How To Solve Reinforcement Learning with a Classifier and a Generative Model

John Langford

Outline

1. The method of reduction and setup

2. The global reduction

3. The local reduction

4. Implications

What is a reduction?

Reductions use solution(s) from problem A to solve a new problem B.

Reductions are thinking-efficient. Algorithms and theory for A applies to B automatically. Future algorithms for A also apply to B.

Reductions exhibit the relationship between problems.

Let “->” mean “reduces to”.

If we have A -> B but not B -> A, then B is a harder problem than A.

If A -> B and B -> A, but B -> A involves more work, B is “harder” than A in some sense.

The method of reduction in machine learning

1. Reduce learning problems to imperfect classification.

2. Apply some classification problem to solve these problems.

3. Apply theory from classification to these problems.

Does it work?: Working reductions

1. “Strong” classification -> “Weak” classification (Boosting, adaboost, etc.. Schapire and many others)

2. Cost sensitive classification -> “Strong” classification (aka “costing”, Bianca, Naoki and I)

3. Multiclass classification -> Binary classification (Guy Lebanon and John Lafferty)

Does it work?: Unverified reductions

1. Reinforcement learning to Regression with a generative model (Sham and I, and Drew Bagnell?, experimentally)

2. Reinforcement learning to classification with a generative model (this talk)

3. Unsupervised learning with classification (under development)

Setup: Classification

X = input space

$Y = \{0, 1\}$ = output space

D = unknown distribution over $X \times Y$

$c : X \to Y$ = classifier

$S = (x, y)^m \sim D^m$ = a training set drawn independently from D

$e_D(c) = \Pr_{x,y \sim D}(c(x) \neq y)$ = "true error rate" = proportion of time c is wrong

Setup: Reinforcement Learning

S = state space

$A = \{0, 1\}$ = actions

$P(s)$ = initial state distribution

$P(s'|a, s)$ = next state distribution

$R(s)$ = reward recieved when entering state s.

T = time horizon

$\pi: S \to A$ = policy.

$\eta(\pi) = E_{(s,a)^T \sim P,\pi} \frac{1}{T} \sum_{i=1}^{T} R(s_i)$ = average expected reward

Outline

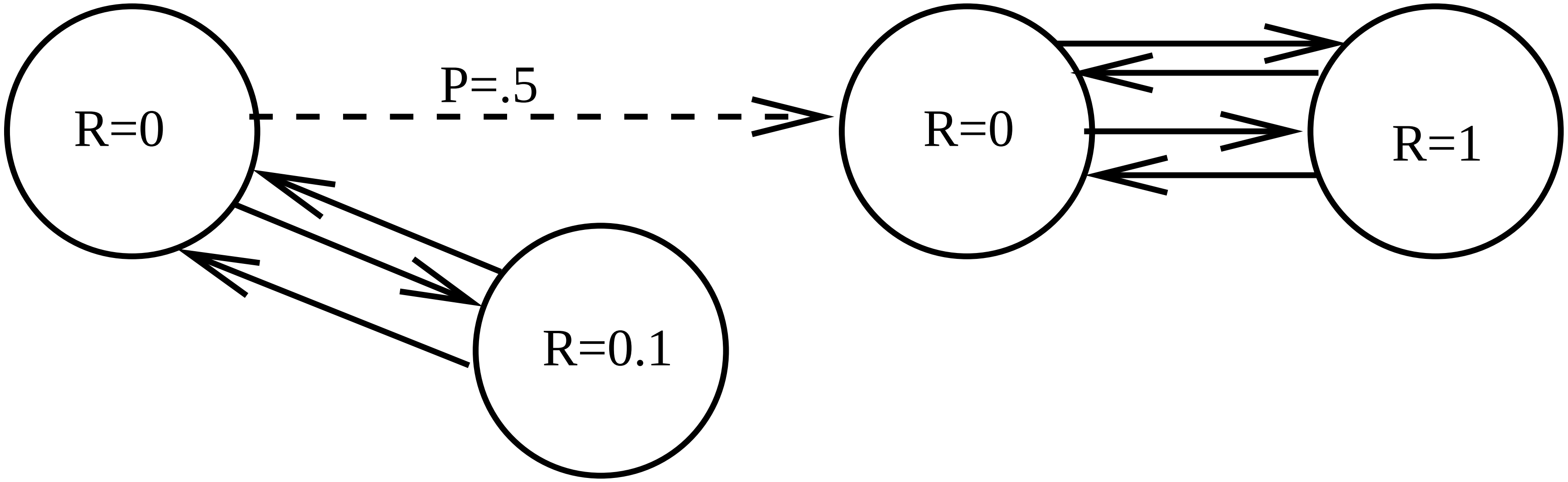
1. The method of reduction and setup

2. The global reduction

3. The local reduction

4. Implications

Something which doesn't work



We can not reduce to one instance of classification for $T = 2$.

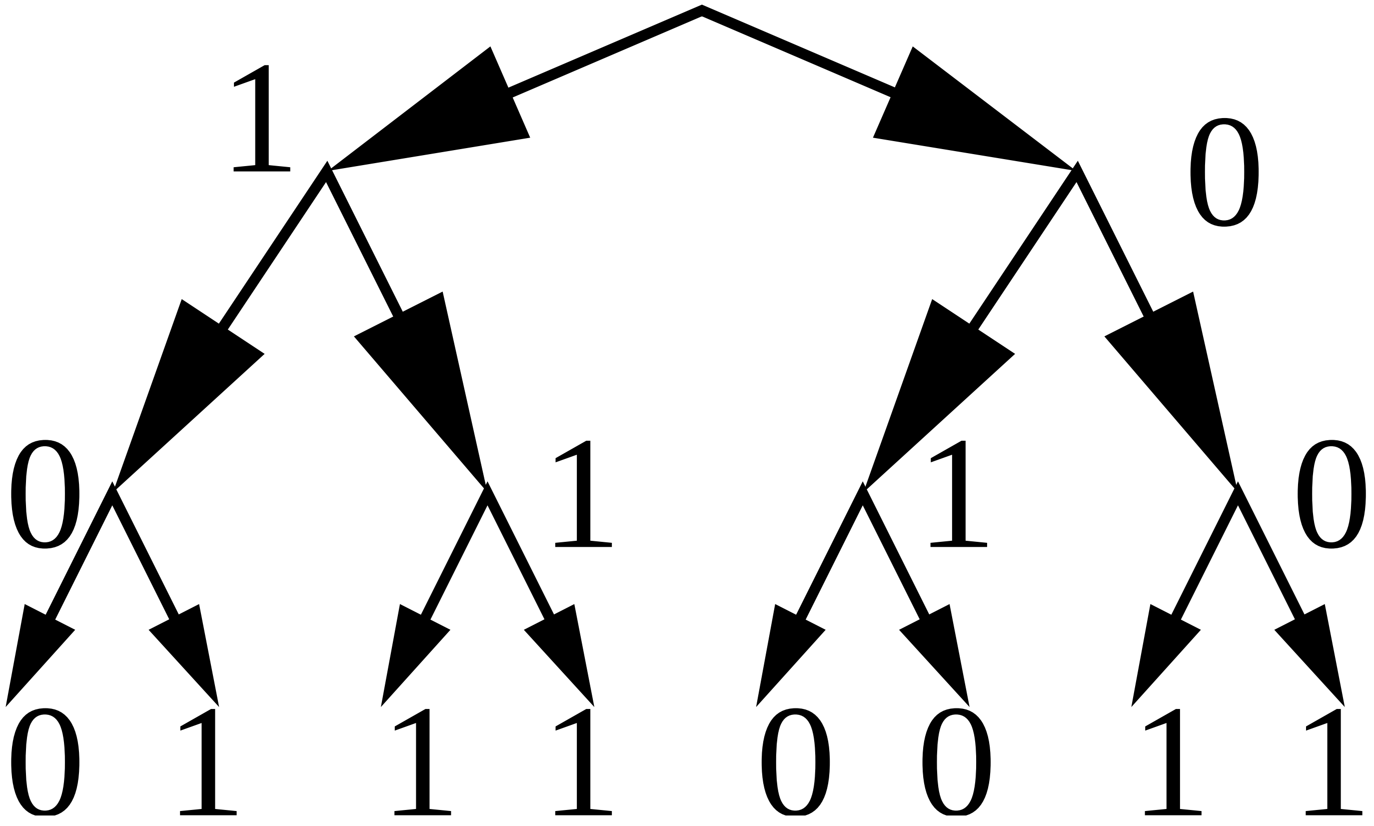
The Global reduction

Reduce to T instances of cost sensitive classification (which can be reduced to classification)

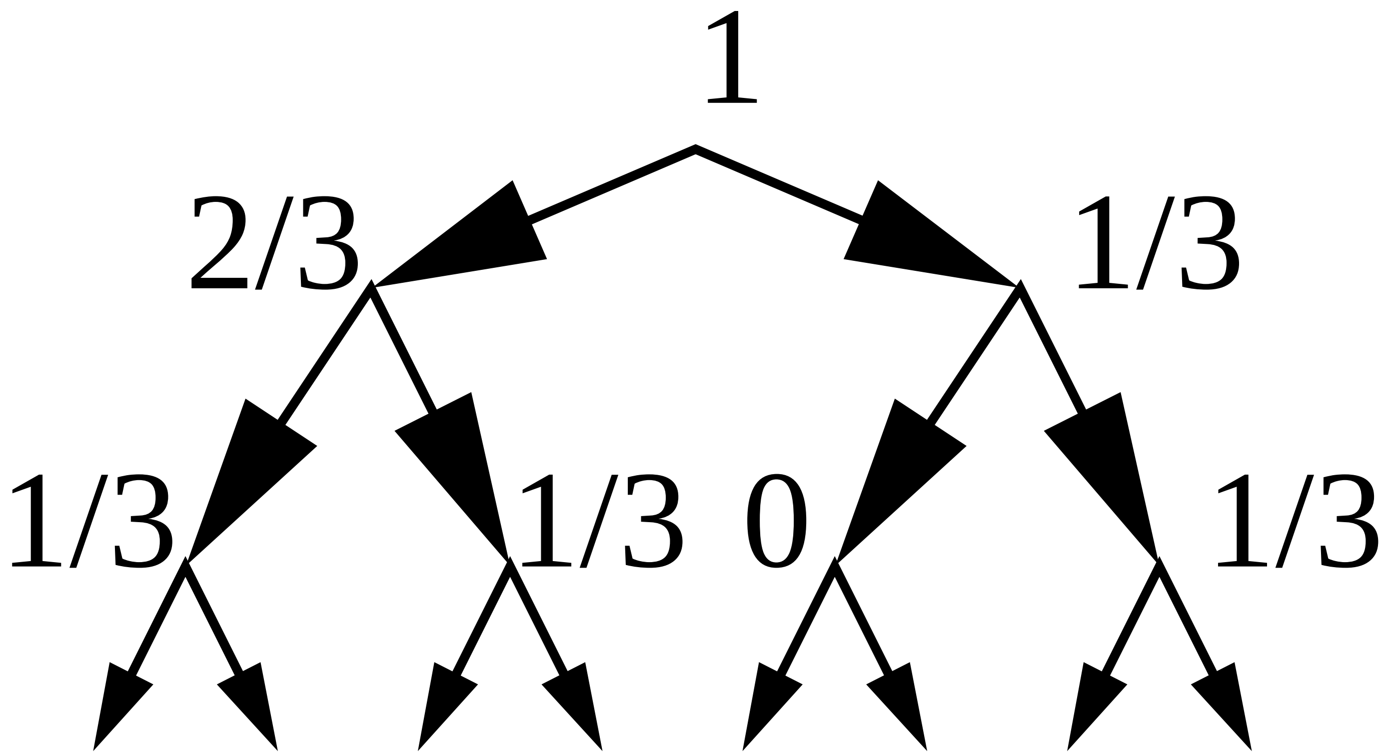
Step 1: draw trajectory trees (as in Kearns, Mansour, and Ng)

Step 2: compile into examples

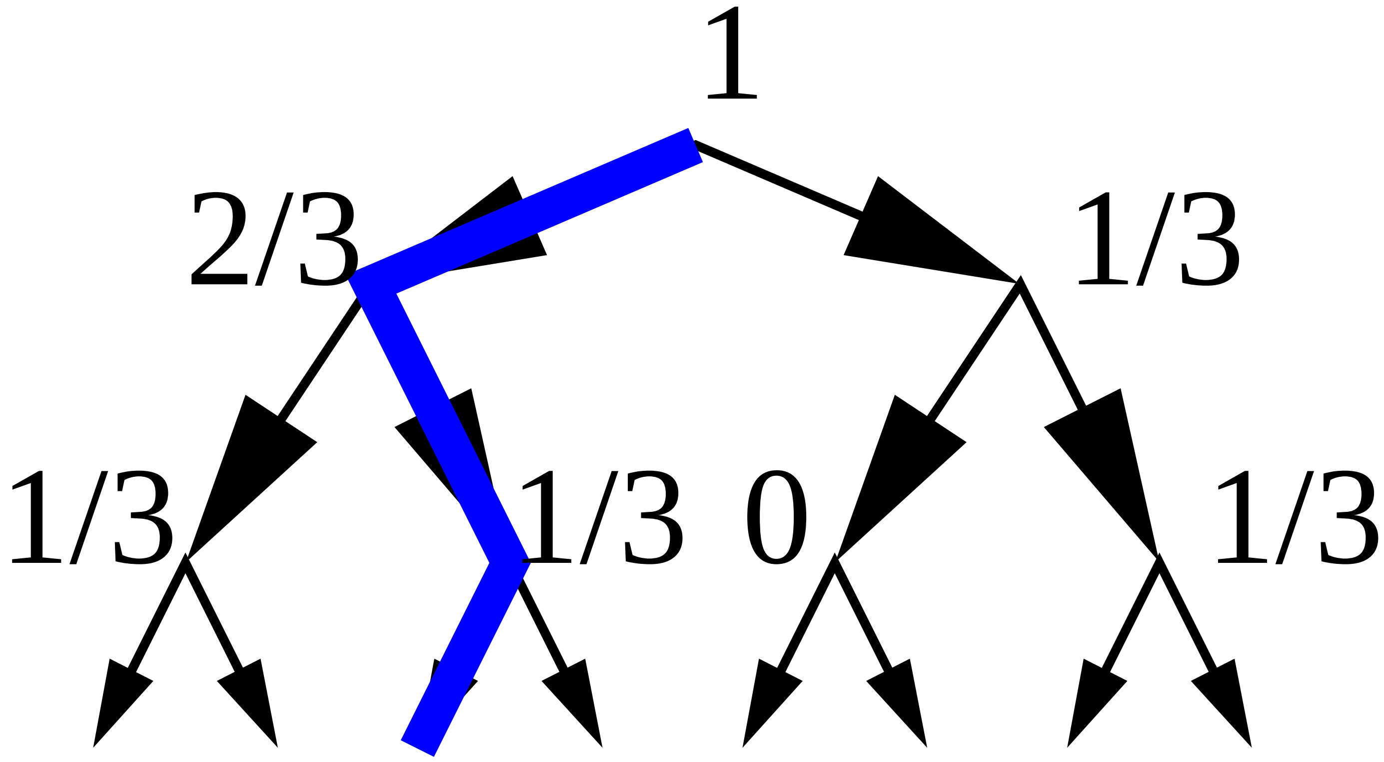
Drawing outcome trees



Construct Empirical version of Q-values



Pick Principle branch



There are T classification problems

For Classification problem i,

$x = s_i$, $y =$ argmax$_a$empirical action value, importance = empirical action value

The classification problem is "stay on the optimal path".

Every Tree $\Rightarrow$ one example per classification problem

Why can't we use more?

1. We don't know the classifier will do well off the path.

2. We dare not make the classifier do well off the path because it may do worse on path.

Many trees $\Rightarrow$ many examples per classification problem.

The Core Theorem

π^* = optimal policy

Theorem (Learning translation): Classifiers have loss rate ϵ or less $\Rightarrow \eta(\pi^*) - \eta(\pi) \leq \frac{T+1}{2}\epsilon$.

Outline

1. The method of reduction and setup

2. The global reduction

3. The local reduction

4. Implications

Global Reduction is very inefficient: Need 2^T samples of which only T are really "used". Can we do better?

Yes: the local reduction

Can't guarantee global optimality.

Can guarantee iterative improvement.

The Local Reduction: observations

Classifier i's performance is dependent on classifier $j < i$ since the distribution over states/inputs is defined by $j < i$.

Classifier i's performance is dependent on classifier $j > i$ since the rewards recieved by i's choice are dependent on j's choice.

The Local Reduction: an iterative algorithm

1. Start with some set of initial classifiers, $c_1, ..., c_T$

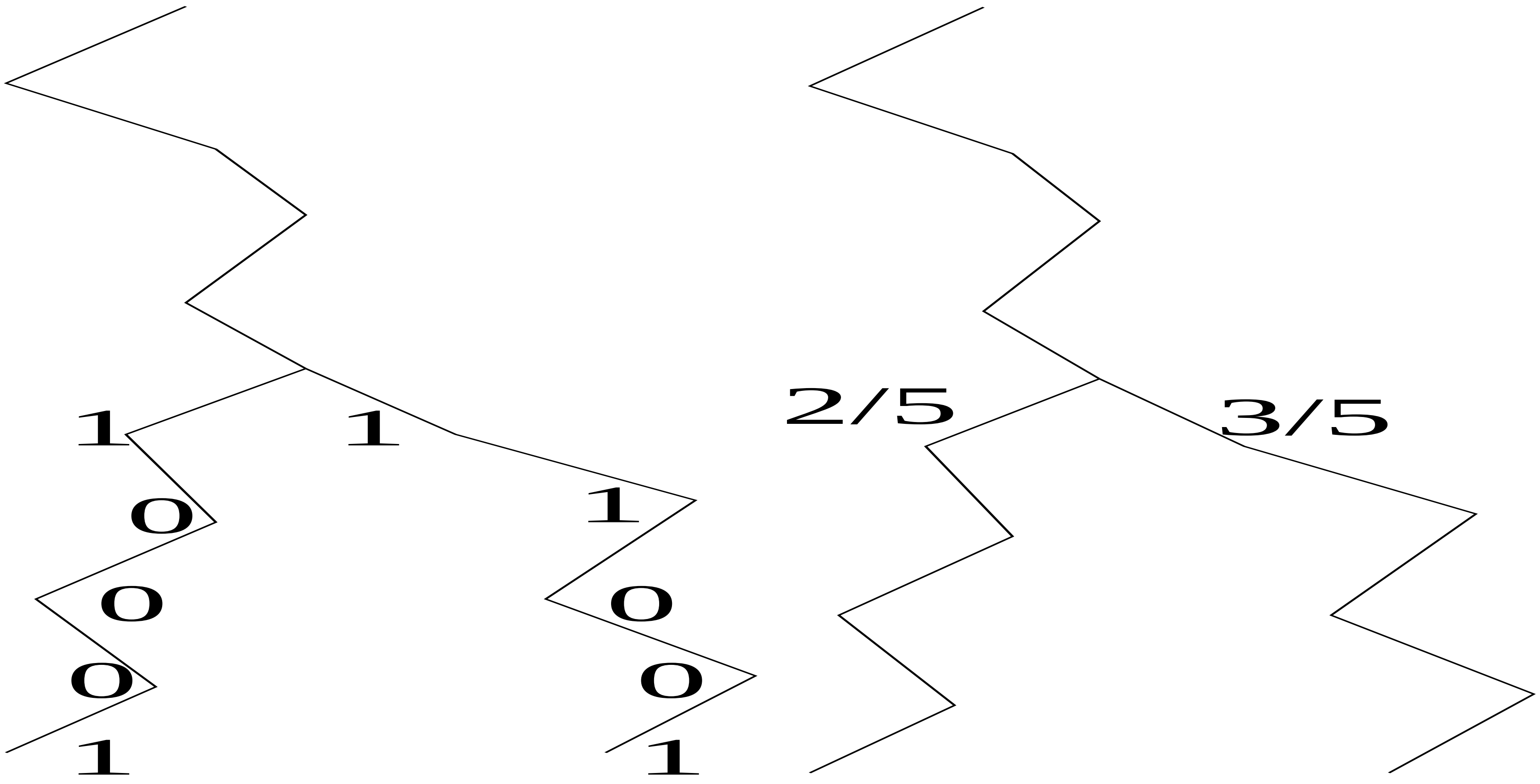
2. Pick an i

   (a) Optimize c given $c_1, ..., c_{i-1}, c_{i+1}, ..., c_T$

   (b) if $e_D(c) > e_D(c_i)$ then $c_i \leftarrow c$

   (c) goto 2

The Local Reduction: step 2.a.



$x = s$, $y = \mathrm{argmax}_a \hat{Q}(s,a)$, importance $= |Q(s,0) - Q(s,1)|$

A big side note

Andrew Ng and Jen <somebody> developed the local reduction seperately and have been (succesfully) using it for helicopter flying.

Outline

1. The method of reduction and setup

2. The global reduction

3. The local reduction

4. Implications

Implications

1. All classification algorithms are reinforcement learning algorithms.
   (a) Support vector machines to solve reinforcement learning?
   (b) Decision trees?
   (c) Neural networks?
   (d) Also motivates a form of "lifelong learning" or "multitask learning"
2. All classification theory is reinforcement learning theory.

Implications

1. All classification algorithms are reinforcement learning algorithms.

2. All classification theory is reinforcement learning theory.

   (a) PAC-Bayes bound +reduction $\Rightarrow$ stochastic policies are less prone to overfitting than might be expected.

   (b) VC theory + reduction $\Rightarrow$ (approximately) Kearns, Mansour, Ng trajectory tree paper.

   (c) PAC-MDL bound (constructed *after* the reduction) implies compressible policies avoid overfitting.