Logarithmic Time Prediction

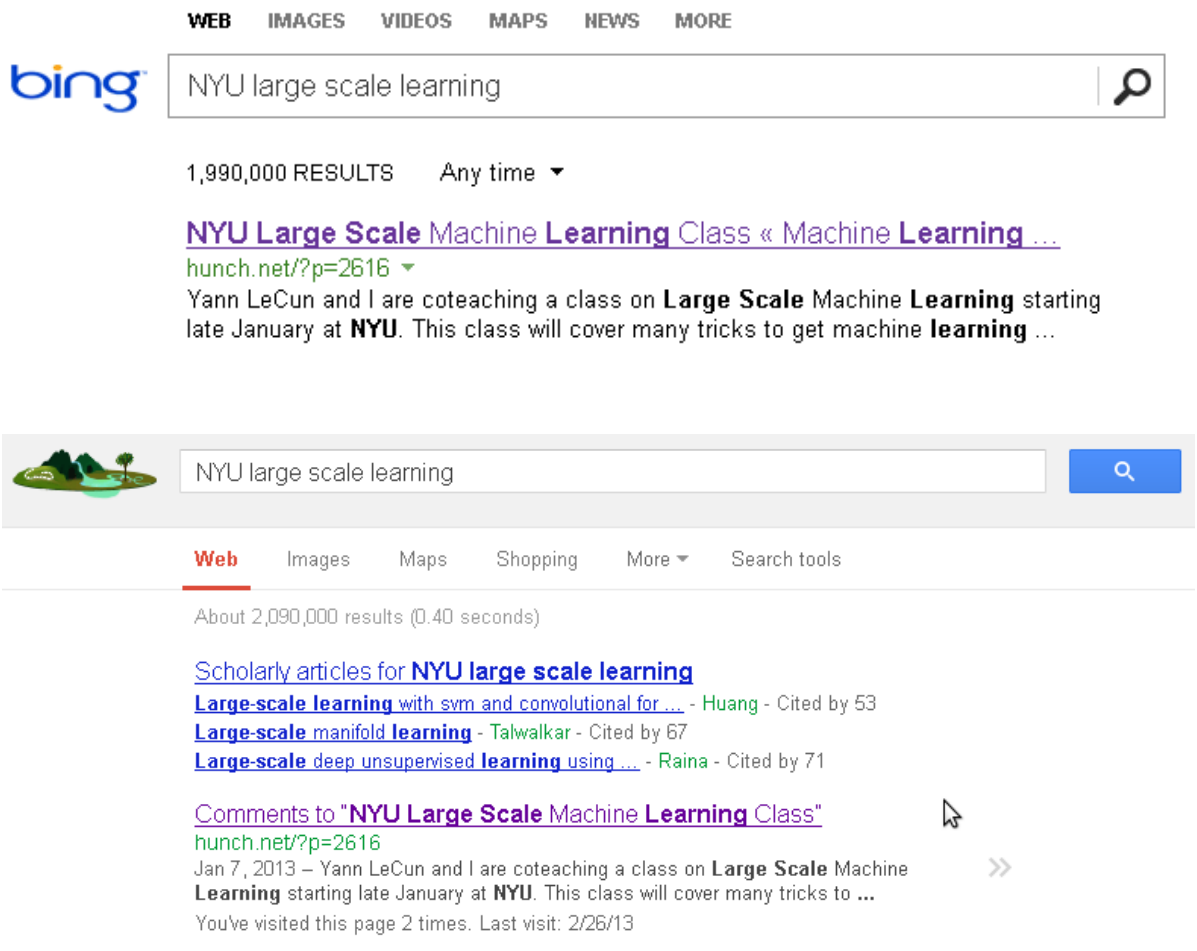
John Langford @ Microsoft Research

(With help from Alina Beygelzimer & Bianca Zadrozny)

May 7, 2013

A Scenario

You have 10^{10} webpages and want to return the best result in 100ms.



How do you do it?

Method 1: Indexing algorithms

1. Inverted Index
2. Weighted And (WAND)
3. Locality Sensitive Hashing
4. Predictive Indexing and other Data-driven variations of the above

See previous lecture on indexing.

Method 2: Logarithmic time prediction

Typical time to choose one of k things with ML algorithms: $O(k)$.

Minimum possible time to choose one of k things: $O(\log k)$.

Method 2: Logarithmic time prediction

Typical time to choose one of k things with ML algorithms: $O(k)$.

Minimum possible time to choose one of k things: $O(\log k)$.

Should we make algorithms which run in $O(\log k)$ time?

Method 2: Logarithmic time prediction

Typical time to choose one of k things with ML algorithms: $O(k)$.

Minimum possible time to choose one of k things: $O(\log k)$.

Should we make algorithms which run in $O(\log k)$ time?

How can we make algorithms which run in $O(\log k)$ time?

Why logarithmic time prediction?

Hypothesis: Indexing + scoring on top candidates is unsound.
Evidence:

1. If you apply a learned quality predictor to all results (on a small subset of queries) you get different results.
2. Machine Learning (well) is hard. Indexing (well) is hard. Getting the two to work together (well) is hard. Maybe a master algorithm can do both better?
3. Common story line for an ML paper: We figured out how to do end-to-end learning, and it worked better.

How do we do logarithmic time prediction?

Approach 1: Build a decision tree with $O(\log k)$ depth.

Difficulties:

How do we do logarithmic time prediction?

Approach 1: Build a decision tree with $O(\log k)$ depth.

Difficulties:

1. Decision trees are slow to train and much data is required.
2. Decision trees alone tend to provide poor performance.

Solution:

How do we do logarithmic time prediction?

Approach 1: Build a decision tree with $O(\log k)$ depth.

Difficulties:

1. Decision trees are slow to train and much data is required.
2. Decision trees alone tend to provide poor performance.

Solution: Use map-reduce with 1000 nodes to train a decision forest.

How do we do logarithmic time prediction?

Approach 1: Build a decision tree with $O(\log k)$ depth.

Difficulties:

1. Decision trees are slow to train and much data is required.
2. Decision trees alone tend to provide poor performance.

Solution: Use map-reduce with 1000 nodes to train a decision forest.

It works! And beats an index+score approach. See WWW 2013 paper.

How do we do logarithmic time prediction?

Approach 2: Reuse existing algorithms multiple times in a log-depth structure.

How do we do logarithmic time prediction?

Approach 2: Reuse existing algorithms multiple times in a log-depth structure.
Existing algorithms solve:

Binary Classification

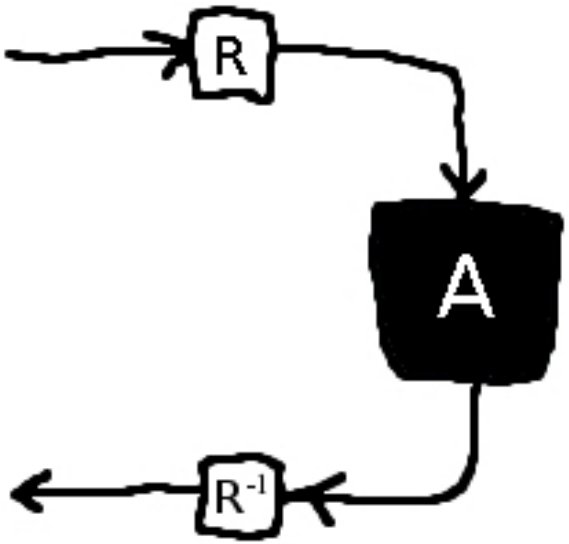
- Given training data $\{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$, produce a classifier $h : X \to \{0, 1\}$.
- Unknown underlying distribution D over $X \times \{0, 1\}$.
- Find h with small 0-1 loss:

$$\ell_{0/1}(h, D) = \mathbf{Pr}_{(x,y)\sim D}[h(x) \neq y]$$

Years of research and development of algorithms for solving this problem. Can we reuse it?

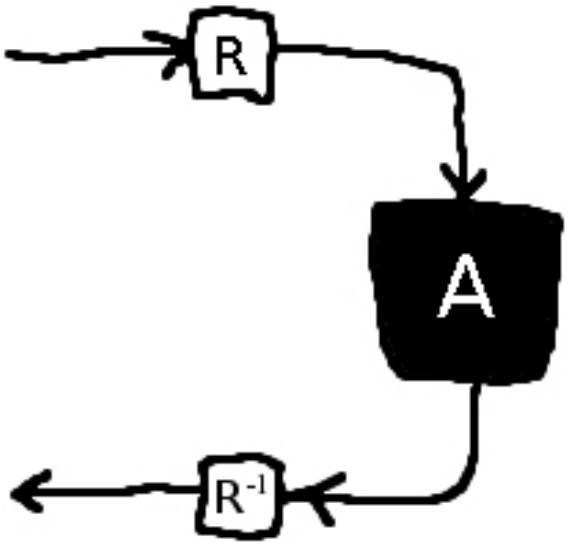
Learning Reductions

Goal: minimize ℓ on D



Learning Reductions

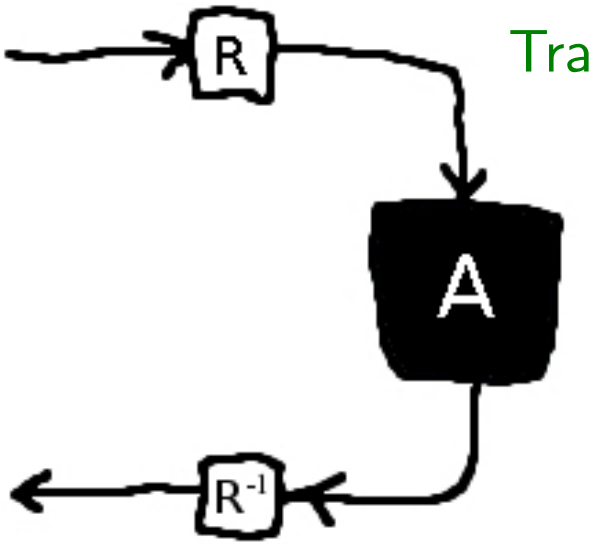
Goal: minimize ℓ on D



Algorithm for optimizing $\ell_{0/1}$

Learning Reductions

Goal: minimize ℓ on D

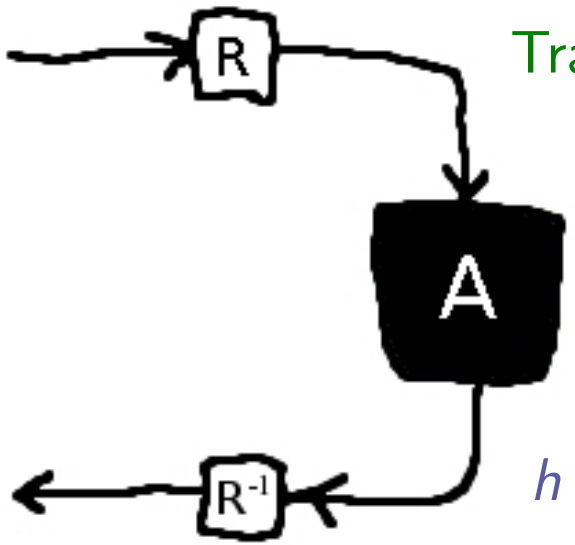


Transform D into D_R

Algorithm for optimizing $\ell_{0/1}$

Learning Reductions

Goal: minimize ℓ on D



Transform D into D_R

Algorithm for optimizing $\ell_{0/1}$

h

Transform h with small $\ell_{0/1}(h, D_R)$ into R_h with small $\ell(R_h, D)$.

such that if h does well on $(D_R, \ell_{0,1})$, R_h is guaranteed to do well on (D, ℓ).

Let's begin with ...

Multi-class classification

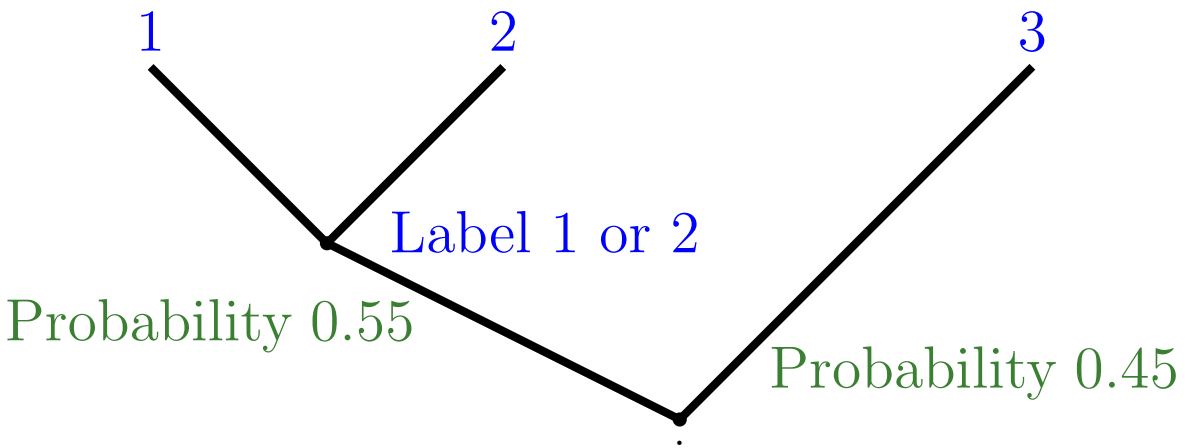
Distribution D over $X \times Y$, where $Y = \{1, \ldots, k\}$.

Find a classifier $h : X \to Y$ minimizing the multi-class loss on D

$$\ell_k(h, D) = \mathbf{Pr}_{(x,y)\sim D}[h(x) \neq y]$$

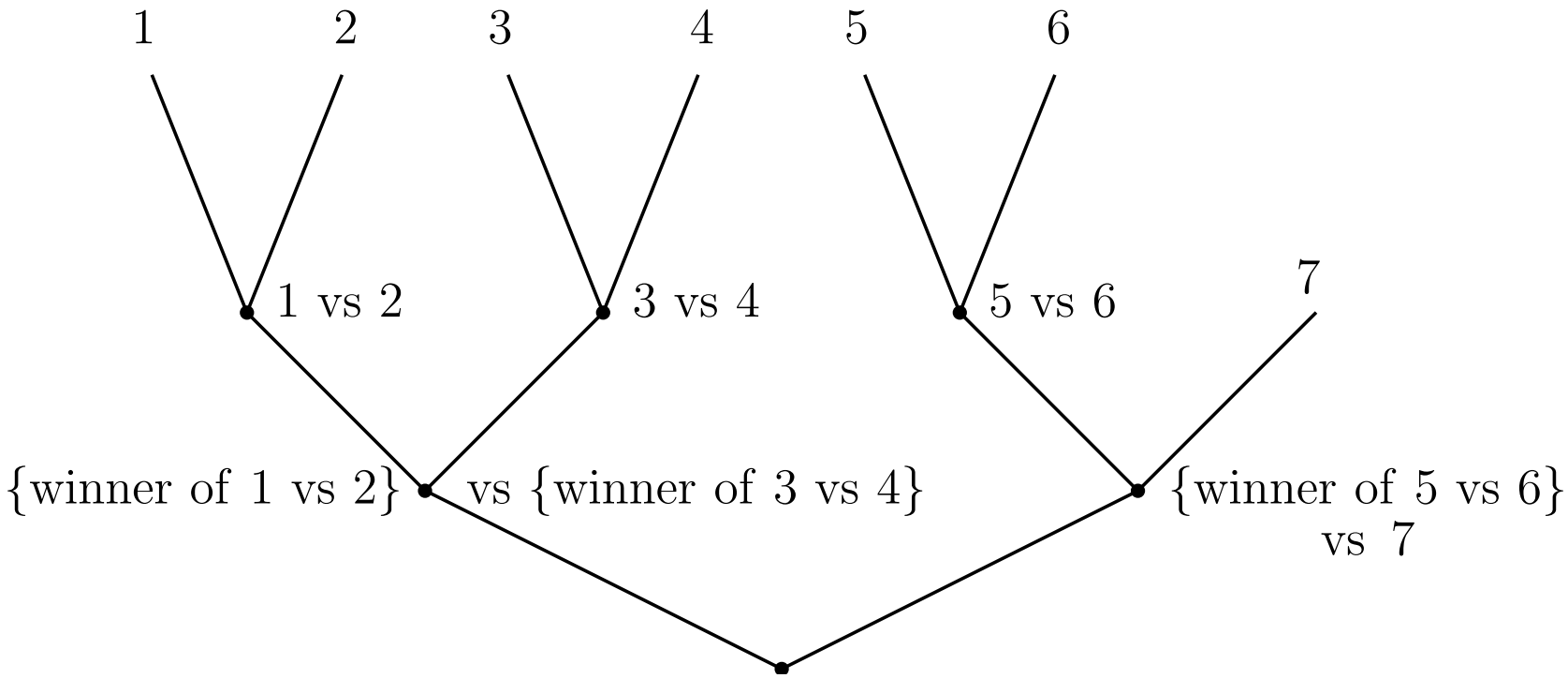
Caution: Trees go wrong

3 choices: $P(1 \mid x) = P(2 \mid x) = 0.275$, $P(3 \mid x) = 0.45$.



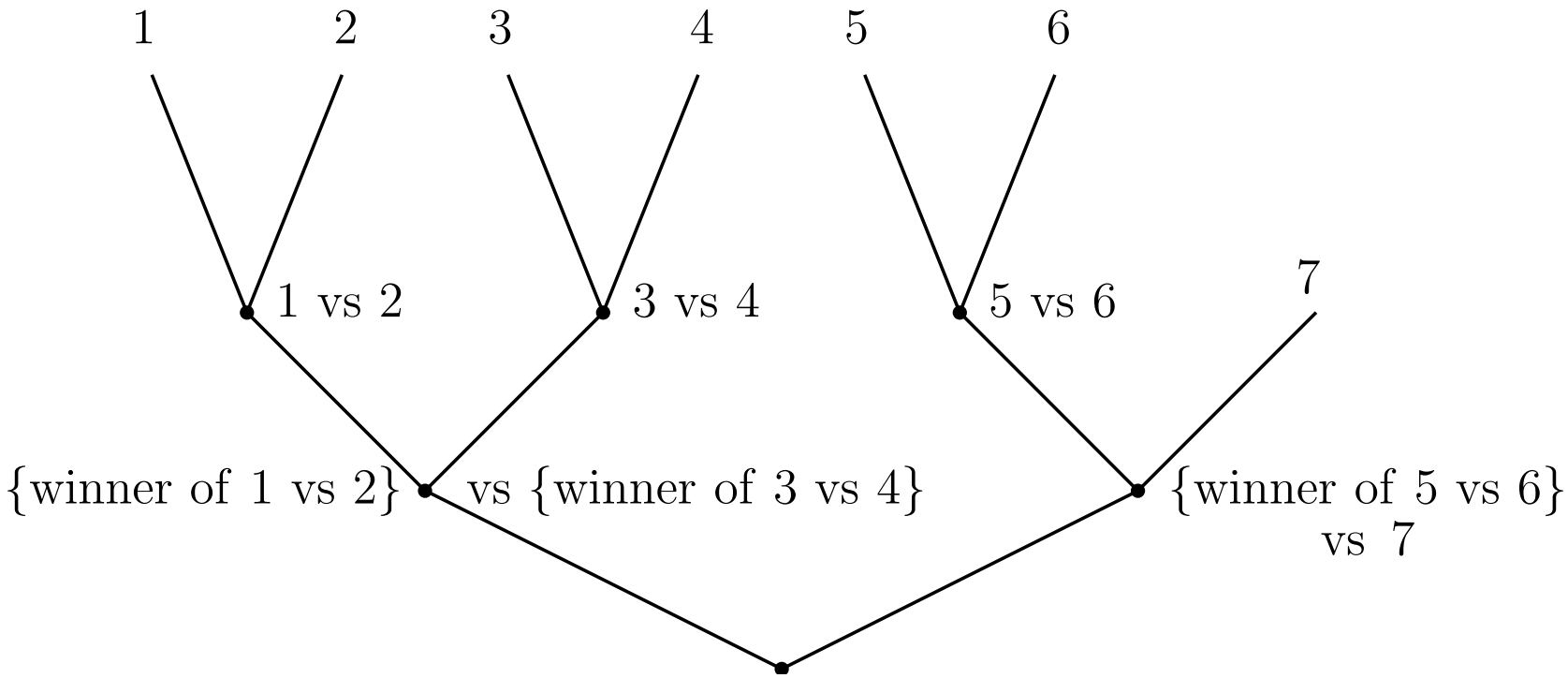
Optimal classifier picks 1 or 2, but choice 3 is best.

The Filter Tree (Single-Elimination Tournament)



Each non-leaf predicts the best of a pair of winners from the previous round

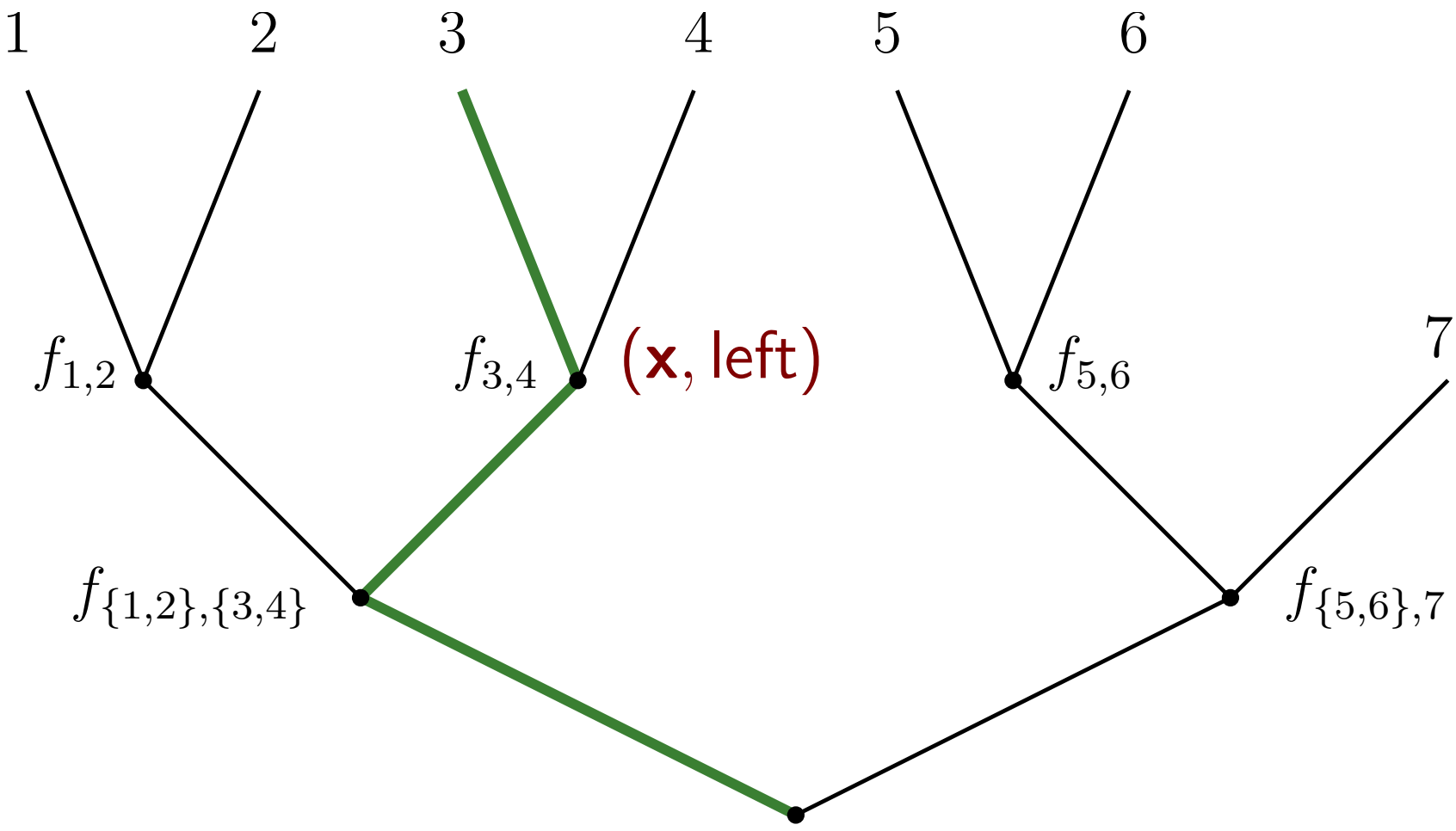
The Filter Tree (Single-Elimination Tournament)



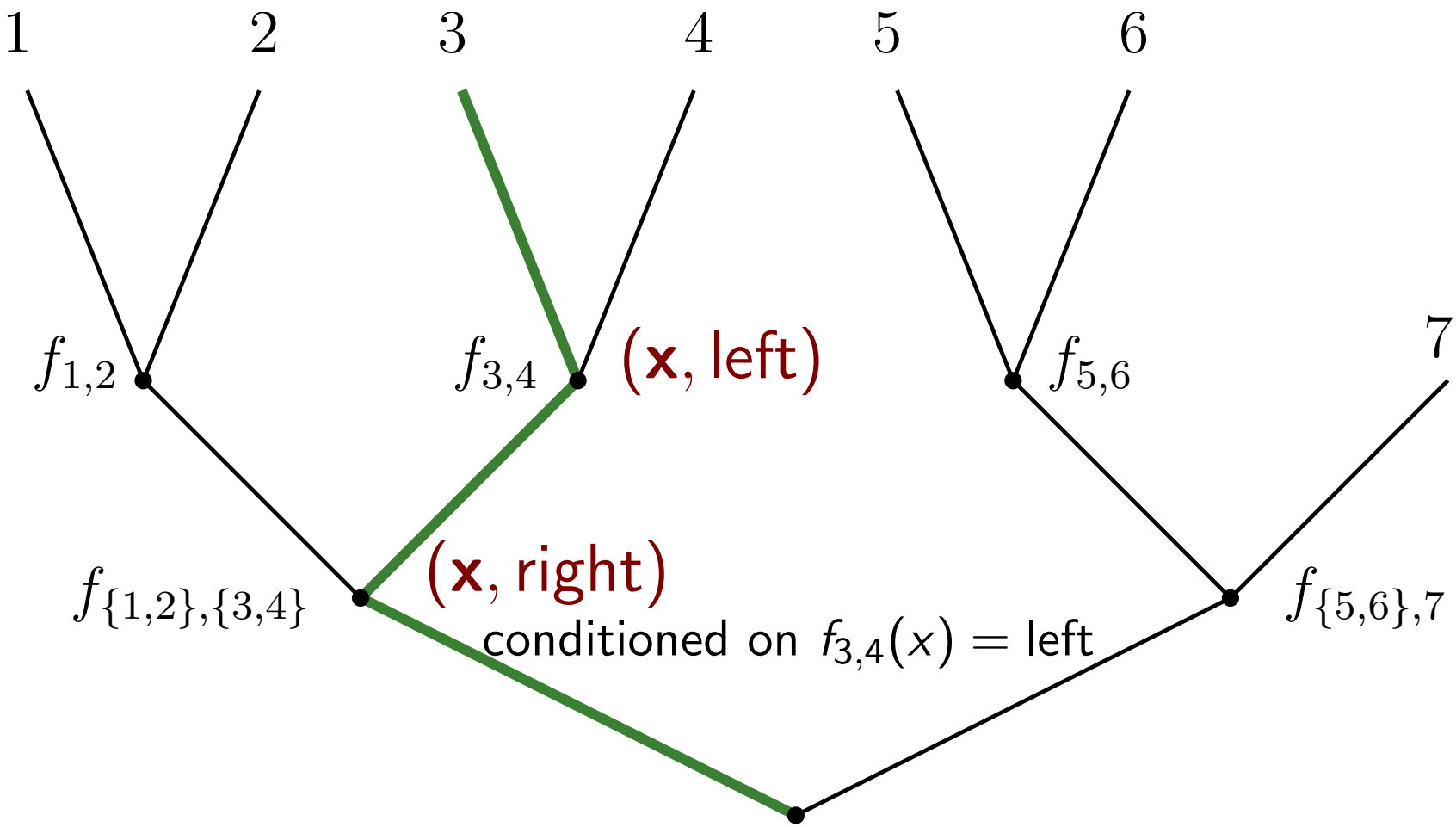
Each non-leaf predicts the best of a pair of winners from the previous round

To predict on x: follow the chain of predictions from root to leaf, output the leaf.

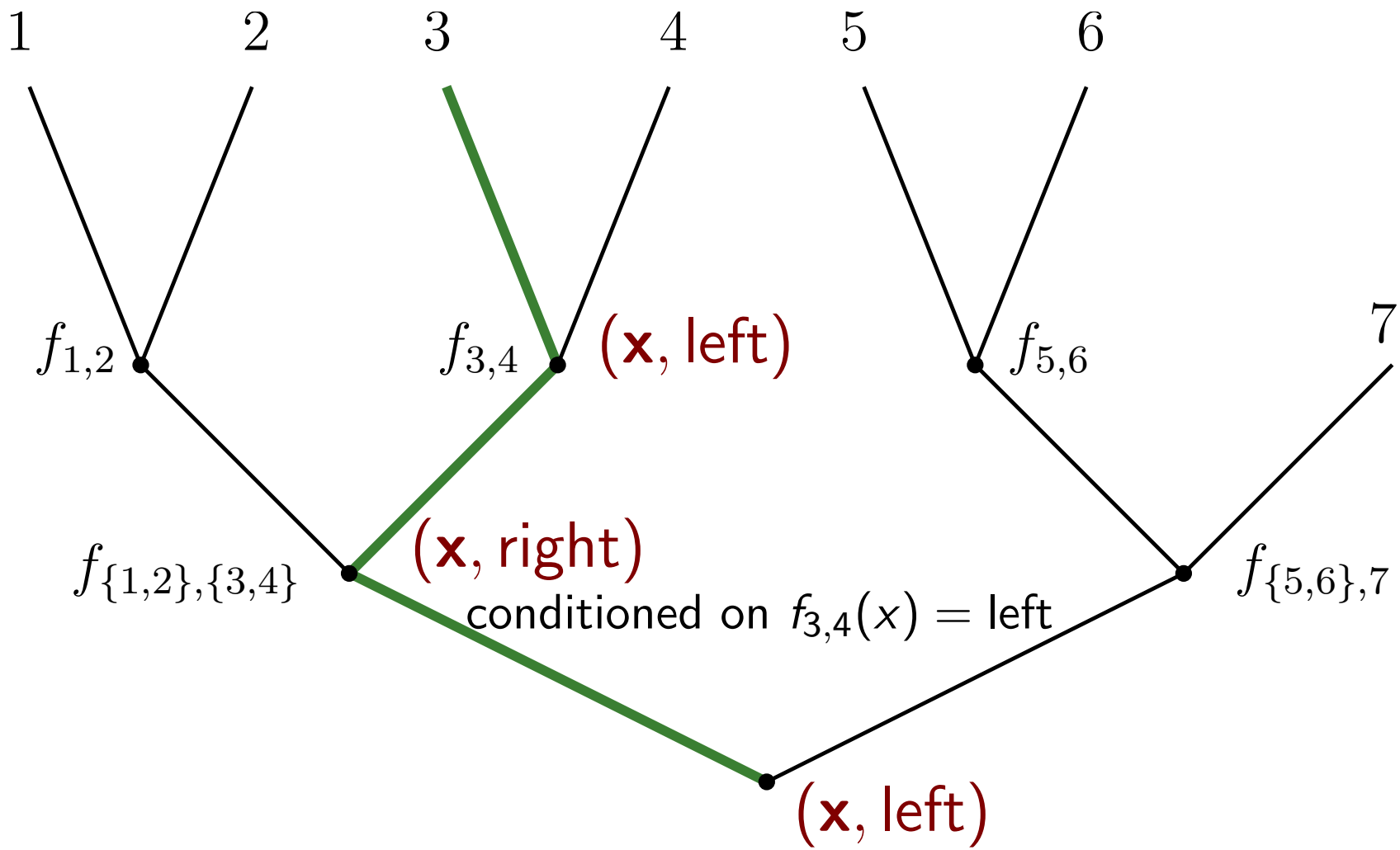
Training on example $(\mathbf{x}, \mathbf{3})$



Training on example $(\mathbf{x}, 3)$



Training on example $(\mathbf{x}, \mathbf{3})$



Training

- Important to form the right training sets: Classifiers from the first level are used to filter the distribution of examples reaching the second level, etc.
- Can be composed with either batch or online base learners.

What about with costs?

Cost-sensitive multi-class classification

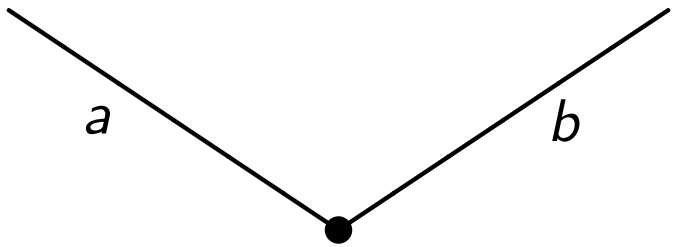
Distribution D over $X \times [0,1]^k$, where a vector in $[0,1]^k$ specifies the cost of each of the k choices.

Find a classifier $h : X \to \{1, \ldots, k\}$ minimizing the expected cost

$$\text{cost}(h, D) = \mathbf{E}_{(x,c)\sim D}[c_{h(x)}].$$

Generalization to the Cost-sensitive Case

To train a non-leaf node on example $(x, c_1, \ldots, c_k)$:



$$\text{Let } y = \begin{cases} \text{left} & \text{if } c_a \leq c_b \\ \text{right} & \text{otherwise} \end{cases}$$

Train on (x, y) with importance weight $|c_a - c_b|$.

Distribution induced at the node

Draw a cost-sensitive example from D, create an importance weighted sample as above.

Analysis

Binary regret:
$\text{reg}_{0/1}(h, D) = \Pr_{(x,y)\sim D}(h(x) \neq y) - \min_{h'} \Pr_{(x,y)\sim D}(h'(x) \neq y)$
Cost-sensitive regret:
$\text{reg}_{CS}(h, D) = E_{(x,y)\sim D}[c_{h(x)}] - \min_{h'} E_{(x,y)\sim D}[c_{h'(x)}]$

Theorem

For all cost-sensitive problems and classifiers at the nodes, the resulting cost-sensitive regret is bounded by average binary regret, times the expected sum of importance weights over the nodes.

$$\text{reg}_{CS}(h_{FT}, D) \leq A(\text{reg}_{0/1}(h, D_{FT}))E_{D_{FT}} \sum_{\text{nodes } n} i_n$$

Alternative bound replaces the sum of importance weights with $k/2$.

The multiclass case

For all multiclass problems and binary classifiers at the nodes, the resulting multiclass regret $\leq$ average binary regret, times $\lceil \log k \rceil$.

Proof: For any example, there is at most one node per level with induced weight 1. Thus tree depth bounds the sum of weights.

The multiclass case

For all multiclass problems and binary classifiers at the nodes, the resulting multiclass regret $\leq$ average binary regret, times $\lceil \log k \rceil$.

Proof: For any example, there is at most one node per level with induced weight 1. Thus tree depth bounds the sum of weights.

Can we make it more robust?

The multiclass case

For all multiclass problems and binary classifiers at the nodes, the resulting multiclass regret $\leq$ average binary regret, times $\lceil \log k \rceil$.

Proof: For any example, there is at most one node per level with induced weight 1. Thus tree depth bounds the sum of weights.

Can we make it more robust?

- Using multiple independent single elimination tournaments is of no help since it doesn't affect the average regret of an adversary controlling the binary classifiers.

The multiclass case

For all multiclass problems and binary classifiers at the nodes, the resulting multiclass regret $\leq$ average binary regret, times $\lceil \log k \rceil$.

Proof: For any example, there is at most one node per level with induced weight 1. Thus tree depth bounds the sum of weights.

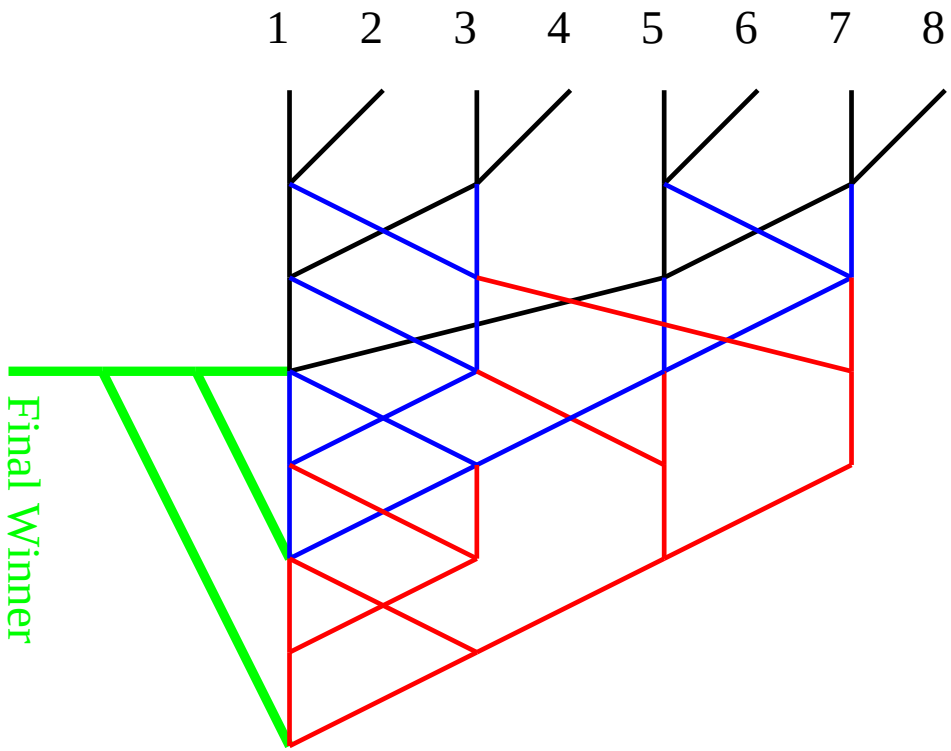
Can we make it more robust?

- Using multiple independent single elimination tournaments is of no help since it doesn't affect the average regret of an adversary controlling the binary classifiers.
- ...But we can have $e = O(\log k)$ single elimination tournaments in $O(\log k)$ rounds, with no player playing twice in the same round.

e-elimination tournament

Once an example loses, it moves to the next tournament. Once an example has lost e times, it is eliminated and no longer influences training.

The e winners from the first phase compete in the final single elimination tournament. To win in round i, each player must defeat its opponent 2^{i-1} times.



$e = 3$

Summary of Multiclass results

	Filter Tree	Error Correcting Tournament
MC Computation	$\log k$	$O(\log k)$
MC Regret ratio	$\log k$	5.5
CS Train	k	$O(k \log k)$
CS Test	$\log k$	$O(\log k)$
CS Regret ratio	$\min\{\frac{k}{2}, \sum_n i_n\}$	??

Contextual Bandit Learning in Logarithmic time

Contextual Bandit Classification

Distribution D over $X \times [0,1]^k$, where a vector in $[0,1]^k$ specifies the cost of each of the k choices.

Find a classifier $h : X \to \{1, \ldots, k\}$ minimizing the expected cost

$$\text{cost}(h, D) = \mathbf{E}_{(x,c)\sim D}[c_{h(x)}].$$

given only observations $(x, a, c_a, p_a)^*$.

The Offset Tree for $k = 2$, $p = 0.5$

Suppose $k = 2$ for the moment and let $a \in \{-1, 1\}$. Create binary importance weighted samples according to:

$$\left(x,\ \text{sign}\left(a\left(\frac{1}{2} - c_a \right) \right),\ \left| \frac{1}{2} - c_a \right| \right)$$

x = side information
$\text{sign}\left(a\left(\frac{1}{2} - c_a\right)\right)$ = label
$\left|\frac{1}{2} - c_a\right|$ = importance weight

Denoising Binary Importance Weighting

Theorem

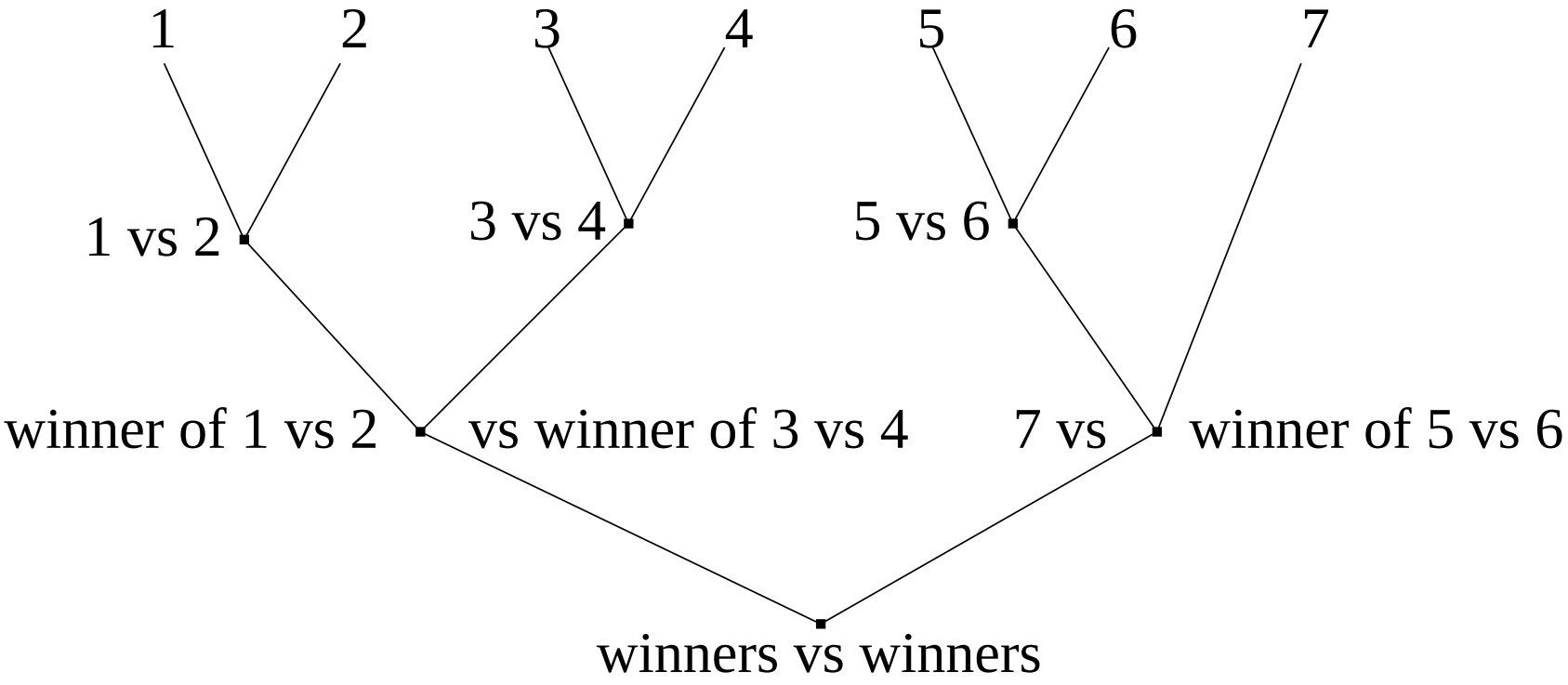
For all Contextual Bandit distributions D with $k = 2$, for all binary classifiers b:

$$\text{policy regret} \leq \text{reg}_{0/1}(b, D_{OT}).$$

The induced problem is often noisy. This trick reduces the maximum noise, giving a factor of 2 improvement in the upper bound.

$\frac{1}{2}$ = minimax value of the median reward. Plugging in the actual median is always better.

Denoising for $k > 2$ arms



Use the same construction at each node. Internal nodes only get an example if all leaf-wards nodes agree with the label (Filtering trick).

Denoising with k arms

D_{OT} = Induced Binary classification problem
b = the classifier which predicts based on both x and the choice of binary problem according to D_{OT}.

Theorem

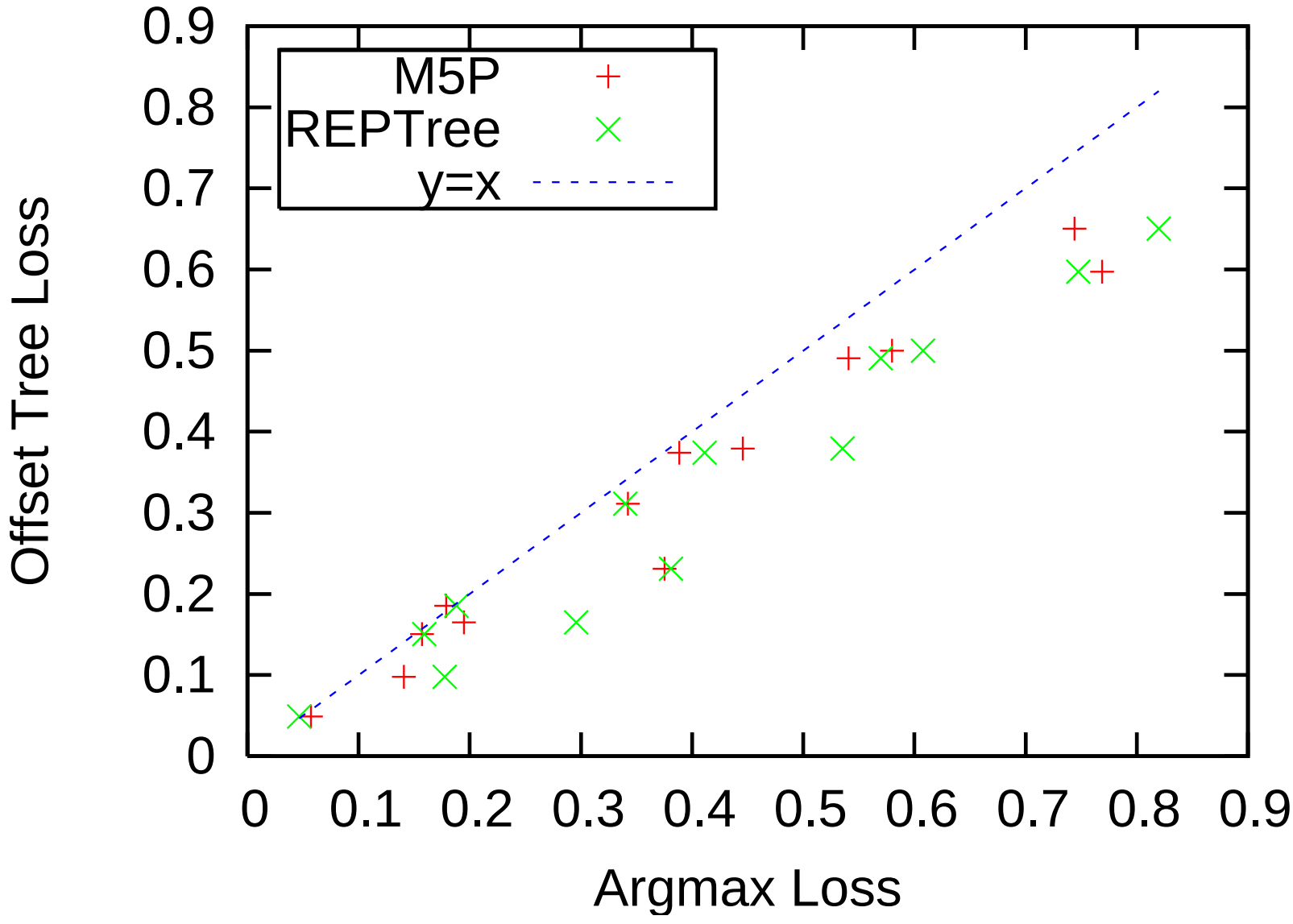
For all k-choice D, binary classifiers b:

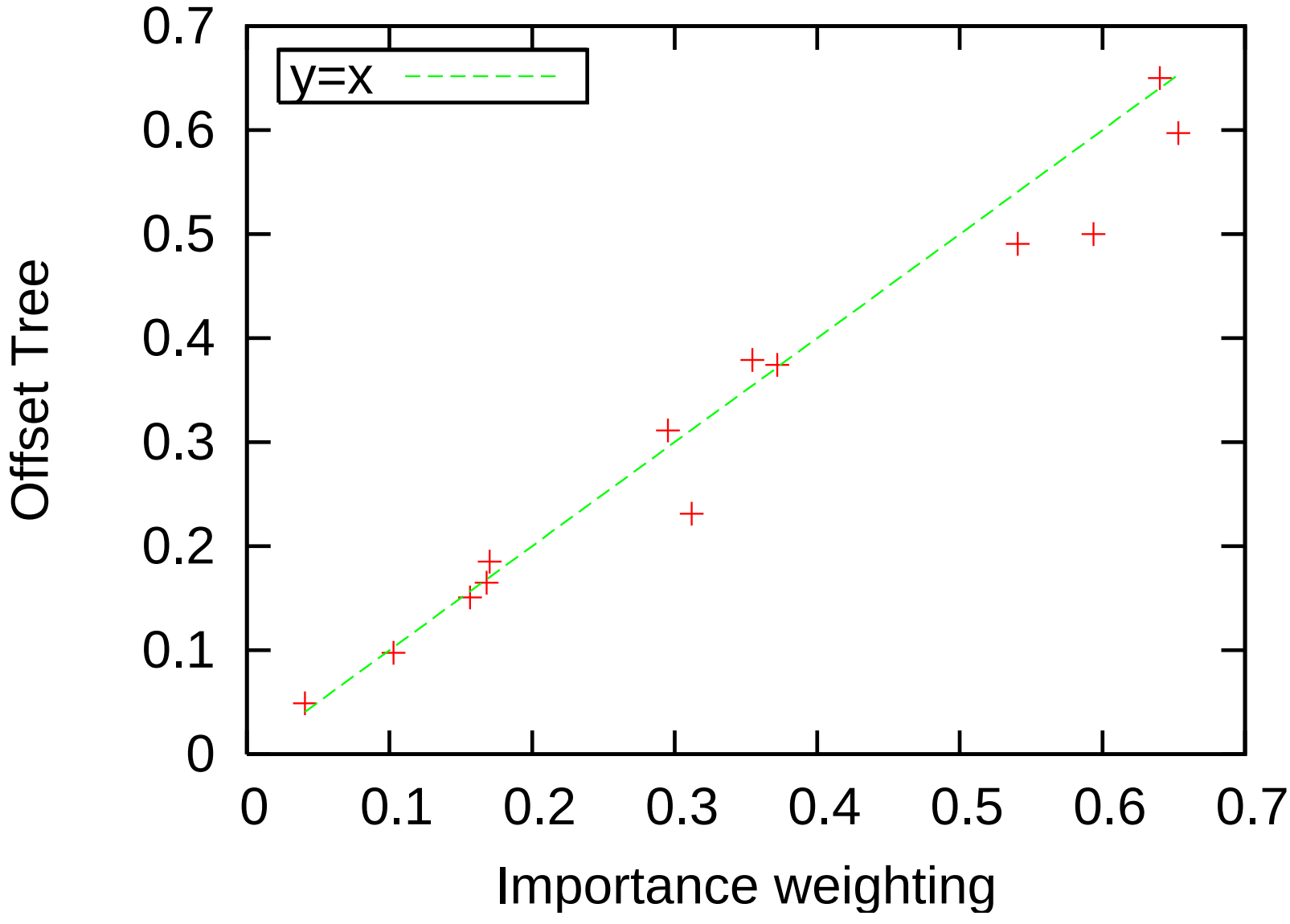
$$\text{policy regret} \leq (k-1)\,\text{reg}_{0/1}(b, D_{OT}).$$

A Comparison of Approaches

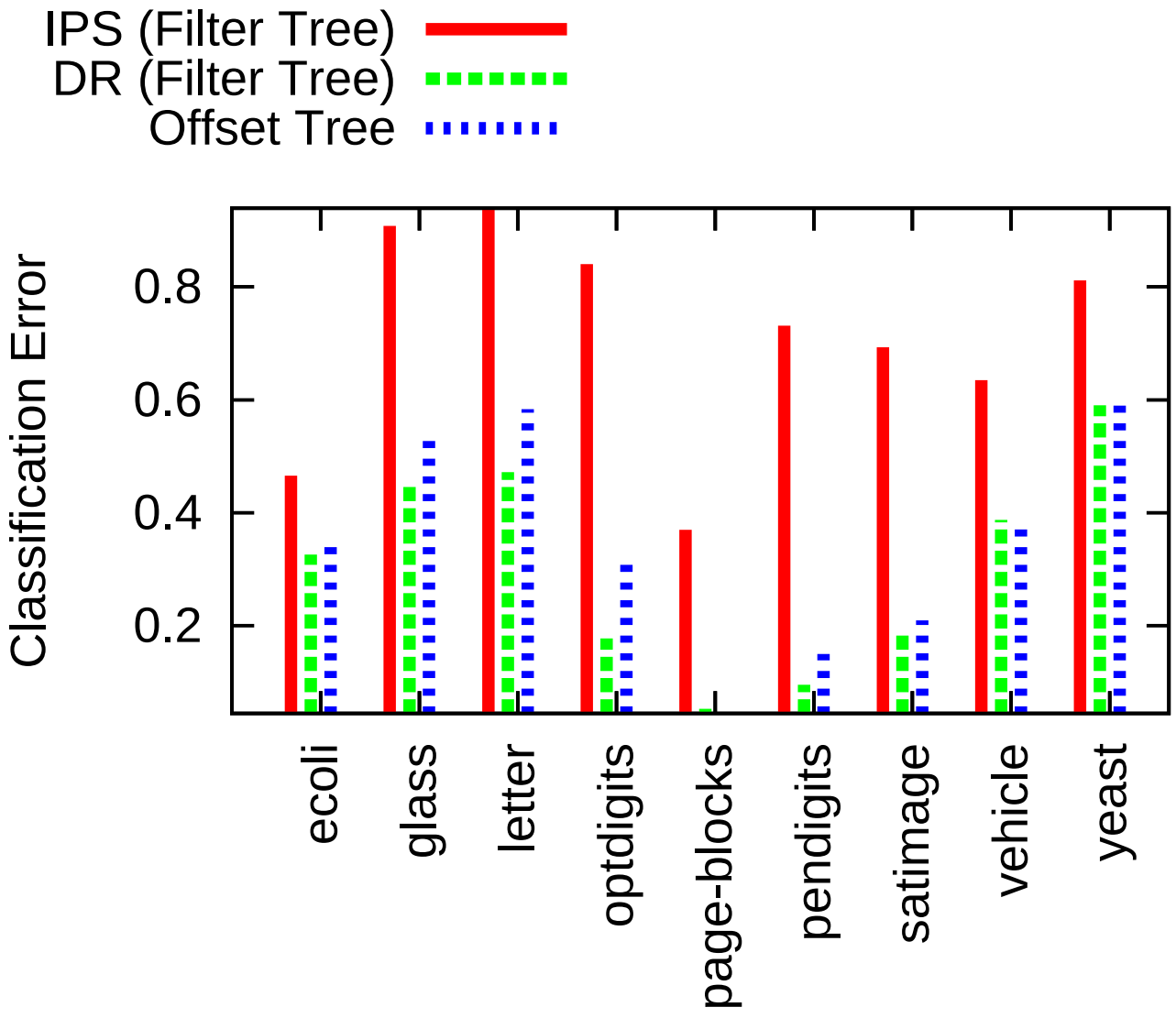
Algorithm	Policy Regret Bound
Argmax	$\sqrt{2k \operatorname{reg}_{0/1}(s, D_{AR})}$
Importance Weighted	$4k \operatorname{reg}_{0/1}(b, D_{IW})$
Offset Tree	$(k-1) \operatorname{reg}_{0/1}(b, D_{OT})$

How do you expect things to work, experimentally?


M5P
REPTree
y=x
Offset Tree Loss
Argmax Loss
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


y=x
Offset Tree
Importance weighting
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

Compare with Double Robust approaches empirically



Double Robust is exponentially slower, but often a bit better.

The last problem

A dataset:

1. $40 * 10^6$ (ad,webpage) pairs ($+ 10 * 10^6$ pairs in test set)
2. $\sim 10^7$ unique webpages
3. $\sim 10^6$ unique ads
4. $\sim 10^6$ unique features.

A problem:
Predict: Pr(ad|webpage) for a given (ad, webpage) pair.

The last problem

A dataset:

1. $40 * 10^6$ (ad,webpage) pairs ($+ 10 * 10^6$ pairs in test set)
2. $\sim 10^7$ unique webpages
3. $\sim 10^6$ unique ads
4. $\sim 10^6$ unique features.

A problem:
Predict: Pr(ad|webpage) for a given (ad, webpage) pair.

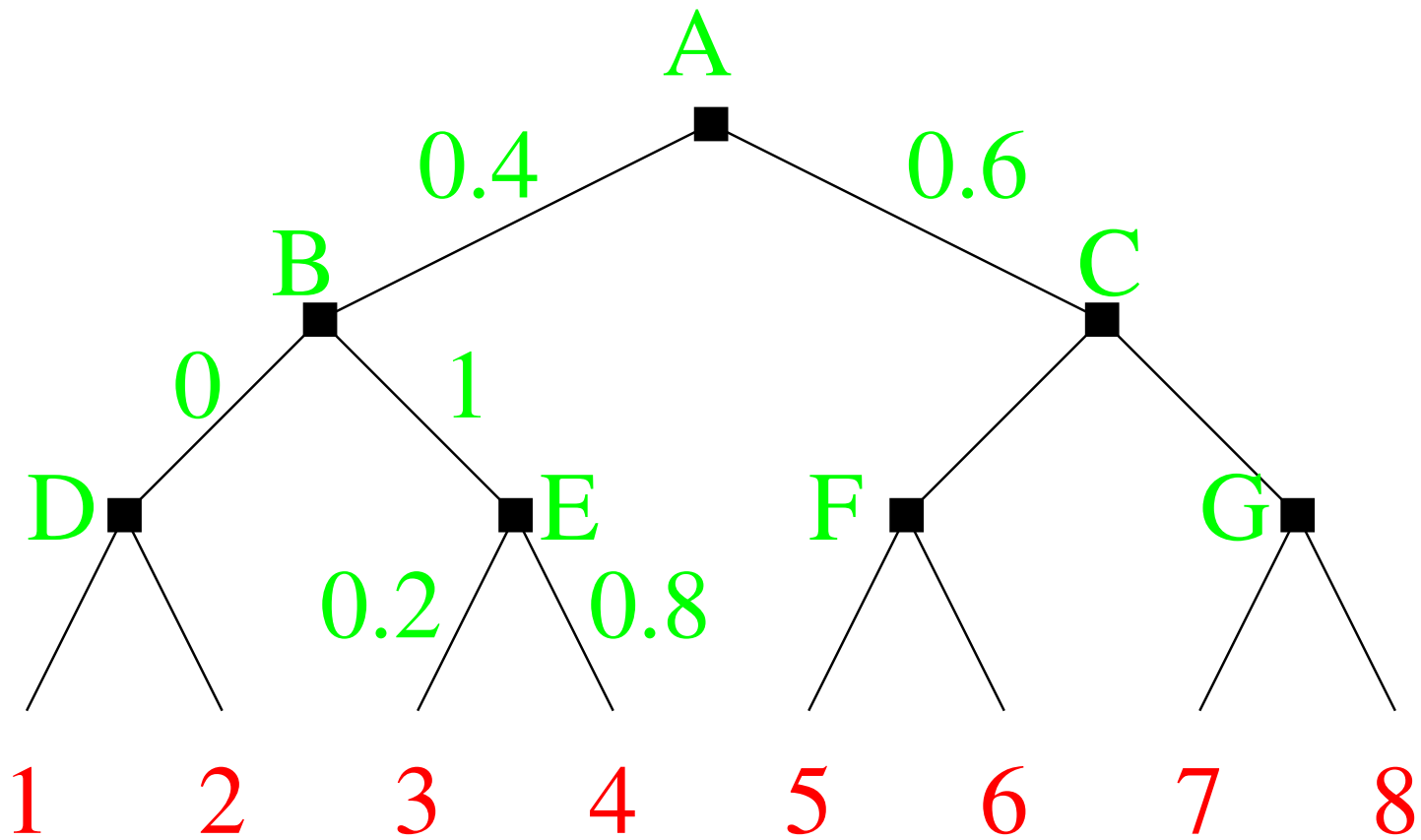
Conditional Probability Estimation

Distribution D over $X \times Y$, where $Y = \{1, \ldots, k\}$.

Find a Probability estimator $h : Y \times X \to [0, 1]$ minimizing squared loss

$$\ell(h, D) = E_{(x,y)\sim D}[(h(y|x) - y)^2]$$

One approach: A Conditional Probability Tree



Every node predicts $P(\text{ad is to the right}|\text{page})$.
Number of nodes to evaluate: $\log_2 |\text{ads}|$.

$$\hat{P}(\text{ad 4}|\text{page}) = (1 - \text{A}(\text{page}))\text{B}(\text{page})\text{E}(\text{page})$$

$$= 0.4 * 1 * 0.8 = 0.32$$

But is the prediction too unstable to work?

Basic fact: minimizer of squared loss = correct probability.
Theorem: For any distribution P, Any regressors Q, and any pair (x, y),

$$(P_Q(y|x) - P(y|x))^2$$

$$\leq \text{depth}^2 E_{i \in \text{Path}}(Q_i(\text{right}_i(y)|\text{page}) - P_i(\text{right}_i(y)|\text{page}))^2$$

$$= \text{depth}^2(\text{average binary regret})$$

A similar statement for one-against-all approaches replaces depth^2 with $|\text{ads}|^2$. This is much worse, although it often is not realized in practice.

Proof

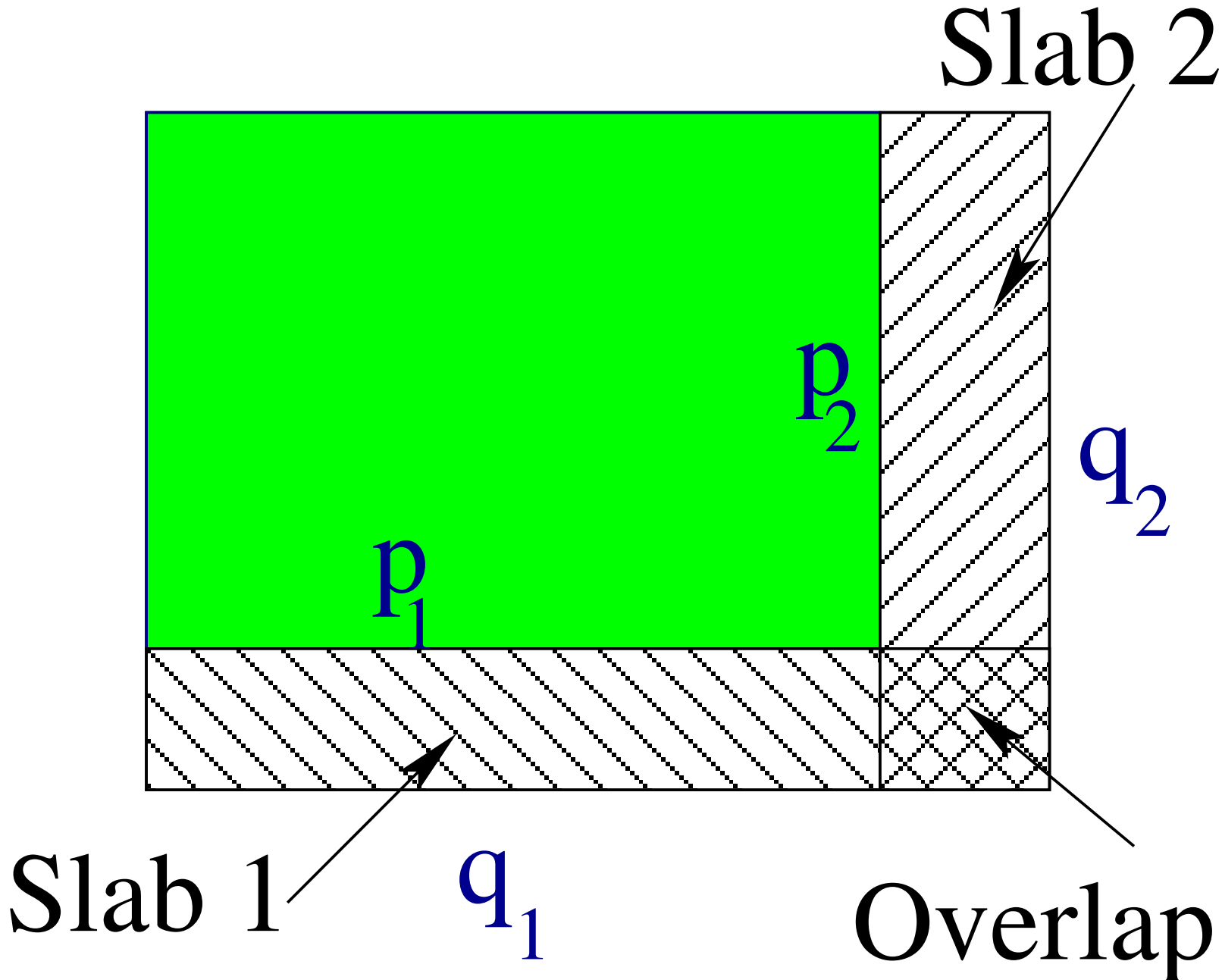
Let $q_i = Q_i(\text{right}_i(y)|x)$ and $p_i = P_i(\text{right}_i(y)|x)$.
$|P_Q(y|x) - P(y|x)| \leq \prod_i \max\{q_i, p_i\} - \prod_i \min\{q_i, p_i\}$
Using geometry, $\prod_i \max\{q_i, p_i\}$, $\prod_i \min\{q_i, p_i\}$ = hypercube volumes.
Small hypercube + slabs $\geq$ big hypercube.

$$\prod_i \max\{q_i, p_i\} - \prod_i \min\{q_i, p_i\} \leq \sum_i |q_i - p_i| \prod_{j \neq i} \max\{q_j, p_j\}$$

$$\leq \sum_i |q_i - p_i|$$

... then apply Jensen's inequality.

The hypercube arbument in 2-d



How do we assign labels to leaves?

How do we assign labels to leaves?

Random left/right decisions: 0.7742 ± 0.0006 (Uniform over ~ 8.32 ads)

Surprisingly good, but surely we can do better than random?

A better than random approach

When a new label is encountered, starting at the root and descend, send the label right if:

$$(1-\alpha)\left(2Q_i(\text{page})-1\right)+\alpha\log_2\frac{|\text{ads left}|+1}{|\text{ads right}|+1}>0$$

Theorem: For all $\alpha\in[0.5,1]$, for all binary regressors,

$$\text{depth}\leq 3+\frac{\log_2|\text{ads}|}{\log_2\left(1+2^{\frac{\alpha-1}{\alpha}}\right)}$$

Proof: Requires a bit of work because of discretization effects.

RAM problems

How do you store and use 10^6 binary regressors each with $\sim 10^6$ features?

We don't have $4 * 10^{12}$ bytes of RAM!

RAM problems

How do you store and use 10^6 binary regressors each with $\sim 10^6$ features?

We don't have $4 * 10^{12}$ bytes of RAM!

Hashing

Conditional Probability Tree on Ad dataset

1. Training time $\sim$ 1 hour on a single machine.
2. RAM usage $\sim$ 100MB.
3. Squared loss 0.7632 ± 0.0006 (equivalent to uniform random over 7.91 ads).

Conditional Probability Tree on RCV1

(Full) RCV1 has:

1. 103 classes
2. 780K examples
3. $\sim 2^{16}$ features

aka: small enough to do the wrong thing

Conditional Probability Tree on RCV1

(Full) RCV1 has:

1. 103 classes
2. 780K examples
3. $\sim 2^{16}$ features

aka: small enough to do the wrong thing

	Conditional Probability Tree	One Against All
Computation	$108s$	$2300s$
Squared Loss	0.56 ± 0.012	0.55 ± 0.012

Losing a small amount of prediction performance for a significant gain.

Choose your baseline

1. Relative to $O(k)$ ML techniques, giant improvements in computation + sometimes better/sometimes worse performance.
2. Relative to index+rank approaches, perhaps a significant improvement in performance.

This area is immature: there are plausibly several theses to write.

Bibliography

Multilabel R. Agarwal, A. Gupta, Y. Prabhu, and M. Varma, Multi-label Learning Learning with Millions of Labels: Recommending Advertiser Bid Phrases for Web Pages, WWW 2013.

Filter A. Beygelzimer, J. Langford, and P. Ravikumar, Error-Correcting Tournaments, ALT 2009.

Offset A. Beygelzimer and J. Langford, The Offset Tree for Learning with Partial Labels, KDD 2009.

Prob A. Beygelzimer, J. Langford, Y. Lifshits, G. Sorkin, A. Strehl, Conditional Probability Tree Estimation Analysis and Algorithms, UAI 2009.