Logarithmic Time Prediction

John Langford

Microsoft Research

DIMACS Workshop on Big Data through the Lens of Sublinear Algorithms

The Multiclass Prediction Problem

Repeatedly

1. See x
2. Predict $\hat{y} \in \{1, ..., K\}$
3. See y

The Multiclass Prediction Problem

Repeatedly

1. See x
2. Predict $\hat{y} \in \{1, ..., K\}$
3. See y

Goal: Find $h(x)$ minimizing error rate:

$$\Pr_{(x,y)\sim D}(h(x) \neq y)$$

with $h(x)$ fast.

Why?

https://www.bing.com/search?q=DIMACS+Workshop+on+Big+Data+through+the+Lens+of+

bing

DIMACS Workshop on Big Data through the Lens of Sublinear Algorit

Web Images Videos Maps News Explore

76,900 RESULTS Any time

DIMACS workshop on Big Data through the Lens of Sublinear …

dimacs.rutgers.edu/**Workshops**/Parallel**Algorithms**

DIMACS Workshop on Big Data through the Lens of Sublinear Algorithms August 27 - 28, 2015 **DIMACS** Center, CoRE Building, Rutgers University Organizers:

my slice of pizza: **Big Data**, **Sublinear Algorithms** ...

mysliceofpizza.blogspot.com/2015/07/**big-data-sublinear-algorithms**...

Jul 24, 2015 · Grigory Yaroslavtsev, **Alexandr Andoni** and I are organizing the **DIMACS workshop on Big Data through the Lens of Sublinear Algorithms**, Aug 27--28, at …

Big Data Through the Lens of Sublinear Algorithms

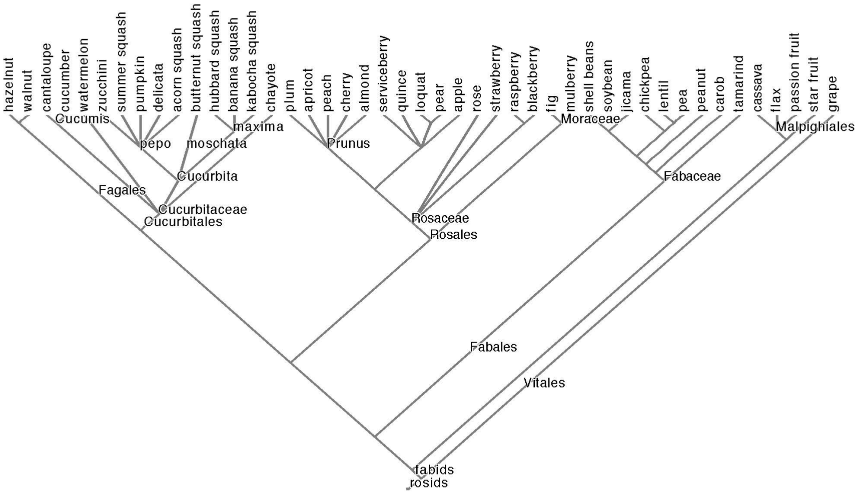
grigory.us/mpc-**workshop-dimacs**.html

Why?

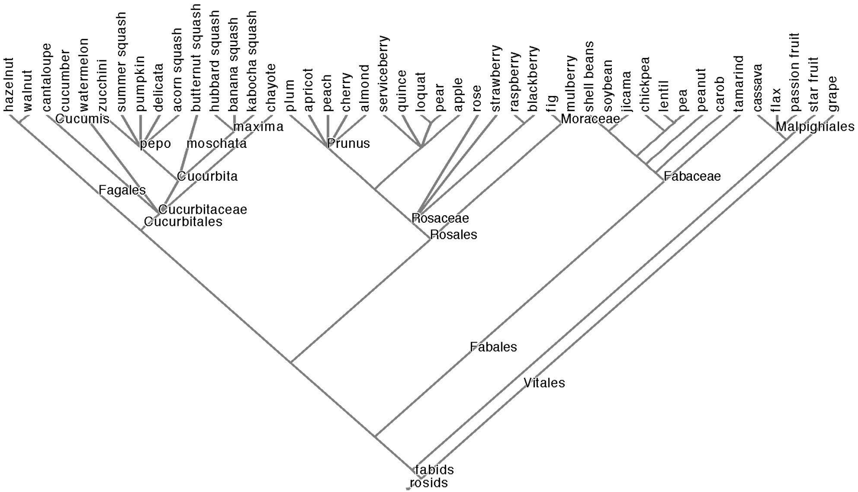
Trick #1

K is small

Trick #2: A hierarchy exists

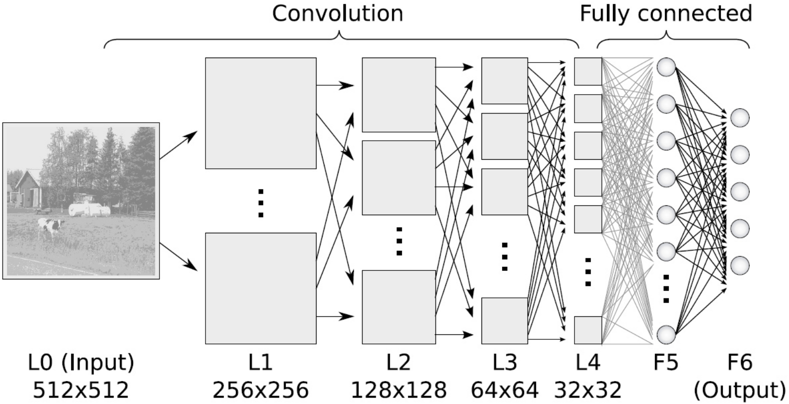


Trick #2: A hierarchy exists

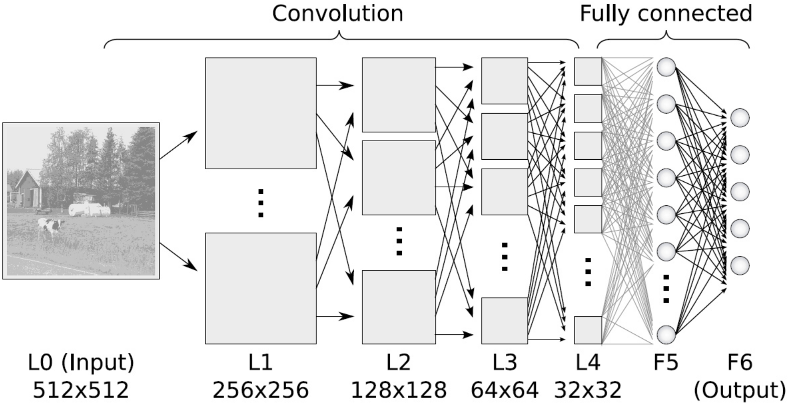


So use Trick #1 repeatedly.

Trick #3: Shared representation

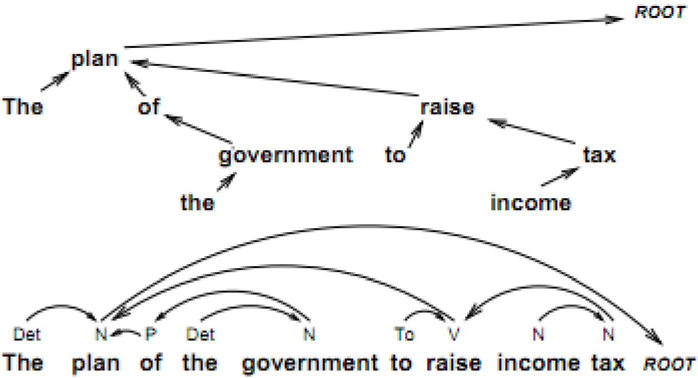


Trick #3: Shared representation

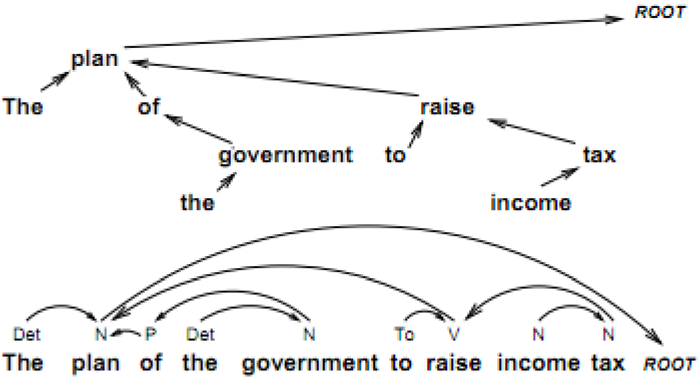


Very helpful... but computation in the last layer can still blow up.

Trick #4: "Structured Prediction"



Trick #4: "Structured Prediction"



But what if the structure is unclear?

Trick #5: GPU

Trick #5: GPU

4 Teraflops is great… yet still burns energy.

How fast can we hope to go?

How fast can we hope to go?

Theorem: There exists multiclass classification problems where achieving 0 error rate requires $\Omega(\log K)$ time to train or test per example.

How fast can we hope to go?

Theorem: There exists multiclass classification problems where achieving 0 error rate requires $\Omega(\log K)$ time to train or test per example.

Proof: By construction

Pick $y \sim U(1, ..., K)$

How fast can we hope to go?

Theorem: There exists multiclass classification problems where achieving 0 error rate requires $\Omega(\log K)$ time to train or test per example.

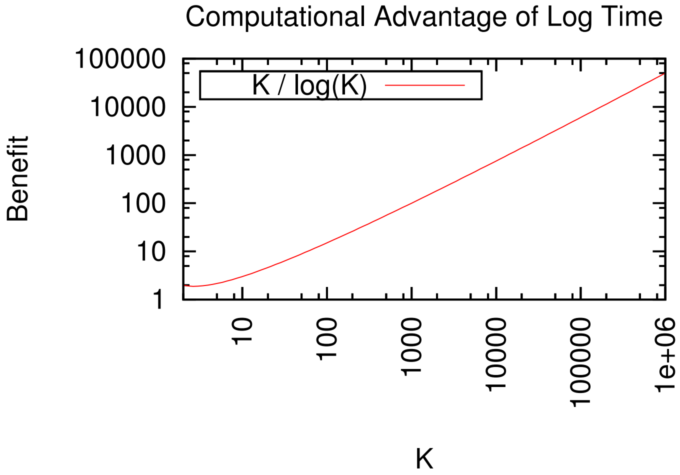
Proof: By construction

Pick $y \sim U(1, ..., K)$

Any prediction algorithm outputting less than $\log_2 K$ bits loses with constant probability.

Any training algorithm reading an example requires $\Omega(\log_2 K)$ time.

Can we predict in time $O(\log_2 K)$?



Not it #1: Sparse Error Correcting Output Codes

1. Create $O(\log K)$ binary vectors b_{iy} of length K

Not it #1: Sparse Error Correcting Output Codes

1. Create $O(\log K)$ binary vectors b_{iy} of length K
2. Train $O(\log K)$ binary classifiers h_i to minimize error rate: $\Pr_{x,y}(h_i(x) \neq b_{iy})$

Not it #1: Sparse Error Correcting Output Codes

1. Create $O(\log K)$ binary vectors b_{iy} of length K
2. Train $O(\log K)$ binary classifiers h_i to minimize error rate: $\Pr_{x,y}(h_i(x) \neq b_{iy})$
3. Predict by finding y with minimal error.

Not it #1: Sparse Error Correcting Output Codes

1. Create $O(\log K)$ binary vectors b_{iy} of length K
2. Train $O(\log K)$ binary classifiers h_i to minimize error rate: $\Pr_{x,y}(h_i(x) \neq b_{iy})$
3. Predict by finding y with minimal error.

Prediction is $\Omega(K)$

Not it #2: Hierarchy Construction

1. Build confusion matrix of errors.

Not it #2: Hierarchy Construction

1. Build confusion matrix of errors.
2. Recursive partition to create hierarchy.

Not it #2: Hierarchy Construction

1. Build confusion matrix of errors.
2. Recursive partition to create hierarchy.
3. Apply hierarchy solution.

Not it #2: Hierarchy Construction

1. Build confusion matrix of errors.
2. Recursive partition to create hierarchy.
3. Apply hierarchy solution.

Training is $\Omega(K)$ or worse.

Not it #3: Unnormalized learning

Train K regressors by
For each example (x, y)

1. Train regressor y with $(x, 1)$.

Not it #3: Unnormalized learning

Train K regressors by
For each example (x, y)

1. Train regressor y with $(x, 1)$.
2. Pick $y' \neq y$ uniformly at random.
3. Train regressor y' with $(x, -1)$.

Not it #3: Unnormalized learning

Train K regressors by
For each example (x, y)

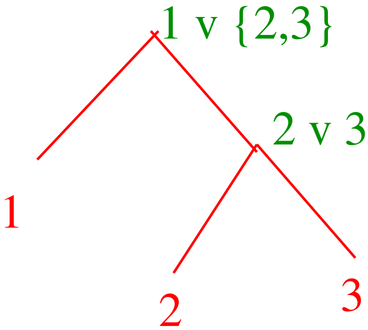
1. Train regressor y with $(x, 1)$.
2. Pick $y' \neq y$ uniformly at random.
3. Train regressor y' with $(x, -1)$.

Prediction is still $\Omega(K)$.

Can we predict in time $O(\log_2 K)$?

Is logarithmic time even possible?

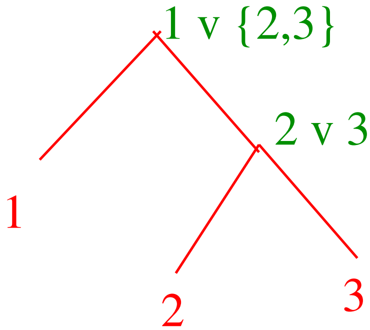
P(y=1) = .4
P(y=2) = .3
P(y=3) = .3



$P(\{2,3\}) > P(1) \Rightarrow$ lose for divide and conquer

Filter Trees [BLR09]

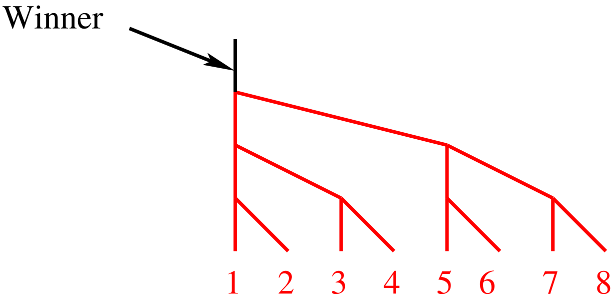
$P(y=1) = .4$
$P(y=2) = .3$
$P(y=3) = .3$



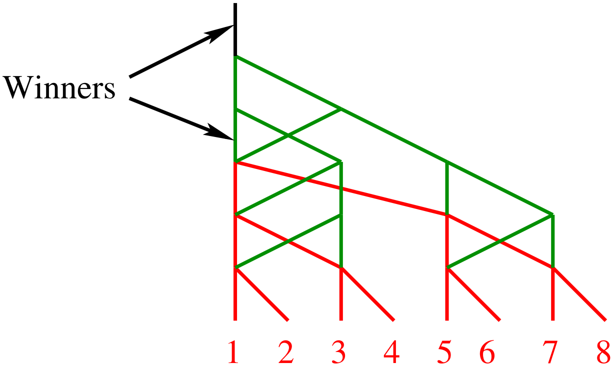
1. Learn $2v3$ first
2. Throw away all error examples
3. Learn 1 v Survivors

Theorem: For all multiclass problems, for all binary classifiers,
Multiclass Regret $\leq$ Average Binary Regret * $\log(K)$

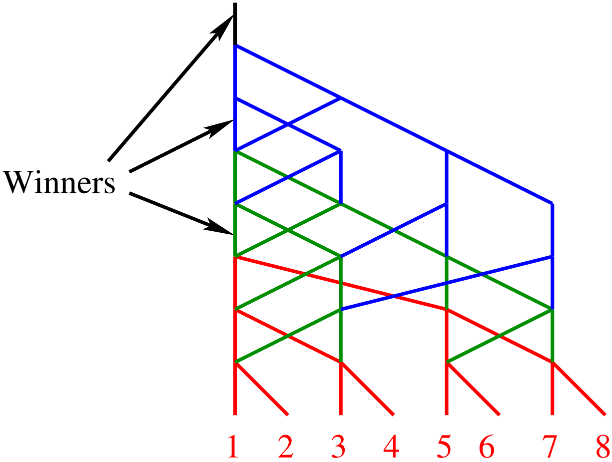
Can you make it robust?



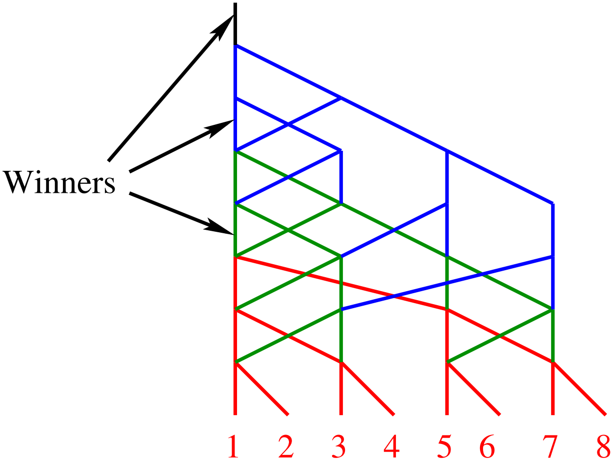
Can you make it robust?



Can you make it robust?



Can you make it robust?



Theorem: [BLR09] For all multiclass problems, for all binary classifiers, a log(K)-correcting tournament satisfies:

Multiclass Regret $\leq$ Average Binary Regret * 5.5

Determined best paper prize for ICML2012 (area chair decisions).

How do you learn structure?

Not all partitions are equally difficult.
Compare $\{1,7\}v\{3,8\}$ to $\{1,8\}v\{3,7\}$
What is better?

How do you learn structure?

Not all partitions are equally difficult.
Compare $\{1, 7\}v\{3, 8\}$ to $\{1, 8\}v\{3, 7\}$
What is better?

[BWG10]: Better to confuse near leaves than near root.
Intuition: the root predictor tends to be overconstrained while the leafwards predictors are less constrained.

The Partitioning Problem [CL14]

Given a set of n examples each with one of K labels, find a partitioner h that maximizes:

$$E_{x,y} |\Pr(h(x) = 1, y) - \Pr(h(x) = 1)\Pr(y)|$$

The Partitioning Problem [CL14]

Given a set of n examples each with one of K labels, find a partitioner h that maximizes:

$$E_x \sum_y \Pr(y) |\Pr(h(x) = 1 | x \in X_y) - \Pr(h(x) = 1)|$$

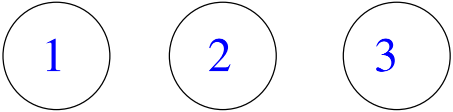
where X_y is the set of x associated with y.

The Partitioning Problem [CL14]

Given a set of n examples each with one of K labels, find a partitioner h that maximizes:

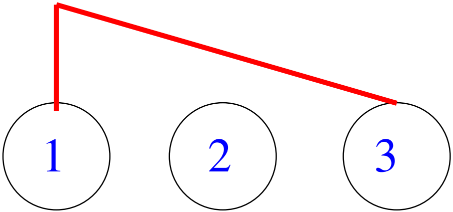
Nonconvex for any symmetric hypothesis class (ouch)

Bottom Up doesn't work



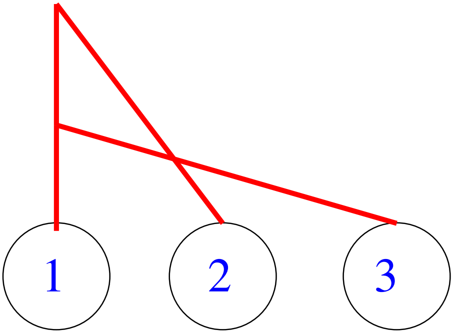
Suppose you use linear representations.

Bottom Up doesn't work



Suppose you use linear representations.
Suppose you first build a 1v3 predictor.

Bottom Up doesn't work



Suppose you use linear representations.
Suppose you first build a 1v3 predictor.
Suppose you then build a 2v{1v3} predictor.
You lose.

Does partitioning recurse well?

Theorem: If at every node n,

$$E_{x,y}\left|\Pr(h(x)=1,y)-\Pr(h(x)=1)\Pr(y)\right|>\gamma$$

then after

$$\left(\frac{1}{\epsilon}\right)^{\frac{4(1-\gamma)^2\ln k}{\gamma^2}}$$

splits, the multiclass error is less than ϵ.

Online Partitioning

Relax the optimization criteria:
$E_{x,y}\left|E_{x|y}\left[\hat{y}(x)\right]-E_{x}\left[\hat{y}(x)\right]\right|$
... and approximate with running average

Online Partitioning

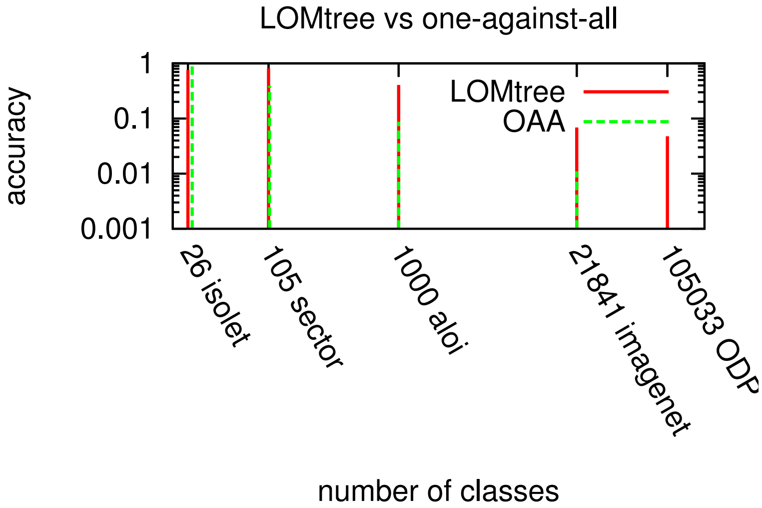
Relax the optimization criteria:
$E_{x,y} \left| E_{x|y}\left[\hat{y}(x)\right] - E_x\left[\hat{y}(x)\right]\right|$
... and approximate with running average

Let $e = 0$ and for all y, $e_y = 0, n_y = 0$
For each example (x, y)

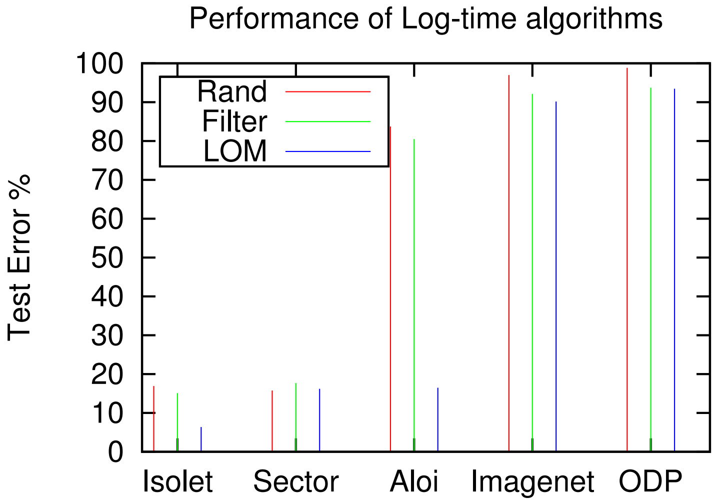
1. if $e_y < e$ then $b = -1$ else $b = 1$
2. Update w using (x, b)
3. $n_y \leftarrow n_y + 1$
4. $e_y \leftarrow \frac{(n_y - 1)e_y}{n_y} + \frac{\hat{y}(x)}{n_y}$
5. $e \leftarrow \frac{(t-1)e}{t} + \frac{\hat{y}(x)}{t}$

Apply recursively to construct a tree structure.

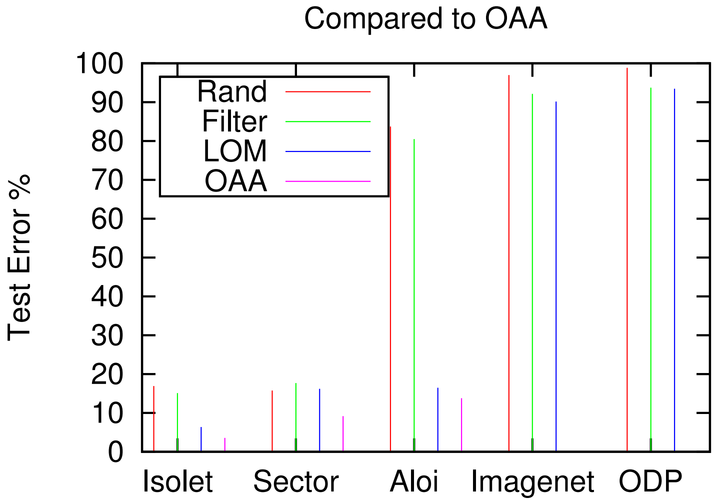
Accuracy for a fixed training time



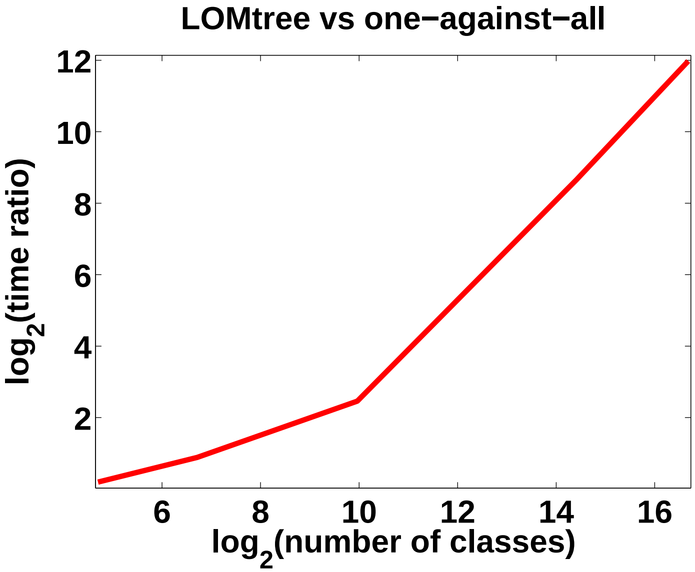
Test Error %, optimized, no train-time constraint



Test Error %, optimized, no train-time constraint



Classes vs Test time ratio



Can we predict in time $O(\log_2 K)$?

Can we predict in time $O(\log_2 K)$?

What is the right way to achieve consistency and dynamic partition?

Can we predict in time $O(\log_2 K)$?

What is the right way to achieve <u>consistency</u> and dynamic partition?

How can you balance representation complexity and sample complexity?

Bibliography

Alina Beygelzimer, John Langford, Pradeep Ravikumar, Error-Correcting Tournaments, http://arxiv.org/abs/0902.3176

Samy Bengio, Jason Weston, David Grangier, Label embedding trees for large multi-class tasks, NIPS 2010.

Anna Choromanska, John Langford, Logarithmic Time Online Multiclass prediction, http://arxiv.org/abs/1406.1822