Boosting: Iterative Tree Construction

"Best off-the-shelf classifier in the world" – Breiman

- Reweight examples for each subsequent tree to focus on **errors**

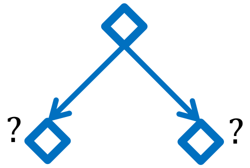


- Numerically: gradient descent in function space
  - Each subsequent tree approximates a step in $-\frac{\partial L}{\partial f}$ direction
  - Recompute *target labels* $\quad y^{(m)} = -\left[\frac{\partial L(y, f(x))}{\partial f(x)}\right]_{f(x)=f^{(m-1)}(x)}$
  - Logistic loss: $L(y, f(x)) = \log(1 + \exp(-yf(x))) \quad y^{(m)} = \frac{y}{1+\exp(yf(x))}$
  - Squared loss: $L(y, f(x)) = \frac{1}{2}(y - f(x))^2 \quad y^{(m)} = y - f(x)$

Efficient Tree Construction

- Boosting is iterative: scaling up = parallelizing tree construction

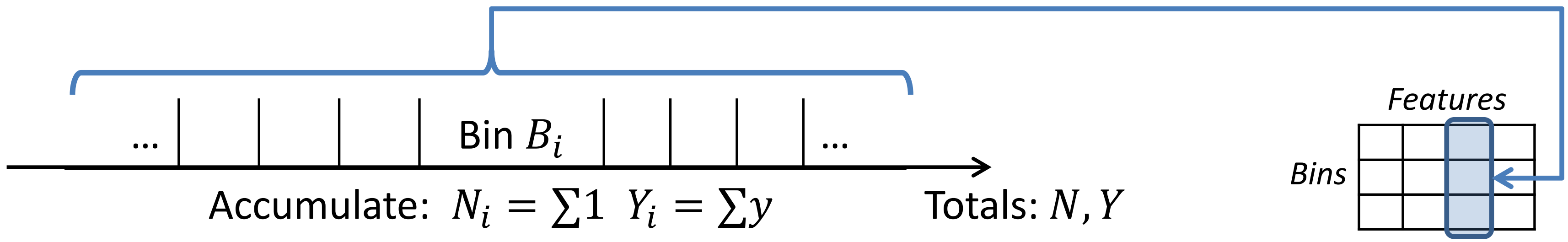
- For every node: pick best feature to split
  - For every feature: pick best split-point
    - For every potential split-point: compute gain
      - For every example in current node, add its gain contribution ~~for given split~~



- Key efficiency: limiting+ordering the set of considered split points
  - Continuous features: discretize into bins, splits = bin boundaries
  - Allows computing split values in a single pass over data

Binned Split Evaluation

- Each feature's range is split into k bins. Per-bin statistics are aggregated in a single pass

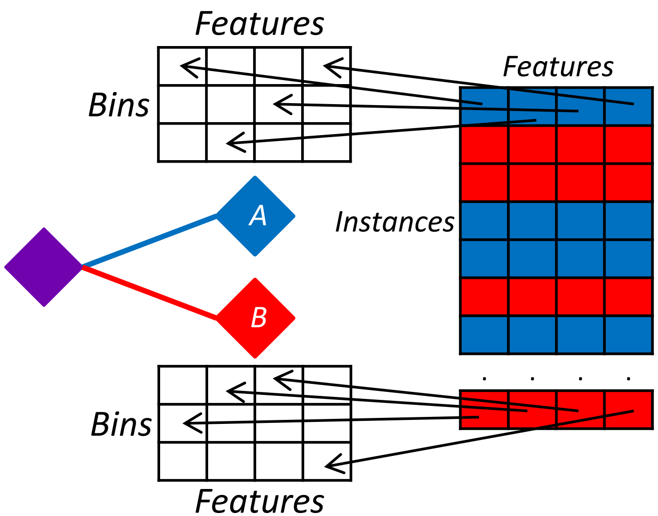


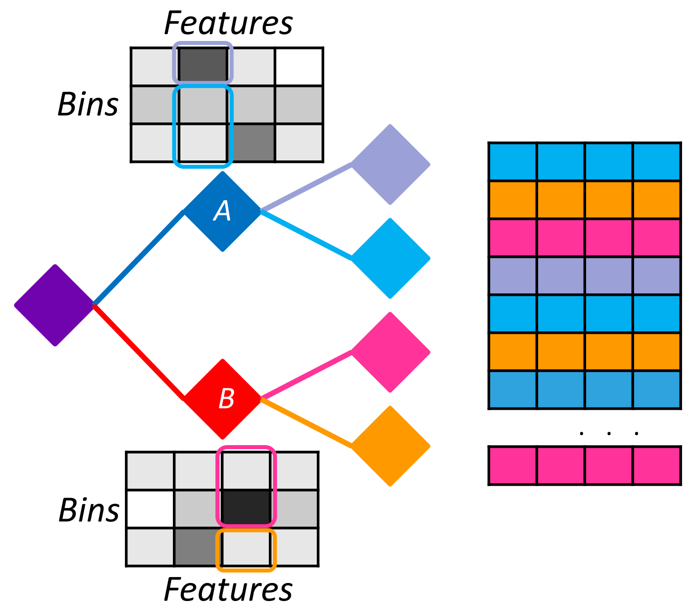
Accumulate: $N_i = \sum 1 \quad Y_i = \sum y$ Totals: N, Y

Split gain: $-\frac{Y_{0:i}^2}{N_{0:i}} - \frac{\left(Y - Y_{0:i}\right)^2}{N - N_{0:i}}$

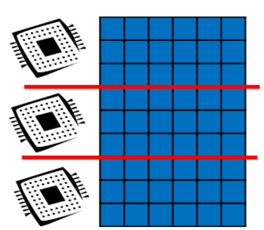
- For each tree node, a two-stage procedure
  (1) Pass through dataset aggregating node-feature-bin statistics
  (2) Select split among all (feature,bin) options

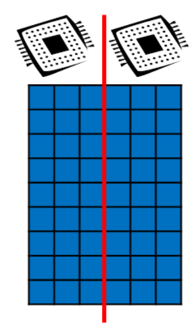
Tree Construction Visualized



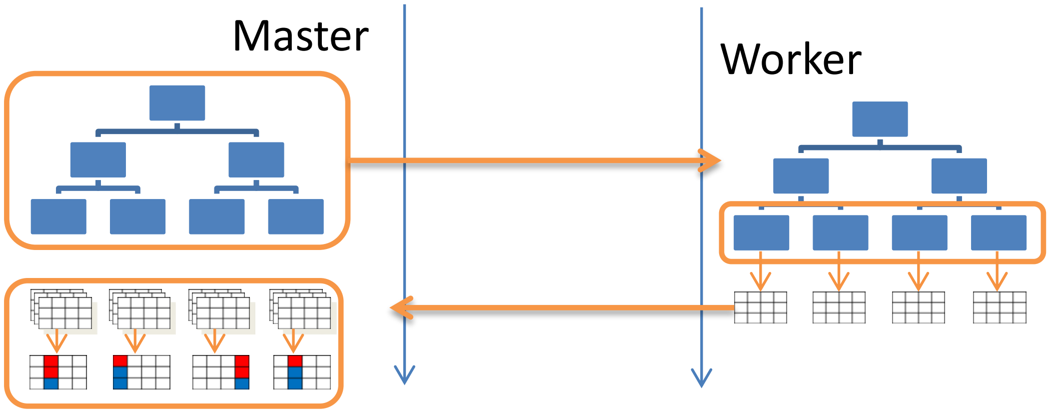


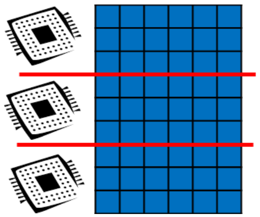
- Observation 1: a single pass is sufficient per tree level
- Observation 2: data pass can iterate by-instance or by-feature
  - Supports horizontally or vertically partitioned data





Data-Distributed Tree Construction

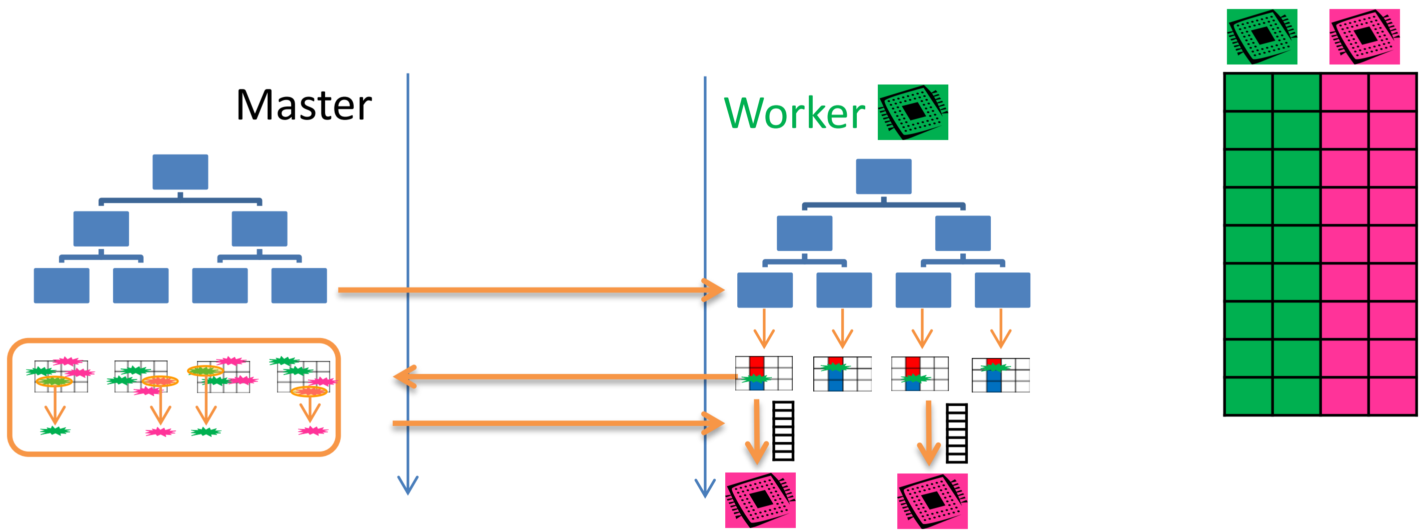




- Master
  1. Send workers current model and set of nodes to expand
  2. Wait to receive local split histograms from workers
  3. Aggregate local split histograms, select best split for every node

- Worker
  - 2a. Pass through local data, aggregating split histograms
  - 2b. Send completed local histograms to master

Feature-Distributed Tree Construction



- Workers maintain per-instance index of current residuals and previous splits
- Master
  1. Request workers to expand a set of nodes
  2. Wait to receive best per-feature splits from workers
  3. Select best feature-split for every node
  4. Request best splits' workers to broadcast per-instance assignments and residuals

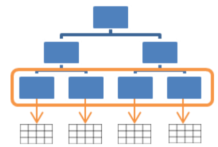
- Worker
  - 2a. Pass through all instances for local features, aggregating split histograms for each node
  - 2b. Select local features' best splits for each node, send to master

PLANET [SUML11-Ch2]

(Parallel Learner for Assembling Numerous Ensemble Trees)

- **Platform**: MapReduce (Google), RPC (!)
- **Data Partitioning**: Horizontal (by-instance)
- **Binning**: during initialization
  - Single-pass approximate quantiles via [Manku et al. ’98]
- Modulate between distributed and single-machine tree construction
  - *MapReduceQueue*: distributed tree construction
  - *InMemoryQueue*: single-worker tree construction (when node small enough)

PLANET: ExpandNodes Mapper



- Go through examples, aggregating summaries $(\sum y, \sum 1)$ for every:
  - Node: total statistics for the node (applies for all features)
  - Split: for every feature-bin

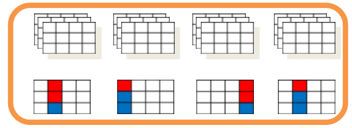
$Mapper.Map(\langle x, y \rangle, model\ M, \text{nodes}\ N)$

$n = \text{TraverseTree}(\text{M}, \mathbf{x})$
If $n \in N$ **then**
  $\text{agg\_tup}_n \leftarrow y$
  For each $X \in \mathcal{X}$ **do**
    $v = \text{Value on } X \text{ in } \mathbf{x}$
  For each Split point s of X s.t. $s < v$ **do**
    $T_{n,X}[s] \leftarrow y$

$Mapper.Finalize()$

For each $n \in N$ **do**
  Output to all reducers$(n, \text{agg_tup}_n)$
  For each $X \in \mathcal{X}$ **do**
  For each Split point s of X **do**
    $\text{Output}((n, X, s), T_{n,X}[s])$

PLANET: ExpandNodes Reducer



- Receives (presorted)
  - (a) Totals for every node
  - (b) Feature-split histogram for every node
- Adds up histograms, finds best split among all possible ones.
- Emits best split found for given node

Input: Key k,Value Set V

If $k == n$ **then**

  // Aggregate agg_tup$_n$'s from mappers by pre-sorting.

  agg_tup$_n$ = Aggregate(V)

Else // $k == n, X, s$

  // Split on ordered feature

  agg_tup$_{left}$ = Aggregate(V)

  agg_tup$_{right}$ = agg_tup$_n$ - agg_tup$_{left}$

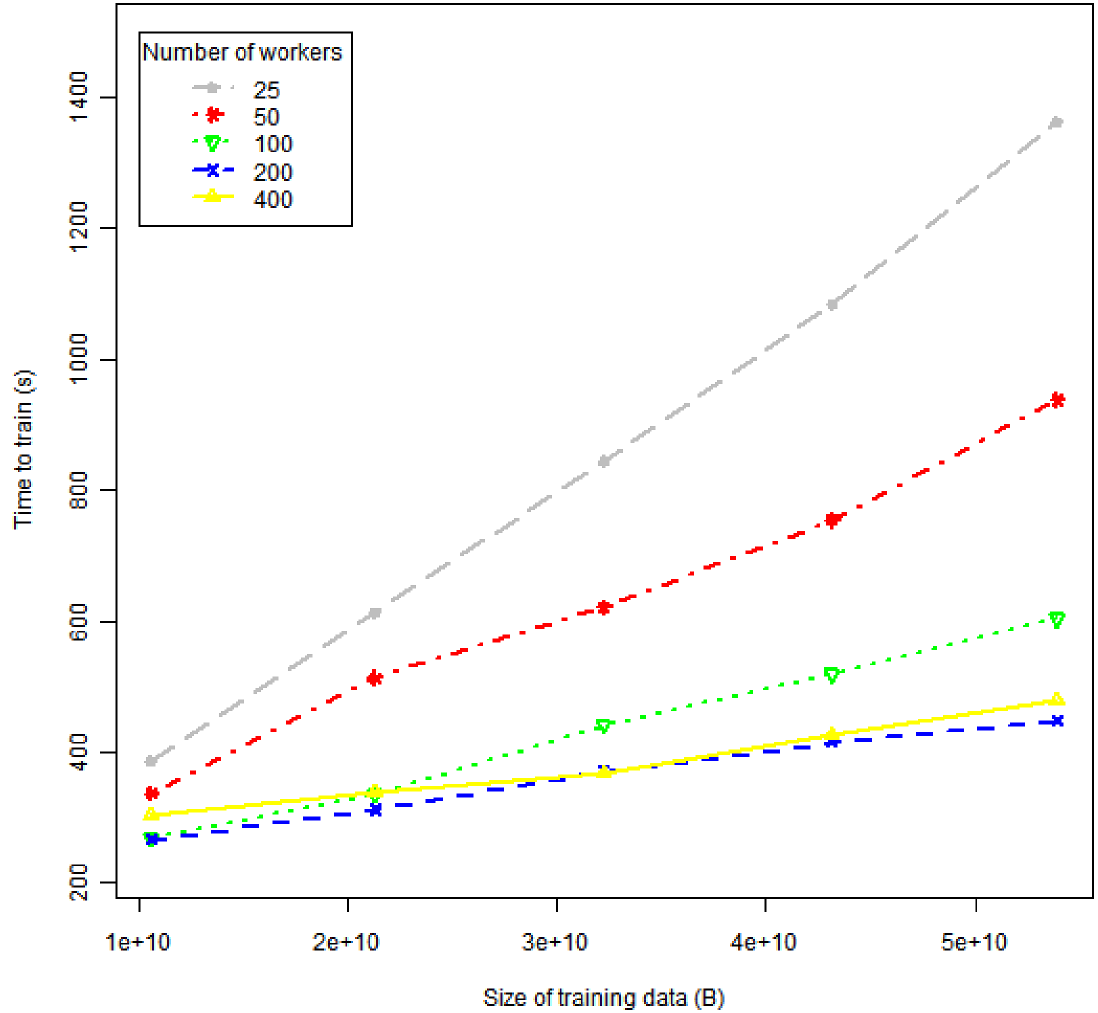
  UpdateBestSplit($S[n]$,X,s,agg_tup$_{left}$, agg_tup$_{right}$)

PLANET: Master (Controller)

- Master (Controller)
  - Creates and sends jobs to *MRQueue* or *InMemoryQueue*
  - Aggregates best splits for each node, updates model
- Engineering is non-trivial, impactful
  - MapReduce per-worker setup and tear-down costs are high
    - Forward-schedule: pre-allocate Mappers and Reducers, standby
    - Controller uses RPC to send jobs
  - Categorical attributes
    - Specially handled (different MR keys; still linear using the “Breiman trick”)
    - Evaluation speedup: fingerprint, store signature with each node

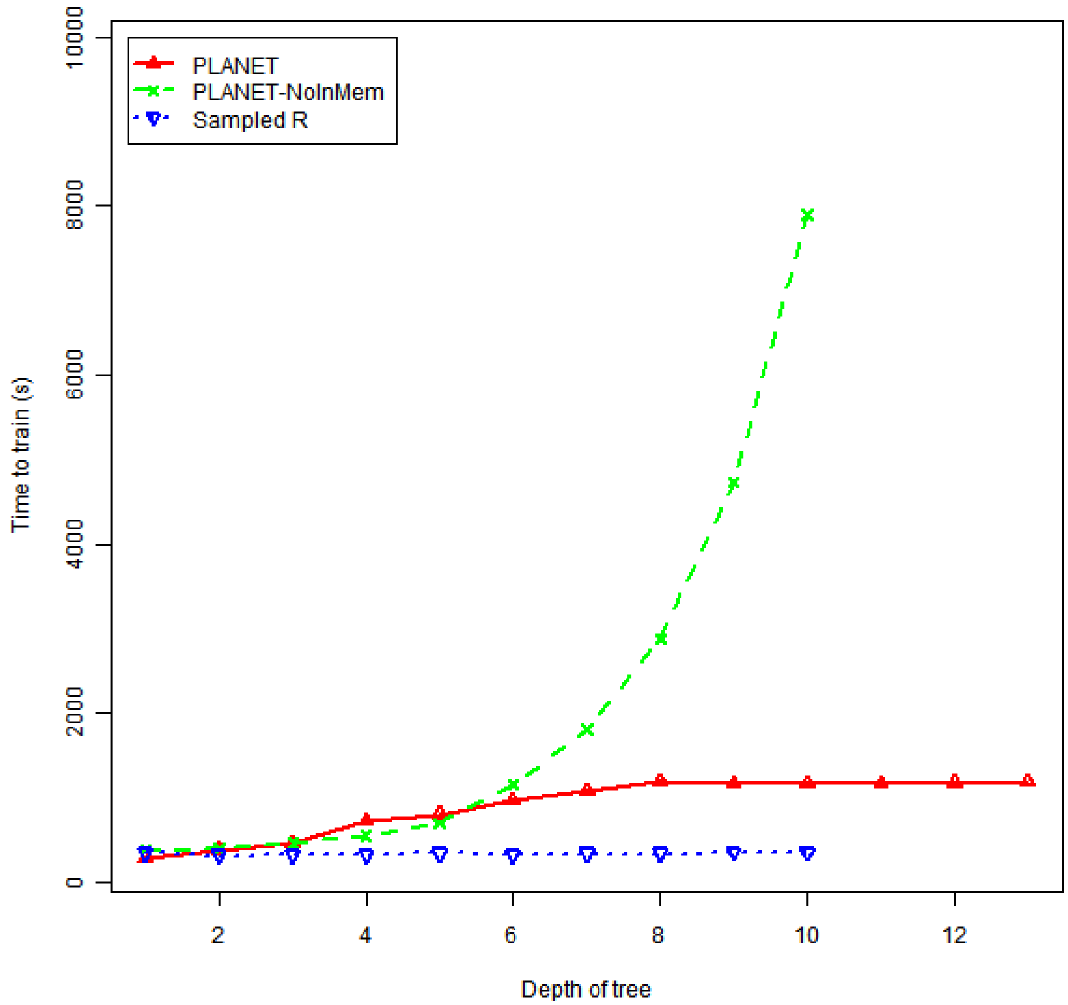
PLANET: Experiments

- Calibrated binary classification (ad bounce rate prediction)
- Features:
  - 4 numeric
  - 6 categorical, $|C| \in [2\text{-}500]$
- 314M instances
- Depth-3 trees



PLANET: Experiments

- Calibrated binary classification (ad bounce rate prediction)
- Features:
  - 4 numeric
  - 6 categorical, $|C| \in [2\text{-}500]$
- 314M instances
- NB: R results on 1/30th of data



Yahoo! GBDT [Y+09]

- **Platform**: Hadoop ~~MapReduce~~ Streaming, MPI
- **Data Partitioning**: horizontal, vertical
  - Vertical partitioning requires a per-instance index on every worker
  - Index aggregates split decisions and residuals for entire dataset
  - Communication $O(n)$ but kept minimal (1 bit + residual)
- Results:
  - MapReduce Horizontal: 211 minutes x 2500 trees = 366 days (100 machines)
  - MapReduce Vertical: 28 seconds x 2500 trees = 19.4 hours (20 machines)
  - MPI: 5 seconds x 2500 trees = 3.4 hours (10 machines)

Distributed LambdaMART [SUML11-Ch8]

- Boosting for Learning to Rank
  - Loss function: $\mathrm{NDCG}@T(q) = \frac{100}{Z}\sum_{r=1}^{T}\frac{2^{l(r)}-1}{\log(1+r)}$
  - Boosting residuals defined via NDCG-based pseudogradient
- **Platform**: MPI on HPC Cluster
- **Binning**: during initialization
- **Data Partitioning**: horizontal and vertical schemes considered

Feature-distributed LambdaMART

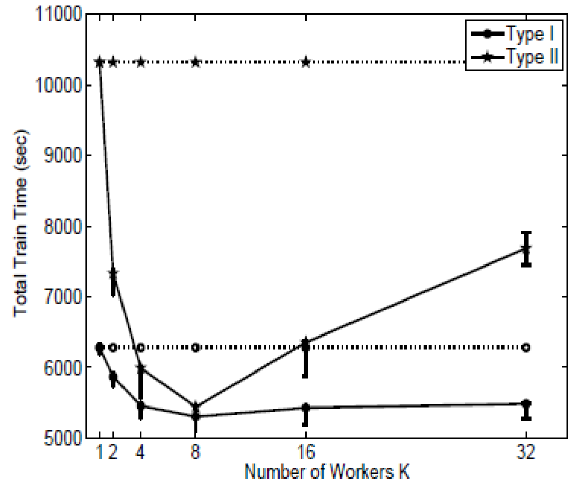
- Each worker stores entire dataset
  - Upper-bound experiment (removes need to communicate per-instance data)
  - With sufficient engineering, 10^{10} instance-features on commodity machines
- No single master, synchronous communication
- Broadcast-based coordination
- Communication cost remains independent of dataset size
  - $O(dK)$ for d features, K workers

Data-distributed LambdaMART

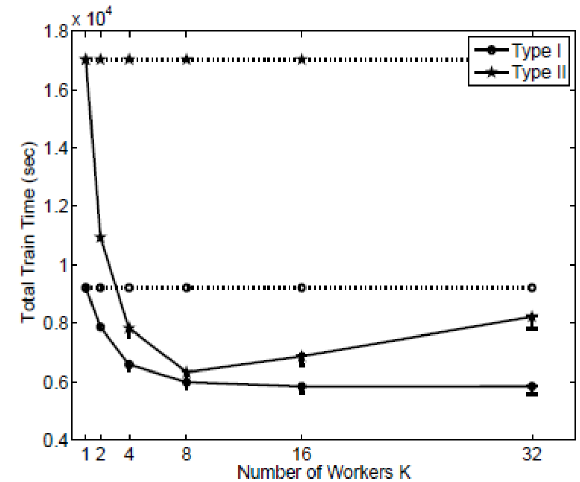
- Evaluate the potential for boosting trees built with sampled data
- Coordinated by master
- Each worker
  - Trains full tree on its partition and broadcasts to all other workers
  - Evaluates other worker's trees on its partition
  - Sends evaluation results to master
- Master selects best tree based on combined evaluation
- Randomized variant: sample worker's hypotheses for evaluation

Feature-Distributed LambdaMART: Results

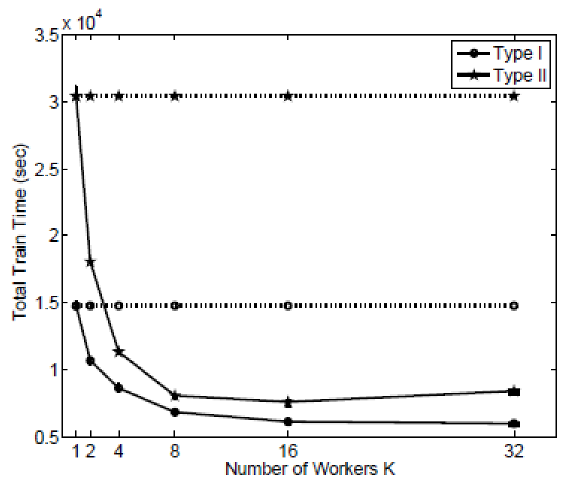
- Ranking (web search)
- 14M query-url pairs
  - 140K queries
- 500-2000 features
- 200-leaf trees
- Communication costs limits gains



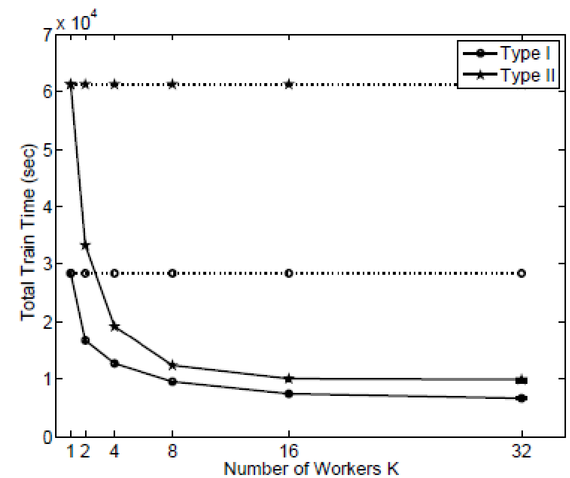
a) 500 Features.



b) 1000 Features.



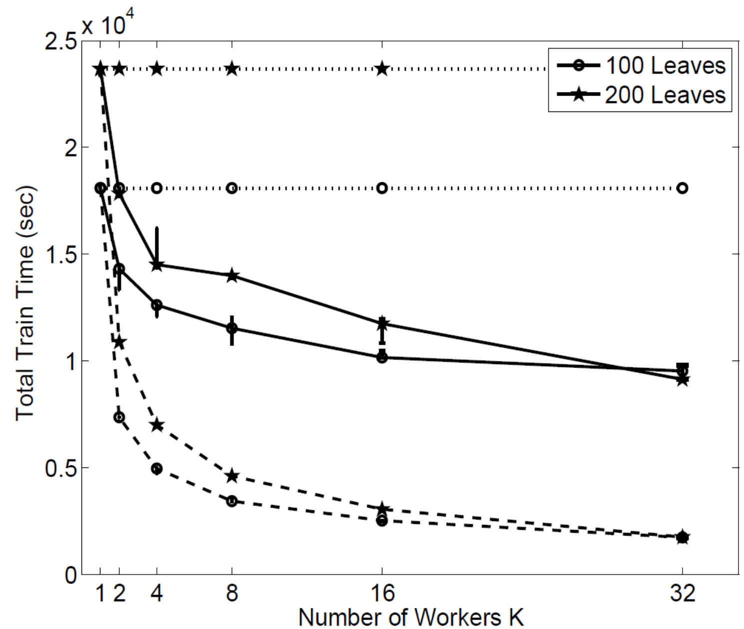
c) 2000 Features.



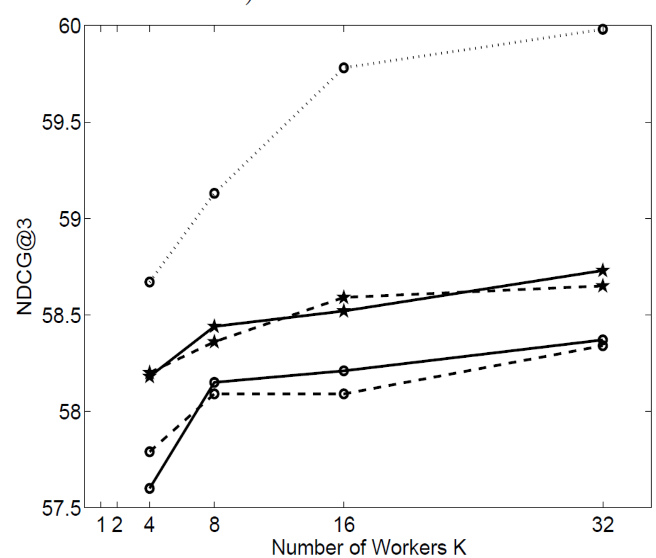
d) 4000 Features.

Distributed LambdaMART: Results

- Ranking (web search)
- 14M query-url pairs,
  - 140K queries
- 500-2000 features
- 200-leaf trees
- Data-distributed:
  - Speedup from sampling comes at accuracy cost



b) NDCG@3.

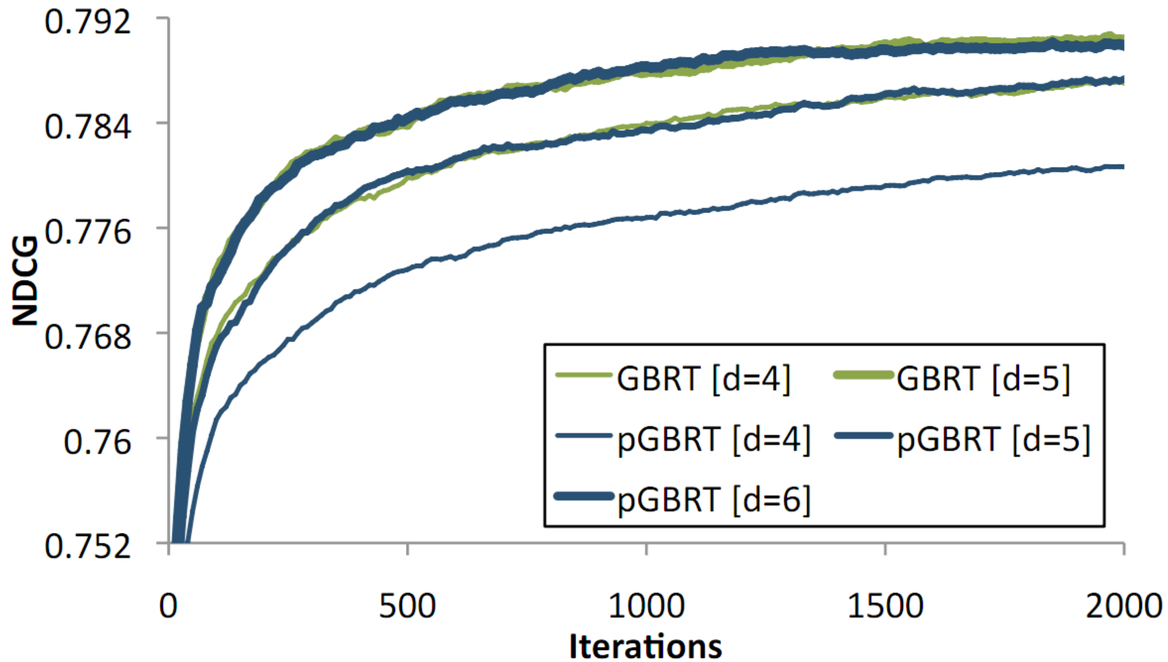


Parallel Boosted Regression Trees [T+11]

- Boosting for Learning to Rank
  - Loss function: squared loss as surrogate for NDCG and ERR
- **Platform**: MPI (multicore and cluster)
- **Data Partitioning**: horizontal
- **Binning**: runtime algorithm [BY10]
  - Each worker constructs initial histogram in a single data pass
  - Split points found by interpolating and approximating uniform-density bins
  - Master aggregation employs same procedure to merge worker histograms
  - Effective alternative: every M iterations re-sample bin centers from data

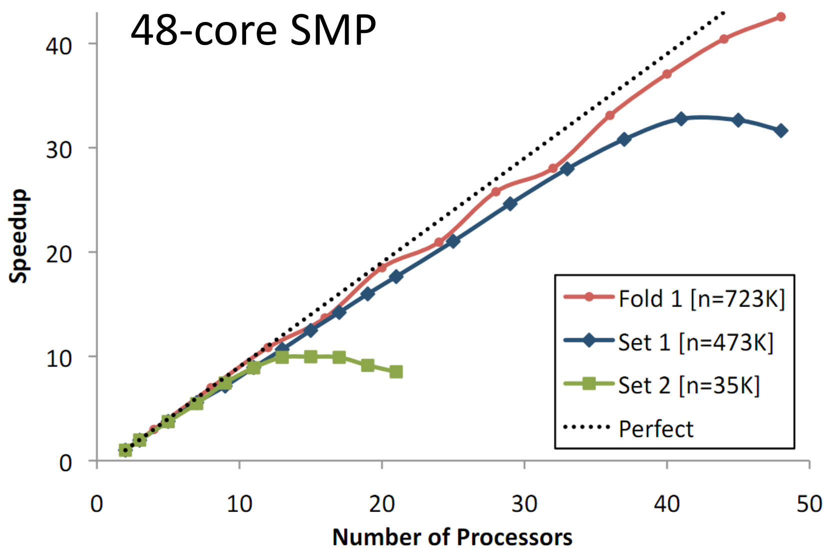
pGBRT Experiments: Accuracy

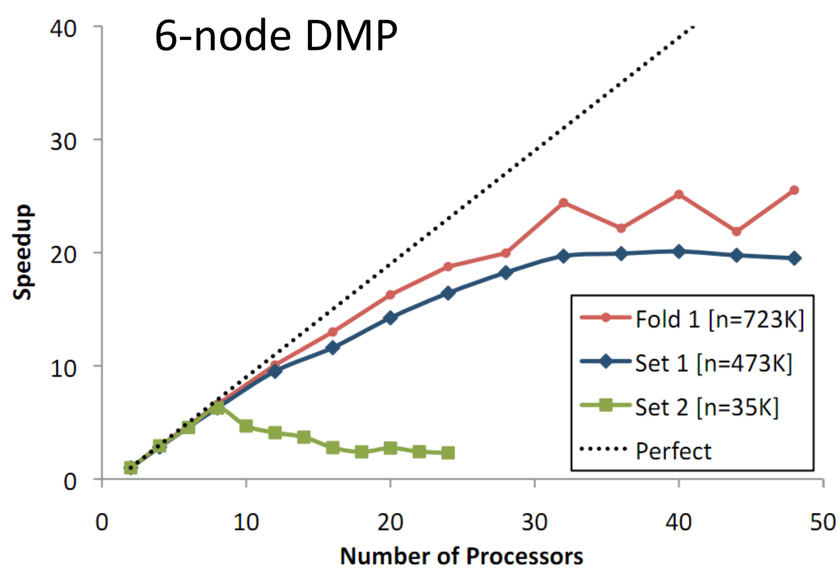
- Yahoo! Learning to Rank Challenge Task I
- 475K documents (20K queries)
- 700 features



pGBRT Experiments: Speedup

- Yahoo! Learning to Rank Challenge dataset
- Network communication limits speedups earlier





Distributed Tree Ensembles: Conclusions

- Boosting can be sped up very effectively on cores and clusters
- MapReduce dataflow is not ideal, overhead is significant
  - Direct communication (MPI/RPC) is required for efficiency
- Feature binning is essential
  - Pre-binning and run-time randomized binning are both effective
- Vertical partitioning is promising but requires delicate engineering
- **Open problem**: cluster-scale parallelization of deeper trees

References

[BY10] Y. Ben-Haim and E. Yom-Tov. A streaming parallel decision tree algorithm. J. of Machine Learning Research, 11:849–872, 2010.

[B96] L. Breiman. Bagging predictors. Machine Learning, 24(2):123–140, 1996.

[B01] L. Breiman. Random forests. Machine Learning, 45(1):5–32, 2001.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1):119-139, 1997.

[F01a] Yoav Freund. An adaptive version of the boost by majority algorithm. Machine Learning, 43(3):293--318, 2001.

[F+03] Yoav Freund, Raj Iyer, Robert Schapire, and Yoram Singer. An Efficient Boosting Algorithm for Combining Preferences. Journal of Machine Learning Research 4: 933-969, 2003.

[F01b] Friedman, J. Greedy function approximation: a gradient boosting machine. Annals of Statistics, 25(5):1189-1232, 2001.

[F+00] Jerome Friedman, Trevor Hastie and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. Annals of Statistics 28(2):337-407, 2000.

[H98] Ho, T. K. The random subspace method for constructing decision forests. IEEE PAMI, 20(8):832–844, 1998.

[P+10] D. Y. Pavlov, A. Gorodilov, and C. A. Brunk. BagBoo: a scalable hybrid bagging-the-boosting model. CIKM-2010.

[S+07] Daria Sorokina, Rich Caruana, Mirek Riedewald. Additive Groves of Regression Trees. ECML-2007.

[SUML11-Ch2] Biswanath Panda, Joshua S. Herbach, Sugato Basu, and Roberto J. Bayardo. MapReduce and its Application to Massively Parallel Learning of Decision Tree Ensembles. In “Scaling Up Machine Learning”, Cambridge U. Press, 2011.

[SUML11-Ch8] Krysta M. Svore and Christopher J.C. Burges. Large-scale Learning to Rank using Boosted Decision Trees. In “Scaling Up Machine Learning”, Cambridge U. Press, 2011.

[SUML11-Ch9] Ramesh Natarajan and Edwin Pednault. The Transform Regression Algorithm. In “Scaling Up Machine Learning”, Cambridge U. Press, 2011.

[T+11] Stephen Tyree, Kilian Q. Weinberger, Kunal Agrawal. Parallel Boosted Regression Trees for Web Search Ranking. WWW-2011.

[Y+09] Jerry Ye, Jyh-Herng Chow, Jiang Chen, Zhaohui Zheng. Stochastic Gradient Boosted Distributed Decision Trees. CIKM-2009.

[Z+08] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun. A general boosting method and its application to learning ranking functions for web search. NIPS 2008.