HAL
archives-ouvertes.fr

MDDAG: learning deep decision DAGs in a Markov decision process setup

D. Benbouzid, Róbert Busa-Fekete, Balázs Kégl

▶ **To cite this version:**

D. Benbouzid, Róbert Busa-Fekete, Balázs Kégl. MDDAG: learning deep decision DAGs in a Markov decision process setup. 25th Annual Conference on Neural Information Processing Systems (NIPS 2011), Dec 2011, Granada, Spain. in2p3-00935607

HAL Id: in2p3-00935607
http://hal.in2p3.fr/in2p3-00935607

Submitted on 23 Jan 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

MDDAG: learning deep decision DAGs in a Markov decision process setup

Djalel Benbouzid
Linear Accelerator Laboratory (LAL)
University of Paris-Sud, CNRS
djalel.benbouzid@gmail.com

Róbert Busa-Fekete
Linear Accelerator Laboratory (LAL)
University of Paris-Sud, CNRS
Research Group on Artificial Intelligence
of the Hungarian Academy
of Sciences and University of Szeged
busarobi@gmail.com

Balázs Kégl
Linear Accelerator Laboratory (LAL),
Computer Science Laboratory (LRI)
University of Paris-Sud, CNRS
balazs.kegl@gmail.com

Abstract

In this paper we propose an algorithm that builds sparse decision DAGs (directed acyclic graphs) out of a list of features or base classifiers. The basic idea is to cast the DAG design task as a Markov decision process. Each instance can decide to use or to skip each base classifier, based on the current state of the classifier being built. The result is a sparse decision DAG where the base classifiers are selected in a data-dependent way. The development of algorithm was directly motivated by improving the traditional cascade design in applications where the computational requirements of classifying a test instance are as important as the performance of the classifier itself. Beside outperforming classical cascade designs on benchmark data sets, the algorithm also produces interesting deep structures where similar input data follows the same path in the DAG, and subpaths of increasing length represent features of increasing complexity.

1 Introduction

The broad goal of deep learning is to go beyond a flat linear combination in designing predictors. Mainstream algorithms achieve this by building nested functions in an iterative, layer-wise fashion. In this paper we propose an alternative approach: we build sparse decision DAGs (*directed acyclic graphs*) using a Markov decision process setup. The basic idea is to cast the feature activation problem into a dynamical setup. The input of the algorithm is a sequence of base classifiers (features). An agent receives an instance to classify. Facing a base classifier in the sequence, the agent can decide between 1) evaluating it and adding it to its pool, 2) skipping it, or 3) quitting the procedure and using its current pool to classify the instance. The decision is based on the output of the current classifier. At the end of the "game" the agent receives a reward which is a function of the classification margin. To make the classifier as "lean" as possible, the agent also receives a penalty for every active feature. Although the final classifier can still be written as a linear combination of simple features, the DAG structure means that features are activated in a data-dependent way.

The development of the algorithm was directly motivated by applications where the computational requirements of classifying a test instance are as important as the performance of the classifier

itself (e.g., object detection in images [1], web page ranking [2], or trigger design in high energy physics [3]). A common solution to these problems is to design *cascade classifiers*. The basic reason of sticking to the cascade structure is that it is simple to understand and easy to design semi-automatically. On the other hand they have numerous disadvantages [4], and once the design is automatic, there remain no arguments for cascades. Indeed, in a recent submission [5] we showed that MDDAG outperforms state-of-the-art cascade classifiers [1, 6, 7, 8, 9] everywhere on the Pareto front of the trade-off defined by classification accuracy and computational cost.

The focus of this paper, however, is not performance but representation. A-priori, we can have as many different paths as training instances, nevertheless, a-posteriori, we observe clustering in the "path-space". Moreover, when we artificially control the clusters within one of the classes, this observed clustering of paths corresponds to the injected sub-classes, which is an appealing feature for, e.g., multi-view object detection. Although we have not yet tried the algorithm in an autoassociative setup, we conjecture that it could also discover interesting structures in the input data in an unsupervised fashion.

MDDAG has several close relatives in the family of supervised methods. It is obviously related to algorithms from the vast field of sparse methods. The main advantage here is that the MDP setup allows us to achieve sparsity in a dynamical data-dependent way. This feature relates the technique to unsupervised sparse *coding* [10, 11] rather than to sparse classification or regression. On a more abstract level, MDDAG is also similar to [12]'s approach to "learn where to look". Their goal is to find a sequence of two-dimensional features for classifying images in a data-dependent way, whereas we do a similar search in a one-dimensional ordered sequence of features. Finally, the fact that our final classifier is a *tree* (more precisely, a directed acyclic graph) where decisions are made by accumulating base classifications *along* the whole path relates our technique to alternating decision trees (ADT) [13], except that ADTs use the output of the base classifiers directly for navigating in the tree, whereas in MDDAG the classifications are only indirectly coupled with control decisions.

The closest relatives to MDDAG are arguably [9] and [14]. Both approaches cast the classifier construction into an MDP framework. The differences are in the details of how the MDPs are constructed and learned. [9] uses a smaller number of more complex "stages" as base classifiers and learn the MDP with a direct policy search algorithm. The main difference between [9] and MDDAG is that they can only decide between quitting and continuing, eliminating thus the possibility of learning sparse classifiers. The model of [14] represents the other extreme: at each base classifier the agent can choose to jump to any other base classifier, effectively blowing up the *action* space. They use an approximate policy iteration algorithm with rollouts. Because of the huge action space, they use the algorithm only on small data sets and restrict the features to the small number (maximum 60) of original features of the data sets. In a sense, our algorithm is positioned halfway between [9] and [14]. Our SKIP action makes it possible to learn sparse DAGs. In theory, our function class is equivalent to that of [14], but the way we design the MDP makes the learner's task easier.

The paper is organized as follows. In Section 2 we describe the algorithm, in Section 3 we show experimental results, and we conclude in Section 4.

2 The MDDAG algorithm

Similarly to SOFTCASCADE [4], we describe MDDAG as a *post-processing* method that takes the output of a trained classifier and "sparsifies" it. Formally, we assume that we are given a sequence of N *base classifiers* $\mathcal{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_N)$. Although in most of the cases cascades are built for binary classification, we describe the method for the more general multi-class case, which means that $\mathbf{h}_j : \mathcal{X} \to \mathbb{R}^K$, where $\mathcal{X}$ is the input space and K is the number of classes. The semantics of $\mathbf{h}$ is that, given an observation $\mathbf{x} \in \mathcal{X}$, it *votes for* class ℓ if its ℓth element $h_\ell(\mathbf{x})$ is positive, and votes against class ℓ if $h_\ell(\mathbf{x})$ is negative. The absolute value $|h_\ell(\mathbf{x})|$ can be interpreted as the confidence of the vote. Although it is not a formal requirement, we will also assume that $\mathcal{H}$ is sorted in order of "importance" or performance of the base classifiers. These assumptions are naturally satisfied by the output of ADABOOST.MH [15], but in principle any algorithm that builds its final classifier as a linear combination of simpler functions can be used to provide $\mathcal{H}$. In the case of ADABOOST.MH or multi-class neural networks, the *final* (or *strong* or *averaged*) classifier defined by the full sequence $\mathcal{H}$ is $\mathbf{f}(\mathbf{x}) = \sum_{j=1}^{N} \mathbf{h}_j(\mathbf{x})$, and its prediction for the class index of $\mathbf{x}$ is $\widehat{\ell} = \arg\max_\ell f_\ell(\mathbf{x})$. In binary *detection*, $\mathbf{f}$ is usually used as a scoring function. The observation $\mathbf{x}$ is classified as signal if

$f_1(\mathbf{x}) = -f_2(\mathbf{x}) > \theta$ and background otherwise. The threshold θ is a free parameter that can be tuned to achieve, e.g., a given false positive rate.

The goal of MDDAG is to build a *sparse* final classifier from $\mathcal{H}$ that does not use all the base classifiers. Moreover, we would like the selection to be data-dependent. For a given observation $\mathbf{x}$ we process the base classifiers in their original order. At each base classifier $\mathbf{h}_j$ we choose among three actions: 1) we either EVALuate $\mathbf{h}_j$ and continue, or 2) we SKIP $\mathbf{h}_j$ and continue, or 3) we QUIT and return the classifier built so far. Let

$$b_j(\mathbf{x}) = 1 - \mathbb{I}\{a_j = \text{SKIP} \;\text{ OR }\; \exists j' < j : a_{j'} = \text{QUIT}\} \tag{1}$$

be the indicator that $\mathbf{h}_j$ is evaluated, where $a_j \in \{\text{EVAL}, \text{SKIP}, \text{QUIT}\}$ is the action taken at step j and the indicator function $\mathbb{I}\{A\}$ is 1 if its argument A is true and 0 otherwise. Then the final classifier built by the procedure is

$$\mathbf{f}^{(N)}(\mathbf{x}) = \sum_{j=1}^{N} b_j(\mathbf{x})\mathbf{h}_j(\mathbf{x}). \tag{2}$$

Inspired by WALDBOOST [7] and the method of [9], the decision on action a_j will be made based on the index of the base classifier j and the output vector of the classifier

$$\mathbf{f}^{(j)}(\mathbf{x}) = \sum_{j'=1}^{j} b_{j'}(\mathbf{x})\mathbf{h}_{j'}(\mathbf{x}). \tag{3}$$

built up to step j.[1] Formally, $a_j = \pi\big((\mathbf{s}_j(\mathbf{x})\big)$, where

$$\mathbf{s}_j(\mathbf{x}) = \big(f_1^{(j-1)}(\mathbf{x}), \ldots, f_K^{(j-1)}(\mathbf{x}), j-1\big) \in \mathbb{R}^K \times \mathbb{N}^+ \tag{4}$$

is the *state* where we are before visiting $\mathbf{h}_j$, and π is a *policy* that determines the action in state $\mathbf{s}_j$. The initial state $\mathbf{s}_1$ is the zero vector with $K+1$ elements.

This setup formally defines a Markov decision process (MDP). An MDP is a 4-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the (possibly infinite) state space and $\mathcal{A}$ is the countable set of actions. $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0,1]$ is the transition probability kernel which defines the random transitions $\mathbf{s}^{(t+1)} \sim \mathcal{P}(\cdot|\mathbf{s}^{(t)}, a^{(t)})$ from a state $\mathbf{s}^{(t)}$ applying the action $a^{(t)}$, and $\mathcal{R} : \mathbb{R} \times \mathcal{S} \times \mathcal{A} \to [0,1]$ defines the distribution $\mathcal{R}(\cdot|\mathbf{s}^{(t)}, a^{(t)})$ of the *immediate reward* $r^{(t)}$ for each state-action pair. A *deterministic policy* π assigns an action to each state $\pi : \mathcal{S} \to \mathcal{A}$. We will only use *undiscounted* and *episodic* MDPs where the policy π is evaluated using the *expected sum of rewards*

$$\varrho = \mathbb{E}\left\{\sum_{t=1}^{T} r^{(t)}\right\} \tag{5}$$

with a finite horizon T. In the episodic setup we also have an *initial* state ($\mathbf{s}_1$ in our case) and a *terminal* state $\mathbf{s}_\infty$ which is impossible to leave.

In our setup, the state $\mathbf{s}^{(t)}$ is equivalent to $\mathbf{s}_j(\mathbf{x})$ (4) with $j = t$. The action QUIT brings the process to the terminal state $\mathbf{s}_\infty$. Figure 1 illustrates the MDP designed to learn the sparse stageless DAG.

To define the rewards, our primary goal is to achieve high accuracy, so we penalize the error of $\mathbf{f}^{(t)}$ when the action $a^{(t)} = \text{QUIT}$ is applied. For the formal definition, we suppose that, at training time, the observations $\mathbf{x}$ arrive with the *index* $\ell \in \{1, \ldots, K\}$ of their class. The *multi-class margin* is defined as $\rho^{(t)}(\mathbf{x}, \ell) = f_\ell^{(t)}(\mathbf{x}) - \max_{\ell' \neq \ell} f_{\ell'}^{(t)}(\mathbf{x})$. With these notations, the classical (0–1) multi-class reward for the QUIT action is

$$r_{\mathbb{I}}^{(t)}(\mathbf{x}, \ell) = \mathbb{I}\Big\{\rho^{(t)}(\mathbf{x}, \ell) > 0\Big\}. \tag{6}$$

For well-known reasons we also use the convex upper bound

$$r_{\text{EXP}}^{(t)}(\mathbf{x}, \ell) = \exp\big(\rho^{(t)}(\mathbf{x}, \ell)\big). \tag{7}$$

[1] When using ADABOOST.MH, the base classifiers are binary $\mathbf{h}_j(\mathbf{x}) = \{\pm\alpha_j\}^K$, and we normalize the output (3) by $\sum_{j=1}^{N} \alpha_j$, but since this factor is constant, the only reason to do it is to make the range of the state space uniform across experiments.

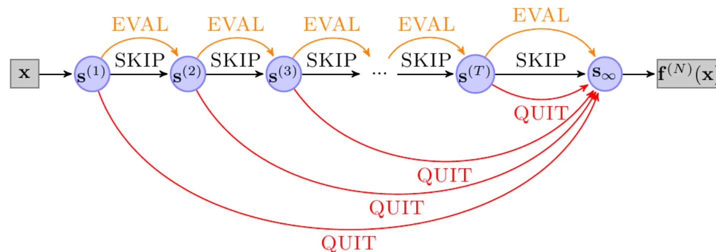


Figure 1: The schematic overview of the process. The input is an instance $\mathbf{x}$ to be classified. During the process the policy decides which base classifier will be evaluated by choosing one of the three available actions in each state. The output is the score vector provided by the classifier $\mathbf{f}^{(N)}(\mathbf{x})$ (2).

In principle, any of the usual convex upper bounds (e.g., logistic, hinge, quadratic) could be used in the MDP framework. The exponential reward is inspired by the setup of ADABOOST [16, 15]. The rewards can easily be adapted to *cost-sensitive* classification when the misclassification cost is different for the classes, although we did not explore this issue in this paper. Note also that in the binary case, $r_{\mathbb{I}}$ and r_{EXP} recover the classical binary notions. From now on we will refer to our algorithm as MDDAG.$\mathbb{I}$ when we use the indicator reward $r_{\mathbb{I}}$ and MDDAG.EXP when we use the exponential reward r_{EXP}.

For encouraging sparsity, we will also penalize each evaluated base classifier $\mathbf{h}$ by a uniform fixed negative reward

$$\mathcal{R}(r|\mathbf{s}, \text{EVAL}) = \delta(-\beta - r), \tag{8}$$

where δ is the Dirac delta and β is a hyperparameter that represents the accuracy/speed trade-off.

The goal of reinforcement learning (RL) in our case is to learn a policy which maximizes the expected sum of rewards (5). Since in our setup the transition $\mathcal{P}$ is deterministic *given* the observation $\mathbf{x}$, the expectation in (5) is taken with respect to the random input point $(\mathbf{x}, \ell)$. This means that the global goal of the MDP is to maximize

$$\varrho = \mathbb{E}_{(\mathbf{x},\ell)\sim\mathfrak{D}} \left\{ r(\mathbf{x}, \ell) - \beta \sum_{j=1}^{N} b_j(\mathbf{x}) \right\} \tag{9}$$

where $r(\mathbf{x}, \ell)$ is one of our margin-based rewards and $\mathfrak{D}$ is the distribution that generates the instances.

Note that, strictly speaking, the rewards (6) and (7) are not stationary given the state $\mathbf{s}^{(t)}$: it is possible that two different policies bring two different subsets of $\mathcal{X}$ with different label distributions into the same state, so the rewards depends on external factor not summarized in the state. This problem could be tackled in an *adversarial* setup, recently analyzed by [17]. However, the results of [17] are rather theoretical, and we found that, similarly to [9] whose method also suffers from non-Markovianness, classical MDP algorithms work well for solving our problem.

2.1 Learning the policy

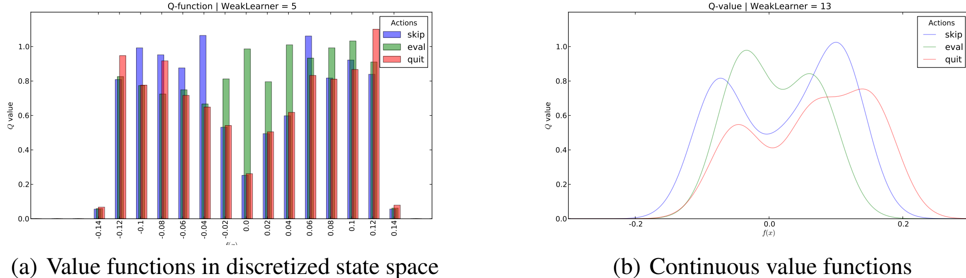
There are several efficient algorithms to learn the policy π using an iid sample $\mathcal{D} = \big((\mathbf{x}_1, \ell_1), \ldots, (\mathbf{x}_n, \ell_n)\big)$ drawn from $\mathfrak{D}$ [18]. When $\mathcal{P}$ and $\mathcal{R}$ are unknown, model-free methods are commonly used for learning the policy π. These methods directly learn a *value function*, (the expected reward in a state or for a state-action pair), and derive a policy from it. Among model-free RL algorithms, *temporal-difference* (TD) learning algorithms are the most commonly used. They can be divided into two groups: *off-policy* and *on-policy* methods. In the case of off-policy methods the policy search method learns about one policy while following another, whereas in the on-policy case the policy search algorithm tries to improve the current policy by maintaining sufficient exploration. On-policy methods have an appealing practical advantage: they usually converge faster to the optimal policy than the off-policy methods.

We use the SARSA(λ) algorithm [19] with *replacing traces* to learn the policy π. For more details we refer the reader to [20]. SARSA(λ) is an on-policy method, so, to make sure that all policies

can be visited with nonzero probability, we use an ϵ-greedy exploration strategy. Concretely, we apply SARSA in an episodic setup: we use a random training instance $\mathbf{x}$ from $\mathcal{D}$ per episode. The instance follows the current policy with probability $1 - \epsilon$ and chooses a random action with probability ϵ. The instance observes the immediate rewards (6), (7), or (8) after each action. The policy is updated during the episode according to SARSA(λ). In preliminary experiments we also tested Q-LEARNING [21], one of the most popular off-policy methods, but SARSA(λ) slightly but consistently outperformed Q-LEARNING.

In all experiments we used ADABOOST.MH to obtain a pool of weak classifiers $\mathcal{H}$. We ran ADABOOST.MH for $N = 1000$ iterations, and then trained SARSA(λ) on the same training set. The hyperparameters of SARSA(λ) were fixed across the experiments. We set λ to 0.95. In principle, the learning rate should decrease to 0, but we found that this setting forced the algorithm to converge too fast to suboptimal solutions. Instead we set the learning rate to a constant 0.2, we evaluated the current policy after every 10000 episodes, and we selected the best policy based on their performance also on the training set (overfitting the MDP was a no-issue). The exploration term ϵ was decreased gradually as $0.3 \times 1/\lceil \frac{10000}{\tau} \rceil$ where τ is the number of training episodes. We trained SARSA(λ) for 10^6 episodes.

In binary classification problems discretizing the state space and representing the value function as a table works well (Figure 2(a)). When the number of classes K (so the number of dimensions of the state space) grows, discretization becomes inefficient. In this case we decided to represent the value functions with radial basis function networks (RBFs – mixtures of Gaussians, Figure 2(b)), and learn the weights using gradient descent [18]. In general, the average value of the actions at a given output $f(\mathbf{x})$ is lower when $|f(\mathbf{x})|$ is small. The EVAL action dominates the region around zero whereas QUIT becomes worthy when $|f(\mathbf{x})|$ is large. This behavior is quite intuitive since $|f(\mathbf{x})|$ is generally related to the confidence of the classification.



(a) Value functions in discretized state space

(b) Continuous value functions

Figure 2: Value functions $Q(s, a)$.

As a final remark, note that maximizing (9) over the data set $\mathcal{D}$ is equivalent to minimizing a margin-based loss with an L_0 constraint. If $r_{\mathbb{I}}$ (6) is used as a reward, the loss is also non-convex, but minimizing a loss with an L_0 constraint is NP-hard even if the loss is convex [22]. So, what we are aiming for is an MDP-based heuristics to solve an NP-hard problem, something that is not without precedent [23]. This equivalence implies that even though the algorithm would converge in the ideal case (decreasing learning rate, proper Markovian rewards), in principle, convergence can be exponentially slow in n. In practice, however, we had no problem finding good policies in reasonable training time.

3 Experiments

Since our focus in this paper is on representation, we show two toy examples to illustrate how MDDAG can discover structure in the input data. In Section 3.1 we first verify the sparsity and heterogeneity hypotheses on a synthetic example. In Section 3.2 we use an MNIST subproblem show “path-wise” clustering.

3.1 Synthetic data

The goal of this experiment was to verify whether MDDAG can learn the subset of “useful” base classifiers in a *data-dependent* way. We created a two-dimensional binary dataset with real-valued features where the positive class is composed of two well-separable clusters (Figure 3(a)). This is a typical case where ADABOOST or a traditional cascade is suboptimal since they both have to use *all* the base classifiers for all the positive instances [4].

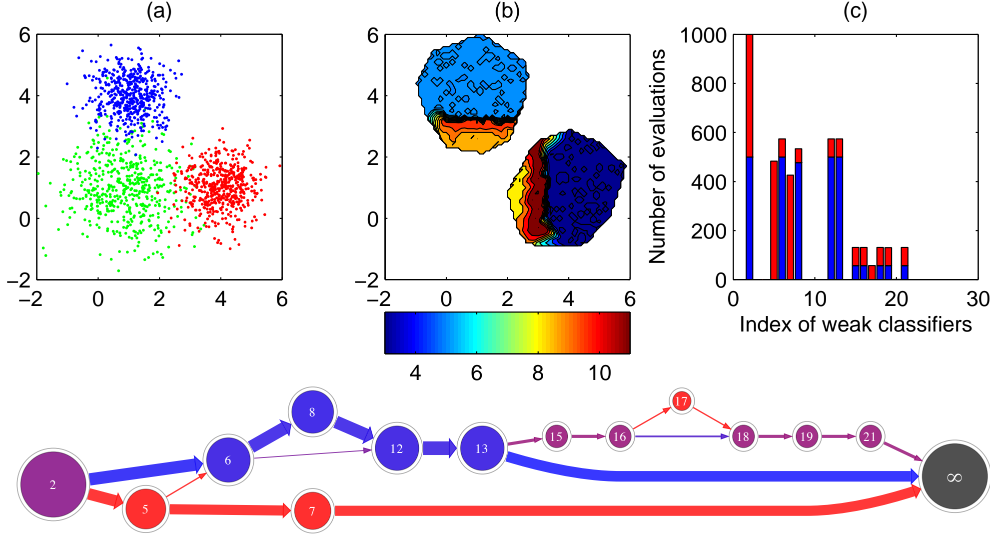


Figure 3: Experiments with synthetic data. (a) The positive class is composed of the blue and red clusters, and the negative class is the green cluster. (b) The number of base classifiers used for each individual positive instance as a function of the two-dimensional feature coordinates. (c) The number of positive instances from the blue/red clusters on which a given base classifier was applied according to the policy learned by MDDAG.$\mathbb{I}$. Lower panel: the decision DAG for the positive class. Colors represent sub-class probabilities (proportions) and the node sizes and arrow widths represent the number of instances in the states and following the actions, respectively.

We ran MDDAG.$\mathbb{I}$ with $\beta = 0.01$ on the 1000 decision stumps learned by ADABOOST.MH. In Figure 3(b) we plot the number of base classifiers used for each individual positive instance as a function of the two-dimensional instance itself. As expected, the “easier” the instance, the less base classifiers it needs for classification. Figure 3(c) confirms our second hypothesis: base classifiers are used selectively, depending on whether the positive instance is in the blue or red cluster. Figure 3(d) shows the actual DAG learned for the positive class. We follow each training instance and “summarize” sequences of SKIP actions into single transitions. Empirical class probabilities are color coded in each node and on each transition. The structure of the DAG also confirms our intuition: the bulk of the two sub-classes are separated early and follow different paths. It is also remarkable that even though the number of possible paths is exponentially large, the number of the realized subpaths is very small. Some “noisy” points along the main diagonal (border between the subclasses) generate rare subpaths, but the bulk of the data basically follows two paths.

3.2 MNIST example

We ran MDDAG.$\mathbb{I}$ with $\beta = 0.0001$ on the 300 Haar stumps [1] trained on 2s and 4s against 6s and 9s. Figure 4 shows the trained decision DAG of the 2-4 class. As in the previous section, we color-code the nodes and the arrows to show how MDDAG automatically separates subclasses without knowing their labels. In Table 1 we enumerate all the paths followed by at least 6 training instances and the pixelwise averages of the corresponding instances. First note that the number of actual paths is tiny compared to the exponentially many possible paths. This mens that even though the nominal complexity of the class of classifiers represented by all the DAGs is huge, the algorithm can successfully control the effective complexity. Second, the average images indicate that

MDDAG finds sub-classes even within the 2s and 4s. Finally, although our goal with this example is to illustrate the structure-learning capabilities of MDDAG, on the performance side MDDAG outperforms AdaBoost by 10%: the decision DAG uses 6.03 base classifiers on average and achieves 91.6% accuracy whereas AdaBoost achieves 80.7% after 6 iterations.

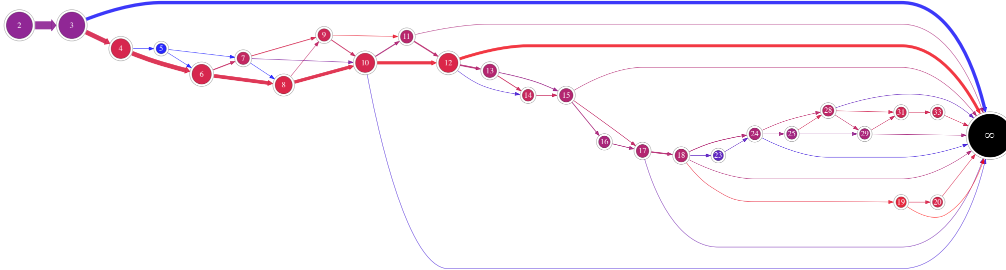


Figure 4: The decision DAG for the 2-4 class. Colors represent sub-class probabilities (blue is 2 and red is 4) and the node sizes and arrow widths represent the number of instances in the states and following the actions, respectively.

path	average image	number of test instances
2-3		835
2-3-4-6-7-9-10		18
2-3-4-6-7-9-10-11-12		28
2-3-4-6-7-9-10-11-12-13-14-15		26
2-3-4-6-7-9-10-11-12-13-14-15-16-17-18		10
2-3-4-6-7-9-10-11-12-13-14-15-17-18-24-25-29		19
2-3-4-6-7-9-10-11-12-13-14-15-17-18-24-28-29		18
2-3-4-6-7-9-10-11-12-13-14-15-17-18-24-28-31-33		44
2-3-4-6-7-9-10-11-12-13-15-16-17-18-19		9
2-3-4-6-7-9-10-11-12-13-15-16-17-18-19-20		12
2-3-4-6-7-9-10-12-13-15-16-17-18		6
2-3-4-6-7-10		18
2-3-4-6-7-10-11		10
2-3-4-6-8-9-11-12-13-14-15-16-17-18		11
2-3-4-6-8-9-11-12-13-14-15-17-18-24-28-31-33		11
2-3-4-6-8-9-11-12-13-15-16-17-18		7
2-3-4-6-8-10-11-12-13-14-15-16-17-18-24		6
2-3-4-6-8-10-11-12-13-15-16-17-18		21
2-3-4-6-8-10-11-12-13-15-17-18-24-25-29		7
2-3-4-6-8-10-11-12-13-15-17-18-24-28-31-33		11
2-3-4-6-8-10-12		699
2-3-4-6-8-10-12-13-15-16-17-18		38
2-3-4-6-8-10-12-13-15-16-17-18-19		9
2-3-4-6-8-10-12-13-15-16-17-18-19-20		12

Table 1: The paths followed by more than 6 test instances, the corresponding average images, and the number of instances.

4 Conclusions

In this paper we introduced an MDP-based design of decision DAGs. The output of the algorithm is a data-dependent sparse classifier which means that every instance "chooses" the base classifiers or features that it needs for predicting its class index. The algorithm is competitive to state-of-the-art

cascade detectors on object detection benchmarks, and it is also directly applicable to test-time-constrained problems involving multi-class classification (e.g., web page ranking). In our view, however, the main advantage of the algorithm is not necessarily its performance but its simplicity and versatility. First, MDDAG is basically a turn-key procedure: it comes with one user-provided hyperparameter with a clear semantics of directly determining the accuracy/speed trade-off. Second, MDDAG can be easily extended to problems different from classification by redefining the rewards on the QUIT and EVAL actions. For example, one can easily design *regression* or *cost-sensitive* classification DAGs by using an appropriate reward in (6), or add a weighting to (8) if the features have different evaluation costs.

The success of this simple algorithm opens the door to a vast field of new designs and extensions. First the current MDP setup is quite simplistic: it assumes that the features are ordered into a sequence and that their computational costs are equal. In a typical physics trigger [3] the base classifiers are small decision trees that use a subset of the raw observables, and the most costly operation at test time is not the evaluation of the tree but the construction of the features. In this case the cost of a base classifier (so its penalty in the MDP) depends on the raw features it uses and the features that have already been constructed in previously evaluated trees. This requires the revisiting of the state definition. Another similar example is when features are embedded in a metric space (for example, filters in object classification in images). In this case the three-action setup may be replaced by a richer configuration where the agent can decide "where to look" next [12, 14]. All these extensions will lead to more complex MDPs, so we will also need to explore the space of RL algorithms to match the specificities of the problem with the strengths of the different RL methods: this is going to be a balancing act between richness of the representation and learnability of the MDP design.

Second, at this point the algorithm is designed for post-processing a fixed output of another learning method. Beside trying it on the output of different methods (e.g., for filtering support vectors), we will also explore if the filtering mechanism could be used within the learning loop. Our ultimate goal is to couple feature construction with designing the DAG in a standalone learning method, either by incorporating the DAG design into the boosting framework or by letting the state and action spaces grow in the RL framework. Finally, it is an open question at this point whether the RL algorithm converges under the condition of our specific non-Markovian rewards; proving convergence is an interesting challenge.

References

[1] P. Viola and M. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, 2004.

[2] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *Yahoo Learning to Rank Challenge (JMLR W&CP)*, volume 14, pages 1–24, Haifa, Israel, 2010.

[3] V. Gligorov. A single track HLT1 trigger. Technical Report LHCb-PUB-2011-003, CERN, 2011.

[4] L. Bourdev and J. Brandt. Robust object detection via soft cascade. In *Conference on Computer Vision and Pattern Recognition*, volume 2, pages 236–243. IEEE Computer Society, 2005.

[5] Same authors. MDDAG: designing sparse decision DAGs using Markov decision processes. In *submitted*, 2011.

[6] R. Xiao, L. Zhu, and H. J. Zhang. Boosting chain learning for object detection. In *Ninth IEEE International Conference on Computer Vision*, volume 9, pages 709–715, 2003.

[7] J. Sochman and J. Matas. WaldBoost – learning for time constrained sequential detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 150–156, 2005.

[8] M. Saberian and N. Vasconcelos. Boosting classifier cascades. In *Advances in Neural Information Processing Systems 23*, pages 2047–2055. MIT Press, 2010.

[9] B. Póczos, Y. Abbasi-Yadkori, Cs. Szepesvári, R. Greiner, and N. Sturtevant. Learning when to stop thinking and do something! In *Proceedings of the 26th International Conference on Machine Learning*, pages 825–832, 2009.

[10] H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, volume 19, pages 801–808. The MIT Press, 2007.

[11] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems 19*, pages 1137–1144. MIT Press, Cambridge, MA, 2007.

[12] H. Larochelle and G. Hinton. Learning to combine foveal glimpses with a third-order Boltzmann machine. In *Advances in Neural Information Processing Systems 23*, pages 1243–1251. MIT Press, 2010.

[13] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *Proceedings of the 16th International Conference on Machine Learning*, pages 124–133, 1999.

[14] G. Dulac-Arnold, L. Denoyer, P. Preux, and P. Gallinari. Datum-wise classification: A sequential approach to sparsity. In *European Conference on Machine Learning*, 2011.

[15] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[16] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[17] G. Neu, A. György, and Cs. Szepesvári. The online loop-free stochastic shortest-path problem. In *Proceedings of the 23th Annual Conference on Computational Learning Theory*, pages 231–243, 2010.

[18] R.S. Sutton and A.G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[19] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University, Engineering Department, 1994.

[20] Cs. Szepesvári. *Algorithms for Reinforcement Learning*. Morgan and Claypool, 2010.

[21] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[22] G. Davis, S. Mallat, and M. Avellaneda. Adaptive greedy approximations. *Constructive Approximation*, 13(1):57–98, 1997.

[23] V. Ejov, J. Filar, and J. Gondzio. An interior point heuristic for the Hamiltonian cycle problem via Markov Decision Processes. *Journal of Global Optimization*, 29(3):315–334, 2004.