Tree Computation for Ranking and Classification

CS240A, T. Yang, 2016

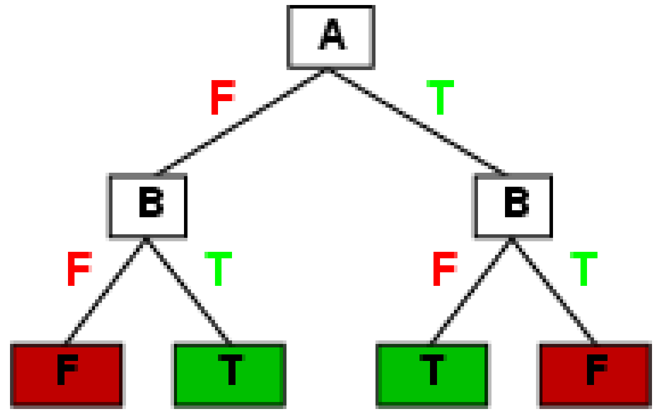
Outlines

- **Decision Trees**
- **Learning Assembles:**
  - Random forest, boosted trees

Decision Trees

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless *f* nondeterministic in *x*) but it probably won't generalize to new examples

- Prefer to find more compact decision trees: we don't want to memorize the data, we want to find structure in the data!

Decision Trees: Application Example

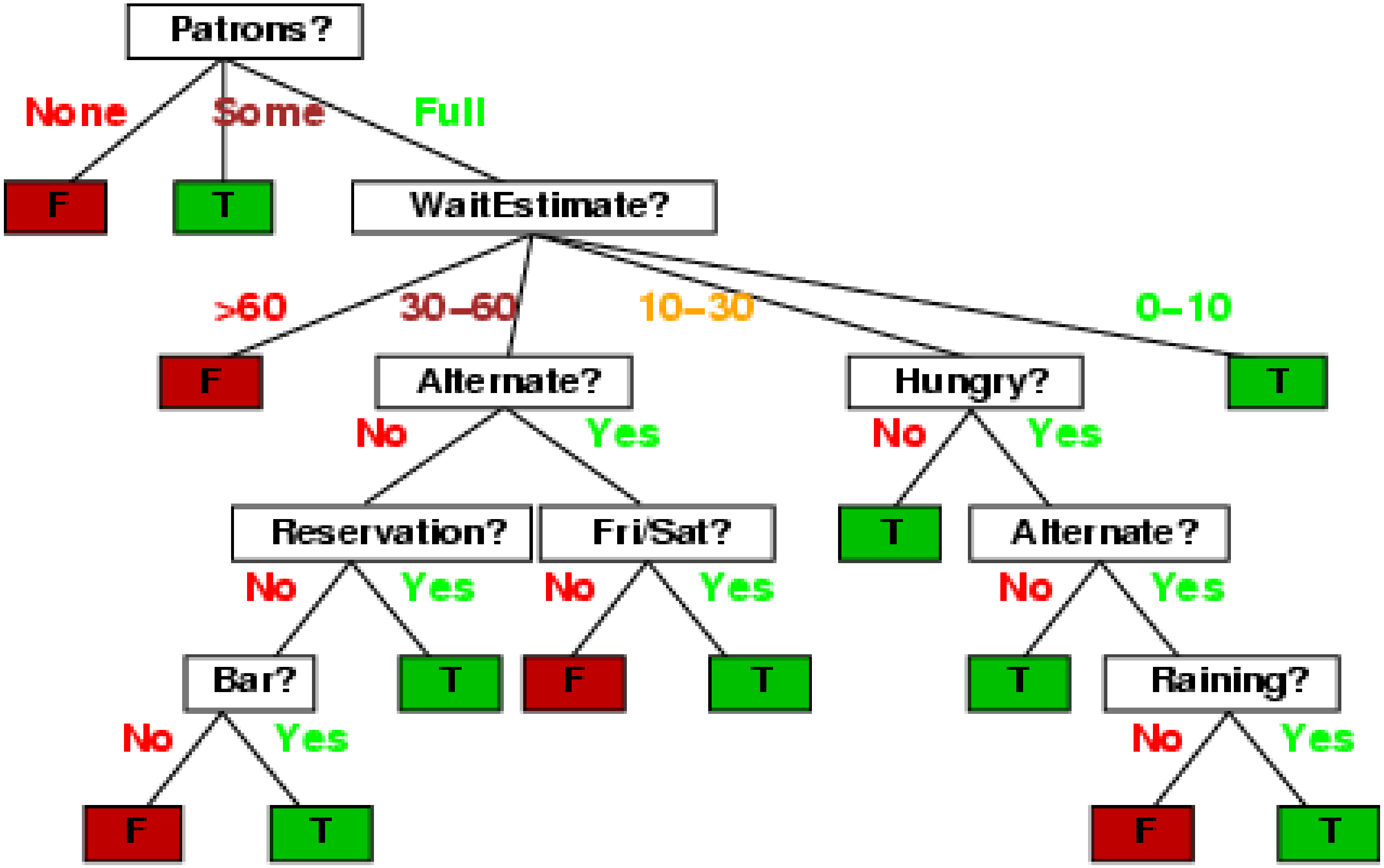
Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

UCSB

A decision tree to decide whether to wait

- **imagine someone talking a sequence of decisions.**



UCSB

Training data: Restaurant example

- **Examples described by attribute values (Boolean, discrete, continuous)**
- **E.g., situations where I will/won't wait for a table:**

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

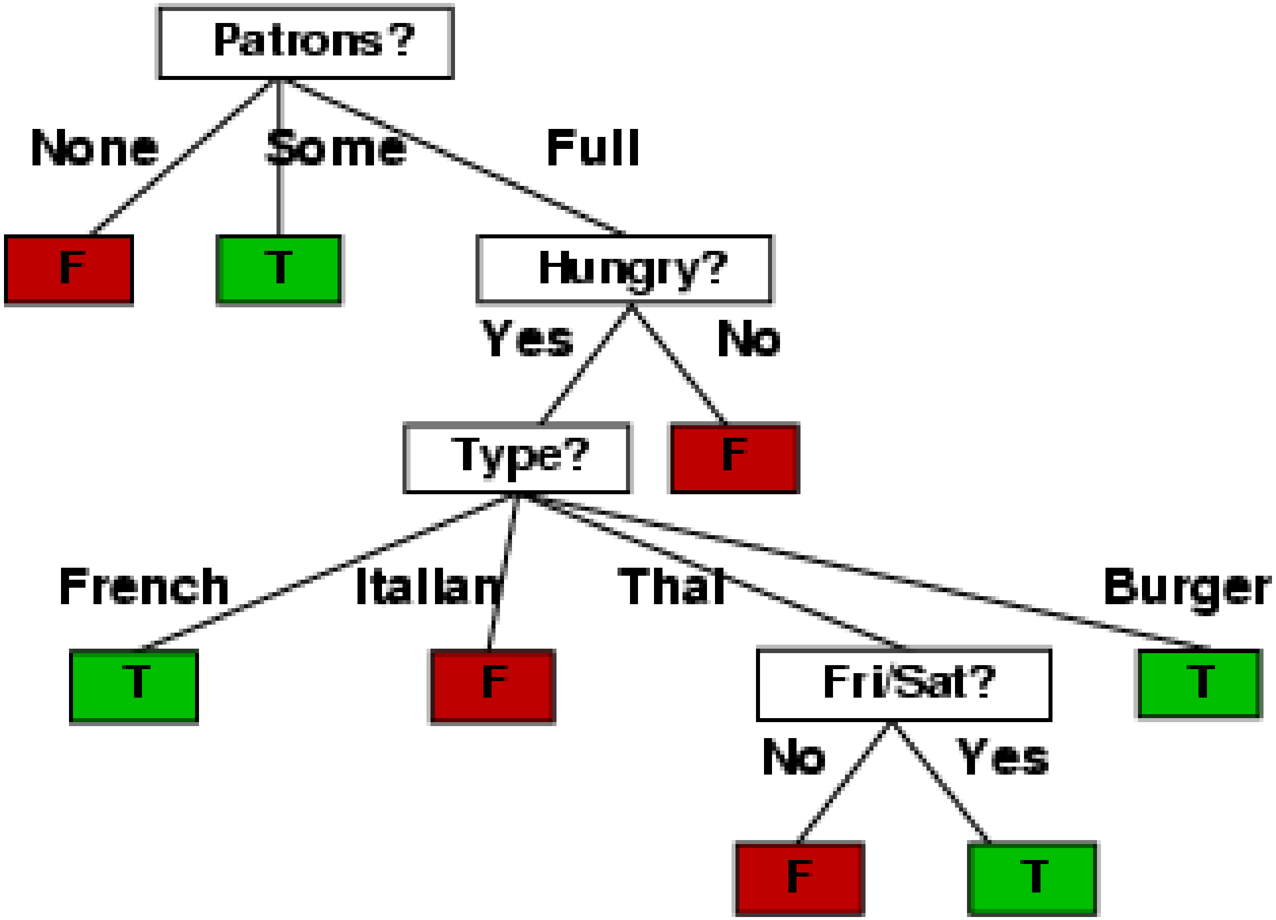
- **Classification of examples is positive (T) or negative (F)**

Decision tree learning

- **If there are so many possible trees, can we actually search this space? (solution: greedy search).**
- **Aim: find a small tree consistent with the training examples**
- **Idea: (recursively) choose "most significant" attribute as root of (sub)tree.**

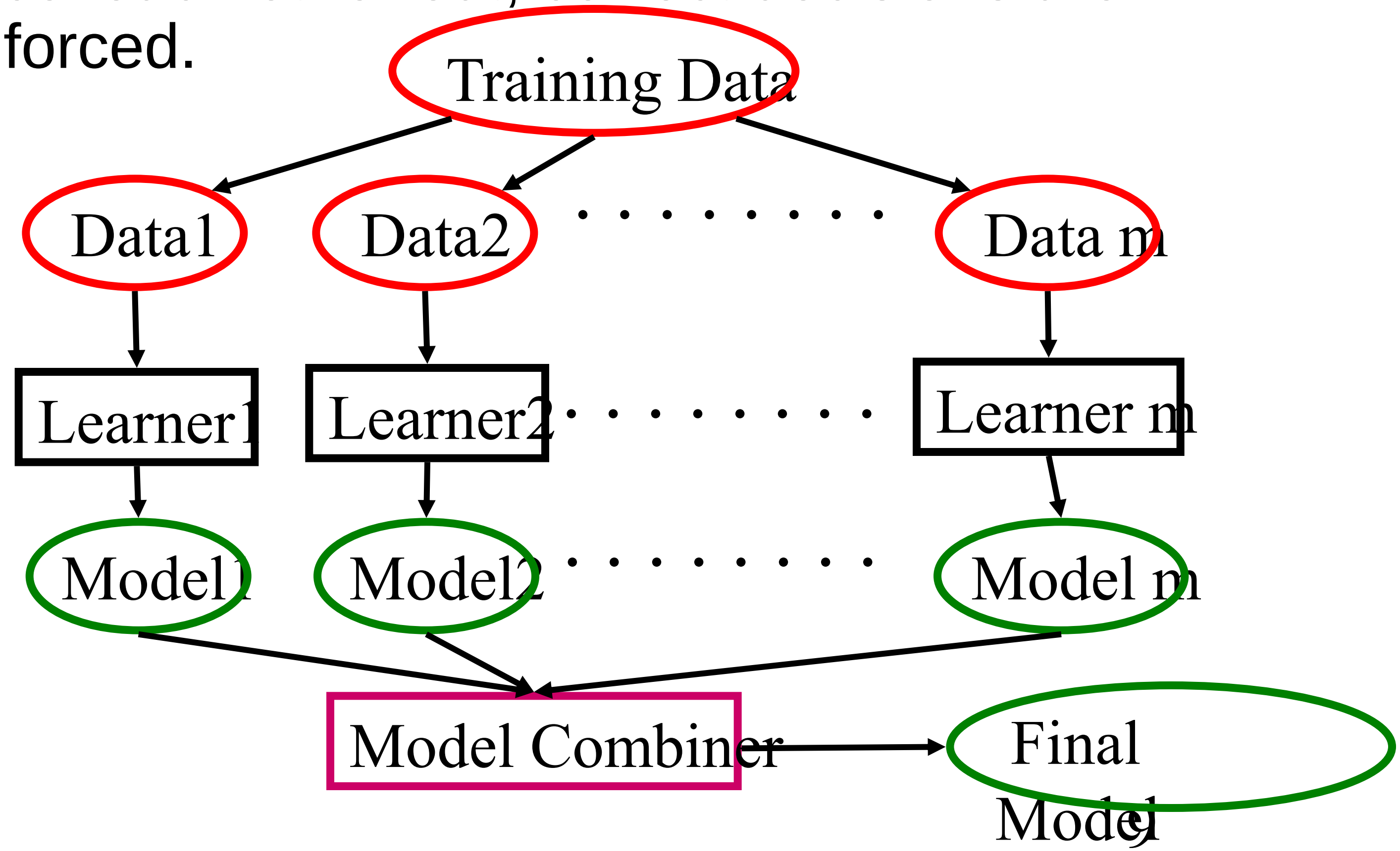
Example: Decision tree learned

- **Decision tree learned from the 12 examples:**



Learning Ensembles

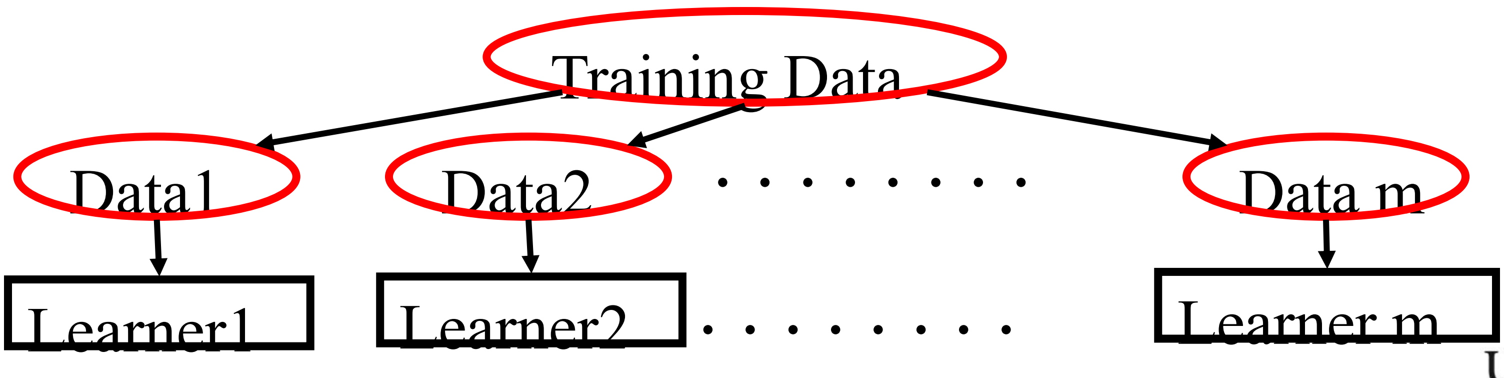
- **Learn multiple classifiers separately**
- **Combine decisions (e.g. using weighted voting)**
- When combing multiple decisions, random errors cancel each other out, correct decisions are reinforced.



UCSB

Homogenous Ensembles

- **Use a single, arbitrary learning algorithm but manipulate training data to make it learn multiple models.**
  - Data1 ≠ Data2 ≠ … ≠ Data m
  - Learner1 = Learner2 = … = Learner m
- **Methods for changing training data:**
  - Bagging: Resample training data
  - Boosting: Reweight training data
  - DECORATE: Add additional artificial training data



Bagging

- Create ensembles by repeatedly randomly resampling the training data (Brieman, 1996).
- Given a training set of size *n*, create *m* sample sets
  - Each *bootstrap sample set* will on average contain 63.2% of the unique training examples, the rest are replicates.
- Combine the *m* resulting models using majority vote.

- Decreases error by decreasing the variance in the results due to *unstable learners*, algorithms (like decision trees) whose output can change dramatically when the training data is slightly changed.

Random Forests

- **Introduce two sources of randomness: “Bagging” and “Random input vectors”**
  - Each tree is grown using a bootstrap sample of training data
  - At each node, best split is chosen from random sample of *m* variables instead of all variables M.
- m is held constant during the forest growing
- Each tree is grown to the largest extent possible
- Bagging using decision trees is a special case of random forests when m=M

UCSB

Random Forests

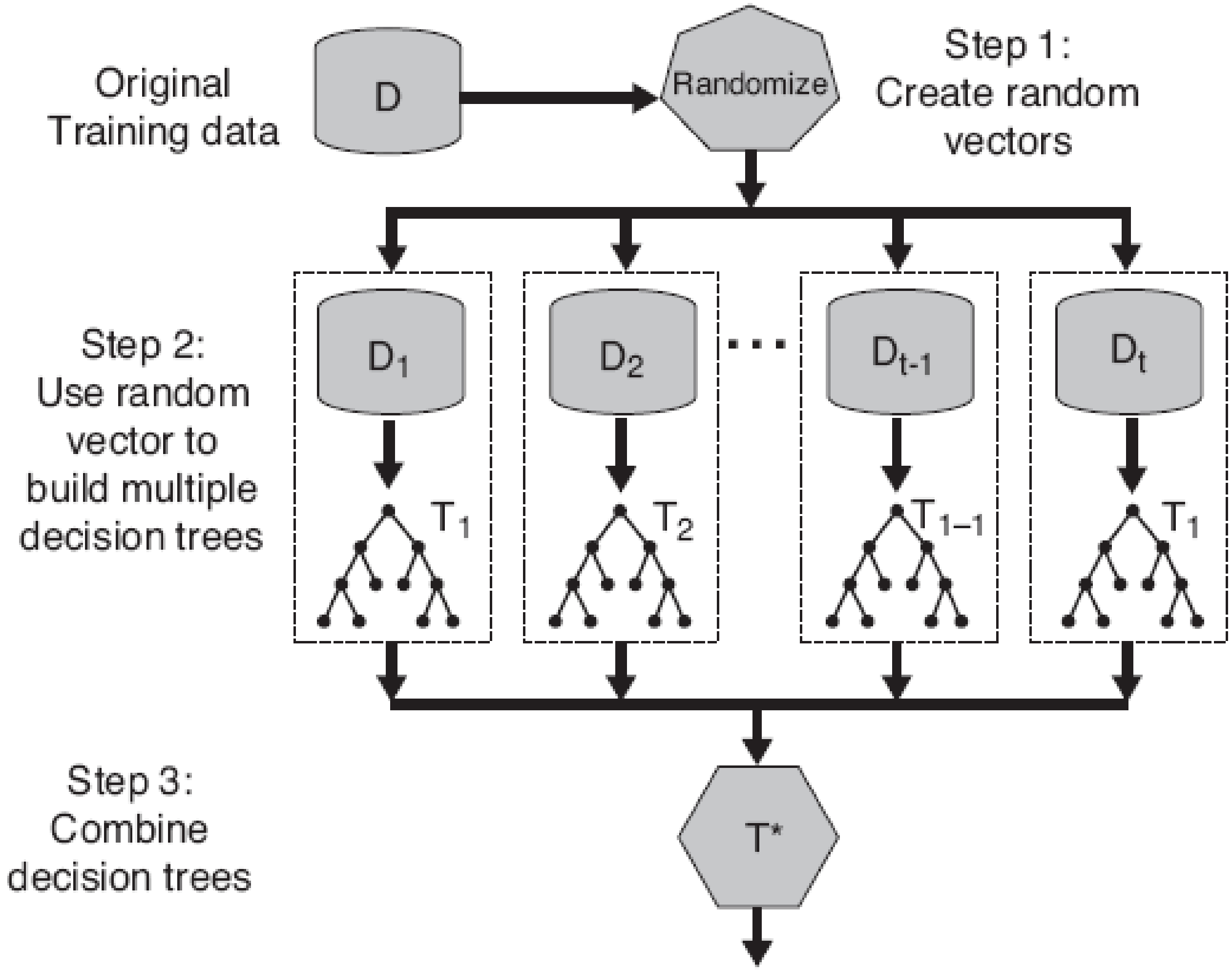


Figure 5.40. Random forests.

Random Forest Algorithm

- Good accuracy without over-fitting
- Fast algorithm (can be faster than growing/pruning a single tree); easily parallelized
- Handle high dimensional data without much problem

Boosting: AdaBoost

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, August 1997.

- Simple with theoretical foundation

Adaboost - Adaptive Boosting

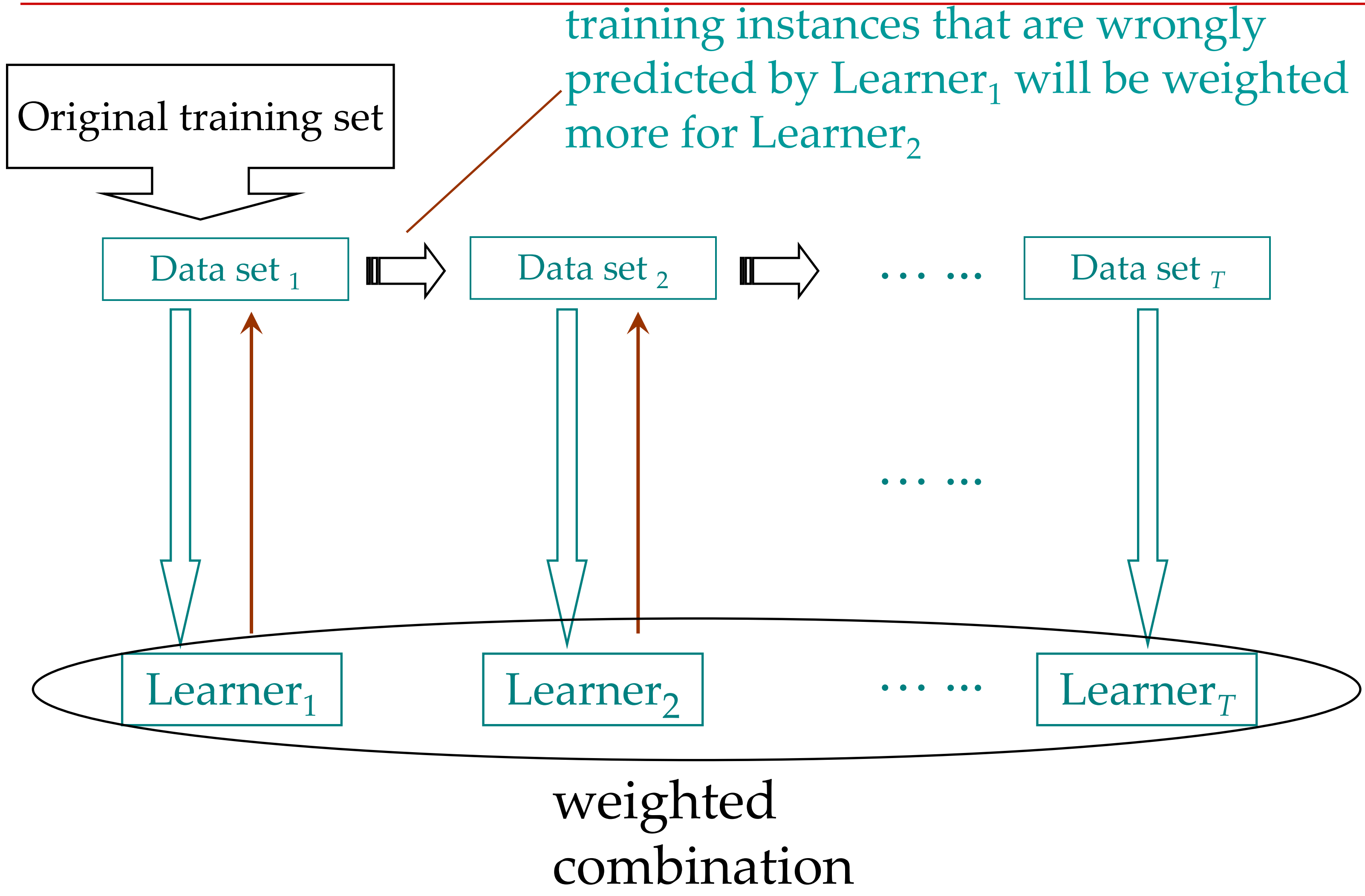
- **Use training set re-weighting**
  - Each training sample uses a weight to determine the probability of being selected for a training set.

- **AdaBoost is an algorithm for constructing a “strong” classifier as linear combination of “simple” “weak” classifier**

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

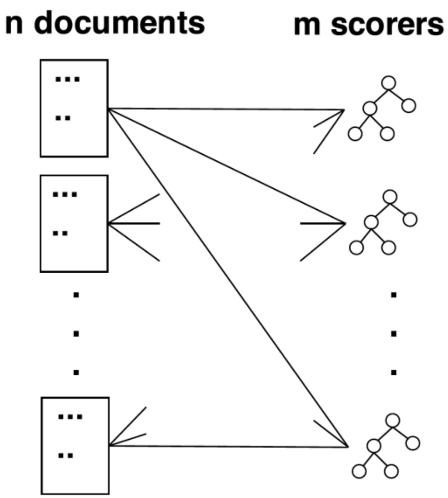
- **Final classification based on weighted sum of weak classifiers**

AdaBoost: An Easy Flow



Cache-Conscious Runtime Optimization for Ranking Ensembles

- **Challenge in query processing**
  - Fast ranking score computation without accuracy loss in multi-tree ensemble models



- **Xun et. al [SIGIR2014]**
  - Investigate data traversal methods for fast score calculation with large multi-tree ensemble models
  - Propose a 2D blocking scheme for better cache utilization with simple code structure

UCSB

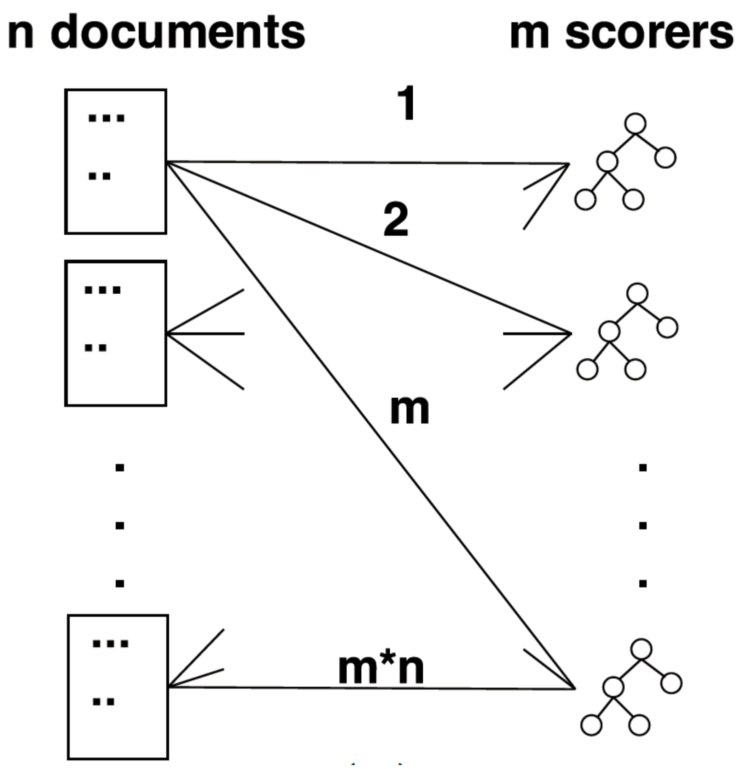
Motivation

- **Ranking assembles are effective in web search and other data applications**
  - E.g. Gradient boosted regression trees (GBRT)
- **A large number of trees are used to improve accuracy**
  - Winning teams at Yahoo! Learning-to-rank challenge used ensembles with 2k to 20k trees, or even 300k trees with bagging methods
- **Time consuming for computing large ensembles**
  - Access of irregular document attributes impairs CPU cache reuse
    - Unorchestrated slow memory access incurs significant cost
    - Memory access latency is 200x slower than L1 cache
  - Dynamic tree branching impairs instruction branch prediction

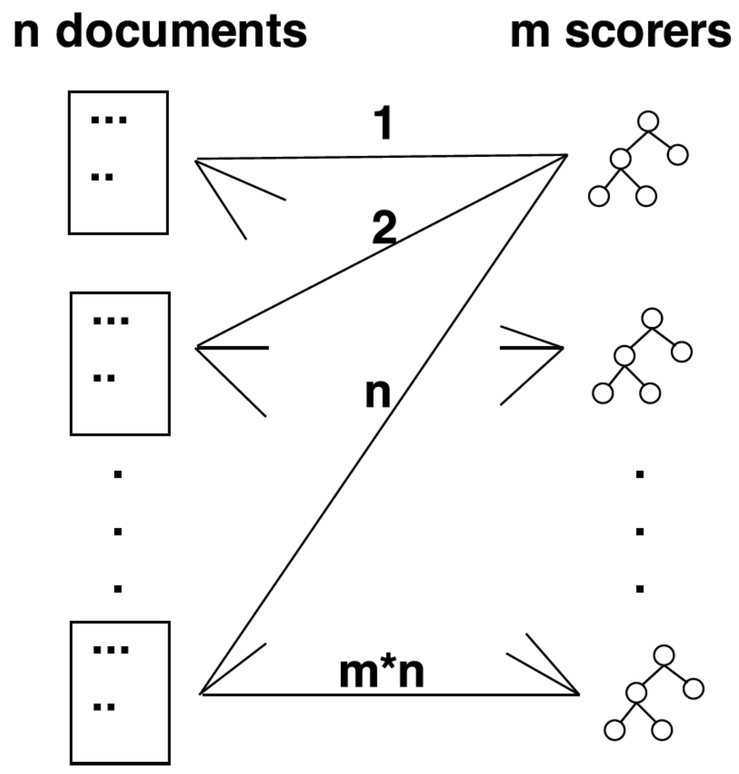
UCSB

Key Idea: Optimize Data Traversal

Two existing solutions:

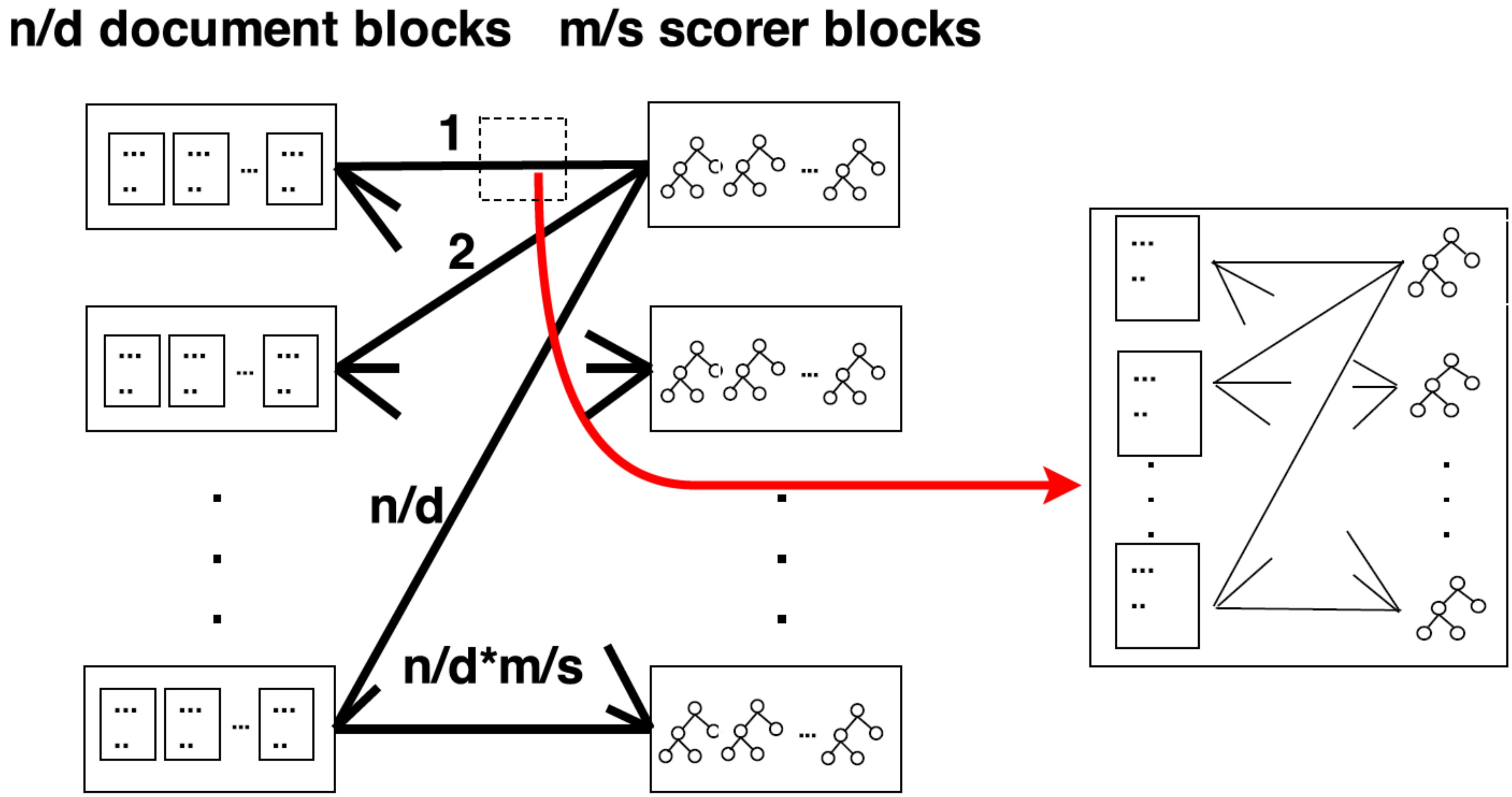


Document-ordered Traversal (DOT)



Scorer-ordered Traversal (SOT)

Our Proposal: 2D Block Traversal



UCSB

Algorithm Pseudo Code

Algorithm 2: 2D blocking with SDSD structure.

for $j = 0$ *to* $\frac{m}{s} - 1$ **do**
    for $i = 0$ *to* $\frac{n}{d} - 1$ **do**
        for $jj = 1$ *to* s **do**
            for $ii = 1$ *to* d **do**
                Compute subscore for document $i \times d + ii$ with tree $j \times s + jj$.
                Update the score of this document.

Why Better?

- **Total slow memory accesses in score calculation**

DOT	SOT	2D Block
$O(m \times n + m)$	$O(m \times n + n)$	$O(m + \frac{m \times n}{s})$

  - 2D block can be up to *s* time faster. But *s* is capped by cache size

- **2D Block** fully exploits cache capacity for better temporal locality
- **Block-VPred**: a combined solution that applies 2D Blocking on top of VPred [Asadi et al. TKDE'13]
  - 159 lines of code vs VPred 22,651 lines for tree depth 51

Evaluations

- **2D Block and Block-VPred implemented in C**
  - Compiled with GCC using optimization flag -O3
  - Tree ensembles derived by *jforests* [Ganjisaffar et al. SIGIR'11] using *LambdaMART* [Burges et al. JMLR'11]
- **Experiment platforms**
  - 3.1GHz 8-core AMD Bulldozer FX8120 processors
  - Intel X5650 2.66GHz 6-core dual processors
- **Benchmarks**
  - Yahoo! Learning-to-rank, MSLR-30K, and MQ2007
- **Metrics**
  - Scoring time
  - Cache miss ratios and branch misprediction ratios reported by Linux *perf* tool

UCSB

Scoring Time per Document per Tree in Nanoseconds

Dataset	Leaves	m	n	DOT	SOT	VPred $[v]$
Yahoo!	50	7,870	5,000	186.0	113.8	47.4 [8]
	150	8,051	2,000	377.8	150.2	123.0 [8]
	400	2,898	5,000	312.3	223.8	136.2 [8]
MSLR-30K	50	1,647	5,000	88.3	41.4	32.6 [8]
MQ2007	50	9,870	10,000	1.79	1.66	2.02 [8]
	200	10,103	10,000	204.1	30.3	43.1 [32]

- **Query latency = Scoring time** * *n* * *m*
  - *n* docs ranked with an *m*-tree model

Query Latency in Seconds

2D blocking [s, d]	Block-VPred [s, d, v]	Latency
36.4 [300, 300]	36.7 [300, 320, 8]	1.43
81.9 [100, 400]	76.1 [100, 480, 8]	1.23
90.9 [100, 400]	86.0 [100, 400, 8]	1.25
26.6 [500, 1,000]	31.1 [500, 1,600, 8]	0.22
1.51 [300, 5,000]	1.94 [300, 5,000, 8]	0.15
28.3 [100, 10,000]	26.2 [100, 5,000, 32]	2.65

Fastest algorithm is marked in gray.

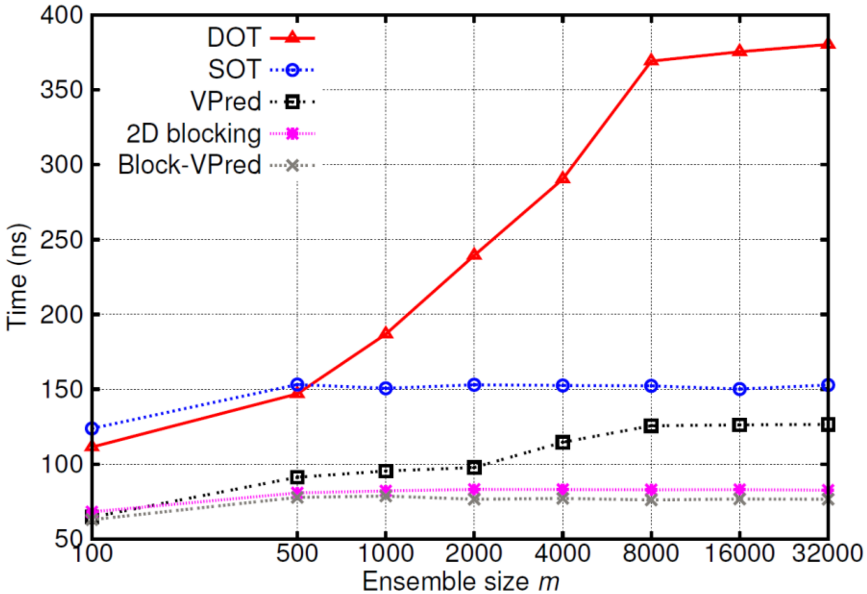
2D blocking

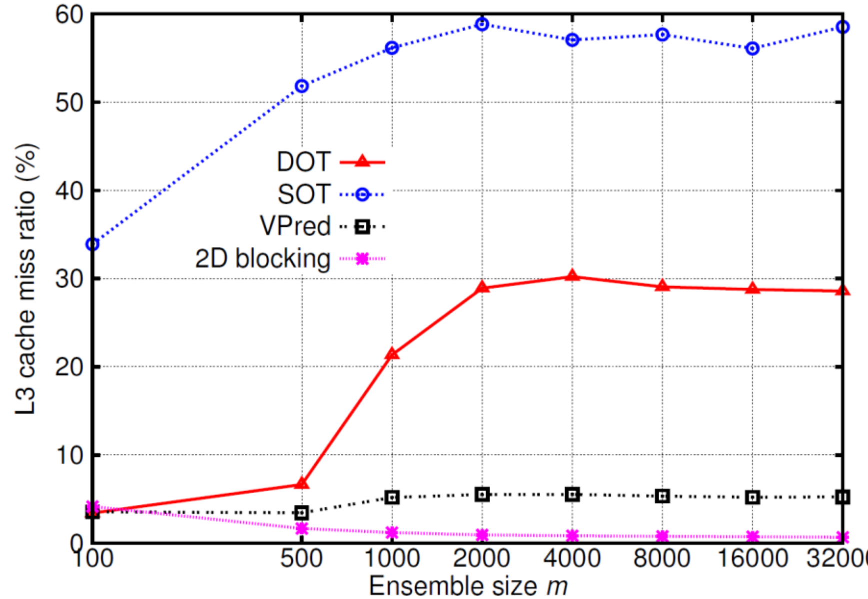
- Up to 620% faster than DOT
- Up to 213% faster than SOT
- Up to 50% faster than VPred

Block-VPred

- Up to 100% faster than VPred
- Faster than 2D blocking in some cases

Time & Cache Perf. as Ensemble Size Varies





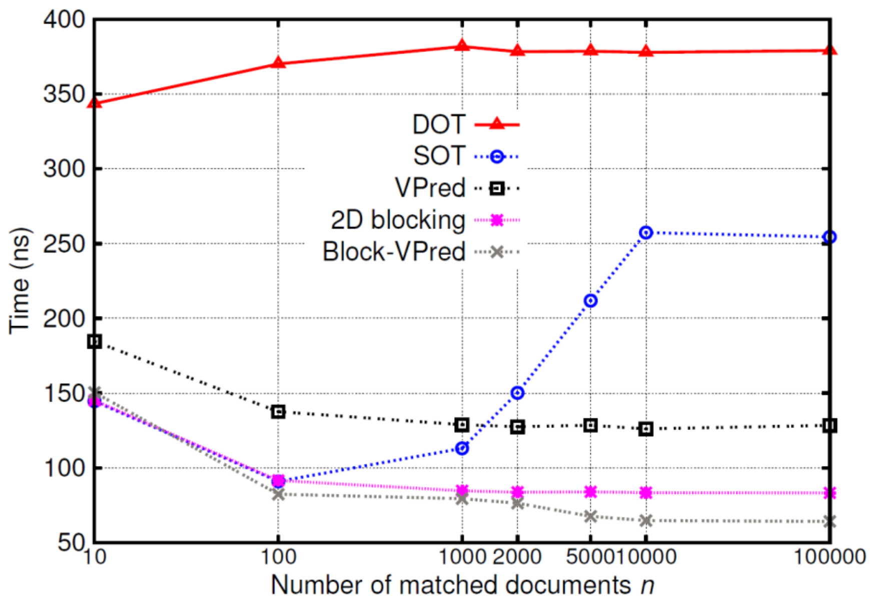
- 2D blocking is up to 287% faster than DOT
- Time & cache perf. are highly correlated
- Change of ensemble size affects DOT the most

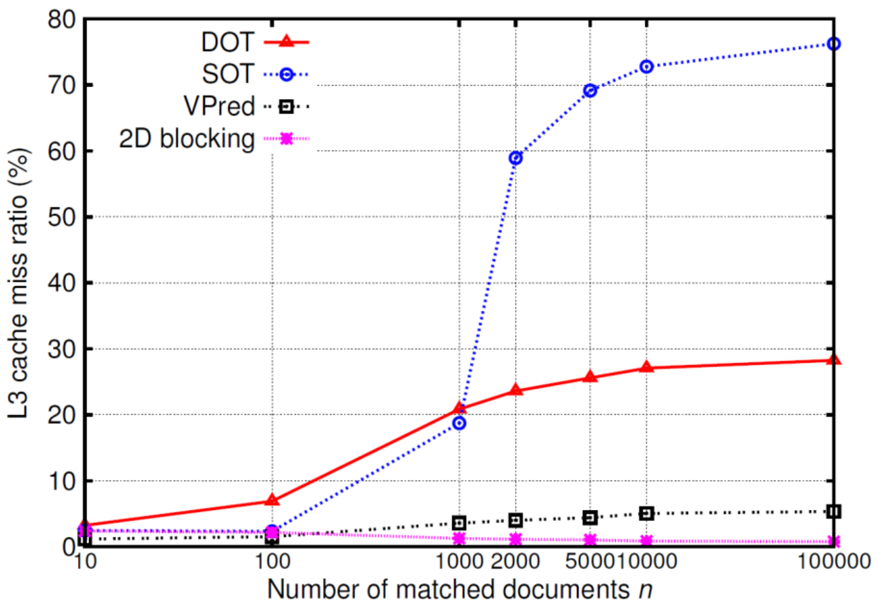
UCSB

Concluding remarks

- **2D blocking data traversal method for fast score calculation with large multi-tree ensemble models**
  - better cache utilization with simple code structure
- When multi-tree score calculation per query is parallelized to reduce latency, 2D blocking still maintains its advantage
- For small *n*, multiple queries could be combined to fully exploit cache capacity.
  - Combining leads to 48.7% time reduction with Yahoo! 150-leaf 8,051-tree ensemble when *n*=10.
- Future work
  - Extend to non-tree ensembles by iteratively selecting a fixed number of base rank models that fit in fast cache

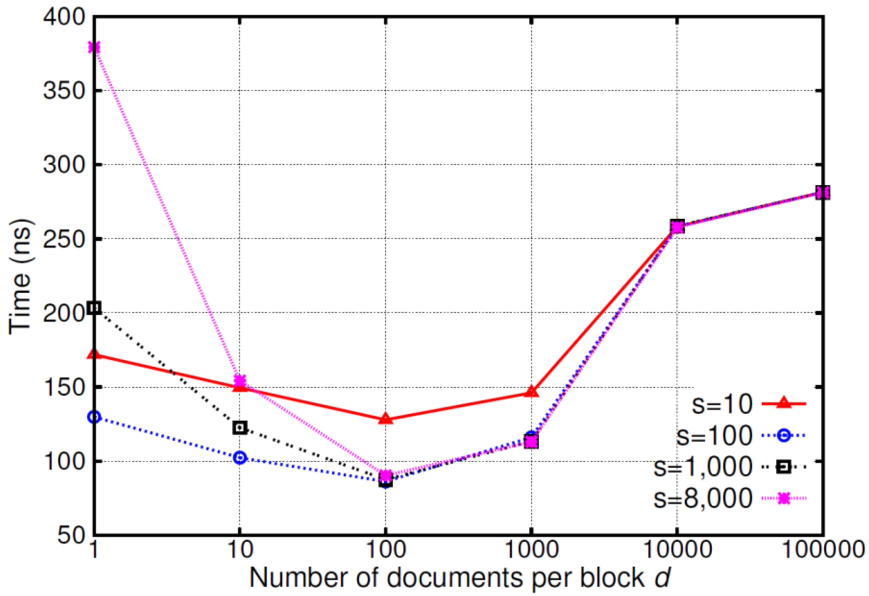
Time & Cache Perf. as No. of Doc Varies

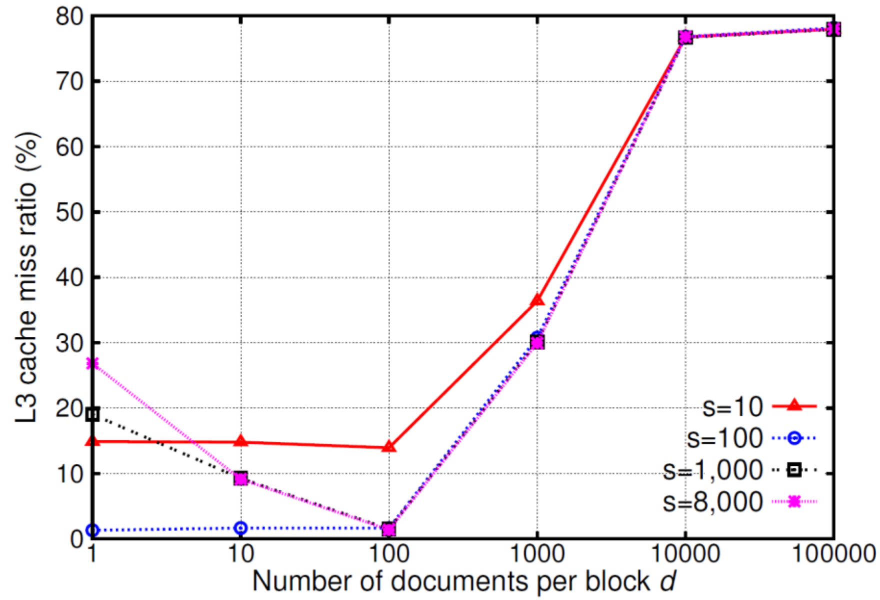




- 2D blocking is up to 209% faster than SOT
- Block-VPred is up to 297% faster than SOT
- SOT deteriorates the most when number of doc grows
- 2D combines the advantage of both DOT and SOT

2D Blocking:
Time & Cache Perf. as Block Size Vary





- The fastest scoring time and lowest L3 cache miss ratio are achieved with block size *s*=1,000 and *d*=100 when these trees and documents fit in cache
- Scoring time could be 3.3x slower if block size is not chosen properly

UCSB

Impact of Branch Misprediction Ratios

MQ2007 Dataset	DOT	SOT	VPred	2D Block	Block-VPred
50-leaf tree	1.9%	3.0%	1.1%	2.9%	0.9%
200-leaf tree	6.5%	4.2%	1.2%	9.0%	1.1%

Yahoo! Dataset	*n*=1,000	*n*=5,000	*n*=10,000	*n*=100,000
2D Block	1.9%	2.7%	4.3%	6.1%
Block-VPred	1.1%	0.9%	0.84%	0.44%

- For larger trees or larger no. of documents
  - Branch misprediction impacts more
  - Block-VPred outperforms 2D Block with less misprediction and faster scoring

UCSB