Machine Learning Reductions Tutorial

Alina Beygelzimer (IBM Research)

John Langford (Yahoo! Research)

Bianca Zadrozny (Fluminense Federal University, Brazil)

July 13, 2009

Applying Machine Learning in Practice

The problem you want to solve is not the problem solved by standard learning algorithms / toolboxes.

Applications gave rise to many different types of machine learning problems:

- Cost-sensitive classification
- Hierarchical classification
- Structured classification
- Machine translation
- Ranking
- ...

Applying Machine Learning in Practice

Approaches:

1. Design new algorithms (or modify existing ones).
2. Think about how to reuse old ones.

This tutorial is about the second approach.

Core Problem

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:

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

There have been years of research and development of algorithms for solving this problem. It would be nice to be able to reuse this for other problems.

What do we want from a reduction?

What do we want from a reduction?

It should be robust. Good 0/1 performance should imply good performance on the problem we care about.

What do we want from a reduction?

It should be robust. Good 0/1 performance should imply good performance on the problem we care about.

It should be modular. Able to plug in any classifier *and* reuse created reductions similarly.

Simple Variation on the Core Problem

Importance-Weighted Classification

- Given training data $\{(\mathbf{x_1}, y_1, c_1), \ldots, (\mathbf{x_n}, y_n, c_n)\}$, produce a classifier $h : X \to \{0, 1\}$.
- Unknown underlying distribution D over $X \times \{0, 1\} \times [0, \infty)$.
- Find h with small expected cost:

$$\ell(h, D) = \mathbf{E}_{(x,y,c)\sim D}[c \cdot \mathbf{1}(h(x) \neq y)]$$

Common scenario: one class is rare but the cost of not recognizing examples of this class is generally high. Can't apply binary classification algorithms directly.

Other Approaches

(1) Make particular classifier learners cost-sensitive

- Need to know the algorithm + implementation
- For a number of algorithms, it's not so easy to perform the conversion correctly

(2) Bayes risk minimization

$$R(y = i \mid x) = D(y = 1 - i \mid x)C(x) \quad \text{for } i \in \{0, 1\},$$

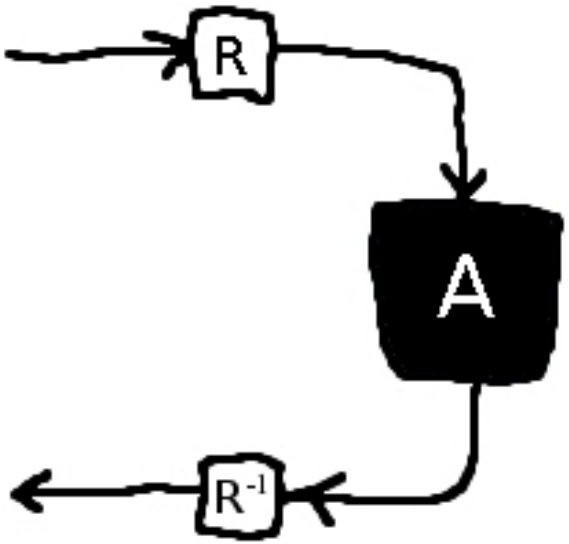
where $C(x)$ is the expected cost of misclassifying x.

- Requires estimating class membership probabilities $D(y \mid x)$ and example-dependent costs.

- Many classifiers yield probability estimates but these are often very poor.

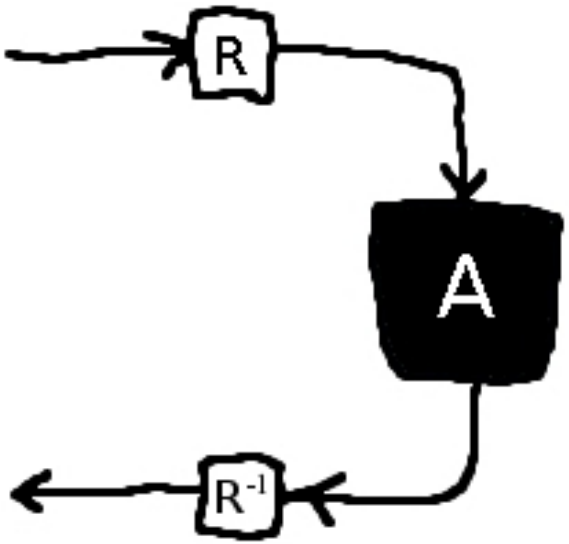
Reductions

Goal: minimize ℓ on D



Reductions

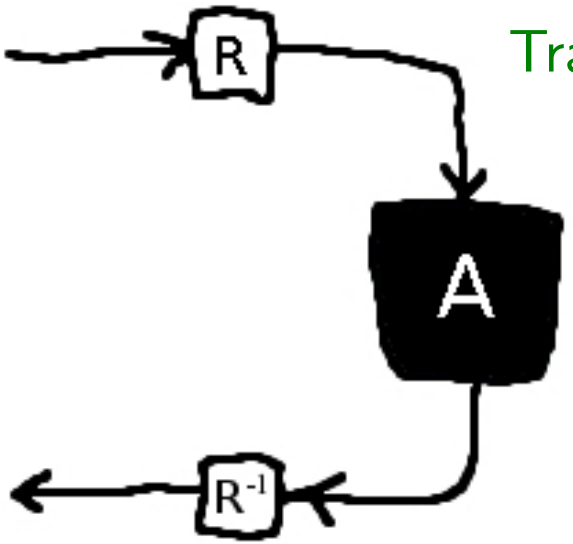
Goal: minimize ℓ on D



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

Reductions

Goal: minimize ℓ on D

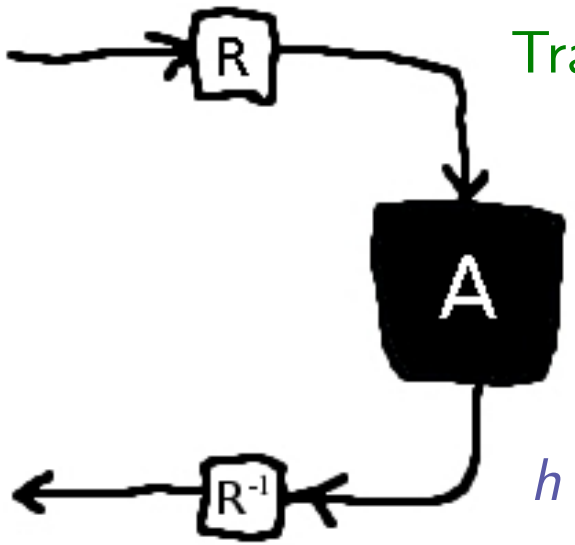


Transform D into D'

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

Reductions

Goal: minimize ℓ on D



Transform D into D'

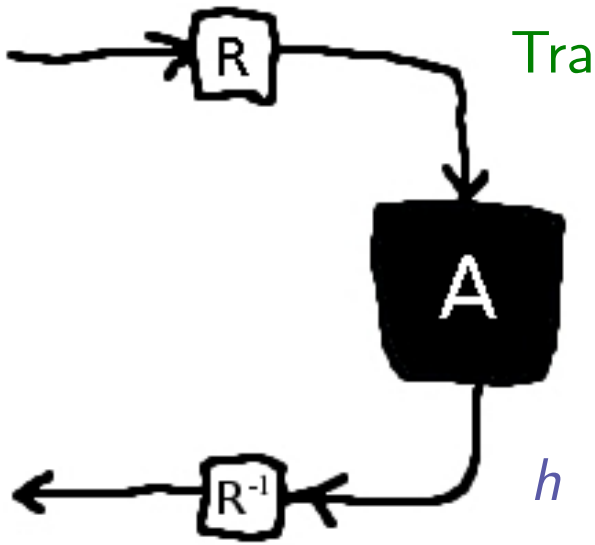
Algorithm for optimizing $\ell_{0/1}$

h

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

Reductions

Goal: minimize ℓ on D



Transform D into D'

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

h

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

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

Example: Importance-weighted Classification

Theorem (Distribution Shift)

For any importance-weighted distribution D and any classifier h, let

$$D'(\mathbf{x}, y, c) = \frac{c}{\mathbf{E}_{(\mathbf{x},y,c)\sim D}[c]} D(\mathbf{x}, y, c),$$

then $\ell(h, D) = \ell_{0/1}(h, D')\mathbf{E}_{(\mathbf{x},y,c)\sim D}[c]$.

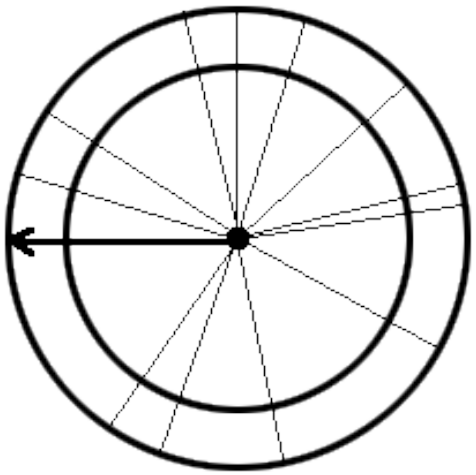
... So choosing h to minimize 0-1 loss under D' is equivalent to choosing h to minimize the expected cost under D.

Proof: Letting $\langle c \rangle = \mathbf{E}_{(\mathbf{x},y,c)\sim D}[c]$,

$$\underbrace{\mathbf{E}_{(\mathbf{x},y,c)\sim D}[c \cdot \mathbf{1}(h(x) \neq y)]}_{\ell(h,D)} = \sum_{(\mathbf{x},y,c)} D(\mathbf{x}, y, c) c \mathbf{1}(h(x) \neq y)$$

$$= \langle c \rangle \sum_{(\mathbf{x},y,c)} D'(\mathbf{x}, y, c)\mathbf{1}(h(x) \neq y) = \langle c \rangle \underbrace{\mathbf{E}_{(\mathbf{x},y,c)\sim D'}[\mathbf{1}(h(x) \neq y)]}_{\ell_{0/1}(h,D')}$$

How do we change the distribution?



Approaches that don't work well:

Resampling with replacement (stratification)

- place examples on roulette wheel with coverage proportional to weight
- spin the wheel many times

Basic problem: duplicate examples = resampled set is not drawn independently from D'

Resampling without replacement

sampling m examples from a set of size m results in the original set, which is drawn from D not D'.

Rejection Sampling

- Pick an upper bound $\hat{c}$ on any importance value in the dataset
- For each example $(\mathbf{x}, y, c) \in S$, flip a coin with bias $c/\hat{c}$. If heads, keep the example; otherwise discard it.

Independence preserving:

Examples in the resampled dataset are drawn independently from D' if examples in the original set are drawn independently from D.

Costing

Train (importance-weighted set S, binary learner A)

For $i = 1$ to 10:

- Rejection sample from S to form S_i
- Train a classifier $h_i = A(S_i)$

Test (example x)

Return

$$h(x) = \text{majority}\left(\{h_1(x), \ldots, h_{10}(x)\}\right)$$

Why reductions?

1. Work well in practice (e.g., multi-class classification using binary learners, boosting)

Why reductions?

1. Work well in practice (e.g., multi-class classification using binary learners, boosting)
2. Can reuse highly optimized learning algorithms and code (including future ones):

$$N \textbf{ base learners} + \textbf{1 reduction} = N \textbf{ different algorithms}$$

Why reductions?

1. Work well in practice (e.g., multi-class classification using binary learners, boosting)
2. Can reuse highly optimized learning algorithms and code (including future ones):

   N base learners + 1 reduction = N different algorithms

3. Reductions *transfer* performance (and theory) from one problem to another:
   - Weak assumptions = wide applicability
   - But only relative performance guarantees

Why reductions?

1. Work well in practice (e.g., multi-class classification using binary learners, boosting)
2. Can reuse highly optimized learning algorithms and code (including future ones):

   N base learners + 1 reduction = N different algorithms

3. Reductions *transfer* performance (and theory) from one problem to another:
   - Weak assumptions = wide applicability
   - But only relative performance guarantees
4. Reductions compose: Complexity is conquered by decomposing big problems into subproblems

Why reductions?

1. Work well in practice (e.g., multi-class classification using binary learners, boosting)
2. Can reuse highly optimized learning algorithms and code (including future ones):

   N base learners + 1 reduction = N different algorithms

3. Reductions *transfer* performance (and theory) from one problem to another:
   - Weak assumptions = wide applicability
   - But only relative performance guarantees
4. Reductions compose: Complexity is conquered by decomposing big problems into subproblems
5. Organize prediction problems

What can you do with a binary classifier learner?

- Importance-weighted classification
- Quantile regression (coming next)
- (Cost-sensitive) multi-class classification
- Ranking
- Learning with partial feedback

Quantile Regression

In many problems, we want *quantiles* (such as the median) of the conditional distribution $D \mid \mathbf{x}$ rather than the mean

- Quantiles are more robust to large outliers
- Describe different segments of the conditional distribution

Next: Quantile regression with any binary classification algorithm

Other methods:

- Linear quantile regression (Koenker)
  - Assumption: conditional quantile is a linear function of $\mathbf{x}$.
- Non-parametric quantile estimation method (kernel estimation + regularization), which reduces to a QP (Takeuchi et al.)
  - No assumed predetermined form; both structure and parameters are data-driven

Conditional q-quantile, $q \in [0,1]$

A function $f = f(\mathbf{x})$ is a q-quantile for D is for every $\mathbf{x} \in X$,

$$D(y \leq f(\mathbf{x}) \mid \mathbf{x}) \geq q \quad \text{and} \quad D(y \geq f(\mathbf{x}) \mid \mathbf{x}) \geq 1 - q$$

1/2-quantile is the median.

Basic observation: The conditional median is

$$\arg\min_f \mathbf{E}_{(\mathbf{x},y)\sim D}|y - f(\mathbf{x})|.$$

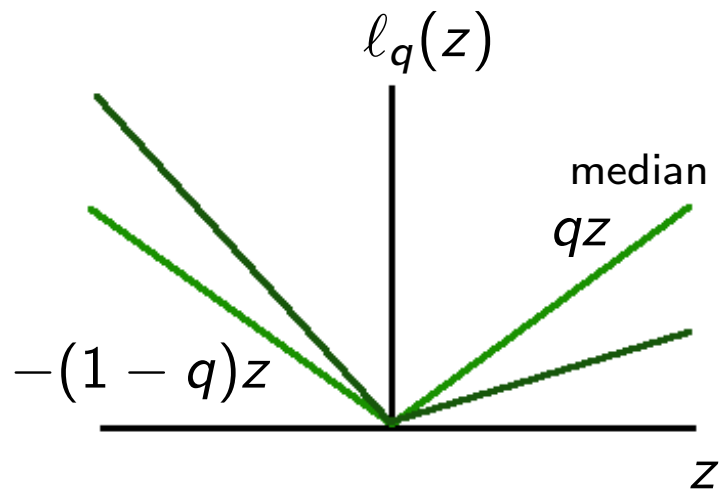
What about other quantiles?

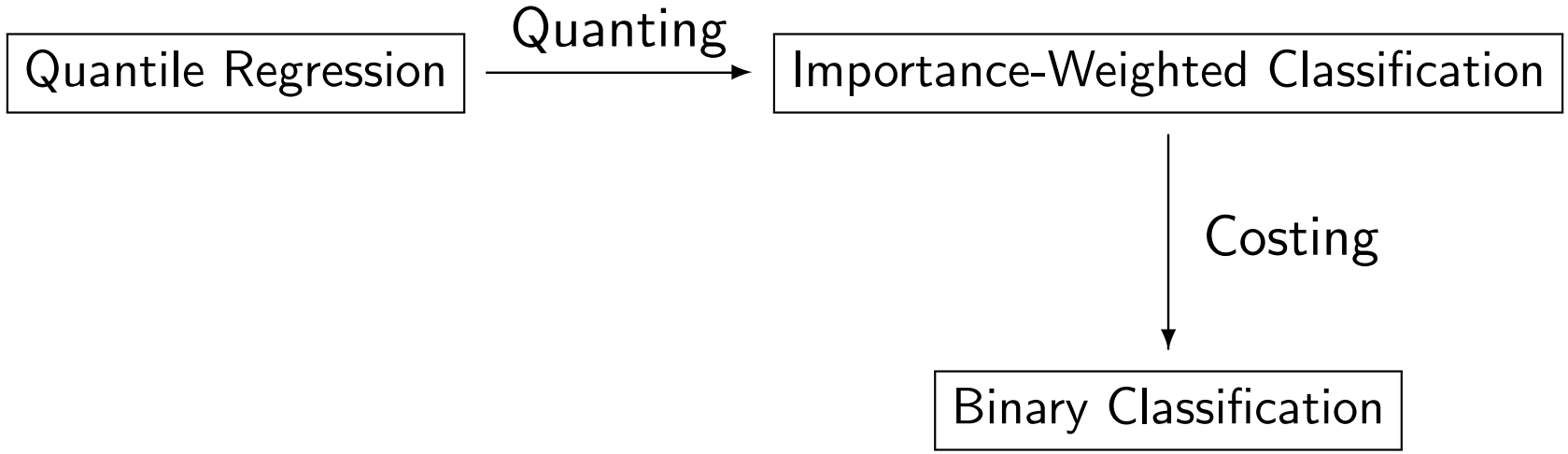
Pinball loss

$$\ell_q(z) = \begin{cases} qz & \text{if} \quad z \geq 0 \\ (q-1)z & \text{otherwise} \end{cases}$$

q-quantile is

$$\arg\min_f \mathbf{E}_{(\mathbf{x},y)\sim D}[\ell_q(y - f(\mathbf{x}))]$$





Quantile Regression Problem

Given a set of examples of the form $(\mathbf{x}, y)$ drawn from D over $X \times [0, 1]$, find a mapping $f : X \to [0, 1]$ that minimizes

$$\mathbf{E}_{(\mathbf{x},y)\sim D}\left[\, \ell_q(y - f(\mathbf{x})) \,\right]$$

$f(\mathbf{x}) =$ estimate of the q-th quantile of $D \mid \mathbf{x}$

q-Quanting — *Training Phase*

Parameters: importance-weighted classifier learner A, training set S

for $t \in [0, 1]$

$S_t = \emptyset$

foreach $(x, y) \in S$

$S_t = S_t \cup \{(x, \mathbf{1}(y \geq t), q \cdot \mathbf{1}(y \geq t) + (1 - q)\mathbf{1}(y < t))\}$

[positive examples get weight q, negative get weight $(1 - q)$]

$h_t = A(S_t)$

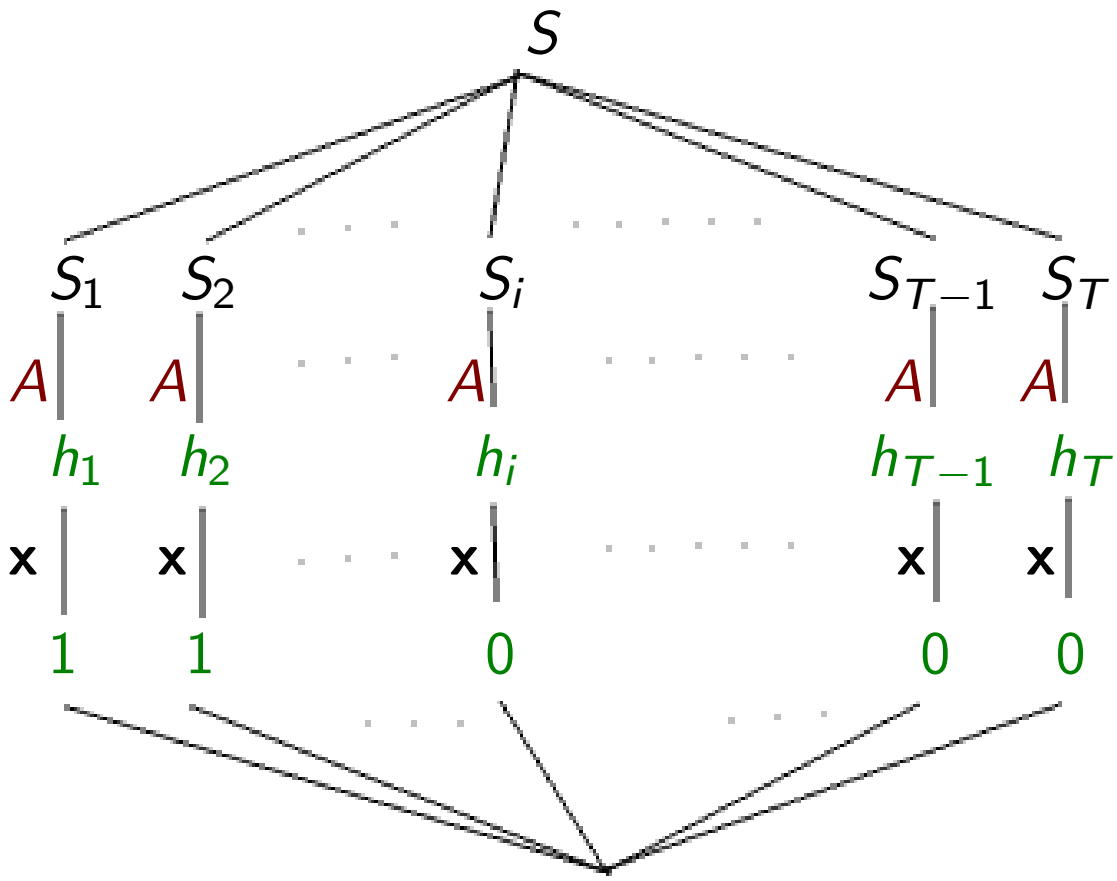
return the set of classifiers $\{h_t\}$

Using $t \in \{0, 1/n, \ldots, (n-1)/n, 1\}$ adds at most $1/n$ term to ℓ_q

q-Quanting (classifiers $\{h_t\}$, test example $\mathbf{x}$) — *Test Phase*

return $f(\mathbf{x}) = \mathbf{E}_{t \sim U(0,1)}[h_t(\mathbf{x})]$

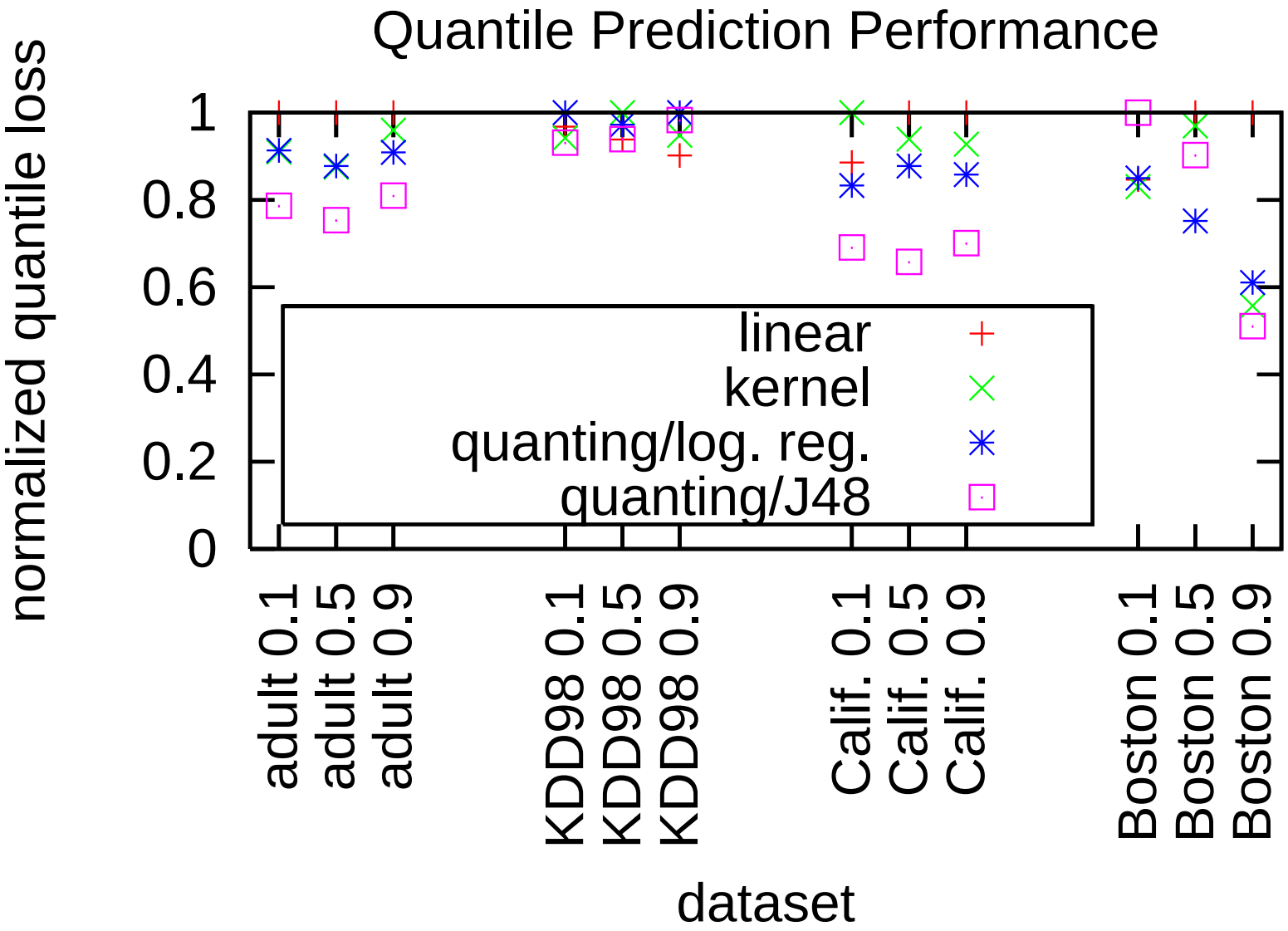
The Quanting Algorithm



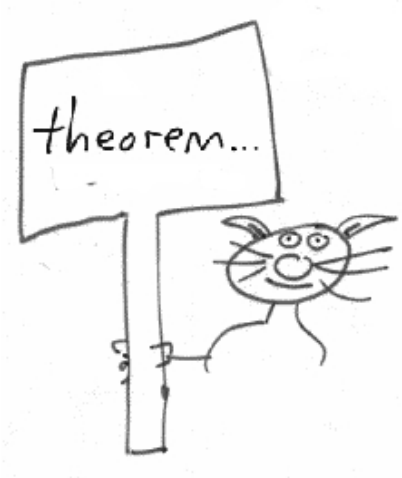
test example

importance-weighted predictions

Single classifier trick: instead of learning different classifiers, we can learn just one $h = \{h_t\}$ with an extra index feature t.


Quantile Prediction Performance
normalized quantile loss
1
0.8
0.6
0.4
0.2
0
linear
kernel
quanting/log. reg.
quanting/J48
adult 0.1
adult 0.5
adult 0.9
KDD98 0.1
KDD98 0.5
KDD98 0.9
Calif. 0.1
Calif. 0.5
Calif. 0.9
Boston 0.1
Boston 0.5
Boston 0.9
dataset

Quanting Analysis



Theorem (for the median)

For all distributions D and all importance-weighted binary classifiers $h : X \times \{1, \ldots T\} \to \{0, 1\}$:

$$\underbrace{\mathbf{E}_{(\mathbf{x},y)\sim D}|y - \text{Quanting}(\mathbf{x})| - \mathbf{E}_{(\mathbf{x},y)\sim D}|y - \text{median}(\mathbf{x})|}_{\text{regret in estimating the median}} \leq \underbrace{\ell(h, D') - \min_{h'} \ell(h', D')}_{\text{importance-weighted regret}}$$

where D' is the importance-weighted binary distribution induced by Quanting on D.

Costing theorem: importance-weighted loss (regret) $= \langle c \rangle \times$ error rate (regret) on the induced binary problem. We have $\langle c \rangle \leq 1$.

Corollary: regret in estimating the median $\leq$ binary regret.

What can you do with a binary classifier learner?

- Importance-weighted classification
- Quantile regression
- (Cost-sensitive) multi-class classification
- Ranking
- Learning with partial feedback

What other loss functions can we optimize with a binary classifier learner?

- *Any* loss function defined on individual examples (assuming discrete, but potentially very large prediction and label spaces)
- A broad class of ranking loss functions (defined on *subsets* of examples)

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

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

One-Against-All (OAA)

Create k binary problems, one per class.
For class i predict "Is the label i or not?"

$$(x, y) \longmapsto \begin{cases} (x, \mathbf{1}(y = 1)) \\ (x, \mathbf{1}(y = 2)) \\ \dots \\ (x, \mathbf{1}(y = k)) \end{cases}$$

Multiclass prediction: evaluate all the classifiers and randomize over those that predict "yes" (or over all if all answers are "no").

Instead, can learn a single classifier f: "Given (x, i), is i the label of x?" Predict according to $\arg\max_i f(x, i)$.

Induced distribution Q: Draw $(x, y) \sim D$, draw a random $i \in \{1, \dots, k\}$, output $((x, i), \mathbf{1}(y = i))$.

How does OAA Transform Errors?

OAA Theorem

For all multiclass distributions D and binary classifiers f,

$$\ell_k(\arg\max_i f(x,i), D) \leq (k-1)\ell_2(f, Q),$$

where Q is the induced distribution.

Proof

- A false negative (FN), no false positives (FP): f induces error rate $(k-1)/k$ with a single binary mistake
- $q > 0$ FP, no FN: f induces error rate $q/(q+1)$ with q binary mistakes (OAA chooses among $q+1$ labels, only one is correct), or $1/(q+1)$ per mistake
- $q > 0$ FP + FN: OAA errs all the time, so f induces error rate $1/(q+1)$ per mistake

Worst case: $(k-1)/k$ times k (there are k opportunities to err).

OAA

Analysis-driven improvement

- Observation: a FN (predicting "no" when the correct label is "yes") is much more disastrous than a FP.
- Use importance weights ($k/2$ for FP and $k-1$ for FN) and compose with Costing: factor of 2 improvement (in theory, and consistent improvement in practice).

Lack of robustness: Dependence on k.

OAA

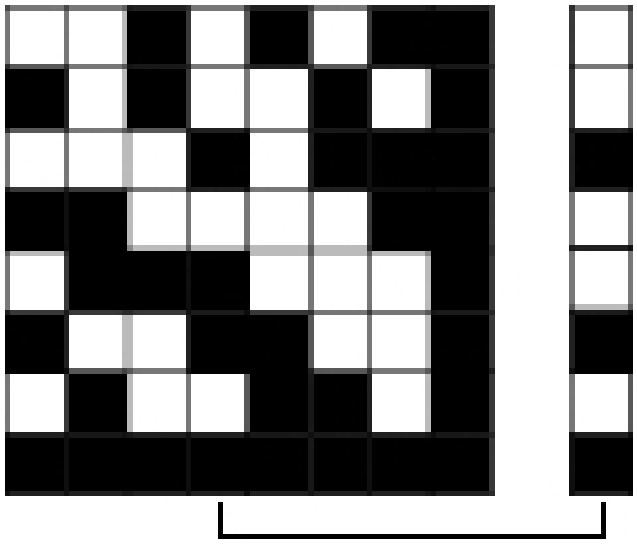
Analysis-driven improvement

- Observation: a FN (predicting "no" when the correct label is "yes") is much more disastrous than a FP.
- Use importance weights ($k/2$ for FP and $k-1$ for FN) and compose with Costing: factor of 2 improvement (in theory, and consistent improvement in practice).

Lack of robustness: Dependence on k.

Error Correcting Output Codes (Dietterich and Bakiri '95): tolerate a constant fraction of binary errors.

Error Correcting Output Codes (ECOC): Multiclass to Binary



One column per class; every two columns differ in at least d positions.

Each row defines a binary problem: "Is the label in the subset defined by the row?"

Decode the vector of binary predictions to the Hamming-closest column.

OAA = ECOC with diagonal matrix

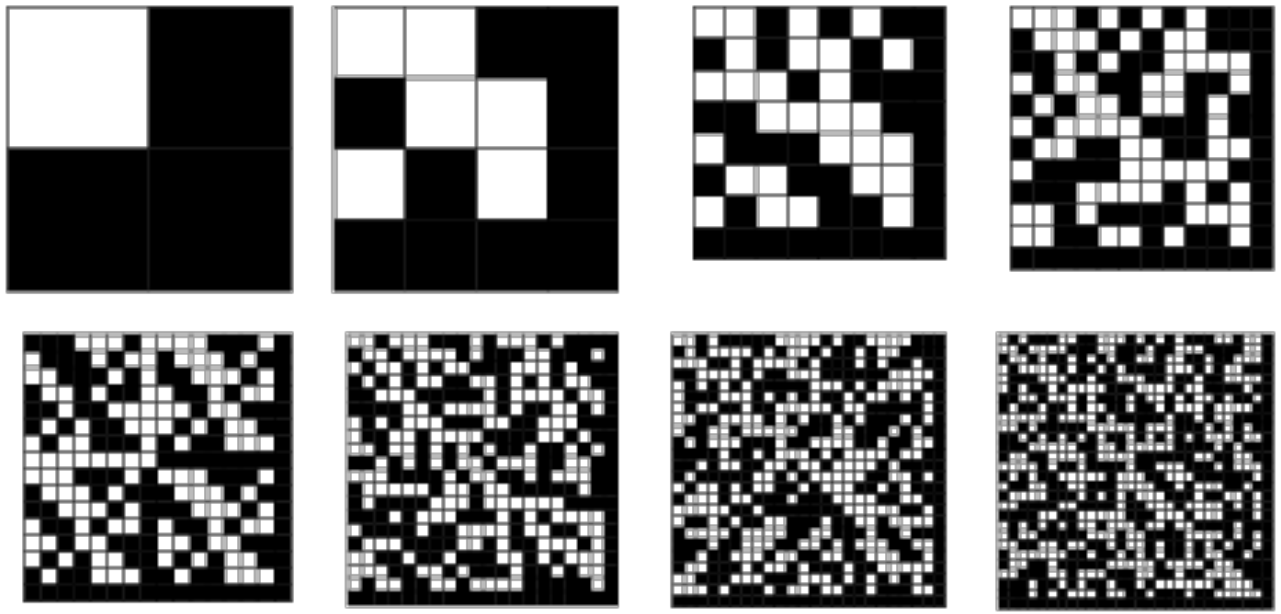
Theorem

For all multiclass problems, for all binary classifiers, ECOC has multiclass loss $\leq 2m\epsilon/d$, where ϵ is the binary error rate.

Proof: At least $d/2$ binary classifiers must err to cause a multiclass error (rate $2/d$ per binary mistake); m opportunities to err. Thus ϵ binary loss can induce at most $2m\epsilon/d$ multiclass loss.

ECOC with Hadamard Matrices

$d = m/2$, thus multiclass loss is at most 4ϵ.



If H is a Hadamard matrix, so is

$$\begin{pmatrix} H & H \\ H & -H \end{pmatrix}$$

Bounding regret

Fear: Error transforms may be vacuous

1. ECOC can create hard binary problems
2. The original problem is inherently noisy, and so is the induced problem.

Regret of classifier f on distribution D with respect to loss function ℓ is

$$r(f, D) = \ell(f, D) - \underbrace{\min_{f^*} \ell(f^*, D)}_{\text{smallest achievable loss on } D}$$

Subtract off inherent noise in both problems. Bound only *excess* loss due to suboptimal prediction.

If we have an optimal binary classifier for the induced problem, we better have an optimal solution to the original problem.

A problem with OAA and ECOC

Inconsistency

Given an optimal binary classifier, the reduction doesn't produce an optimal multiclass classifier.

Example (no majority class):

$\frac{1}{2} - \delta$	$\frac{1}{4} + \frac{\delta}{2}$	$\frac{1}{4} + \frac{\delta}{2}$	optimal predictions
1	0	0	no
0	1	0	no
0	0	1	no

Randomizing over the three labels results in multiclass regret $\frac{2}{3}(\frac{1}{2} - \delta - (\frac{1}{4} + \frac{\delta}{2})) > \frac{1}{8}$

Regret Transforms

OAA: regress on the conditional probability of each label and predict according to the argmax of the predicted probabilities.

PECOC: regress on the probability of being in each subset; decode to the l_1 closest column (label).

	Regret	Computation per example
regression-OAA	$\sqrt{2(k-1)r}$	$O(n)$
PECOC	$4\sqrt{r}$	$O(n^2)$

Regression can be reduced to binary classification using the Probing reduction, with

$$\text{squared error regret} \leq \text{classification regret}$$

Questions:

1. Is there a consistent reduction that does not have a $\sqrt{r}$ dependence?
2. Is there a consistent reduction that requires just $O(\log k)$ computation per example (matching the lower bound)?
3. Can the above be achieved with a reduction that performs only pairwise comparisons between classes?

Questions:

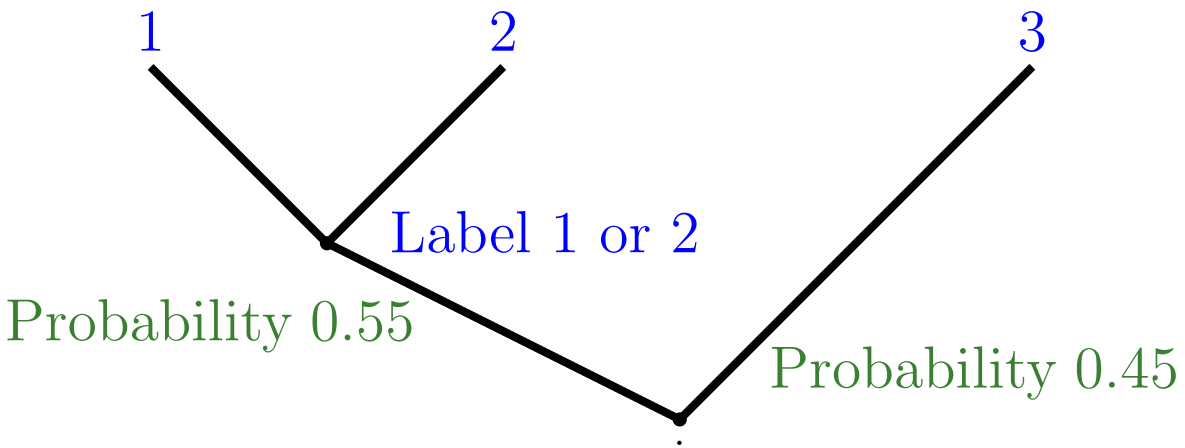
1. Is there a consistent reduction that does not have a $\sqrt{r}$ dependence?
2. Is there a consistent reduction that requires just $O(\log k)$ computation per example (matching the lower bound)?
3. Can the above be achieved with a reduction that performs only pairwise comparisons between classes?

Error-correcting tournaments:

Regret bound	Computation
$5.5r$	$O(\log k)$
$4r$	$O(k)$

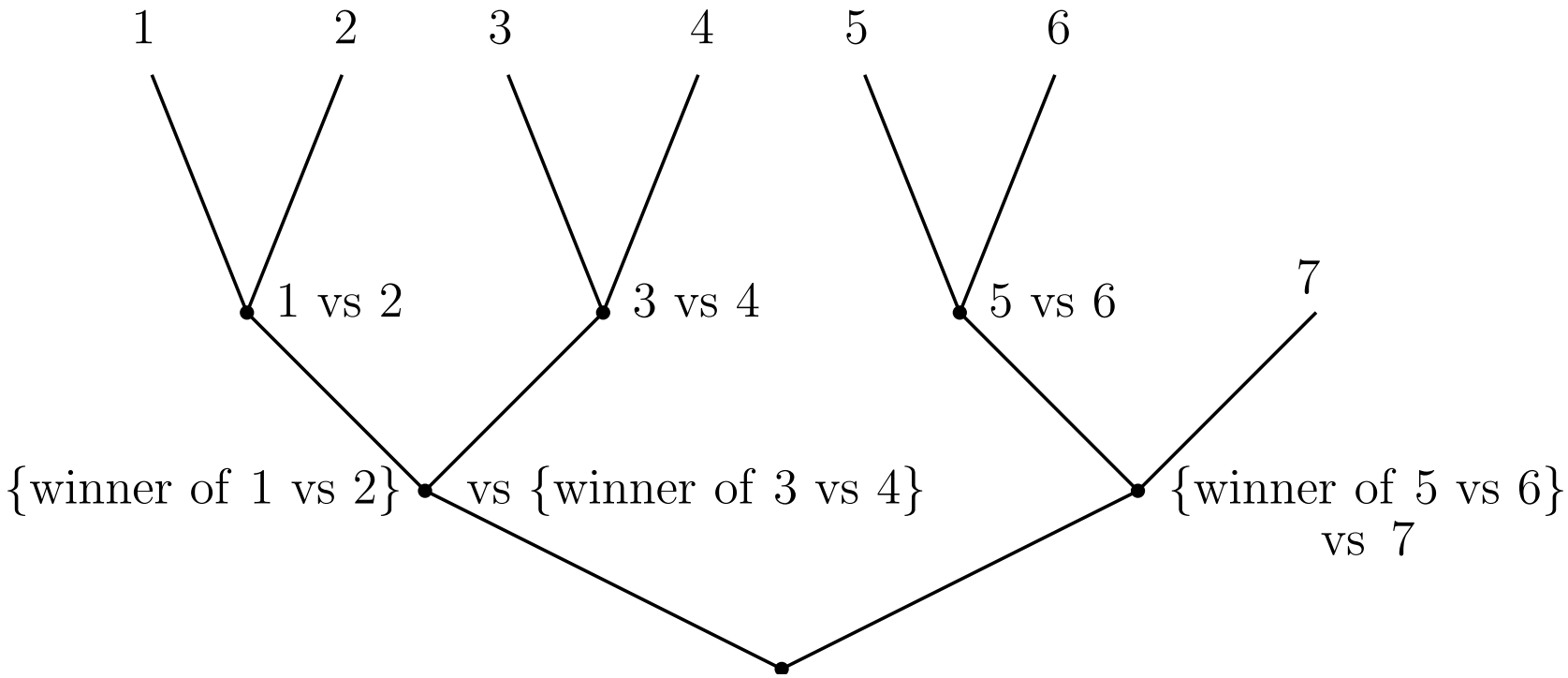
Tree Approach

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



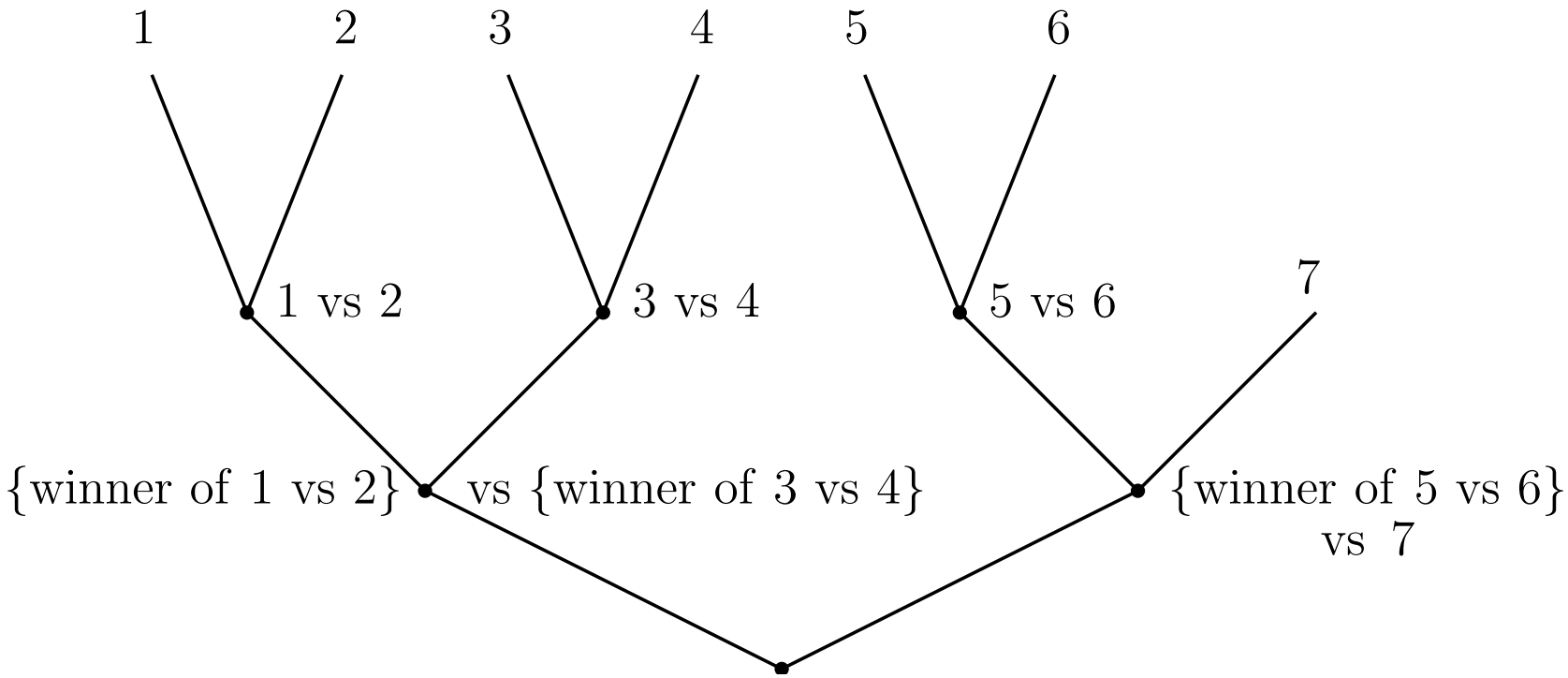
Optimal classifier picks 1 or 2, but choice 3 is best.

The Filter Tree (Single-Elimination Tournament)



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

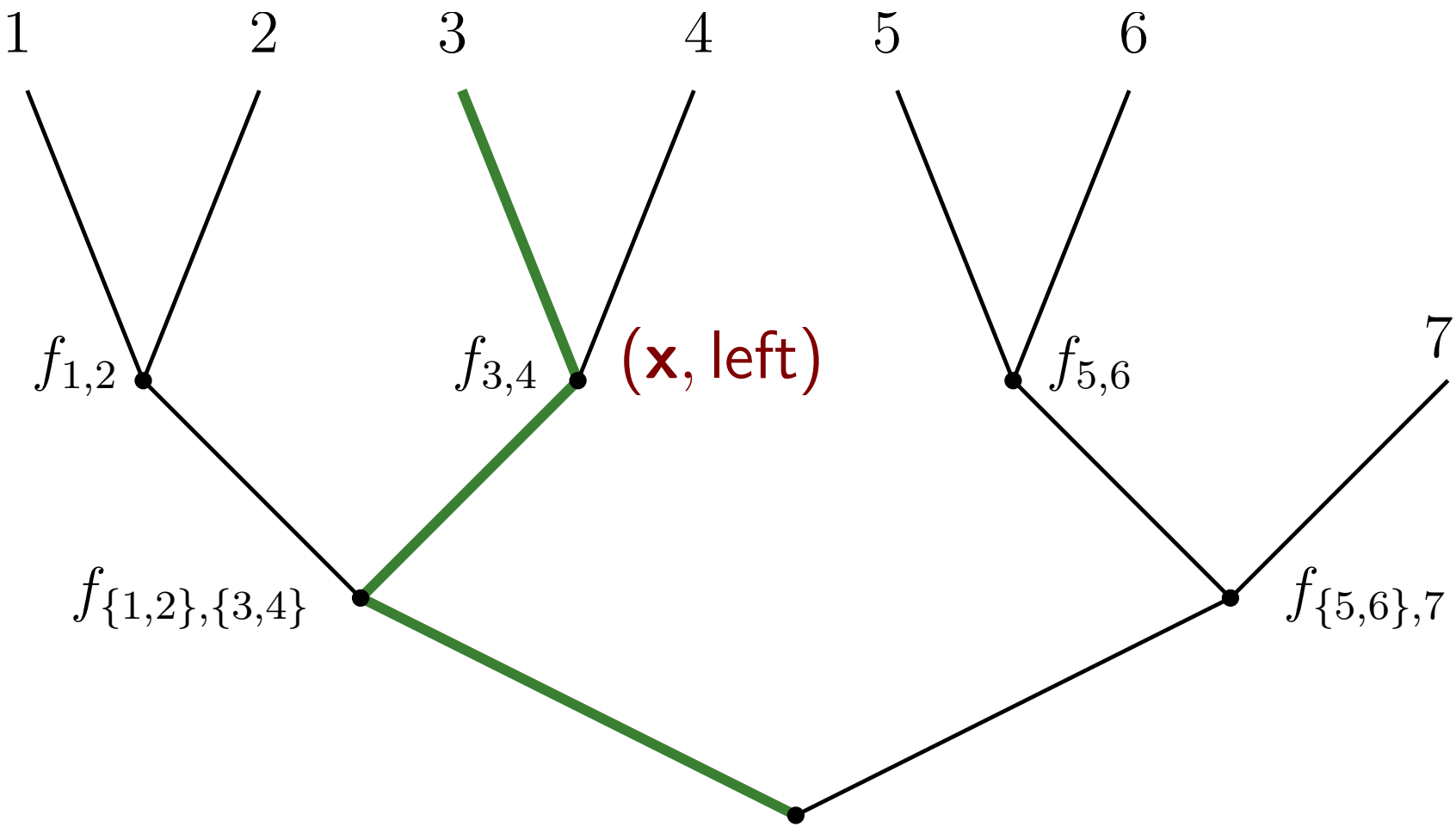
The Filter Tree (Single-Elimination Tournament)



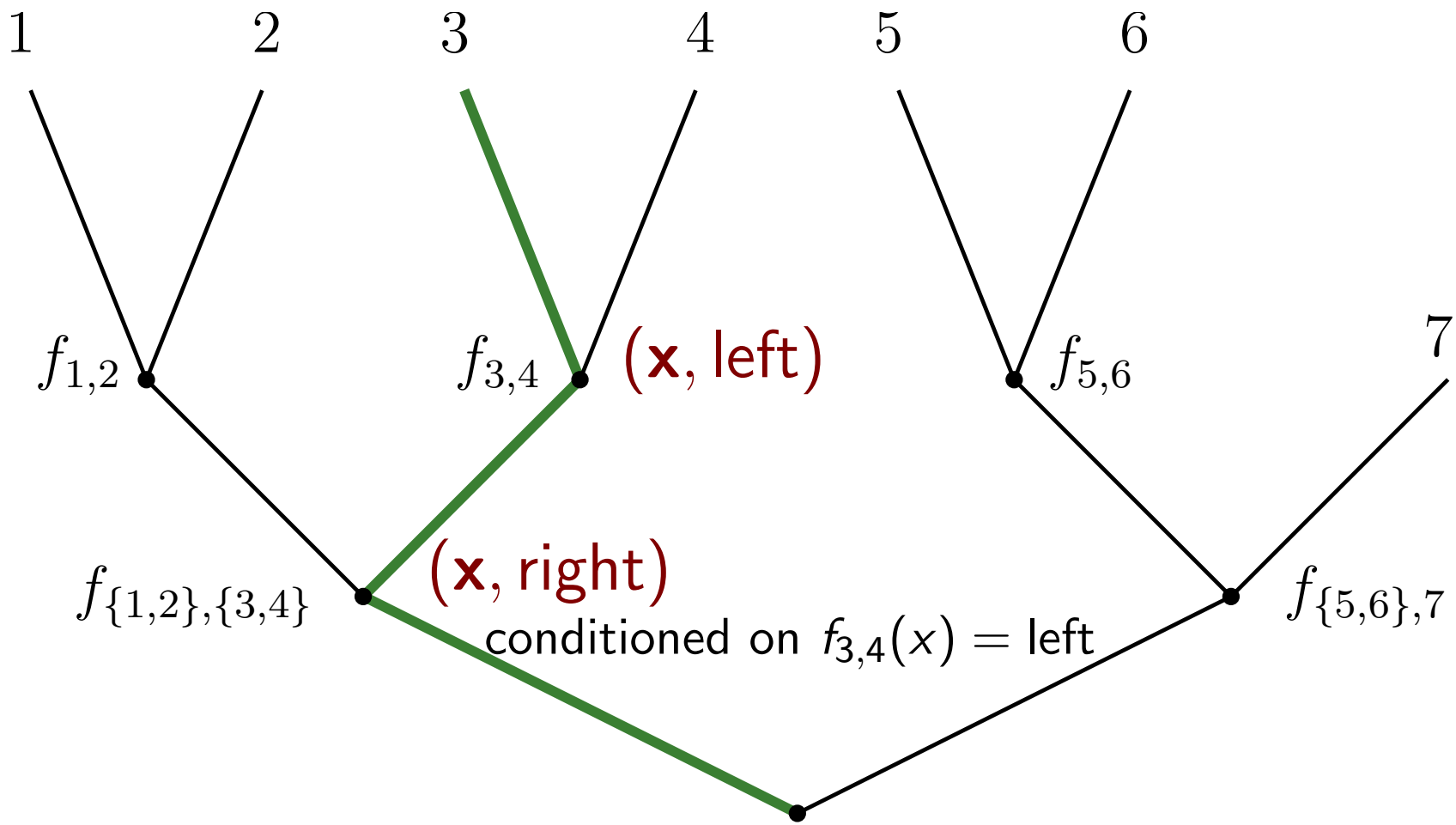
Each non-leaf predicts the best of a pair of winners from the previous round

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

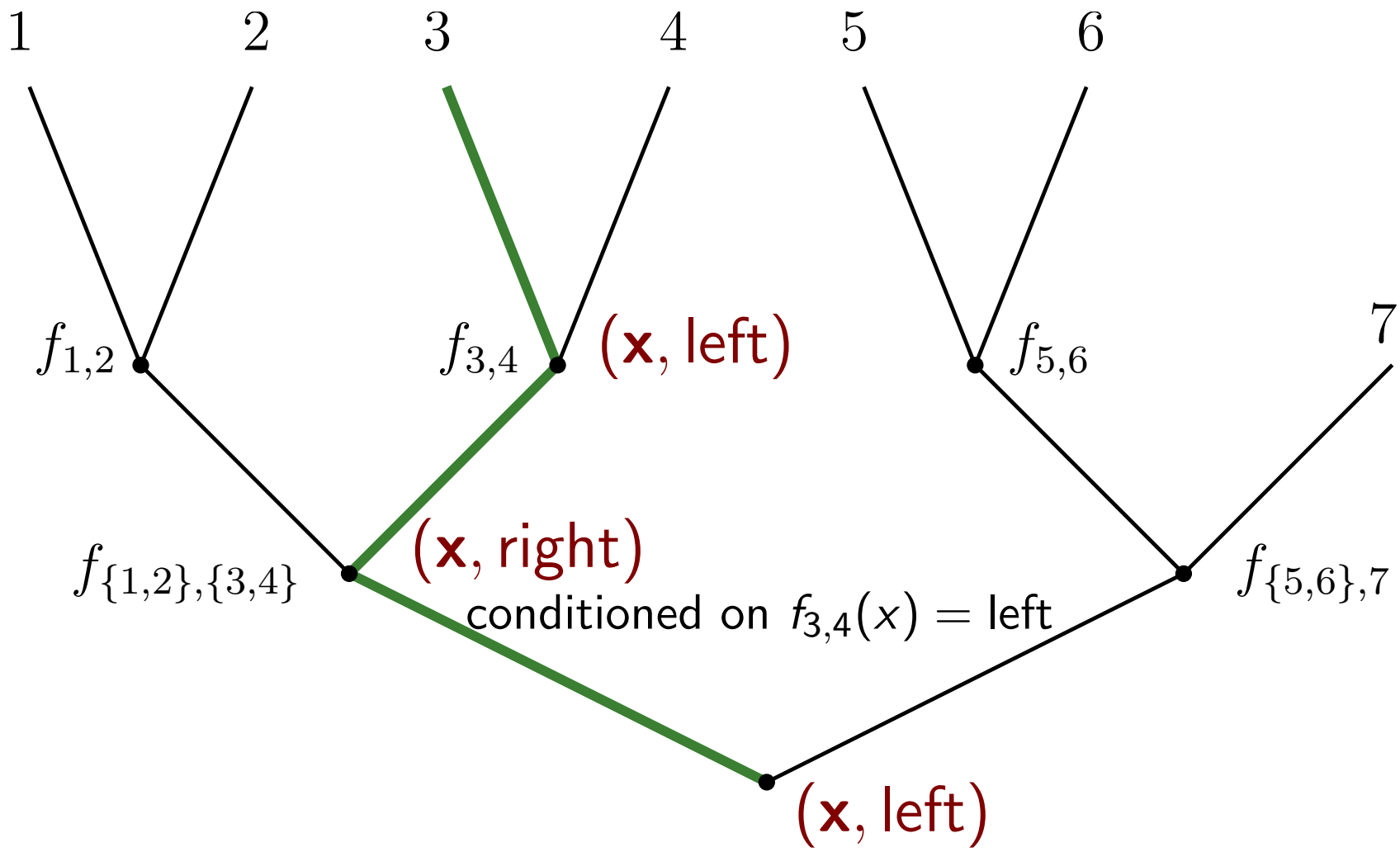
Training on example $(\mathbf{x}, \mathbf{3})$



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.
- Multiclass to binary regret ratio bounded by $\lceil \log k \rceil$.
- Next: generalization to the cost-sensitive case.

Cost-sensitive multi-class classification

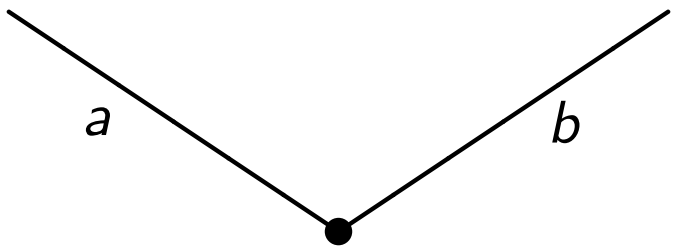
Distribution D over $X \times [0,1]^k$, where a vector in $[0,1]^k$ specifies the cost of each of the k choices.

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 $|c_a - c_b|$. Compose with Costing (cost-proportionate rejection sampling) to remove the weights.

Distribution induced at the node

Draw a cost-sensitive example from D, create an importance weighted sample as above, and resample according to Costing.

Analysis

Theorem

For all cost-sensitive problems and classifiers at the nodes, the resulting cost-sensitive regret $\leq$ average binary regret, times the expected sum of weights over the nodes.

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

Analysis

Theorem

For all cost-sensitive problems and classifiers at the nodes, the resulting cost-sensitive regret $\leq$ average binary regret, times the expected sum of weights over the nodes.

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

Computational analysis:

	computation per example
Cost-sensitive training	$O(k)$
Multiclass training	$O(\log k)$
Testing	$O(\log k)$

The multiclass case

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

There is no need to apply Costing since all weights are either 0 (example is filtered out) or 1.

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 $\lceil \log k \rceil$.

There is no need to apply Costing since all weights are either 0 (example is filtered out) or 1.

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 $\lceil \log k \rceil$.

There is no need to apply Costing since all weights are either 0 (example is filtered out) or 1.

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 $\lceil \log k \rceil$.

There is no need to apply Costing since all weights are either 0 (example is filtered out) or 1.

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

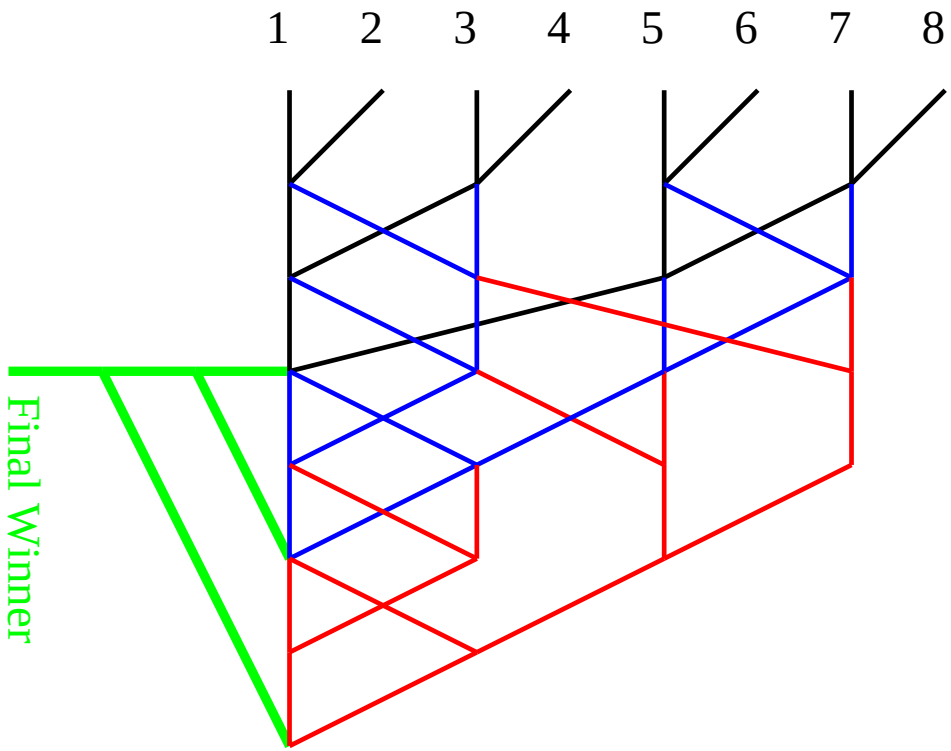
Can we make it more robust?

- 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.



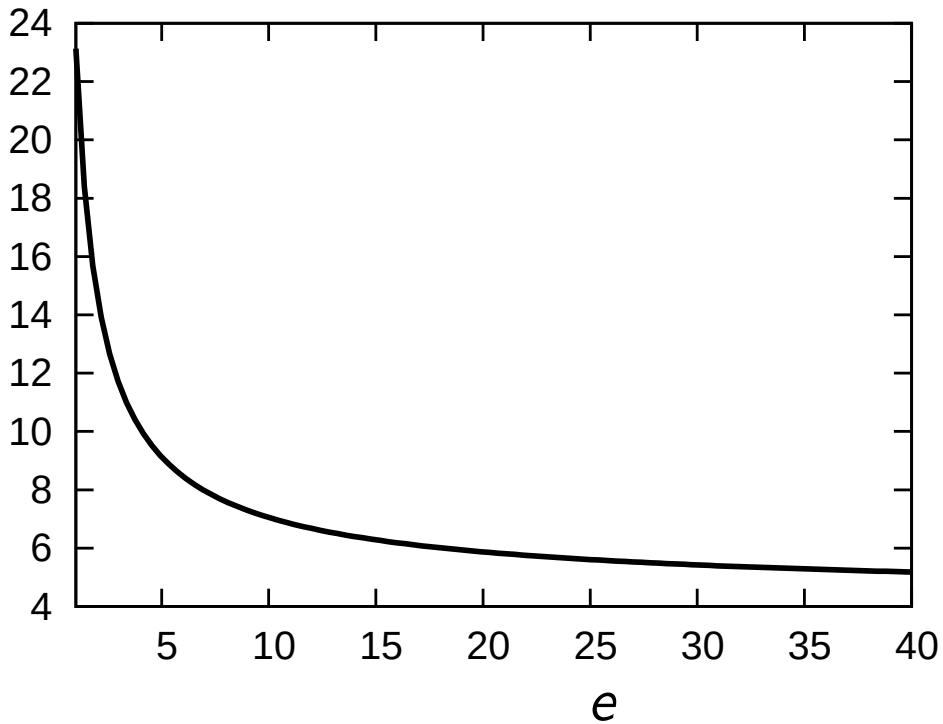
$e = 3$

e-Elimination Tournaments

$O(e + \log k)$ computation per multiclass example.

For $m \leq 4 \log k$, regret ratio is bounded by

$$4 + 2\frac{\ln k}{e} + 2\sqrt{\frac{\ln k}{e}}.$$



e-Elimination Tournaments

Summary:

1. A consistent reduction from multiclass to binary classification. For $e = 4 \ln k$, regret ratio is upper bounded by 5.5.

2. Computation required is just $O(\log k)$ per multiclass example.

3. The binary problems involve only pairwise comparisons between classes.

What can you do with a binary classifier learner?

- Importance-weighted classification
- Quantile regression
- (Cost-sensitive) multi-class classification
- Ranking (next)
- Learning with partial feedback

Ranking

Given a subset $S \subset U$ of elements (from some uknown distribution over subsets), sort S so that after the labels are revealed all the elements are sorted.

AUC loss

Normalized bubble sort distance to the sorted order

Two general approaches:

1. Learn a *scoring function* $f : U \to \mathbb{R}$, inducing a total linear ordering of all of U. Sort S according to f.
2. Learn a *classifier* h to predict, given two elements in U: Should the first be ranked above the second? Sort S using h, noting that h may not induce a linear ordering on S. Motivation: Good total linear ordering may be impossible to achieve, while such an h can be learned well.

Results in the Bipartite Case

Randomized reduction:

Quicksort (Ailon and Mohri)

Expected AUC regret is bounded by the classification regret.

Deterministic reduction:

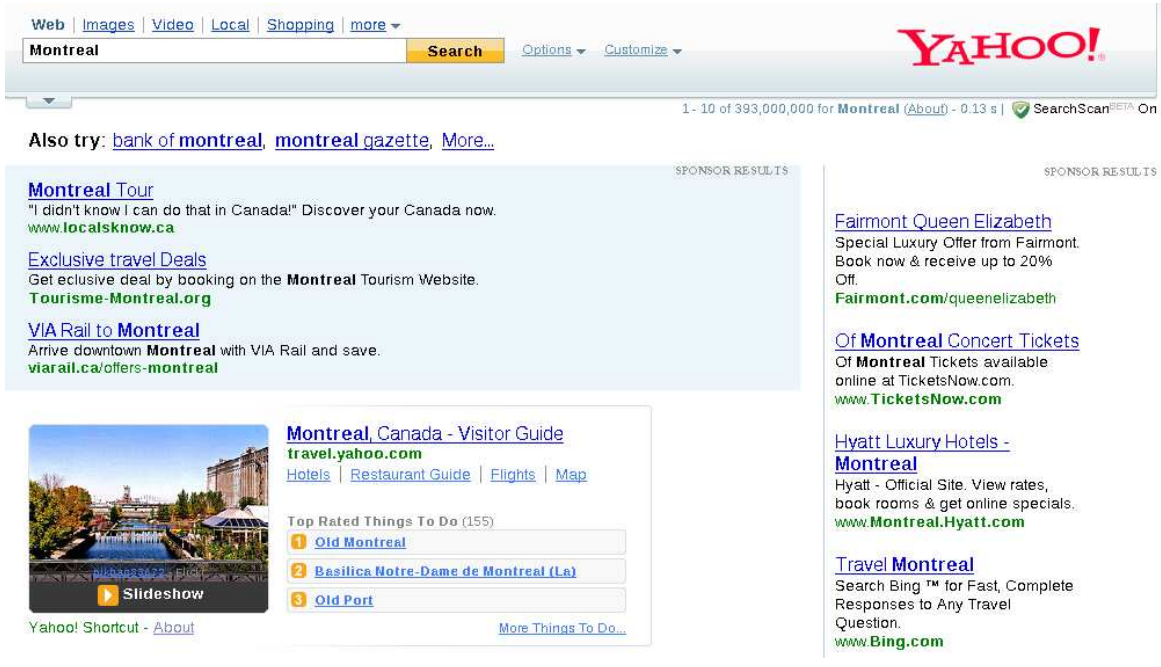
Order by the number of wins

Worst-case AUC regret is bounded by twice the classification regret.

What can you do with a binary classifier learner?

- Importance-weighted classification
- Quantile regression
- (Cost-sensitive) multi-class classification
- Ranking
- Learning with partial feedback (next)

One Way Yahoo! Makes Money



1. A user with some hidden interests make a query on Yahoo.
2. Yahoo chooses an ad to display.
3. The user either clicks on the ad or not (= a payoff to Yahoo or not).

Lots of other details: computational, network, adaptivity constraints. Multiple ads.

A Mathematical Description

1. The world chooses $(x, r_1, ..., r_k)$ and reveals x.
2. You choose a in $\{1, ..., k\}$.
3. The world reveals r_a.

Loss is unknown even at training time! Exploration required, but still simpler than reinforcement learning.

Solution Approaches

1. Argmax Regression
2. Importance Weighted Classification
3. Offset Tree

The Regression Approach

Important fact: the minimizer of squared error is the conditional mean.

1. Learn a regressor f to predict r_a given (x, a).
2. Let $\pi_f(x) = \arg\max_a f(x, a)$

Regression Analysis

Induced distribution D': draw $(x, r_1, ..., r_k) \sim D$, then choose a random $a \in \{1, \ldots, k\}$ to get $((x, a), r_a)$.

Squared error regret of f on D':

$$\text{reg}_{\text{sq}}(f, D') = E_{(x,a)\sim D'}(f(x, a) - f^*(x, a))^2$$

where $f^*(x, a)$ is the mean r_a given (x, a).

Regret of π_f on D: $\text{reg}(\pi_f, D) = E_{(x,\vec{r})\sim D}\left[r_{\pi^*(x)} - r_{\pi_f(x)}\right]$, where π^* is the optimal policy.

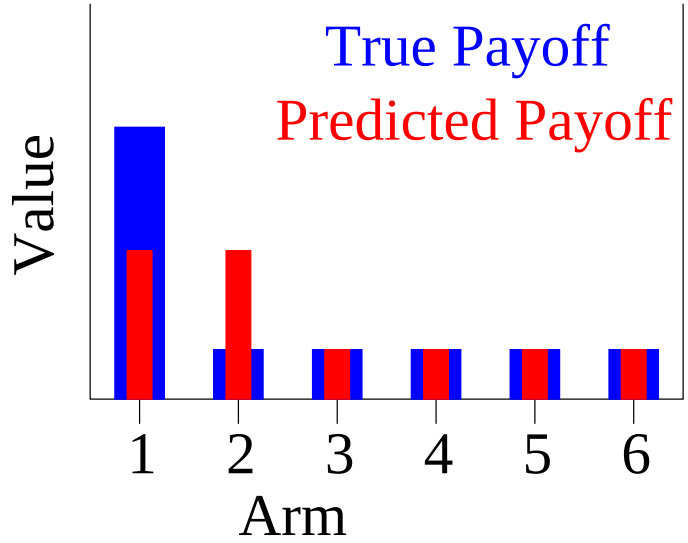
Theorem

For all D and f:

$$\text{reg}(\pi_f, D) \leq \sqrt{2k \cdot \text{reg}_{\text{sq}}(f, D')}$$

$$(\text{policy regret} \leq \sqrt{2k \cdot \text{binary regret}})$$

Proof sketch: Fix x. Worst case:



The regressor's squared error regret is $2\left(\frac{E_{\vec{r}\sim D|x}\left[r_{h^*(x)}-r_{h(x)}\right]}{2}\right)^2$, out of k regression estimates. Thus the average squared error regret is

$$\frac{1}{2k}\left(E_{\vec{r}\sim D|x}\underbrace{\left[r_{h^*(x)}-r_{h(x)}\right]}_{\text{policy regret}}\right)^2.$$

Solving for policy regret, finishes the proof.

Solution Approaches

1. Argmax Regression
2. Importance Weighted Classification
3. Offset Tree

Importance-Weighted Classification Approach (Z'03)

Training:

1. For each (x, a, r) example, create an importance weighted multiclass example (x, a, rk).
2. Reduce importance weighted multiclass to binary using Costing and ECT for multiclass to binary reduction.

Testing:
Make a multiclass prediction.

Importance-Weighted Classification Analysis

Let D' = induced binary distribution.

Theorem

For all D and binary classifiers b,

$$\text{policy regret} \leq 4k\text{reg}(b, D').$$

Importance-Weighted Classification Analysis

Let D' = induced binary distribution.

Theorem

For all D and binary classifiers b,

$$\text{policy regret} \leq 4k\text{reg}(b, D').$$

Proof: A uniform random $a \in \{1, ..., k\}$ implies the expected importance weighted cost for choosing a instead of a' is: $\frac{1}{k}(r_a k - r_{a'} k) = r_a - r_{a'}$ = policy regret.

Compose with costing $\Rightarrow$ multiply by $Er_a k \leq k$.

Compose with ECT $\Rightarrow$ multiply by 4.

Solution Approaches

1. Argmax Regression
2. Importance Weighted Classification
3. Offset Tree

The Offset Tree for $k = 2$

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

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

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

Denoising Binary Importance Weighting

Theorem

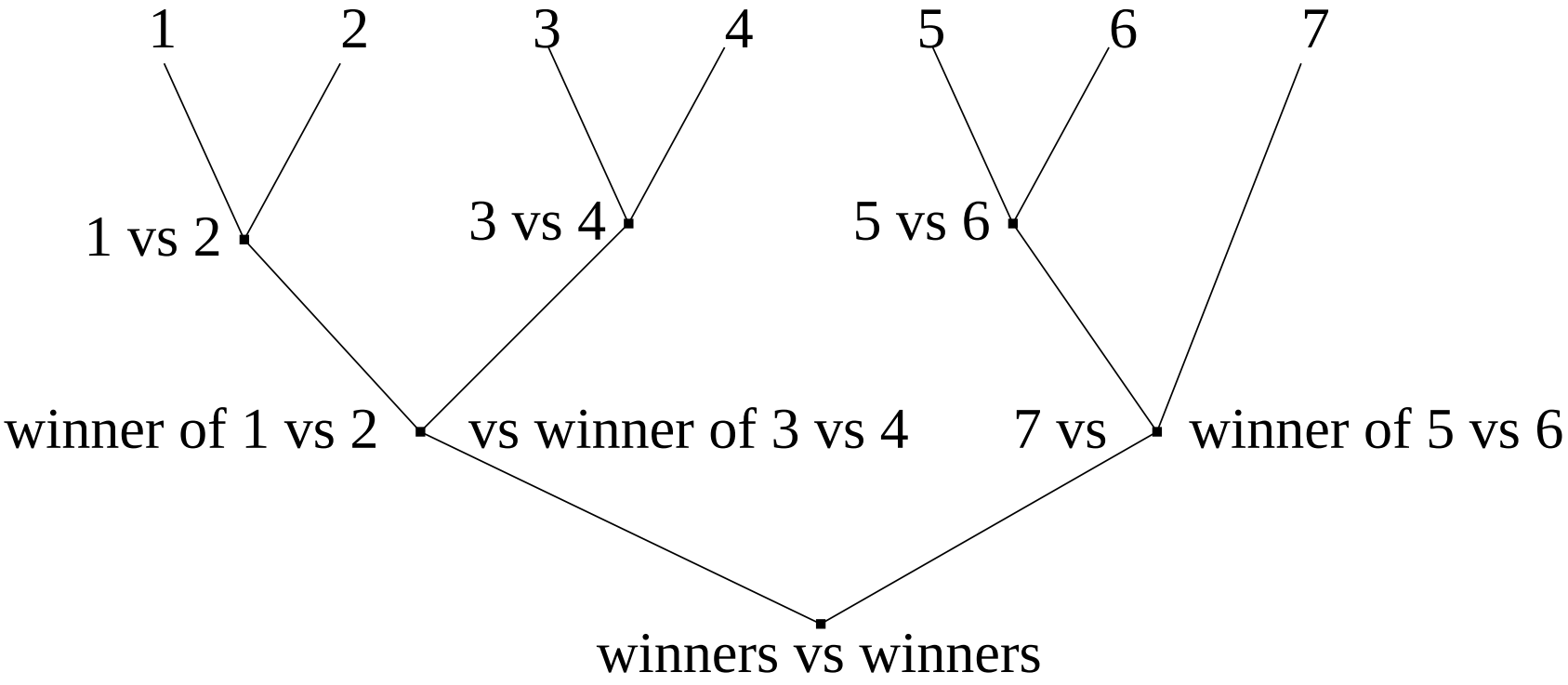
For all distributions D with $k = 2$, for all binary classifiers b:

$$\text{policy regret} \leq \text{reg}(b, D').$$

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' = random binary problem according to chance that binary problem is fed an example.
b = the classifier which predicts based on both x and the choice of binary problem according to D'.

Theorem

For all k-choice D, binary classifiers b:

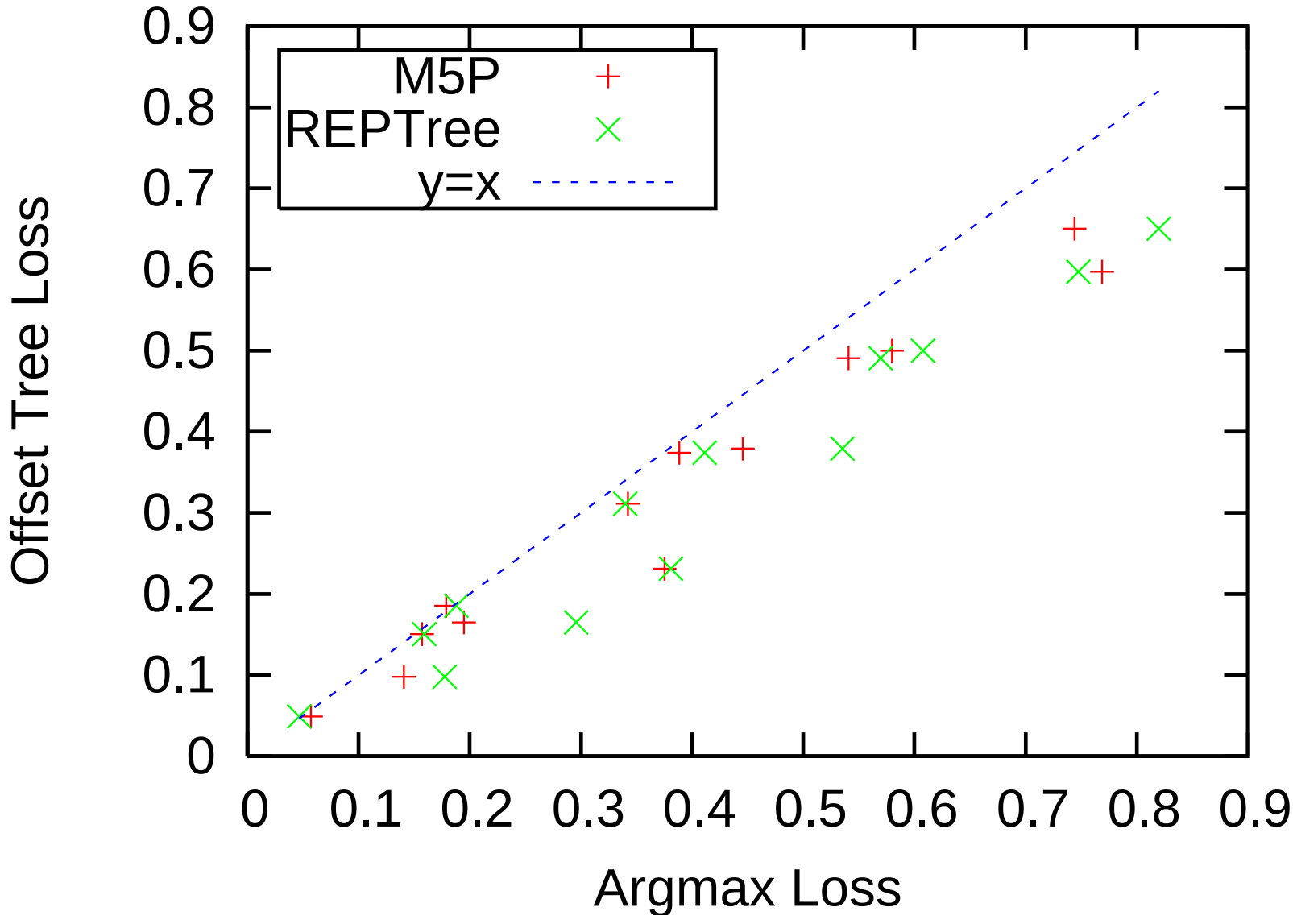
$$\text{policy regret} \leq (k-1)\text{reg}(b, D').$$

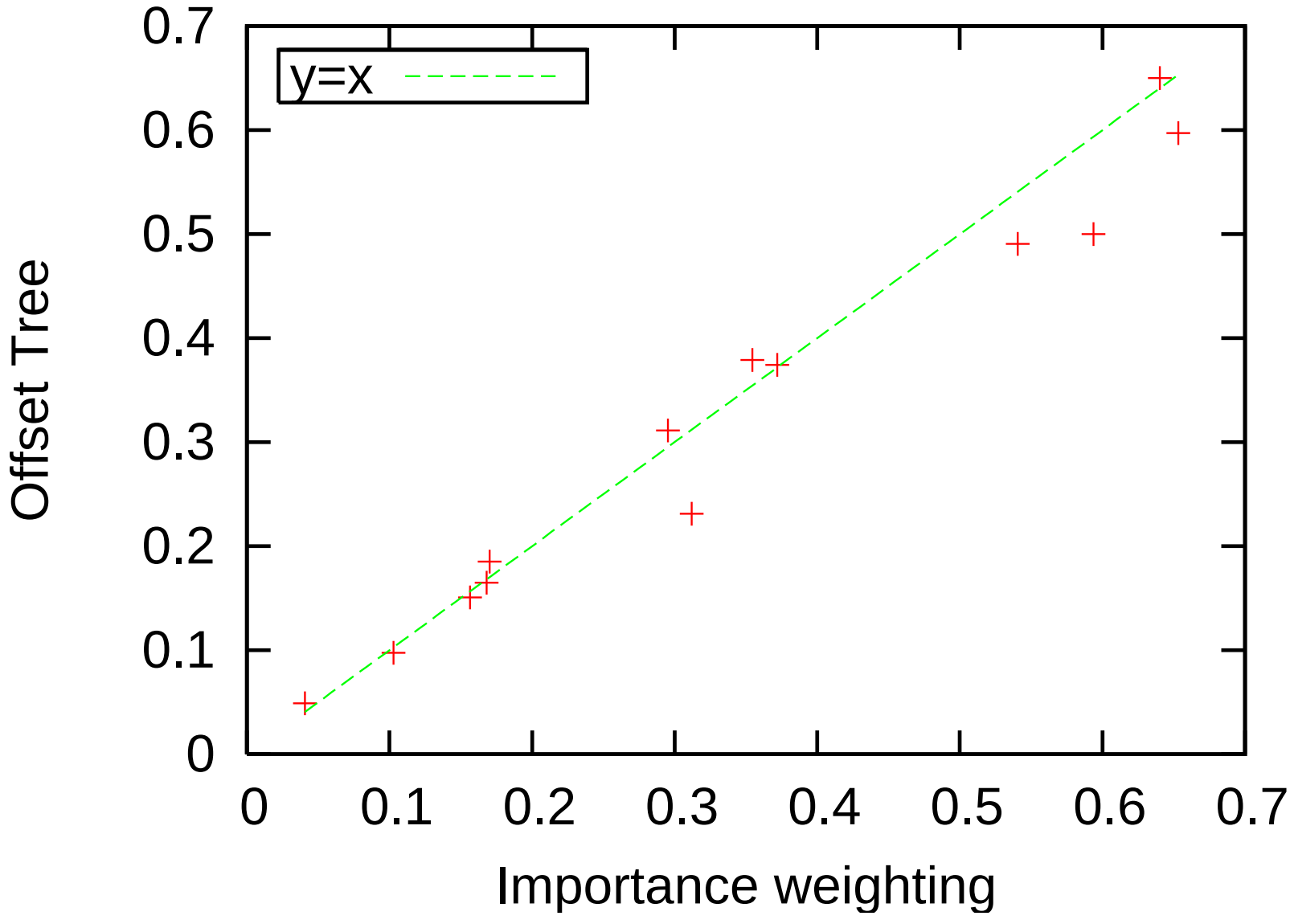
A dependence on k, but at least not k^2.

A Comparison of Approaches

Algorithm	Policy Regret Bound
Argmax Regression	$\sqrt{2k\text{reg}(s, D_{\text{AR}})}$
Importance Weighted Classification	$4k\text{reg}(b, D_{\text{IWC}})$
Offset Tree	$(k-1)\text{reg}(b, D_{\text{OT}})$

How do you expect things to work, experimentally?

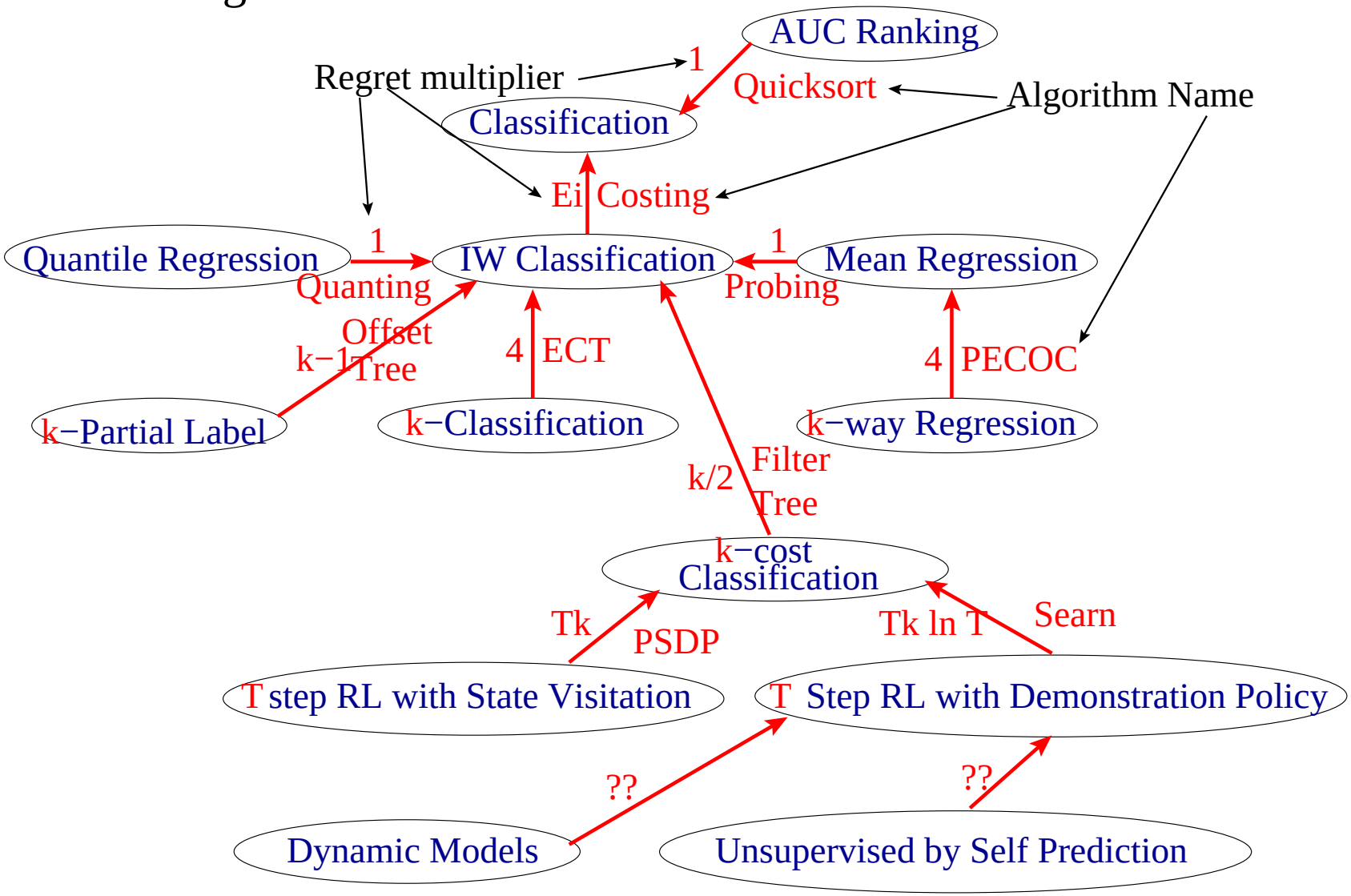

M5P
REPTree
y=x
Offset Tree Loss
Argmax Loss
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


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

The Reduction Viewpoint

1. Reductions = tool for machine learning architect.
2. Good reductions *preserve* prediction ability.
3. After you have a reduction use {multitask, active, semisupervised, Bayesian, deep} learning to achieve best-possible prediction ability.
4. A new definition of learnable: we can learn if we can tightly reduce to binary.

Regret Transform Reductions



Future Directions

Primary value: Providing fast effective solution outlines to complex prediction problems. (See Hal Daume's ICML talk for an example.)

Theoretical questions

1. Is there a better notion of learning reduction?
2. Can maximally robust reductions have minimal computational requirements?
3. What are the {computaional, robustness} limits of adaptive vs. nonadaptive reductions?