Sequential prediction for budgeted learning : Application to trigger design

Djalel Benbouzid

LAL 14-24

UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE D'INFORMATIQUE

THÈSE DE DOCTORAT

Soutenue le 20 Février 2014 par

Djalel BENBOUZID

Sequential prediction for budgeted learning

Applications to trigger design

Discipline : Informatique
Préparée au Laboratoire de l'Accélérateur Linéaire de l'Université
(Paris-Sud XI), dans les équipes APPSTAT et TAO

sous la direction de Balázs KÉGL

Jury

Rapporteurs	Ludovic DENOYER	Université Paris VI
	Kilian WEINBERGER	Washington University in St.Louis
Directeur	Balázs KÉGL	Université Paris-Sud & CNRS
Examinateurs	Florence D'ALCHÉ-BUC	Université d'Evry-Val d'Essonne
	Damien ERNST	Université de Liège
	Vladimir GLIGOROV	CERN
	Michèle SÉBAG	Université Paris-Sud & CNRS
Invité	Guy WORMSER	Université Paris-Sud & CNRS

Last modified on February 28, 2014

LAL 14-24

UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE D'INFORMATIQUE

THÈSE DE DOCTORAT

Soutenue le 20 Février 2014 par

Djalel BENBOUZID

Prédiction séquentielle en apprentissage statistique avec contraintes de budget

Applications à la conception de *trigger* en physique des particules

Discipline : Informatique
Préparée au Laboratoire de l'Accélérateur Linéaire de l'Université
(Paris-Sud XI), dans les équipes APPSTAT et TAO

sous la direction de Balázs KÉGL

Jury

Rapporteurs	Ludovic DENOYER	Université Paris VI
	Kilian WEINBERGER	Washington University in St.Louis
Directeur	Balázs KÉGL	Université Paris-Sud & CNRS
Examinateurs	Florence D'ALCHÉ-BUC	Université d'Evry-Val d'Essonne
	Damien ERNST	Université de Liège
	Vladimir GLIGOROV	CERN
	Michèle SÉBAG	Université Paris-Sud & CNRS
Invité	Guy WORMSER	Université Paris-Sud & CNRS

Acknowledgments
Remerciements

Il est difficile d'exprimer sa gratitude en quelques lignes lorsque l'on doit tant et à autant de personnes. Je suis particulièrement reconnaissant envers mon directeur de thèse, Balázs Kégl, dont l'enthousiasme scientifique, la gestion "googlienne" de l'équipe et la constante bienveillance ont été et seront, pour moi, d'une inspiration incommensurable. Merci pour ta confiance Balázs.

Un grand merci aussi à Róbert Busa-Fekete, alias Robi, qui a accompagné mes débuts au sein de l'équipe APPSTAT et avec qui j'ai collaboré et appris. Une pensée particulière va aussi à mon ami Rémi Bardenet qui a le don de toujours présenter les mathématiques avec simplicité et clarté.

Je voudrais aussi remercier les personnes que j'ai eu le privilège de côtoyer tout au long de mon doctorat pour les échanges passionnants et enrichissants dont j'ai pu bénéficier, l'atmosphère scientifique fertile que chacun entretient et leur bonne humeur invariable. Merci à Marcel Urban, Sylvie Dagoret-Campagne, Cécile Germain, François-David Collin, Diego Garcia Gamez, Yacine Bekri, Sourav Prasad Mishra et à Darko Veberic. Vous m'inspirer tous beaucoup d'humilité.

J'aimerais aussi chaleureusement remercier Damien Ernst et son équipe de recherche pour la semaine fructueuse passée à Montefiore. J'exprime aussi ma sincère gratitude envers Vladimir (Vava) Gligorov avec qui j'ai eu le privilège de collaborer et dont les précieux conseils m'ont été extrêmement bénéfiques.

Il y a deux enseignants qu'il me tient à coeur de remercier, et avec beaucoup d'humilité, monsieur Slimani pour sa passion et son enseignement des mathématiques (et de l'éthique d'ailleurs...) au lycée "sbitar" et madame Michèle Sébag qui est à l'origine de mon engouement pour l'apprentissage statistique.

Je remercie chaleureusement les personnes avec qui j'ai eu l'honneur d'enseigner à l'université Paris-Sud et dont j'ai beaucoup appris, notamment Daniel Etiemble, Mathieu Dubois, Michel Menou et Frédérique Coquelle.

Je remercie également mes rapporteurs ainsi que tous les membres de mon jury de thèse d'avoir accepté de participer à la concrétisation de ces trois dernières années de travail.

J'aimerais aussi rendre hommage à tout le personnel administratif du LAL dont le travail a grandement contribué au bon déroulement de cette thèse. Merci à Sylvie Prandt, Valentine Marcuola, Brigitte Renard, au service mission et bien sûr à Françoise Maréchal pour son travail remarquable à la bibliothèque du LAL. Merci à Stéphanie Druetta au secrétariat de l'école doctorale d'informatique. Merci aussi à Estelle pour les jus d'orange pressés qui ont été essentiels durant les derniers moments de la rédaction. Un grand merci au service informatique et à l'équipe de StratusLab, notamment à Cal et à Mohammed.

Je suis aussi éternellement endetté envers mes parents à qui je dois tout, littéralement. La distance ne les a jamais empêché d'être présents tout au long de cette thèse. Mes pensées vont aussi à ma famille qui m'a toujours soutenu et encouragé, à mes grandmas pour leur *du'aa* (qui faisaient compétition avec ceux de Hani) ainsi qu'à toutes les personnes qui sont "devenus" ma famille et qui ont aussi, chacune à sa manière, contribué à la réalisation de cette thèse, Mohamed, Brahim, Roberto, Zaki, Imad et tant d'autres !

Merci, Lilia, pour ton admirable soutien et pour ta constante affection.

Enfin, je me dois de rendre hommage à toute la communauté du logiciel libre et open-source ainsi qu'à la fondation Wikimedia à qui nous devons tous beaucoup.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Contents

1.1 Preamble

The human being always sought to extend his capabilities. Despite being endowed with limited sensory organs, the creative faculties of his brain, that unveiled no boundaries yet, have always allowed him to go far beyond his sensory limitations and dig out secrets of the world that would have remained deeply buried otherwise. He, for instance, can see but had to go beyond the limits of his eyes to invent the microscope and investigate extremely small objects; he went even further than the visible wavelengths and invented the electron microscope that no longer "sees with light". The same is true for extremely far objects that are now observed with telescopes. From that perspective, we can look at the invention of computers as yet another step towards extending the human capacities with the exception that, this time, the extensions concern the brain itself. This idea in fact is not new and dates back to the appearance of the first computers, back in the 40's, when scientists began to foresee that the machines would replace at some point the human being for certain tasks. The high speed at which processors execute instructions is an asset, in and of itself, that offers a great advantage for repetitive and simple programmable tasks, however, one fundamental goal that scientists have been striving for over decades is to exhibit *intelligent* behavior out of machines, thus the emergence of a whole field of research named *Artificial Intelligence* (AI) (Russell and Norvig, 2009).

Machine learning (ML) is a descendant of AI and as such, it shares with its ancestor the goal of making machines mimicking human beings. In particular, it quickly became a more concrete realization of the idea that machines can extend human brain. Being inductive by definition, ML methods are *fed* with real data, directly taken from nature. So does the human brain. Furthermore, ML algorithms remain powerful even when this data lies in high dimensional spaces and depend on a large number of dimensions. For comparison, a normally constituted adult under normal conditions can hardly exceed a couple of criteria taken altogether when it comes to making decisions and reasoning. An average person can typically hold about 4[1] items simultaneously in mind (Miller, 1956; Cowan, 2001), ML algorithms, on the other hand, produce better results with more data. In that respect, we see ML as a way to, once again, extend our capabilities in territories where our senses fail but machines excel.

A concrete example of this idea is also found in practice. In modern particle physics experiments, physicists usually have to deal with enormous amounts of data and measurements, in order to study a given phenomenon. The typical approach in analyzing this data is to isolate regions of interest in the multidimensional space of measurements on the basis of the theoretical properties of the studied phenomenon. Traditionally, these *cuts* on the variables values have been done manually and for every dimension separately, however, ML techniques, such as Boosted Decision Trees, have shown during the past years to be very effective during the phase of data exploration and analysis (Hoecker et al., 2007). Besides being objectively driven by the data, their most appealing aspect lies in the multivariate discriminant functions they produce, which obviously tend to consistently outperform the traditional univariate approach.

Targeted audience

This manuscript is mainly targeted to the machine learning community, however, we also address the particle physics community since the scientific environment in which the presented works were conducted, the initial motivating problem and the main practical application are all related to the field of high energy particle physics. It is not intended to present a unified language which both communities could refer to, but we try nevertheless to make the content accessible to the particle physics community.

It is always a bit delicate to address two different communities, in particular when they have an overlapping terminology, sometimes used for different concepts. Since our main contributions are in machine learning, we made the choice to alleviate the terminological ambiguities mainly in the introduction and keep a pure

[1] A bigger number is sometimes cited but evidence tend to show that beyond 4 items, the brain uses smart trickery instead of actually storing the items.

ML terminology in the remaining chapters. Also, we summarize the different terminological parallels in Table 1.1 at the end of this chapter.

1.2 Machine learning

Machine learning refers to a broad family of algorithms that inductively extracts models from data in order to perform some specific tasks. If this definition is generic, it nevertheless reflects the wide range of applications this sort of algorithms can lead to, in particular in a time where data is preponderant, both in industry and scientific research (Mitchell, 2006). Industry is nowadays completely penetrated by machine learning algorithms with diverse applications like advertisement placement, face recognition systems, text translation and categorization, speech recognition, fraud detection in banking transactions as well as many applications in robotics. Scientific research is no exception in the use of machine learning such as in genetics, drug design, but also in fields like cosmology and high energy physics. In some cases, this adoption even led to the institutionalization of almost fully new independent fields like biostatistics and, more recently, data science.

When categorizing machine learning algorithms, we often distinguish three major paradigms, mainly depending on the data we have to deal with, namely

- Supervised Learning,
- Unsupervised Learning,
- and Reinforcement Learning.

In supervised learning, the data is presented as a set of pairs, each of which comprises the observation (also called instance or example) and its label[2]. The goal is to infer a function that maps the observations to their labels through a process called the training or learning phase. The learned function, sometimes referred to as the *predictor*, is meant to provide the correct label for any new data that it similar to the already known one. In other words, it must *generalize* the prediction to unknown observations.

Getting labels is often expensive, error prone or even impossible. When the available data is not labeled, we refer to the corresponding learning methods as unsupervised learning. The objective is usually to discover hidden structure in the data, find its compounding clusters or reduce its dimensionality before some further processing or for visualization purposes.

[2] We sometimes find in the literature that the instance refers to both the observation and label together, however, we find that distinguishing the instance from its label is more coherent with the fact that we also classify an instance at test time, even though its label is unknown.

The third category, Reinforcement Learning (RL), is sometimes described as an intermediate paradigm between supervised ans unsupervised learning. The data come in fact from the interaction of a learning agent with a source of information called the environment and is said to be *evaluative* data (and not informative), i.e, unlike in supervised learning, the environment does not provide a definitive answer or label but only gives a feedback to the actions that the learning agent takes. By analogy with reinforcement learning in behaviorist psychology, the feedback is usually called *reward*. Another important line of distinction between RL and the other types of learning is the sequential dimension of the interaction between the agent and the environment. We talk about episodes, usually modeled as state machines, which consist in alternating state transitions and actions taken by the learning agent. In short, the agent takes an action and the environment responds with a reward that evaluates that action along with a new state. The goal of RL methods is either to estimate the expected sum of the rewards over an episode for the different states, which is known as *planning*, or to optimize the decision making of the agent so as to maximize the expected sum of rewards. The latter case is known as a *control* problem.

Transversely, the research has focused on specific cases such as when only a subset of the available data is labeled, raising the question of how supervised learning methods can still take advantage of the unlabeled part, this is referred to as *semi-supervised* learning and if, furthermore, the learner can actively query for new data points, the problem is known as *active* learning. Other transverse family of algorithms concern the cases wherein the data is provided one instance at a time and not as a bunch, which is known as *on-line* learning or whether one can build a model for a type of problems and still apply it to another related problems, commonly known as *transfer* learning questions.

1.3 Supervised learning notation

In supervised learning, we learn a function $g : \mathcal{X} \to \mathcal{L}$ from a dataset

$$\mathcal{D} = \big\{(\mathbf{x}_1, \ell_1), \dots, (\mathbf{x}_n, \ell_n)\big\} \in (\mathcal{X} \times \mathcal{L})^n$$

comprised of pairs of observations and labels. The elements $x^{(j)}$ of the d-dimensional *feature* vector $\mathbf{x}$ are either real numbers or they come from an unordered set of cardinality M_j. When the label space $\mathcal{L}$ is real-valued, the problem is known as *regression*, whereas when the labels come from a finite set of classes, we are talking about *classification*.

In *multi-class* classification, the label ℓ of the observation $\mathbf{x}$ comes from a finite set. Without loss of generality, we will suppose that $\ell \in \mathcal{L} = \{1, \dots, K\}$. We will refer to the label of the observation $\mathbf{x}$ as $\ell(\mathbf{x})$. For technical reasons that will become clear later, we will encode the label using a K-dimensional binary vector $\mathbf{y} \in \mathcal{Y} = \{\pm 1\}^K$. In the classical multi-class setup, $\mathbf{y}$ is known as the *one-hot* representation: the $\ell(\mathbf{x})$th element of $\mathbf{y}$ will be 1 and all the other elements will be -1, that is,

$$y_\ell = \begin{cases} +1 & \text{if } \ell = \ell(\mathbf{x}), \\ -1 & \text{otherwise.} \end{cases}$$

This representation has the advantage to be generalizable to *multi-label* learning when and observation $\mathbf{x}$ can belong to several classes. From now on we will call $\mathbf{y}$ and ℓ the label and the label *index* of $\mathbf{x}$, respectively.

The general multi-class training data is thus

$$\mathcal{D} = \big\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\big\} \in (\mathcal{X} \times \mathcal{Y})^n,$$

and the goal of learning is to infer a vector-valued multi-class classifier $\mathbf{g} : \mathcal{X} \to \mathcal{Y}$ from $\mathcal{D}$. Sometimes we will use the notion of an $n \times d$ *observation matrix* of $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and an $n \times K$ *label matrix* $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ instead of the set of pairs $\mathcal{D}$.

As in binary classification, learning algorithms usually output a multi-class *discriminant* function $\mathbf{f} : \mathcal{X} \to \mathbb{R}^K$. The semantics of $\mathbf{f}$ is that the higher the ℓth element $f_\ell(\mathbf{x})$ of $\mathbf{f}(\mathbf{x})$, the more likely is that the real label index of $\mathbf{x}$ is ℓ. The vector-valued classifier $\mathbf{g}$ is formally obtained by simply thresholding the elements of $\mathbf{f}(\mathbf{x})$ at 0, that is,

$$g_\ell(\mathbf{x}) = \operatorname{sign}\big(f_\ell(\mathbf{x})\big), \quad \ell = 1, \ldots, K.$$

The single-label output of the algorithm is then the class index ℓ for which $f_\ell(\mathbf{x})$ is maximal, that is,

$$\ell_{\mathbf{f}}(\mathbf{x}) = \arg\max_\ell f_\ell(\mathbf{x}).$$

The classical measure of the performance of the multi-class discriminant function $\mathbf{f}$ is the *single-label one-loss*

$$L_{\mathbb{I}}\big(\mathbf{f}, (\mathbf{x}, \ell)\big) = \mathbb{I}\left\{\ell \neq \ell_{\mathbf{f}}(\mathbf{x}_i)\right\}$$

that defines the single-label training error

$$\widehat{R}_{\mathbb{I}}(\mathbf{f}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\left\{\ell(\mathbf{x}_i) \neq \ell_{\mathbf{f}}(\mathbf{x}_i)\right\}. \tag{1.3.1}$$

1.4 The training workflow

In supervised learning, a common way to obtain an estimation of the future performance of a trained classifier once it is used with real data, is to split the dataset $\mathcal{D}$ in two folds, $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$ so that the learning is done on $\mathcal{D}_{train}$ and the output classifier is tested on $\mathcal{D}_{test}$. Another benefit of this setup is to detect *overfitting*. Overfitting occurs whenever the trained models fits so well the training data that it also encodes its random variations. When the learner overfits, its performance on the training set carry on decreasing while it increases on the test set. In other words, it no longer *generalizes*. Figure 1.1 illustrates this phenomenon with a 2D toy problem[3]. In order to accurately detect when this happens, it is usual to plot the so called *learning curve*, depicting the training set error (or another performance measure) against the running time (or the iteration, for iterative algorithms, or other complexity parameters). An example of such a learning curve is shown in Figure 1.2.

During the learning, the parameters of the trained model are tuned (or added) in order to achieve the learning goal, however, the learning algorithms themselves can have parameters that influence the learning and that must be set before the training starts. Examples of such parameters are the number of decision nodes in a decision tree or the number of units in a neural network. They are called *hyperparameters* and in order to correctly tune them for a given problem, we add a third step to the two steps of training and test, that is the *validation* step. The principle is to split the data set in three folds, $\mathcal{D}_{train}$, $\mathcal{D}_{valid}$, and $\mathcal{D}_{test}$ so that the training is done with $\mathcal{D}_{train}$ as usual, the selection of the hyperparameters is done on $\mathcal{D}_{valid}$, and $\mathcal{D}_{test}$ is used to estimate the generalization error using the selected set of hyperparameters.

1.5 Fast and budgeted classification

The subject of this dissertation fits into the supervised learning category and tackles the problem of classification. In particular, we are interested in cases where the classification process is subject to strong constraints such as computational and time constraints. These questions, often raised when it comes to apply classification algorithms in practice, have not known as much interest as for the generic theoretical framework; so far, the focus has been usually put on the classification performance of the learning algorithms and their statistical guarantees. Object detection in images, such as face detection, is a typical example of constrained classification problems because it is often applied in embedded environments and requires a real-time classification. The problem appears to be even harder when we know that the common process for detecting an object inside a image consists

[3] http://commons.wikimedia.org/wiki/File:Overfitting.svg

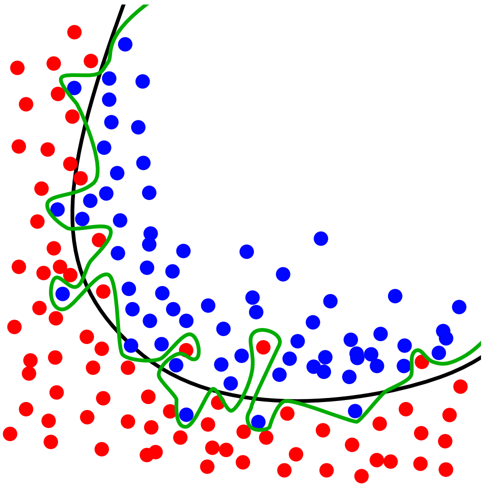


Figure 1.1 – A depiction of overfitting in classification with a two-dimension binary problem. The black decision boundary is more likely to generalize to new data points than the green one, despite the fact that the green curve makes fewer errors on the training points.

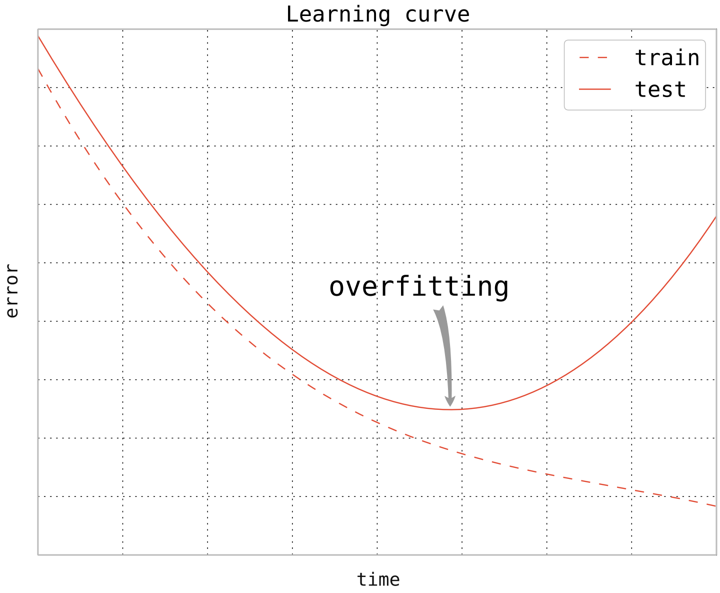


Figure 1.2 – An example of a learning curve with overfitting.

of sweeping this image with a sliding window that covers all locations in the image at different scales. Thus, the problem is no longer that of classifying an image accurately, for which a plethora of solutions already exists, but how to correctly classify thousands of images within a frame rate between 10 and 24 images per second.

Even though early object detectors in images could achieve satisfying performance Roth et al. (2000); Schneiderman and Kanade (2000), the breakthrough in real-time face detection happened with the work of Viola and Jones (2004), introducing cascade classifiers (Section 2.3). Conceptually, the idea was to turn the classification into smaller sequential classification steps, called stages, that can be early preempted. If simple and fast classifiers are put at the early stages, they can allow most of the observations to be classified with a small complexity, by sequentially processing them and discard them as soon as a certain classification confidence is attained. In the Viola-Jones cascade, the discarded instances were the non face images and the overall classifier was efficient because of the highly skewed nature of the detection problems in term of class imbalance and because the non faces were easier to classify that the face images. As it is pointed out by Schneiderman (2004), the idea of cascade classifiers dates back to the 70s and 80s as it was widely used for automatic target recognition, however, the seminal work of Viola and Jones as well as the subsequent improvements extended the initial concept and it became since then the framework of reference for a variety of problems where fast classification is involved, such as in web page ranking (Cambazoglu et al., 2010) and structured prediction (Weiss et al., 2012).

1.6 Similarity with particle physics experiments problems

Interestingly, a very similar type of problems to that of detecting objects in images arises in modern particle physics experiments in which, not surprisingly, the same concepts of cascaded classification are employed. The Large Hadron Collider (LHC) for example produces on average 600 million collisions per second, corresponding roughly to 1 Petabyte of data per second but due to limited storage capacities, only a minor fraction of these *events* can be kept, thus an extremely fast classifier, called the *trigger*, must operate on-line in order to detect the "most interesting" events and discard the rest, usually labeled as *background*. For comparison, in ATLAS, one of the four experiments of the LHC, the trigger is expected to only select 200 events per second. This can seen as a big decrease in the so called *trigger rate* but it is still relatively high, knowing that the Higgs boson, the targeted particle, is expected to be produced at a rate of 0.01 Hz. The triggers in modern particle colliders are also structured in stages (referred to as *levels* in the physics terminology) where the *acceptance rate*, i.e, the number of retained events, decreases step-wise from a level to another. Naturally, the lowest levels use raw measurements in order to perform a fast classification but the high levels, as they receive less data to process, employ constructed features that are more discriminative but also more expensive to compute. In particular, some of them take more time to compute than the other. When the features incur an acquisition cost and, in particular, when a subset of the features is cheaper to acquire than the others, we usually talk about budgeted learning. The idea behind budgeted learning is to use the least costly features as much as possible so as to decrease the average classification cost.

Besides real-time and budgeted classification, we also find a number of other motivations for investigating fast classifiers. The problem is, for example, related to medical diagnosis where one needs to obtain an answer based on different medical tests but, because the tests might have secondary effects on the patient or because some of them (usually the most informative ones) are expensive, we need to prescribe as few tests as possible, while maintaining a mandatory level of accuracy. Another issue of interest concerns the reduction of energy waste. The motivation behind can be either purely technical, as for the spatial stations that must keep strict energy policies, pecuniary or even ecological, the driving principle being that reducing computation also reduces energy consumption.

In order to avoid dilution, our focus in this manuscript will be particularly on particle physics experiments, as it was our primary motivation, and also on object detection in images, because of all the similarities that exist between these two types of problems and the shared nature of solutions that have been proposed for both types of problems. Within these two applications, we investigate what constitutes the key ingredient in the efficiency of the cascades, the sequential classification, and propose a method that tackles the problem in a novel way.

1.7 The LHCb trigger

We are initially motivated by another of the four experiments inside the LHC, namely the LHCb experiment. In this experiment, the data is produced at a rate of 16 MHz and the trigger must only let the data to be stored at a rate of 2 kHz. This tremendous amount of data to discard, which reminds of the original motivation behind cascade classifiers, is not the only learning challenge. In fact, the features that the classification is based on have different computational costs that depends on many factors. The acquisition of some features requires the computation of other features beforehand which, overall, creates a network of dependency among the features. Also, there is a subset of features the cost of which depends on their actual value (reflecting the idea that some high values are easier to compute and necessitate less computational time, such as particle momenta). Furthermore, in order to compute the classification cost of an observation, one must also take into account the fact that the observations come in groups (or bags) corresponding to the same physics event and that among instances of the same group, some feature computations are shared. These constraints are atypical in machine learning and, in order to provide a fully data-driven, automatic procedures to train efficient classifiers in that context, one must investigate new learning methods. As we show in later chapters, our direction of investigation points towards sequential models.

1.8 Overview

In the next chapter, we introduce boosting methods. We start with this chapter for mainly two reasons, this family of learning algorithms has been employed recurrently when it concerned fast classification. Their additive nature allows them to precisely control the complexity of the prediction and, furthermore, they are employed in most of the works on cascade classifiers and fast classifiers. The second reason is that boosting methods are an important part in our approach. Even though theoretically not necessary, they contribute greatly to the application of our method in practice for tackling fast classification. Chapter 2 also reviews cascade classifiers, their origin and the different subsequent works. Budgeted learning in the framework of boosting methods is also reviewed, as it is directly related to fast classification and in particular to our motivating problem. We also describe in this chapter an efficient and modular library, implementing some boosting and cascades algorithms, that we maintained and extended during the course of this thesis.

In Chapter 3, we describe a method for fast classification that was initially inspired by cascade classifiers but finally differs in many ways from classical approaches. We first define a fast classification framework that lead us to introduce sequentiality in order to satisfy all our requirements, then we propose our Algorithm, named MDDAG (for Markov Decision Directed Acyclic Graph), that represents a concretization of the aforementioned framework. The main idea behind MDDAG is that we explicitly cast the classification as a decision making procedure and solve the new problem through a reinforcement learning approach. The resulting algorithm is fast and accurate but also extremely flexible in the sense that it can accommodate various practical constraints such as feature costs. Throughout this chapter, we illustrate the different steps of building MDDAG using illustrative problems and, later, show experiments conducted on different classical machine learning datasets in order to confirm the efficiency of the proposed method. At the end of the chapter, we also review other works that, similarly to ours, cast the initial prediction problem into a decision making task.

In Chapter 4, we apply MDDAG to our initial particle physics problem. After explaining the non trivial cost calculations used for the features employed within the LHCb trigger, we adapt MDDAG accordingly and show that it perfectly takes the new logic into account.

Finally, 5 concludes this dissertation and provides an overview for future works.

Machine Learning	**High Energy Physics**
Instance, example, observation	Candidate, data point
Label	Response
Bag of instances	Event
Attribute, column, feature	Variate, covariate, variable
Training, Learning	Fitting
Generalization	Test set performance
Hyperparameter	Parameters of the learning algorithm itself, by contrast with those of the output model. Eg. depth of the trees, number of iterations etc.
Validation	The selection of the appropriate set of the hyperparameters, usually measured on a separate dataset. This is why we usually speak about a three-fold data splitting, referring to training, validation and test.
Base classifier	Either a cut on one feature augmented with a decision or a small classification tree.
Strong (, Final, or Ensemble) classifier	The classifier that combines multiple base classifiers.

Table 1.1 – Machine learning vs high energy physics glossary.

CHAPTER 2

Boosting based frameworks

Contents

Suppose that, for diagnosing a disease, we do not know any good doctor who can provide a reliable answer most of the time with a high confidence, but instead, we have access to a large, possibly infinite, number of "bad" doctors that only provide the right diagnosis barely more than half of the time, and on different subsets of patients for each doctor individually. The question that arises then is: can we could pick up a subset of these doctors, endow each of them with a weight and combine them so that their weighted majority vote would provide a reliable diagnosis?

The question of combining simple decision rules in order to produce an overall reliable predictor was first raised in Machine Learning by Kearns and Valiant (1994), following a previous work on PAC models (Kearns and Vazirani, 1994), a family of models suitable for analyzing machine learning algorithms. Boosting algorithms came later to answer this question positively. The gist of boosting methods is to iteratively construct an ensemble of *weak classifiers*, for example an ensemble of "bad" doctors, so that their combination forms an overall reliable classifier, called the *final* (or *strong*) classifier. In every iteration, a subroutine called the *weak learner* is invoked to select the appropriate weak classifier for a given subset of the observations, and once selected, the weak classifier is assigned a coefficient and then added to the overall pool. More generally, the weak learner can be given the entire training dataset along with a distribution over the observations, which is often referred to as the examples weights.

The *weakness* of a classifier literally means that we do not assume its performance to be strong, i.e, close to perfect classification, nevertheless, it is not to be confused with a trivial classifier that does not conveys any relevant information. In the binary classification for instance, the "weak learnability" is defined by the condition for a weak learner to output a classifier with a smaller error rate than 50%, in other words, it must perform better than if we simply flipped a coin at random. Note that we also use the term "base" instead of "weak" to emphasize the compositional aspect of the weak learner, both terms remain equivalent however.

Boosting methods have met a great success during the past decades and many boosting algorithms were proposed to solve different kinds of problems, mainly in multi-class classification, regression and ranking. The main axes of variation between the different algorithms lie in the following two question:

- how to set the weights of the examples before the weak learning,
- and how to set the coefficient of every weak classifier.

The flexibility that the boosting framework provides and its additive nature makes it a privileged framework for fast classification. In this dissertation, we mainly review the boosting algorithms that are related our issue of interest, i.e, fast and budgeted classification. For a comprehensive coverage of boosting, the literature is full of excellent reviews (Schapire, 2003; Meir and Rätsch, 2003), we particularly refer the reader to the book of Freund and Schapire (2012).

2.1 Adaboost

AdaBoost (Freund and Schapire, 1997) was historically the first efficient polynomial-time boosting algorithm. It has been since then subject to a throughout theoretical study and led to a plethora of successful applications like in text categorization, face detection or even experimental analysis in particle physics (Gligorov and Williams, 2012).

2.1.1 Description

Figure 2.1 depicts the pseudocode of AdaBoost in the binary case. The algorithm takes as input a training set

$$\mathcal{D}_m = \big\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\big\}$$

where $\mathbf{x} \in \mathcal{X}$ and $y \in \{-1, +1\}$. It also expects a base learner $\textsc{Base}(\cdot, \cdot)$, and a number of iterations T.

The main idea behind ADABOOST learning is to maintain a set of positive weights on the examples. This set of weights sums to 1 so we call it a distribution and we denote the weight of the i^{th} example by $\mathbf{w}^{(i)}$. Initially, the examples are equally weighted (line 1) and are updated in every iteration.

The learning procedure consists of calling $\textsc{Base}(\cdot, \cdot)$ repeatedly, for T iterations, with the same training dataset but with varying weights on the examples (line 3). In the simplest case, $\textsc{Base}(\cdot, \cdot)$ outputs a binary classifier $h(\mathbf{x}) \in \{\pm 1\}$. Once, in a given iteration t, a weak classifier $h^{(t)}$ is found, it is assigned a coefficient $\alpha^{(t)}$ proportional to its performance. Note that the performance is measured with respect to the *weighted error* $\varepsilon^{(t)}$, defined by the sum of the weights of misclassified examples. Intuitively, the $\alpha^{(t)}$ coefficient represents the importance that particular weak classifier in the final decision, lower errors will naturally correspond to more important base classifiers.

At the end of the iteration, we increase the weight of the misclassified examples and the correctly classified examples get their weight decreased (lines 8 and 10 respectively). This reweighting scheme has the effect of letting the next iteration emphasize more on *hard* examples when choosing the base classifier.

The final classifier corresponds to the weighted sum of different votes of the base classifiers

$$f^{(T)}(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot)$$

The empirical efficiency of ADABOOST has been widely demonstrated in various applications. Furthermore, it has been shown that ADABOOST minimizes the training error exponentially fast as the number of combined weak classifiers grows and that it guarantees a bounded generalization error (Freund and Schapire, 2012). ADABOOST has been extensively studied theoretically but one element appears to be particularly important to emphasize in order to fully understand the subsequent sections, namely the study of ADABOOST as an optimization procedure.

2.1.2 Gradient descent boosting

Many machine learning algorithms contain or consist of an optimization procedure with an explicit *loss function* to minimize. Even though ADABOOST was not designed explicitly as such, Mason et al. (2000) proved that ADABOOST fits into a more generic optimization framework, which they call *AnyBoost*. In the specific case of ADABOOST, the loss function that is minimized is so called the exponential loss function

$$\frac{1}{m} \sum_{i=1}^{m} \exp\left(-y_i f^{(T)}(\mathbf{x}_i)\right),$$

```
AdaBoost$(\mathcal{D}_m, \textsc{Base}(\cdot,\cdot), T)$
$\mathbf{w}^{(1)} \leftarrow (1/m, \dots, 1/m)$    ▷ initial weights
for $t \leftarrow 1$ to $T$
    $h^{(t)} \leftarrow \textsc{Base}(\mathcal{D}_m, \mathbf{w}^{(t)})$    ▷ base classifier
    $\varepsilon^{(t)} \leftarrow \sum_{i=1}^{m} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\}$    ▷ weighted error of the base classifier
    $\alpha^{(t)} \leftarrow \frac{1}{2} \ln\left(\frac{1-\varepsilon^{(t)}}{\varepsilon^{(t)}}\right)$    ▷ coefficient of the base classifier
    for $i \leftarrow 1$ to $n$    ▷ re-weighting the training points
        if $h^{(t)}(\mathbf{x}_i) \neq y_i$ then    ▷ error
            $w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2\varepsilon^{(t)}}$    ▷ weight increases
        else    ▷ correct classification
            $w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2(1-\varepsilon^{(t)})}$    ▷ weight decreases
return $f^{(T)}(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot)$    ▷ weighted "vote" of base classifiers
```

Figure 2.1 – The pseudocode of binary AdaBoost.

This view of AdaBoost allowed to derive new boosting algorithms by modifying the loss function to minimize. In fact, both the choice of the base classification and its coefficient in each iteration, as well as the reweighting scheme of the examples, result directly from minimizing this loss function.

Given a loss function L, the choice of the base classifier h and the coefficient α at each the iteration $t+1$ result from the following minimization

$$h_{t+1} = \arg\min_{h} \left. \frac{\partial L\big(f^{(t)} + \alpha\, h\big)}{\partial \alpha} \right|_{\alpha=0}$$

$$\alpha_t = \arg\min_{\alpha} \frac{\partial L\big(f^{(t)} + \alpha\, h_{t+1}\big)}{\partial \alpha}$$

and the new weight of the example $\mathbf{x}_i$ corresponds to

$$\frac{\partial L\big(f^{(t)}\big)}{\partial f^{(t)}(\mathbf{x}_i)}.$$

2.1.3 Multi-class ADABOOST

ADABOOST can be extended to multi-class classification in several ways. The seminal paper of Freund and Schapire Freund and Schapire (1997) described two multi-class extensions of ADABOOST, namely ADABOOST.M1 and ADABOOST.M2 but they both required a quite strong assumption from the base learner's performance. For the training error to be decreased, the base learner is required to achieve a minimum training error of 50%, which can be problematic in the multi-class et in particular given the hard distributions that the algorithm produces. A particularly successful multi-class extension, however, came later in Schapire and Singer (1999). Schapire and Singer described, among other interesting algorithms, ADABOOST.MH (Figure 2.2) which can be seen as a way to solve K *one-vs-all* classification problems by maintaining a $m \times K$ weight matrix over the training examples and the classes

$$\mathbf{W} = \left[w_{i,\ell}\right]_{i=1\dots,m}^{\ell=1\dots,K}$$

The boosting procedure then seeks to minimize the *weighted multi-class exponential loss*

$$\widehat{R}_{\text{EXP}}\big(\mathbf{f}^{(T)}, \mathbf{W}\big) = \frac{1}{n}\sum_{i=1}^{n} L_{\text{EXP}}\big(\mathbf{f}, (\mathbf{x}, \mathbf{y}), \mathbf{w}_i\big) \tag{2.1.1}$$

with the convex loss L_{EXP} defined as

$$L_{\text{EXP}}\big(\mathbf{f}, (\mathbf{x}, \mathbf{y}), \mathbf{w}\big) = \sum_{\ell=1}^{K} w_\ell \exp\big(-f_\ell(\mathbf{x}) y_\ell\big).$$

that upperbounds the classical multi-class hamming loss

$$L_{\text{H}}\big(\mathbf{f}, (\mathbf{x}, \mathbf{y}), \mathbf{w}\big) = \sum_{\ell=1}^{K} w_\ell \mathbb{I}\left\{\text{sign}\big(f_\ell(\mathbf{x})\big) \neq y_\ell\right\}$$

ADABOOST.MH outputs a vector-valued discriminant function

$$\mathbf{f}^{(T)}(\mathbf{x}) = \sum_{t=1}^{T} \mathbf{h}^{(t)}(\mathbf{x}) \tag{2.1.2}$$

as a sum of T multi-class base classifiers $\mathbf{h}^{(t)} : \mathcal{X} \to \mathbb{R}^K$ returned by a *base learner* algorithm $\text{BASE}\big(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)}\big)$ in each iteration t.

2.2 MULTIBOOST

The simplicity of ADABOOST makes it extremely easy to implement, especially in the binary case, so one might expect to find a large number of implementations

```
AdaBoost.MH($\mathbf{X}, \mathbf{Y}, \mathbf{W}, \text{Base}(\cdot,\cdot,\cdot), T$)
    $\mathbf{W}^{(1)} \leftarrow \frac{1}{n}\mathbf{W}$
    for $t \leftarrow 1$ to $T$
        $\big(\alpha^{(t)}, \mathbf{v}^{(t)}, \varphi^{(t)}(\cdot)\big) \leftarrow \text{Base}\big(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)}\big)$
        $\mathbf{h}^{(t)}(\cdot) \leftarrow \alpha^{(t)}\mathbf{v}^{(t)}\varphi^{(t)}(\cdot)$
        for $i \leftarrow 1$ to $n$ for $\ell \leftarrow 1$ to $K$
            $w_{i,\ell}^{(t+1)} \leftarrow w_{i,\ell}^{(t)} \frac{\exp\left(-h_\ell^{(t)}(\mathbf{x}_i)y_{i,\ell}\right)}{\underbrace{\sum_{i'=1}^{n}\sum_{\ell'=1}^{K} w_{i',\ell'}^{(t)} \exp\left(-h_{\ell'}^{(t)}(\mathbf{x}_{i'})y_{i',\ell'}\right)}_{Z\left(\mathbf{h}^{(t)}, \mathbf{W}^{(t)}\right)}}$
    return $\mathbf{f}^{(T)}(\cdot) = \sum_{t=1}^{T}\mathbf{h}^{(t)}(\cdot)$
```

Figure 2.2 – The pseudocode of the AdaBoost.MH algorithm. $\mathbf{X}$ is the $n \times d$ observation matrix, $\mathbf{Y}$ is the $n \times K$ label matrix, $\mathbf{W}$ is the user-defined weight matrix used in the definition of the weighted exponential margin-based error (2.1.1), Base$(\cdot,\cdot,\cdot)$ is the base learner algorithm, and T is the number of iterations. $\alpha^{(t)}$ is the base coefficient, $\mathbf{v}^{(t)}$ is the vote vector, $\varphi^{(t)}(\cdot)$ is the scalar base (weak) classifier, $\mathbf{h}^{(t)}(\cdot)$ is the vector-valued base classifier, and $\mathbf{f}^{(T)}(\cdot)$ is the final (strong) discriminant function.

freely available but in reality, there is a real lack of implementations that are both efficient and extensible, as far as we know.

MultiBoost (Benbouzid et al., 2012a) is boosting toolbox that aims to fill this gap by providing a framework that efficiently implements some boosting algorithms in C++ as well as providing the basic "buiding blocks" to implement new algorithms.

The toolbox mainly focuses on AdaBoost.MH to provide an off-the-shelf multi-class/multi-label boosting software but also implements other boosting based algorithms as well

- FilterBoost (Bradley and Schapire, 2008): an online "filtering" booster.
- VJCascade (Viola and Jones, 2004): an algorithm that learns a cascade classifier (c.f section 2.3) by running AdaBoost at each node.
- SoftCascade (Bourdev and Brandt, 2005): another cascade learner that starts with a set of base classifiers, reorders them, and augments them with rejection thresholds.

MultiBoost was designed with modularity so that the implementation of the boosting algorithms (the strong learners) and the weak learners are clearly decoupled. New weak learners can be implemented without modifying the existing code and would immediately be available to combine with the existing strong learners.

2.2.1 Implemented base learners

MULTIBOOST implements the following base learners.

- The SINGLESTUMP learner is a one-decision two-leaf tree learned on real-valued features. It is indexed by the feature it cuts and the threshold where it cuts the feature.
- SELECTOR is a one-decision two-leaf tree learned on nominal features. It is indexed by the feature it cuts and the value of the feature it selects.
- INDICATOR is similar to SELECTOR but it can select several values for a given feature (that is, it can *indicate* a subset of the values).
- HAARSTUMP is a STUMP learned over a feature space generated using rectangular filters on images.
- TREE is a *meta* base learner that takes any base learner as input and learns a vector-valued multi-class decision tree using the input base learner as the basic cut.
- PRODUCT is another meta base learner that takes any base learner as input and learns a vector-valued multi-class decision product (Kégl and Busa-Fekete, 2009) using the input base learner as terms of the product.

The code of MULTIBOOST has been fully documented in Doxygen.[1] It is available under the GPL licence at `multiboost.org`.

2.3 Cascade classifiers

Most of the successful learning algorithms for classification produce classifiers with an algorithmic complexity such that they can not be directly deployed in real-time situations or when some strong constraints are imposed on the classification time. The boosting family of classifiers is no exception. As we combine more classifiers, we might improve the classification accuracy but we also increase the time needed for classifying each observation. Take the example of face detection in images. Building a face detector by boosting decision stumps on the pixel values, i.e, without any feature engineering, typically necessitates a couple of hundreds base classifiers before reaching an accuracy beyond 90% (Figure 2.3). Using

[1] See `www.doxygen.org`.

more elaborate features, such as Haar-like features, improves the performance but still produce relatively complex classifiers (Figure 2.4). We are indeed far from the dozens of base classifier typically needed to obtain real-time performance. It is not trivial therefore, to satisfy both the accuracy and the complexity requirements, even though we have access to an accurate algorithm on the one hand, and a fine control on its complexity on the other hand.

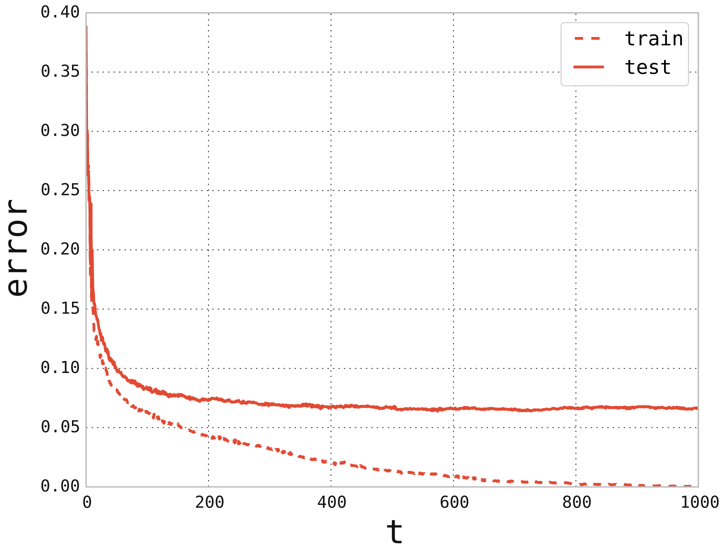


Figure 2.3 – ADABOOST learning curve on a face detection dataset. The base learner consist in a decision stump on the pixel values.

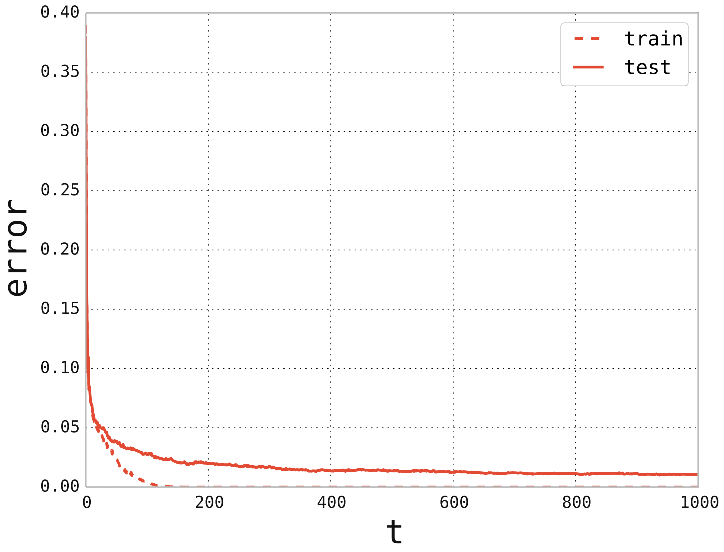


Figure 2.4 – ADABOOST learning curve on a face detection dataset, with Haar-like features.

The fact that boosting algorithms, being additive models by nature, offer a direct and precise control on the complexity of the final classifier actually was the keystone for a whole family of solutions to the problem of combining accuracy with speed in classification, namely the concept of *cascade classifiers*.

The crux of the cascade idea is to decompose the whole classification process into a set of smaller classification steps, called stages, that each observation would have to traverse sequentially. Then, if the early stages of the cascade are simple enough and the observations are allowed to stop this sequential process as soon as some confidence level about the final decision is reached, we end up lowering the average number of evaluated stages over all the observations, thus potentially obtaining fast classifiers without sacrificing the accuracy. Boosting techniques particularly fit into this framework as the number of base classifiers in each stage directly determines its complexity and can be set to gradually increase with the stage index.

Figure 2.5 depicts an example of this structure containing 4 stages (the rectangles), each of them is a strong classifier. The plain circles in each stage represent base classifiers and the arrows between stages show the path that each example can take, in this case, either access the next stage or stop the classification (represented by the ∞ circle). The cascaded classification is effective when most of the examples can be classified in the early stages and only a minor subset arrives to the last stage. This situation occurs for example when a subset of the classes is prevalent, by several orders of magnitude, over the other classes or when one of the classes, such as noise, uninteresting physics events, or simple background in the case of image classification, needs to be "filtered" out. In the general case, classifying sequentially becomes interesting when some observations can be more *easily* classified than others, requiring a less complex classification in order to predict their label correctly.

2.3.1 The original Viola-Jones cascade

The seminal and first successful work on cascade classifiers was introduced by Paul Viola and Michael Jones (VJ) (Viola and Jones, 2004) in which they used the cascade structure to design the first real-time face detector. The success of their application was mainly due to a precise control of the classifier's complexity *via* the following building blocks

- a pertinent choice of image features, namely the Haar-like features[2], which not only avoid the limitations of the raw pixels, but can also be computed efficiently;
- the choice of ADABOOST to learn a classifier by selecting an exact number of features per stage;

[2] In the MULTIBOOST package, Haar-like features correspond to the HAARSTUMP base learner.

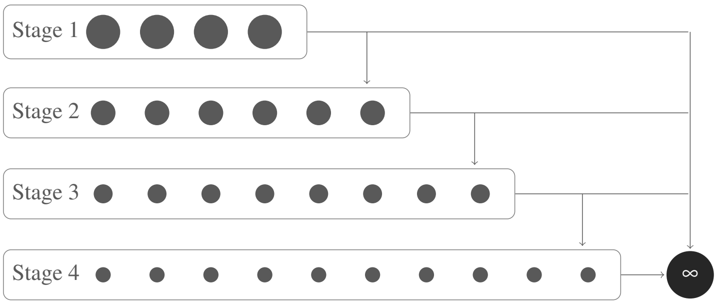


Figure 2.5 – The structure of the Viola-Jones cascade. Rectangles represent stages, circles represent base classifiers. The width of the base classifiers in each stage represents the portion of examples that gets into that stage. Each observation is sequentially classified by the stages, starting from stage 1, and at the end of each stage, it either calls the evaluation of the next stage or it stops the classification, ending at the ∞ circle. Thus, every observation follows a specific *path* along the cascade. In Chapter 3, we keep this visualization in order to show how we can give more flexibility to the paths taken by the observations.

- and the cascade structure of the whole detector.

Given an image that contains a face at an unknown location, the common approach for the detection is to sweep over the whole image with a sliding window over all the possible scales and locations. Because this process generates an enormous amount of non-face sub-images (the negatives, also called *background*), the cascade in this case acts as a stage-wise filter, eliminating a portion of the background at each stage, while letting almost all of the positive sub-images, the faces, go to the subsequent stage. Thus, each stage can only classify the negative instances except the last stage which can also classify positives as well. Given an image that contains a face at an unknown location, the common approach for the detection is to sweep over the whole image with a sliding window over all the possible scales and locations. Because this process generates an enormous amount of non-face sub-images (the negatives, also called *background*), the cascade in this case acts as a stage-wise filter, eliminating a portion of the background at each stage, while letting almost all of the positive sub-images, the faces, go to the subsequent stage. Thus, each stage can only classify the negative instances except the last stage which can also classify positives as well. Because the number of base classifiers in the first two stages is relatively small, most of the background sub-images ends up being classified with a small complexity. The algorithm in Figure 2.6 shows the classification algorithm of the VJ cascade provided an input of two vectors, one containing the stages and the other containing the corresponding thresholds.

```
VJCascadeClassifier($\mathcal{F}, \Theta, \mathbf{x}$)
for $s \leftarrow 1$ to $||\mathcal{F}||$                ▷ iterating on the element of the vector $\mathcal{F}$
    $f_s(\cdot) \leftarrow \mathcal{F}(s)$             ▷ fetch the $s^{th}$ classifier from the vector $\mathcal{F}$
    $\theta_s \leftarrow \Theta(s)$                   ▷ fetch the $s^{th}$ threshold from the vector $\Theta$
    if $f_s(\mathbf{x}) < \theta_s$ then return $-1$
return $+1$
```

Figure 2.6 – The pseudocode of the Viola-Jones Cascade classifier. The input of the algorithm are $\mathcal{F}$, the vector of stages and Θ, the vector of thresholds. The stages are evaluated sequentially. As soon as the score the observation falls below the stage's threshold, it is classified as negative. Otherwise, after the evaluation of the last stage, it is classified as positive.

Training a VJ cascade

Training an optimal VJ cascade is far from being trivial. The stages are trained individually and greedily until the *overall false positive rate* falls below a prespecified level $F_{overall}$. Two other hyper-parameters that concern the stages individually must be set as well in order to control the cascade accuracy and complexity, namely the maximum acceptable false positive rate r and the minimum acceptable detection rate d. During the training of a stage, the base classifiers keep being added until the maximum acceptable false positive rate r is satisfied. r, however, is not controlled directly. Instead, the temporary strong classifier being constructed is thresholded in order to satisfy d, the minimum acceptable detection rate, and this thresholding affects the false positive rate. Once a stage is trained, it is fixed for the rest of the learning. The algorithm is described with more details in Figure 2.7

Because the hyperparameters only control the false positive and detection rates, there is no direct control on the accuracy/complexity trade-off and there is no guarantee on whether the resulting cascade is optimal or not. Moreover, the training necessitates a considerable amount of negative observations, otherwise, the negative class can quickly become underrepresented in the latest stages. To overcome this issue, Viola and Jones proposed what they call the *bootstrapping* (line 17 in Figure 2.7) which consists in injecting new false positive instances after the training of each stage. Drawn for a large database of negatives, the instances are passed through the cascade and the ones that make it through as positives are used in the training of the subsequent stage.

```
VJCascadeLearner$(\mathcal{D}_m, \text{Base}(\cdot,\cdot), r, d, F_{overall})$
$F_0 \leftarrow 1.0, D_0 \leftarrow 1.0$    ▷ initial stage-wise rates
$s \leftarrow 0$    ▷ stage index
$\mathcal{F} \leftarrow \emptyset, \Theta \leftarrow \emptyset$    ▷ vector of stages and thresholds respectively
while $F_s > F_{overall}$
    $s \leftarrow s + 1, n_s \leftarrow 0$
    $F_s \leftarrow F_{s-1}$
    while $F_s > r \times F_{s-1}$
        $n_s \leftarrow n_s + 1$
        $f_s(\cdot) \leftarrow$ AdaBoost$(\mathcal{D}_m, \text{Base}(\cdot,\cdot), n_s)$
        $(F_s, D_s) \leftarrow$ Evaluate$(\mathcal{D}_m, \mathcal{F}, \Theta, f_s(\cdot))$
        $\theta \leftarrow 0$
        while $D_s < d \times \mathcal{D}_{s-1}$
            $\theta_s \leftarrow$ DecreaseThreshold$(\mathcal{D}_m, \mathcal{F}, \Theta, f_s(\cdot), \theta)$
            $(F_s, D_s) \leftarrow$ Evaluate$(\mathcal{D}_m, \mathcal{F}, \Theta, f_s(\cdot))$
    $\mathcal{F} \leftarrow \mathcal{F} \cup f_s$
    $\Theta \leftarrow \Theta \cup \theta_s$
    if $F_s > F_{overall}$ then Bootstrap$(\mathcal{D}_m, \mathcal{F}, \Theta)$
return VJCascadeClassifier$(\mathcal{F}, \Theta, \cdot)$
```

Figure 2.7 – The pseudocode of the Viola-Jones Cascade training algorithm. EVALUATE is a simple function that counts the number of false positives and false negatives. DECREASETHRESHOLD simply decreases θ by a certain amount, it is left generic on purpose so that it's efficiency depends on its actual implementation. It is left generic. BOOTSTRAP augments the training dataset with new false positives.

2.3.2 Improvements and variations

The Viola-Jones cascades inspired many subsequent works that also followed the approach of classifying the instances sequentially, trying to overcome the limitations of the initial work. Some focused on the feature level, others on the training algorithm used, and some works proposed new cascade architectures. We are particularly interested in the cascade structure itself, since it is this architecture that allows to classify sequentially, but we briefly review some improvements in the feature employed since it can help understand further sections.

Haar-like features and variants

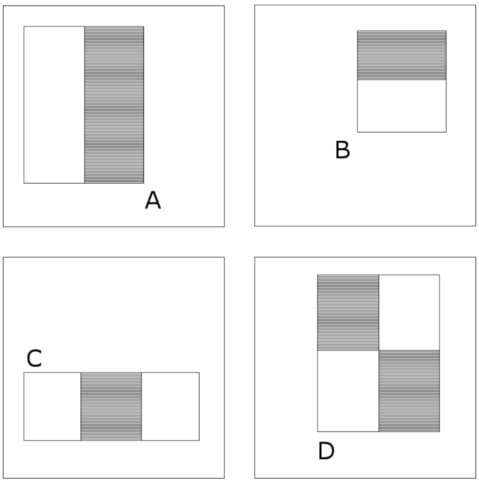


Figure 2.8 – The four categories of the original Haar-like features. The grey area of each feature is summed up and subtracted from the sum of the white area. The resulting value is then thresholded with the decision stump algorithm.

The features used in VJ CASCADE, the Haar-like features, consist summing the pixels within rectangles and applying basic addition/subtraction operations on them. Viola and Jones for example proposed to use 4 types of features (Figure 2.8) where the difference in the sum of the pixels within adjacent rectangles is computed and thresholded to form the base classifier. This type of features, although simple, shows good performance when boosted.

In order to compute the sums of the pixels efficiently, Viola and Jones preprocess all the images before the training. They replace every pixel $image(x,y)$ in an image i by the sum of the pixels above and to the left of (x,y), producing what they call an *integral image* ii:

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x,y)$$

This transformation can be applied in one pass by applying the following recurrences (Viola and Jones, 2004)

$$\begin{aligned} s(x,y) &= s(x,y-1) + i(x,y) \\ ii(x,y) &= ii(x-1,y) + s(x,y) \end{aligned}$$

$s(x,y)$ being the cumulative row sum, $s(x,-1) = 0$, and $ii(-1,y) = 0$.

Applying the integral image transformation allows to compute the Haar-like features in constant time by only using the rectangle corner pixels. The rectangle's sum in Figure 2.10 for example simply consists of the following pixels' value sum $C + A - B - D$

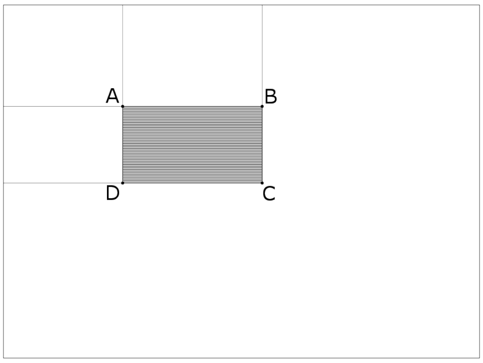


Figure 2.9 – The sum of pixels within the rectangle $ABCD$ can be calculated only with the value of its corner pixels and is equal to C + A - B - D.

Figure 2.10 – The sum of pixels within the rectangle $ABCD$ can be calculated only with the value of its corner pixels and is equal to C + A - B - D.

Haar-like features, although being simple, are able to capture some type of correlation in pixels, corresponding to difference of contrasts along lines, edges etc. Lienhart and Maydt (2002) extended the original 4 categories of features by adding rotated features (Figure 2.11) in order to capture 45° edges. Many other works focused on improving the features selected by the boosting algorithm during the training of a stage. Li et al. (2006) allowed to place a distance between rectangles in the same feature, allowing more flexibility. Viola et al. (2005) augmented the Haar-like features with a one-pixel-shift operator, encoding the local variations that occur inside videos and motions in the same region within consecutive frames. Since it is not our main subject to review extensively the features used for detecting objects in images, we refer the reader to the excellent review of Zhang and Zhang (2010).

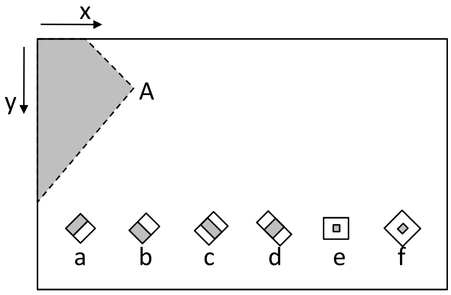


Figure 2.11 – Rotated Haar-like features.

The learning algorithm

While some works were interested in the features employed inside the boosted classifiers, others focused particularly on the cascade itself, its shape, how to set its thresholds and the boosting algorithm within. Viola and Jones (Viola and Jones, 2001) noticed that the requirements of each stage in terms of minimum detection rate and maximum false positive rate can be difficult to satisfy altogether. If, for example, a 10-stage is to have an overall detection rate of 0.95%, the average minimum detection rate per stage must be equal to $\sqrt[10]{0.95} \approx 0.995$. This is not obvious to obtain if, at the same time, the cascade targets an overall false positive rate of 10^{-6}, requiring each stage to discard 25% of the observations. If this issue particularly concerns the later stages, another phenomenon hinders the training of the first stages. A strong classifier with N base classifiers will output at most 2^N different score values for all the example. If N is small, it can be difficult to threshold the corresponding strong classifier in order to increase the detection rate without making big increasing "jumps" in the false positive rate. Viola and Jones (2001) introduced a variation of ADABOOST that takes into account the different mis-classification costs among the classes. Although this cost-sensitive version of ADABOOST is not directly based on theoretical justifications (cf. Section 2.1.2), it already improves the cascade's performance by inducing a bias towards classifying the positives correctly. The concept of including cost-sensitivity into boosting algorithms has been studied by a number of papers (Fan et al., 1999; Ting, 2000; Sun et al., 2005), and in particular in the context of cascade classifiers (Hou et al., 2006; Pham and Cham, 2007), but most of them consists in heuristically upweighting the positive examples during each boosting iteration in order to induce a bias. One noticeable work, however, provides theoretical justifications through the gradient-descent framework of boosting (Masnadi-Shirazi and Vasconcelos, 2011) wherein the authors directly minimize a cost-sensitive loss function.

Another direction in the attempt to improve the boosting algorithm inside the cascade has been to investigate the use of *real* ADABOOST (also known as REALBOOST) which allows the base classifiers to output real valued scores, corresponding the classification confidence (Xiao et al., 2003; Li et al., 2002). In their empirical analysis, Lienhart et al. (2003) conclude that GENTLEBOOST, another variation of ADABOOST, provides better performance than its two experimental contenders, ADABOOST and REALBOOST, in addition to using less base classifiers. Furthermore, Li et al. (2002) were also motivated by reducing the number of base classifiers in each stage and in order to reach this goal, they included the idea of FLOATING SEARCH, originally used in feature selection, into the boosting algorithm in order to backtrack the least relevant base classifiers and remove them from the pool. The resulting algorithm, named FLOATBOOST uses less classifiers but has been shown empirically not to provide a stable decrease in the error rate, as the number of base classifiers increases (Wu et al., 2004).

Stage-less cascades

In the original VJ CASCADE, the stages are independent boosted classifiers, therefore, when an instance is passed to the next stage, its score is set back to zero before the classification starts in new stage. In other words, the information, in terms of classification score and confidence, that every stage computes is not carried onto the subsequent stages and is definitively lost. A number of papers addressed this issue and tried to take advantage of the cumulated knowledge within a cascade.

Various approaches were proposed, Wu et al. (2004) introduced the "boosting chain" cascade in which every stage L_{i+1} $(i > 0)$ is initialized with its predecessor L_i. In other words, L_i plays the role of the first base classifier trained in L_{i+1}. The resulting cascade is then similar to a classical boosted classifier augmented with intermediate rejection thresholds, which provides it with the same generalization properties as ADABOOST.

Xiao et al. (2003) proposed a very similar idea to the "boosting chain" (Wu et al., 2004) under the name "nesting structured cascade". As far as we know, the only difference between these two cascades lies in the use of *real* ADABOOST inside the nesting-structured cascade, in replacement of ADABOOST (also called *discrete* ADABOOST)[3].

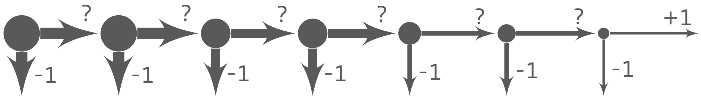


Figure 2.12 – The structure of the Soft Cascade. The stages consist of only one base classifier each, which allows to exploit the cumulative knowledge of the cascade.

Similarly, the SOFT CASCADE (Bourdev and Brandt, 2005) has the form of a long boosted classifier with intermediate rejection thresholds, with the exception that every stage consists of only one base classifier and hence there are as many thresholds as there are classifiers (Figure 2.12). The training of the SOFT CASCADE is done in two steps, first ADABOOST is trained to obtain a series of base classifiers and then, the base classifiers are reordered so that the most discriminative ones are placed at the head of the chain. While reordering the base classifiers, their thresholds are set following an exponential function family parametrized by one hyperparameter. This hyperparameter explicitly determines if the cascade rejection policy is conservative, thus slow, or if it is permissive and fast. Sun and Zhou (2013) go further in the exploration of the effects of reordering the base classifiers. They first propose a representation of the score evolution as the base classifiers are sequentially evaluated, from which they analyze different policies for early classification (for example whenever the score of an observation reaches a certain value so that the subsequent base classifiers can not alter the classification decision any-

[3] However, most of the papers reviewing both works treat them as two distinct type of cascades.

more), and then they propose a reordering method called SIFTING in order to optimize the early classification.

Noticeably, the general idea of reusing the cumulated information across the cascade has consistently shown improvements in the cascade performance, both in terms of accuracy and speed. Our algorithm (Chapter 3) also follows this approach.

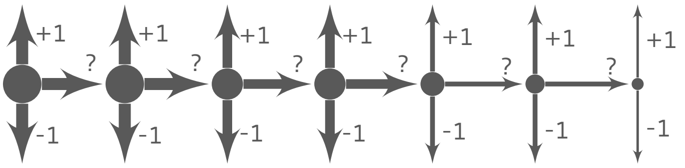


Figure 2.13 – The structure of Waldboost. The stages, are endowed with two thresholds in order to discard both positives and negatives.

```
WALDBOOSTCLASSIFIER($\mathcal{F}, \Theta, \mathbf{x}$)

for $s \leftarrow 1$ to $||\mathcal{F}||$                ▷ iterating on the element of the vector $\mathcal{F}$
    $f_s(\cdot) \leftarrow \mathcal{F}(s)$              ▷ fetch the $s^{th}$ classifier from the vector $\mathcal{F}$
    $(\theta_s^A, \theta_s^B) \leftarrow \Theta(s)$     ▷ fetch the $s^{th}$ couple of thresholds from $\Theta$
    if $f_s(\mathbf{x}) < \theta_s^A$ then return $-1$
    if $f_s(\mathbf{x}) > \theta_s^B$ then return $+1$
```

Figure 2.14 – The pseudocode of Waldboost classifier. The input of the algorithm are $\mathcal{F}$, the vector of stages and Θ, the vector of threshold couples. Each stage is endowed with two thresholds for classifying both positives and negatives.

Tuning the thresholds

One fundamental question related to designing cascades is how to set the rejection policy. The most naive rejection policy (Viola and Jones, 2004; Bourdev and Brandt, 2005) consists of a single threshold per stage that is set to let most of the positives carry on their *path* to the next stages while only a fraction of the negatives is correctly classified and discarded. (Zhang and Viola, 2011) improve the speed of the cascade through a simple idea: the positive examples that are anyway misclassified by the last stage can be discarded at earlier stages without decreasing the training error, thus the thresholds are set accordingly which allows the cascade to discard more negatives per stage. They further improve the cascade speed *via* a multiple-instance framework, in which they focus only on a subset of positive sub-images, corresponding to adjacent sub-windows in the original image, hence the term of *multiple instance pruning*.

Following the same structure as in the SOFT CASCADE, the authors of WALDBOOST (Wu et al., 2004) propose to set two thresholds for discarding both positives and negatives (Figure 2.13). Based on the Wald's sequential probability ratio test, they optimize these thresholds for every base classifier and show that the final classifier is faster since it can also early classify positives, however, WALDBOOST does not show exceptional performance in terms of accuracy, comparing to other cascades. The classification algorithm of a cascade with two thresholds is described in Algorithm 2.14. Similarly, Póczos et al. (2009) augment the VJ CASCADE with a couple of threshold per stage in order to early classify positives as well as negatives. In this work, and contrary to WALDBOOST, the training procedure does not need to estimate the likelihood ratios and the problem is set into a Reinforcement Learning framework. In Sun and Zhou (2013), the authors do not directly compute thresholds but instead define an "escape interval" of every base classifier, which corresponds to the range of scores in which the classification decision becomes definitive, even when only a subset of the base classifiers has been evaluated.

The automatic training and the optimality of the cascade

Most of the cascades (Viola and Jones, 2004; Xiao et al., 2003; Bourdev and Brandt, 2005; Brubaker et al., 2008) are tuned through a set of hyperparameters which do not directly control the trade-off between the accuracy and the complexity. As for the original VJ CASCADE, one needs to set the targeted overall accuracy as well as the stage-wise performance. Not only this makes the training of the cascade tedious and particularly long[4] because of the important number of features (often Haar-like features) and instances needed to achieve good results, but it raises also the question about how to get the optimal performance, given a limit on the complexity for example or for a specific accuracy-complexity trade-off in general. The question of the cascade's optimality has not been directly addressed in early works, in fact, most of the algorithms that were proposed for designing the first cascades consisted of heuristics that focused on decreasing the overall classification cost. Thus, no theoretical work addressed both the accuracy and the complexity of the final classifier and provided guarantees on its performance.

Within a generic framework for cascades, Brubaker et al. (2006, 2008) propose a cascade analysis, both in terms of classification accuracy and complexity, from which they derive an automatic procedure for the training and for setting the different thresholds. The stages, however, are still trained sequentially in a greedy fashion. Saberian and Vasconcelos (2010) construct a global loss function, taking into account the overall accuracy, as well as the overall cascade complexity including all the stages. They combine the new loss function with the gradient boosting framework (Section 2.1.2) in order to obtain a fully automatic cascade learning

[4] It usually necessitates a couple of weeks to manually tune a cascade but it used to take month with the early works.

where the base classifiers are no more added to the last stage exclusively but can also update the other stages as well. The resulting algorithm, named FCBoost, consists of boosting iterations and is tuned by a central hyperparameter which directly controls the accuracy-complexity trade-off. (Lefakis and Fleuret, 2010) also propose a joint learning of all the stages, based on a noisy-and probabilistic model that is minimized with a boosting algorithm.

Another critical aspect of the cascade learning is the use of *bootstrapping* (Sung and Poggio, 1998). By definition the role the cascade structure is to filter as many negatives as possible, therefore, as the number of stages increases, the number of available negatives for training becomes insufficient. On the other hand, training AdaBoost with a tremendous amount of negatives with the purpose to overcome the latter issue can quickly degrade the quality of the learned classifier, as the learning objective would tend to constantly classify as negative. The solution that is commonly used, the bootstrapping, is to have at disposal an enormous database of negative instances that we pass through the cascade at each stage in order to collect the false positives and use them for the training of a new stage. Wu et al. (2004) drew a parallel between bootstrapping and the weighting scheme of boosting algorithms by considering the former as a weighted sampling of negative instances. In the case of FCBoost (Saberian and Vasconcelos, 2010), the stages are not constructed sequentially so the authors only replace a fraction (another hyperparameter, usually fixed once for all) of the negatives by new false positives at each iteration.

Multi-class cascades

After frontal face detector were proven to be efficient and applicable to real-time detection, the focus has also been put on extending this work to multi-pose face detectors. Since the cascade was originally designed for binary problems, where one of the two classes was designated as background and meant to be step-wise filtered out, it appears that when it comes to detecting varying faces, or objects in general, the original cascade architecture does not provide as good performance as for upright frontal faces. In fact, it is difficult to take into account a large inner-class variability while assuring an acceptable false positive rate, so the problem has to be turned into a multi-class classification but it is not straightforward to adapt the cascade efficiency to multi-class problems. A number of papers, motivated by multi-pose face detection and multi-class cascades in general, addressed the issue by coining new cascade architectures. The most straightforward architecture is to simply have parallel cascades, one for each class except the background that is to be discarded. Obviously, this is the least effective solution and it goes against the initial objectives of the cascades in terms of speed as we have more classes. For multi-pose face detection, this can still be applicable since the different poses are not completely separate classes and one cascade can be trained to handle a whole range of pose angles but for large multi-class problems in general where each class is distinct, this solution is not applicable. Wu et al. (2004), along with the nesting-structured cascade, propose a variant of the parallel cascade approach by including a pose estimator at the beginning of each cascade, making possible for the instances to jump from a cascade to another at the exit of the pose estimator. This estimator has itself a nesting-structure and prefixes the corresponding cascade. During the classification, the observation is sent to all the stages and after the first base classifiers that perform pose estimation, the instance is potentially redirected towards another cascade. Jones and Viola (2003) also include a post estimator at the beginning of parallel cascades, the estimator is trained with a decision tree and the observations are finally treated by only one cascade. The drawbacks of such an approach are obviously that errors made by the pose estimator are definitive and can drastically decrease the overall accuracy of the detector. In Li and Zhang (2004), the authors apply FLOATBOOST to multi-pose face detection by hierarchically split the pose estimation following a coarse-to-fine approach. This *pyramid detector* refines the pose estimation as the observation goes from a level of the pyramid to a lower level and can also discard the observation at the end of each level.

All the architectures made of distinct cascades for the different classes suffer from the drawback of not exploiting the inter-class correlations, which hints towards the fact that this architecture can be optimized in order to be more efficient. Zehnder et al. (2008) describe an algorithm for constructing a parallel cascade having shared features among the stages. The resulting cascade has a Directed Acyclic Graph shape and exhibits a sub-linear scalability with the number of classes.

Lin and Liu (2005); Huang et al. (2005) propose two similar approaches, MBH-BOOST and VECTOR BOOSTING respectively, based on a vector-valued boosting algorithm (Section 2.1.3). The crux of these ideas is to make the base classifiers share the same features across the classes and thus reduce the overall computational cost. Zhang and Zhang (2009) use a fully boosted classifier for pose estimation. The originality of their boosting algorithm, named WTA-MCBOOST, is to assign virtual labels to the training examples so their label gets adapted to the real class as the learning proceeds. Their algorithm is particularly suited for large datasets which is typically the case in object detection and images.

Note: In particle physics experiments, one often needs to measure different phenomena and select observations of interest through the on-line classifier (the trigger). In fact, it often happens that even the background to be discarded is composed of different classes. Thus, coupling the efficiency of the cascades with multi-class classifiers is of particular interest for our motivating problem. We address this issue when defining our approach in Chapter 3 as well as in our main application in Chapter 4.

Tree-shaped boosted classifiers

In Kim et al. (2011), a trained boosted classifier is transformed into a tree structure by casting the problem into a boolean expression optimization. The authors show that this tree classifier performs as good as the original strong classifier with a considerable decrease in the evaluation time. Although not motivated by fast classification, Freund and Mason (1999) also propose an algorithm for constructing tree-shaped strong classifiers which can be viewed as a generalization of decision trees. Each node of the tree is a base classifier and instead of summing up the weighted vote of the all the base classifiers, the score of a classified observation only involves the traversed nodes, giving at the same time more intelligibility to the classification. In Tu (2005), the author proposes a procedure to grow a tree of classifiers in which the nodes are full strong classifier. The resulting classifier follows a divide-and-conquer strategy and can also handle multi-class classification by splitting the data at each node based on estimated posterior probabilities.

2.4 Budgeted learning

Another type of problems which is related to fast classifications appears when we are to manage a certain budget that is consumed as we evaluate the different features; we speak then about the *feature costs* and *budgeted* classification. The type of the budget varies from an application to another, it can be some computational constraints as in particle physics experiments, a set of different tests in medical diagnosis, or more generally, when one must provide an answer within a limited time frame. The difference with the problem of fast classification treated above appears in particular when the features have different costs, so that the total cost of classifying an observation depends on which subset of features was in fact evaluated during the process.

Once again, the family of additive models in general and boosting methods in particular appears to provide a suitable framework for budgeted classification problems. (Reyzin, 2011) proposes an adaptation of AdaBoost in which a strong classifier is first trained, and during test-time, this pool of base classifiers is sampled following a distribution induced by their weights. In a variant algorithm from the same paper, the feature costs also influence the sampling distribution making the cheapest features more likely to be sampled. This work is supported by both theoretical foundation and empirical analysis and the resulting classifier is shown to perform almost identically as AdaBoost while only using 20% of the features.

Grubb and Bagnell (2012) also adopt the boosting framework, through minimizing a tailored loss function with functional gradient descent (Section 2.1.2). The algorithm, named SpeedBoost, minimizes a loss function that incorporates the base classifiers' costs and at each iteration, the base classifier with the best ratio between the loss decrease and complexity is selected. The authors further apply this algorithm to build a cascade following the SOFT CASCADE structure.

Very similarly, "The Greedy Miser" (Xu et al., 2012b) is trained through a variation of the gradient boosting framework that directly takes into account the feature costs in the construction of the CART trees (Breiman et al., 1984) that are selected in each iteration. The difference with the work of Grubb and Bagnell (2012) mainly lies in the fact that the construction of the CART trees also involves the cost minimization so the algorithm potentially constructs more efficient trees. On the other hands, SPEEDBOOST incorporates a filtering procedure which makes the evaluation of a base classifier for a given observation only occurs if its score is low, giving a cascade aspect to the whole classifier.

In an orthogonal work, Chen et al. (2012) assume a sequence of decision trees already obtained from a boosting algorithm that belongs to the gradient boosting family and then optimizes again the coefficients of the trees in order to take into account the cost of the features employed within each tree, in addition to the performance of the overall classifier. For further efficiency, a cascade is produced by considering a global optimization of all the stages at once, which has the effect of re-ordering the decision trees in order to promote early exists with cheap features.

Xu et al. (2012a) take advantage of the tree structure in order to tackle the budgeted learning through hierarchically splitting the input space. They build a Cost Sensitive Tree of Classifiers (CSTC) in which each node is an ensemble of CART trees, obtained through a gradient boosting algorithm. They avoid the combinatorial NP-hard problem of constructing the CSTC by assuming the CART trees to be already available and through the use of soft thresholding at each node. This allows to tune the CART trees parameters along with the node thresholds *via* a global optimization procedure where the objective function includes the different feature costs. The resulting classifier is shown to be efficient, especially when large groups of instances use similar features. This is, in fact, not the first time that the hierarchical tree structure is shown to be suitable for cost-sensitive problems. Turney (1995) also build cost-sensitive trees that are optimized with genetic algorithms.

(Xu et al., 2013) tackle the problem of budgeted learning but they also propose an *anytime* solution to the problem. Anytime algorithms improve the quality of their results as they have more computational time, hence, they can be stopped whenever the user wants and provide intermediate results. (Xu et al., 2013) combine the anytime aspect of boosted classifiers with the well-known good generalization properties of large-margin classifiers (Cortes and Vapnik, 1995). In particular, they use "The Greedy Miser" (Xu et al., 2012b), that is already a budget-sensitive boosting algorithm, in order to map the input space into the scores of boosted CART trees. Then, within this new feature space, they optimize the parameters of a Support Vector Machine to maximize the margin. During the learning, the parameters of both CART trees and SVM are optimized jointly and "snapshots" of the state of these parameters are taken whenever a new feature cost is involved.

These snapshots are stored so that during test time, the CART trees are evaluated sequentially, like stages, in order to affine the input representation gradually and the corresponding SVM (embedded in the snapshot) is used for the classification.

2.5 Conclusion

In this chapter, we reviewed the main methods proposed in the literature to deal with both the accuracy and speed of the classification. We saw that the cascade classifiers succeed in combining these two criteria and are particularly suited for real-time detectors. The principles we can draw from the cascade literature is that their success is due to three factors,

- the classification is divided into sub-classification steps, each of which can be quickly executed;
- these sub-classifications are done sequentially;
- and the whole process has the ability to early-stop, for example because a certain level of confidence is reached.

Our work, presented in Chapter 3 is inspired by this *sequentiality* paradigm. It extends the typical tree structure of the cascades into a more generic Directed Acyclic Graph. Hence, it can be viewed as generalization of the cascades which provides an automatic way to deal with time-constrained classification.

Furthermore, we saw that the problem of fast classification is in fact a special case of a broader family of problems known as budgeted learning problems. In the case of fast classification, the budget corresponds to a limited time span or equivalently to some limited computational resources to comply with, however, the distinction with general budgeted learning problems becomes clearer when the different features incur different costs and, at the same time, are more relevant to different subset of observations, hence pointing even more to the fact that the classification must not be done "the same way" for all the observations. In this precise context, the sequential aspect of our approach appears to provide an advantage in terms of flexibility, in addition to exposing an intuitive and intelligible way to address this family of learning problems.

CHAPTER 3

Making decisions for classification

Contents

In Chapter 2, we reviewed a family of work that tackled the problem of fast and budgeted classification through supervised learning. These methods often consist in casting the learning problem into an optimization procedure, as in the Anyboost framework (2.1.2), by tailoring a loss function that takes into account both the classification accuracy and the evaluation cost of the final classifier. One orthogonal way to achieve the fast and budgeted learning goals is to design a cascaded classifier, where fractions of the examples are discarded stepwise. One can see the cascade classifiers as a decision framework in which subparts of the classifier *choose* whether to send the current observation for further processing or to classify it given the current information. In that respect, cascades are a step towards a more abstract framework where the classification consists of sequential decisions that vary from an instance to another. In the case of the classical cascades, the decisions are simply based on thresholding the score of the observation at a given point but the principle of having a dynamic processing of the observations can be generalized.

In this chapter, we introduce a work published in (Benbouzid et al., 2012b) where we actually extend the concept of cascades to a more generic category of fast classifiers. We explicitly cast the classification as a sequential procedure and use Reinforcement Learning algorithms to solve the resulting problem. Similarly to cascade classifiers, decisions are made for each observation individually with the purpose to produce a fast classifier but unlike cascades, the decisions are allowed to be more complex, they are not restricted to simple thresholds but instead, they are implemented as a mapping from the information available at a given point to a set of possible actions. We discuss both the advantages and disadvantages of such an approach as well as the different variations that it can lead to. This work mainly appears in (Benbouzid et al., 2012b) but some parts of it are discussed within other publications that we mention later. After describing our main algorithm, MDDAG, we also review other papers that use sequential models for prediction tasks.

3.1 Designing fast sequential classifiers

We introduce our main algorithm in this section. We follow a step by step description of the different building blocks that constitute the overall approach, in order to clearly motivate our choices. Thus, we start by defining a framework that corresponds to our motivating problem, then we refine it before actually proposing an concrete instantiation of this resulting framework. We will also often alternate formalization and concrete experiments for pedagogical purposes.

3.1.1 Instance-dependent sparsity

Let us first clearly define our objectives. Our goal is to produce classifiers which are

1. accurate and competitive with state-of-the-art performance;

2. simple and fast enough to be used in real-time;

3. flexible enough to incorporate different domain-specific constraints like feature acquisition costs.

The accuracy is, naturally, the most important criteria as it is one of the main reasons that made data-driven methods and machine learning in particular the *de facto* solutions for classification problems. Considering only the accuracy criteria, data-driven methods have proven to be extremely effective, like for producing nearly perfect classification (LeCun and Cortes, 1998) or even beating human performance Netzer et al. (2011).

However, producing accurate classifiers often comes at the price of heavy computations whereas many industrial applications require the classification of an observation to be executed in a limited time, thus with a limited amount of computation. This is typically the case for object detection in images where the classification is done in real-time or in particle physics experiments where the amount of data generated can not be entirely stored in the hard disks so it becomes necessary to only retain some kind of events. In general, the pervasiveness of mobile devices (with the evolution of embedded computing) and the new user requirements in term of of instance response, in one hand, and the enormous amounts of data that we produce nowadays, in the other hand, make the complexity issues as important as the accuracy itself.

Our last criteria comes directly from our original motivation in particle physics experiments where one not only needs to design a fast classifier but also needs to take into account the cost of acquiring the different features and, furthermore, some specific dependencies within the computation of the features. Hence, the need of flexibility in our final classifier. It is important to remember these criteria, i.e, the accuracy, the speed, and the flexibility, as they drive the different choices we make throughout this section. At the end of the chapter, we review others works that, motivated by other constraints, made different choices.

Additive models

As mentioned in Chapter 2, additive models already offer a level of control on the algorithmic complexity of the resulting classifier, both through the choice of the base classifiers in each term and the number of terms employed in the final classifier. Additionally, a common practice that makes the final classifier less prone to overfitting and consequently provides better generalization properties consists in penalizing the complexity of the final classifier. *Regularization* is a way to achieve it.

Assume a finite sequence of classifiers

$$\mathcal{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_N), \qquad \mathbf{h}_j : \mathcal{X} \rightarrow \mathbb{R}^K.$$

The corresponding additive model which combines them, given an instance $\mathbf{x} \in \mathcal{X}$, is

$$\mathbf{f}(\mathbf{x}) = \sum_{j=1}^{N} b_j \mathbf{h}_j(\mathbf{x}) \tag{3.1.1}$$

where $\mathcal{X}$ is the input space, K is number of classes, and $b_j \in \mathbb{R}$ are the coefficients of the additive model. Given a loss function L to minimize that is relative to the accuracy of the classifier $\mathbf{f}$, the regularization consists in inducing an additional cost in the loss function through a vector space norm L^p on the coefficient vector $\mathbf{b} = [b_1, \ldots, b_N]$, weighted by a trade-off parameter β, so that the overall function to minimize is

$$\sum_{i=1}^{m} L(y, \mathbf{f}(\mathbf{x}_i)) + \beta \left\|\mathbf{b}\right\|_p$$

The L^0 and L^1 norms defined respectively by[1]

$$\left\|\mathbf{f}\right\|_0 = \sum_{j=1}^{N} \mathbb{I}\left\{b_j \neq 0\right\}$$

and

$$\left\|\mathbf{f}\right\|_1 = \sum_{j=1}^{N} |b_j|$$

are known to yield *sparse* classifiers with a fraction of null coefficients which, in addition to reduce the overfitting, perform a feature selection and allow the informative features (or classifiers in our case) to emerge and give intelligibility to the final classifier. These norms work particularly well when we know that the final classifier is sparse. In our case, sparsity is used to control the complexity of the final classifier and by nullifying some coefficients, it allows the "economy" of the evaluation of the corresponding base classifiers. This, in fact, constitutes a step towards satisfying our speed criteria, however, depending on the complexity requirements for the given problem, this complexity reduction can be insufficient and, in our case in particular, it becomes quickly difficult to obtain a satisfying accuracy as we approach the extreme nature of the sparsity required for our targeted

[1] The L^0 "norm" is not a proper norm since it is not homogeneous with respect to multiplication by a scalar. However, this abuse of terminology is often made in the literature.

problems. In other words, increasing the level of sparsity with classical methods can not come without a drastic decrease in classification accuracy. Because of this strong antagonistic relationship between the accuracy and speed of the final classifier, we conjecture that to satisfy our first two criteria together, namely both the accuracy and speed, the final classifier must be sparse but also *instance-dependent*. Intuitively, instance-dependency in our case means that the complexity with which the different instances are processed varies depending on the how much resources are needed to achieve a correct classification. The resource in question can be the number of actually evaluated classifiers in the simplest case and it can also take into account the *cost* of acquiring a specific information about the instance, when one has to deal with a limited budget for example. As we describe it later in Chapter 4, this occurs in the LHCb trigger where every observation needs to be classified within a couple of milliseconds.

This *instance-dependent sparsity* is in contrast with the atomic aspect of "classical" machine learning methods (Support Vector Machines, Neural Networks etc.) where all the observations undergo the same processing and, therefore, lead to new types of learning problems. Formally, we define an instance-dependent classifier as an additive model where the coefficients b_j depend on $\mathbf{x}$, the current observation

$$\mathbf{f}_N(\mathbf{x}) = \sum_{j=1}^{N} b_j(\mathbf{x})\mathbf{h}_j(\mathbf{x}), \tag{3.1.2}$$

where

$$b_j(\mathbf{x}) = \begin{cases} 1 & \text{if the classifier } \mathbf{h}_j \text{ is evaluated} \\ 0 & \text{otherwise} \end{cases} \qquad , j = 0 \ldots N,$$

become indicator functions that induce the *instance-dependent sparsity* in the final classifier. Note that b_j is henceforth binary and the problem then corresponds to an L^0 optimization. The fact that $\mathbf{h}_j$ belongs to $\mathbb{R}$ allows us to consider its coefficient to be part of its output value, preventing us from loosing generality.

During the classification, for a given observation $\mathbf{x} \in \mathcal{X}$, we select a subset of $\mathcal{H}$ to evaluate, where each selected classifier $\mathbf{h} \in \mathcal{H}$ can *vote* for or against the class ℓ by setting its ℓth component $h_\ell(\mathbf{x})$ to, respectively, a positive or negative value, following the same semantics introduced in Section 1.3; the absolute value, again, $|h_\ell(\mathbf{x})|$ can be interpreted as the confidence of the vote. [2]

Learning the functions

$$b_j : \mathbf{x} \mapsto \{1, 0\}$$

in order to minimize

$$\sum_{i=1}^{m} \Big(L(y, \mathbf{f}(\mathbf{x}_i)) + \beta \sum_{j=1}^{N} b_j(\mathbf{x}_i) \Big) \tag{3.1.3}$$

[2] A binary classifier can still be transformed into a multiclass classifier as described in Gao and Koller (2011b) as well as in the online MULIBOOST documentation (Benbouzid et al., 2012a).

makes the regularization trivial during the training phase and offers no generalization guarantees. In fact, it would suffice to evaluate all the classifiers and then select, for each observation independently, a subset of the *correct* classifiers which satisfies the Lagrangian term within the sum in function 3.1.3. This "postdictive" selection is of course unusable since one has to evaluate each base classifier before deciding whether or not to select it.

One way to make the setup non-trivial and potentially usable is to formulate the problem as a multi-class classification where the target corresponds to the b_j functions.

Setup

Assuming a data set

$$\mathcal{D} = \big\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\big\} \in (\mathcal{X} \times \mathcal{Y})^m$$

where $\mathcal{X}$ is the input space and $\mathcal{Y} = \{\pm 1\}^K$ is the K-multi-label output space, we construct a matrix $\mathbf{B} \in \{0,1\}^{m \times N}$, that include all the b_j, s.t

$$\mathbf{B}(i,j) = \begin{cases} 1 & \text{if} \quad h_j(\mathbf{x}_i) = +1 \\ 0 & \text{otherwise,} \end{cases}$$

that is, $\mathbf{B}(i,j)$ indicates whether h_j is correctly classifying $\mathbf{x}_i$. We generate an *auxiliary* dataset by assigning to each instance $\mathbf{x}_i$ a new label $\mathbf{y}_i \in \{\pm 1\}^N$ whose k^{th} component is

$$y_i^k = (\mathbf{B}(i,k) \times 2) - 1$$

and train a classifier $\boldsymbol{\pi}$ (the control classifier) on this auxiliary data set

$$\boldsymbol{\pi} : \mathcal{X} \to \{\pm 1\}^N.$$

We use AdaBoost.MH to train both the classifier 3.1.1 (the data classifier) and the control classifier so the final classifier is

$$\mathbf{f}(\mathbf{x}) = \sum_{j=1}^{N} \big(\pi_j(\mathbf{x}) + 1\big)/2 \times \mathbf{h}_j(\mathbf{x})$$

Experiments

In order to empirically illustrate our approach, we will be using two datasets throughout this Section (c.f table 3.1). The first is a multinomial synthetic dataset (that we will refer to as the Toy dataset) that we will use to illustrate the case of binary classification and as the proof of concept. The generation of this dataset was partly inspired by the Madelon dataset (Guyon, 2003; Pedregosa et al., 2011): to generate a d-dimensional dataset with K classes, we generate m Gaussian clusters such that $K \times m \leq 2^d$, the clusters are then randomly spread over the classes. The second dataset is the Pendigits dataset from the UCI repository (Bache and Lichman, 2013) for classifying handwritten digits. This dataset is meant to serve as a realistic multi-class dataset and it is is described on the UCI website as follows:

> We create a digit database by collecting 250 samples from 44 writers. The samples written by 30 writers are used for training, cross-validation and writer dependent testing, and the digits written by the other 14 are used for writer independent testing. We use a WACOM PL-100V pressure sensitive tablet with an integrated LCD display and a cordless stylus. The input and display areas are located in the same place. Attached to the serial port of an Intel 486 based PC, it allows us to collect handwriting samples. The tablet sends x and y tablet coordinates and pressure level values of the pen at fixed time intervals (sampling rate) of 100 miliseconds. These writers are asked to write 250 digits in random order inside boxes of 500 by 500 tablet pixel resolution. Subject are monitored only during the first entry screens. Each screen contains five boxes with the digits to be written displayed above. Subjects are told to write only inside these boxes. If they make a mistake or are unhappy with their writing, they are instructed to clear the content of a box by using an on-screen button. The first ten digits are ignored because most writers are not familiar with this type of input devices, but subjects are not aware of this. In our study, we use only (x, y) coordinate information. The stylus pressure level values are ignored. First we apply normalization to make our representation invariant to translations and scale distortions. The raw data that we capture from the tablet consist of integer values between 0 and 500 (tablet input box resolution). The new coordinates are such that the coordinate which has the maximum range varies between 0 and 100. Usually x stays in this range, since most characters are taller than they are wide. In order to train and test our classifiers, we need to represent digits as constant length feature vectors. A commonly used technique leading to good results is resampling the (x_t, y_t) points. Temporal resampling (points regularly spaced in time) or spatial resampling (points regularly spaced in arc length) can be used here. Raw point data are already regularly spaced in time but the distance between them is variable. Previous research showed that spatial resampling to obtain a constant number of regularly spaced points on the trajectory yields much better performance, because it provides a better alignment between points. Our resampling algorithm uses simple linear interpolation between pairs of points. The resampled digits are represented as a sequence of T points $(x_t, y_t)_{t=1}^T$, regularly spaced in arc length, as opposed to the input se-

	Number of features	**Number of classes**	**Number of instances**
Toy	10	2	8000
Pendigits	16	9	10992

Table 3.1 – The properties of the illustrative datasets.

Data	$\|\mathcal{H}\|$	**AdaBoost**	**regularized**	**CC complexity**	**DC complexity**
Toy	10	0.363	0.314	1000	5.23
	100	0.345	0.34	1000	50.074
Pendigits	10	0.375	0.235	1000	6.325
	100	0.125	0.136	1000	54.278
	1000	0.057	0.085	545	527.57

Table 3.2 – The results of an instance-dependent sparse classifier (regularized) in comparison with the full AdaBoost with different complexities. The table also shows the number of base classifier in the Control Classifier (CC) and the average number of evaluated base classifiers in the Data Classifier (DC).

> quence, which is regularly spaced in time. So, the input vector size is $2 \times T$, two times the number of points resampled. We considered spatial resampling to $T = 8, 12, 16$ points in our experiments and found that $T = 8$ gave the best trade-off between accuracy and complexity.

Table 3.2 shows that the b_j functions can be learned efficiently, i.e, we can skip the evaluation of nearly half of the classifiers on average while still achieving an accuracy comparable to the dense classifier, if not better when the dense classifier overfits. However, mapping the observation $\mathbf{x}$ to the functions b_j can be costly. The control classifier in our case ends up containing more classifiers than the data classifier, so it is not useful if the goal is budgeted classification.

3.1.2 Sequential instance-dependent sparsity

In order to keep our data-dependent sparsity while simplifying the computation of the b_j functions, we impose a sequentiality constraint on the evaluation of the b_j functions so that the information available before evaluating every b_j is limited to *past* b_i, $i = 1 \ldots j-1$ as well as the output of the evaluated $\mathbf{h}_j$. This means that the controller can use more information than what is available in $\mathbf{x}$, but only up to the point where the inclusion of $\mathbf{h}_j$ must be decided (so no postdiction is allowed). The classification, then, consists of iterating over the classifiers and sequentially deciding whether to evaluate the current classifier or to skip it, based on the information acquired “on the way”. The problem consists of learning a sequence of functions of the form

$$b_j : \left(\mathbf{x}, b_1, \mathbf{h}_1, \ldots, b_{j-1}, \mathbf{h}_{j-1}\right) \mapsto \{0, 1\}.$$

Note that the sequentiality constraint implies putting a total order on the classifiers but, as described further, this requirement is not difficult to satisfy with a reasonable criteria.

Retrospectively, we find that the same intuition motivated the Viola-Jones cascade to become the first real-time face detector. Recall that Viola and Jones had to deal with a large number of instances to classify per second where the non-face images were prevalent but *easier* to classify than the face images.

In a sense, Viola and Jones proposed a framework to make decisions at the instance level, choosing sequentially whether to discard the image early on or to carry on the classification process if a certain level of confidence was not yet reached. In that respect, the framework we propose relates to the cascade paradigm introduced by Viola and Jones except that we aim to provide more flexibility by allowing more complex instance-level decisions while keeping the overall complexity acceptable enough to satisfy all of our three key criteria together.

3.1.3 MDDAG: Markov Decision Directed Acyclic Graph

Having explained the motivations that, in turn, allowed us to define an appropriate framework, we introduce MDDAG (for Markov Decision Directed Acyclic Graph), an implementation of this framework that allows us to learn the sequential b functions.

We use AdaBoost.MH (Schapire and Singer, 1999) to provide the set of classifiers $\mathcal{H}$. In principle, any algorithm that builds its final classifier as a linear combination of simpler functions can be used to provide $\mathcal{H}$. However, there are key advantages in using base classifiers provided by AdaBoost.MH:

- The base classifiers can have a low complexity that is suitable for our performance criteria. They typically consist of simple DecisionStumps or small trees.

- The first base classifiers output by AdaBoost have usually the best performance which will help accelerate the learning.

- The base learner decomposition in the multi-class case will help us simplify our algorithm.

As a reminder, the *final* (or *strong*) 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

$$\widehat{\ell} = \arg\max_{\ell} f_\ell(\mathbf{x}).$$

Note that in the binary case, $f_1(\mathbf{x}) = -f_2(\mathbf{x}) = f(\mathbf{x})$ and the observation $\mathbf{x}$ is classified as positive if $f(\mathbf{x}) > \theta$ and negative otherwise. The threshold θ is a free parameter that can be tuned to achieve, for instance, a specific false positive rate.

Experiments

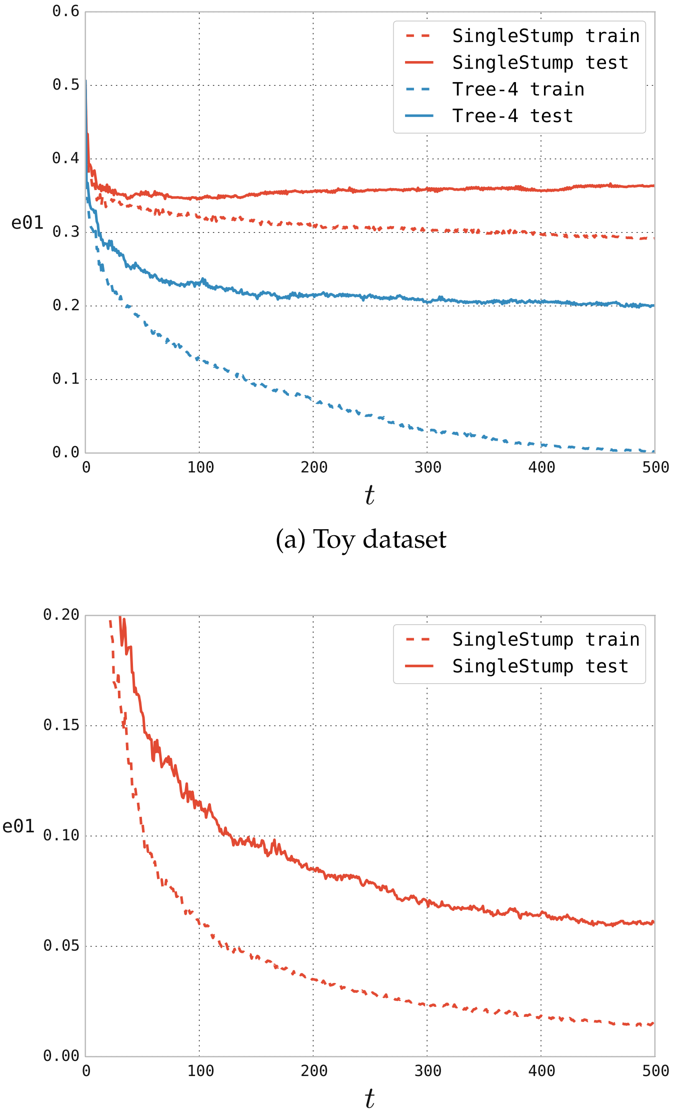
Running ADABOOST.MH on the illustrative datasets gives the learning curves in Figure 3.1. The SINGLESTUMP algorithm appears to give satisfying results on Pendigits, however, the learner quickly overfits on the Toy dataset. For this reason, we also use small 4 nodes trees (TREE-4) with the Toy dataset, which illustrate also the possibility to use different classifiers. Figure 3.1 can also be seen as a Pareto front between the accuracy of ADABOOST.MH and its complexity at different iterations. Note that we did not run ADABOOST.MH for more than 500 iterations since we are interested in very low complexity "regimes".

3.1.4 Learning $b_j(\mathbf{x})$ from delayed rewards

Sequential problems are usually tackled with Reinforcement Learning (RL) methods. In RL, the main source of information is the environment and the goal is to learn how to interact with this environment in order to perform a certain task or maximize an objective function. Each step of the interaction is defined by a *state*, in which a learning *agent* takes *actions* that influence the environment and receives some feedback about his actions and his state. The different steps that result from this interaction are called an *episode*. Unlike in supervised learning, one does not have access to *instructive* labels on the data but only to some *evaluative* feedback from the environment.

RL problems are often formalized with a Markov Decision Process in order to apply learning methods. An MDP is a 5-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where

- $\mathcal{S}$ is the state space. At time t, the agent is in the a state $\mathbf{s}_t$.
- $\mathcal{A}$ is the countable set of actions. The action that the agent takes at time t is denoted a_t.



(a) Toy dataset

(b) Pendigits dataset

Figure 3.1 – AdaBoost.MH learning curves. The curves represent the 0-1 error for the training and test set.

- $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ 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.

- $\mathcal{R} : \mathbb{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ defines the distribution $\mathcal{R}(\cdot|\mathbf{s}_t, a_t)$ of the *immediate reward* r_t for each state-action pair.

- and $\gamma \in [0, 1]$ is a discount factor. In our case, it is always set to 1.

A *deterministic policy* π assigns an action to each state

$$\pi : \mathcal{S} \to \mathcal{A}.$$

When the state transition and the reward probabilities are not known, the main approach consists in estimating the value of each action-state pair and then apply a greedy selection of the actions in each state.

The value of a state-action pair $(\mathbf{s}_t, a)$, following a policy π is defined by the expected sum of rewards starting at state $\mathbf{s}_t$, taking action a, and following the policy π,

$$Q^{\pi}(\mathbf{s}, a) = E_{\pi}\{r_t \mid \mathbf{s}_t = \mathbf{s}, a_t = a\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, \mathbf{s}_t = \mathbf{s}, a_t = a \right\}$$

In order to achieve the sequential instance-dependent sparsity, we endow each classifier $\mathbf{h}_j$ with a decision (or action), and, during test-time, we sequentially select the classifiers to evaluate for a given observation. At a given classifier $\mathbf{h}_j$, and depending on the output of the previously selected classifiers *w.r.t.* the observation, we choose an action a_j among three possible actions:

$$\begin{cases} \textsc{Eval}\textit{uate } \mathbf{h}_j, \\ \textsc{Skip } \mathbf{h}_j, \textit{or} \\ \textsc{Quit} \text{ and return the classifier built so far.} \end{cases}$$

Hence, the initial problem is cast as a decision making problem, and learning the $b_j(\mathbf{x})$ functions boils down to training a deterministic policy (or agent) π which maps a given state to one of the aforementioned actions:

$$\pi\big(\underbrace{\mathbf{x}, \mathbf{h}_1(\mathbf{x}), \ldots, \mathbf{h}_{j-1}(\mathbf{x})}_{\text{state descriptor}}\big) \mapsto \{\underbrace{\textsc{Eval}, \textsc{Skip}, \textsc{Quit}}_{\text{actions}}\}$$

Having defined a state space $\mathcal{S}$, a set of actions $\mathcal{A}$ and the dynamics of the system $\mathcal{P}$, it only remains to define the *rewards* $\mathcal{R}$ in order to complete the formalization of an episodic Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ and hence, apply RL algorithms.

During the learning, the observation $\mathbf{x}$ is drawn randomly from the distribution $\mathfrak{D}$ to form the initial state $\mathbf{s}_1$ and is fixed throughout the episode. The terminal state $\mathbf{s}_\infty$ finishes the classification with the instance-dependent score function (3.1.2) with

$$b_j(\mathbf{x}) = 1 - \mathbb{I}\left\{a_j = \textsc{Skip} \vee \exists j' < j : a_{j'} = \textsc{Quit}\right\}$$

The rewards

The crux of RL methods lies in the fact that we do not need to explicitly describe the "right way" of doing a task, unlike in supervised learning. Instead, we only provide informative feedback (the rewards) in response to the agent's actions. The agent, in turn, seeks to maximize the expected sum of the rewards (the *undiscounted return*) over the *finite horizon T* of action-state pairs

$$\varrho = \mathbb{E}\left\{\sum_{t=1}^{T} r_t\right\}. \tag{3.1.4}$$

This RL property translates into a great level of flexibility in MDDAG and satisfies our third criteria.

As the primary goal is to optimize the accuracy, we simply penalize the misclassifications of the agent at the end of the episode[3]. When the agent maximizes the return, it also minimizes a well know loss function in supervised learning, namely the *multi-class 0-1 loss function*

$$L_{\mathbb{I}}(\mathbf{f},(\mathbf{x},\ell)) = \mathbb{I}\left\{f_\ell(\mathbf{x}) - \max_{\ell'\neq\ell} f_{\ell'}(\mathbf{x}) < 0\right\}$$

Furthermore, we want to foster the sparsity of the final classifier, so we penalize the EVAL action by a factor β. The overall loss function that MDDAG minimizes is

$$\mathbb{E}_{(\mathbf{x},\ell)\sim\mathfrak{D}}\left\{L_{\mathbb{I}}\big(\mathbf{f},(\mathbf{x},\ell)\big) + \beta\sum_{j=1}^{N} b_j(\mathbf{x})\right\}. \tag{3.1.5}$$

By modifying the classification reward, $L_{\mathbb{I}}$ can be replaced by other loss functions, such as the commonly used

- multi-class exponential loss function (Schapire and Singer, 1999)

$$L_{\text{EXP}}(\mathbf{f},(\mathbf{x},\ell)) = \exp\left(\sum_{\ell'\neq\ell}^{K} f_{\ell'}(\mathbf{x}) - f_\ell(\mathbf{x})\right),$$

- or the multi-class logistic loss (Friedman et al., 1998)

$$L_{\text{LOGIT}}(\mathbf{f},(\mathbf{x},\ell)) = -\ln\left(\frac{e^{f_\ell(\mathbf{x})}}{\sum_{l'=1}^{K} e^{f_{\ell'}(\mathbf{x})}}\right)$$

[3] The penalization can be constant across the classes or can privilege a subset of classes over the rest by assigning a bigger misclassification cost.

3.1.5 The state representation

One of the key ingredients that make the RL methods successful is the right representation of the state space. Intuitively, if the state space is too complex, the learning phase might be impractical; if it is too simple, the agent might lack information to make the right decision. In our case, the state complexity plays an important role as the computation cost of the state must not overshadow the efficiency gain of the sparsity.

The *raw* state at a given time t is

$$\mathbf{s}_t = (\mathbf{x}, \phi_1(\mathbf{x}), \ldots, \phi_{t-1}(\mathbf{x}), 0, \ldots, 0)$$

where

$$\phi_t(\mathbf{x}) = \begin{cases} \mathbf{h}_t(\mathbf{x}) & \text{if } \mathbf{h} \text{ was evaluated} \\ 0 & \text{if } \mathbf{h} \text{ was skipped} \end{cases}$$

It represents all the information available to the agent and it consists of $N \times K$ continuous variables.

Note: Henceforth, we deliberately omit the observation $\mathbf{x}$ since it is fixed throughout the episode as well as the trailing zeros corresponding to the remaining classifiers at time t so the state can be expressed as

$$\mathbf{s}_t = (\phi_1(\mathbf{x}), \ldots, \phi_{t-1}(\mathbf{x})).$$

Without losing information, we can simplify the state space decomposing the base learner into

$$\mathbf{h}_t(\mathbf{x}) = \alpha_t \mathbf{v}_t \varphi_t(\mathbf{x}),$$

where $\varphi_t(\mathbf{x}) \in \{\pm 1\}$ is a binary function and $\mathbf{v}_t \in \{\pm 1\}^K$ is K-valued vector. Intuitively, $\mathbf{v}_t$ corrects the anti-correlations between $\varphi_t(\mathbf{x})$ and the label vector. Since only φ_t depends on $\mathbf{x}$, ϕ_t becomes

$$\phi_t(\mathbf{x}) = \begin{cases} +1 & \text{if } \varphi(\mathbf{x}) = +1 \\ -1 & \text{if } \varphi(\mathbf{x}) = -1 \\ 0 & \text{if } \mathbf{h} \text{ was skipped} \end{cases}$$

Using our multi-class formulation of AdaBoost.MH reduces the state space to N ternary variables.

Experiments

We run this first version of MDDAG on the two illustrative dataset as a first check of feasibility. The choice of the RL learning algorithm used is orthogonal to the state representation, for this reason, we describe it further in Section 3.1.6. Since both the accuracy and the complexity of the final classifier are crucial, we plot the Pareto front which opposes these two measures where the complexity is characterized by the average number of evaluated base classifiers. We adopt this complexity measure mainly because it does not depend on any hardware. The only condition that makes it valid, however, is that the complexity of the agent stays insignificant comparing to the overall complexity. We discuss this issue in Section 3.3.3.

Figure 3.2 shows that the learning is effective and the agent succeeds in combining the base classifiers in order to improve the accuracy of the final classifier. Figure 3.3 shows that for the same number of base classifiers used, MDDAG achieves a significantly better accuracy than ADABOOST.MH, for both the binary and multiclass case.

The drawback of this state representation is still the size of the state space which impacts the scalability of the approach wrt. the number of input classifiers to use. In fact, we quickly end up with tens of thousands of different states which, although still very far from the 3^N theoretical possible states, prevents a better generalization of the RL algorithm to new states.

Improving the state space

In order to overcome the generalization problems of the discrete state space, we *summarize* the history of an episode through the intermediate classification score up to time t:

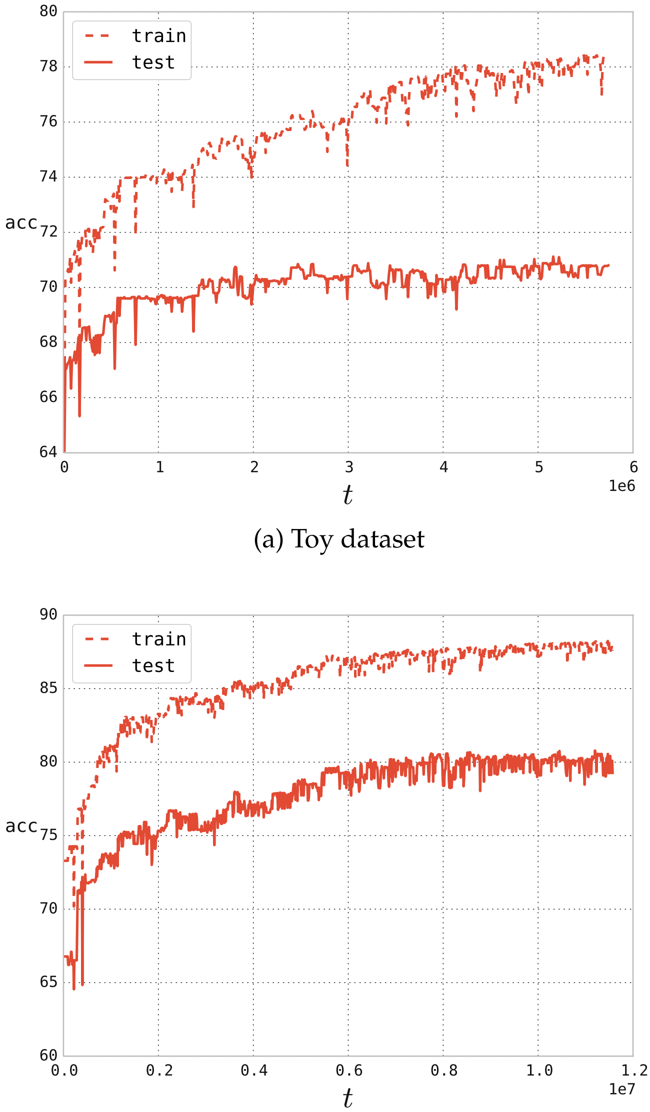
$$\mathbf{f}^{(t)}(\mathbf{x}) = \sum_{\tau=1}^{t} b_\tau(\mathbf{x})\mathbf{h}_\tau(\mathbf{x}) \qquad t < N \tag{3.1.6}$$

so the new state $\mathbf{s}_t$ becomes

$$\mathbf{s}_t = \left(t, \mathbf{f}^{(t-1)}(\mathbf{x})\right)$$

This representation means that the decisions are independent among the classifiers and that they are based upon the *current confidence*. Although this representation proves empirically the effectiveness of the approach for the binary class, where $\mathbf{f}^{(t)}(\mathbf{x}) = f^{(t)}(\mathbf{x})$ is a single continuous variable, it scales poorly in the multi-class case as the number of continuous variables in the state space grows linearly with the number of classes. Thus, we opt for a variation where we only retain the identity of the two winning classes at time t along with their class-wise score difference. Formally, defining the two winning classes at time j

$$\ell_1^{(j)} = \arg\max_{\ell} f_\ell^{(j-1)}(\mathbf{x})$$



(a) Toy dataset

(b) Pendigits dataset

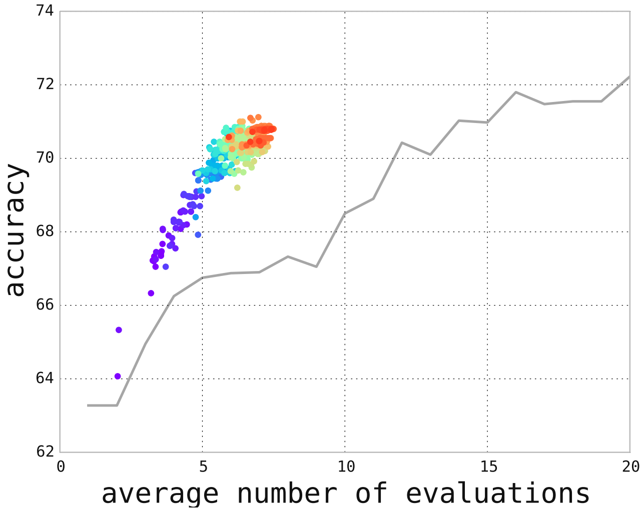
Figure 3.2 – Learning curve: the accuracy (acc) of the final classifier vs the episode number (t).

$$\ell_2^{(j)} = \arg\max_{\ell, \ell \neq \ell_1^{(j)}} f_\ell^{(j-1)}(\mathbf{x})$$

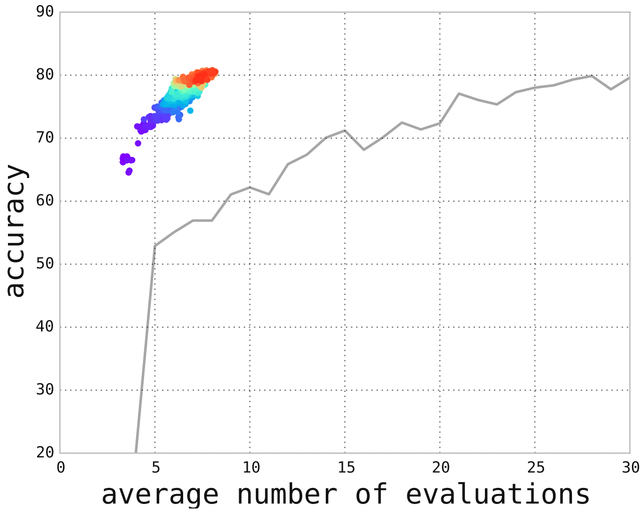
and their score difference

$$\Delta^{(j)}(\mathbf{x}) = f_{\ell_1^{(j)}}(\mathbf{x}) - f_{\ell_2^{(j)}}(\mathbf{x})$$

the state $\mathbf{s}_t$ is characterized as follows



(a) Toy dataset

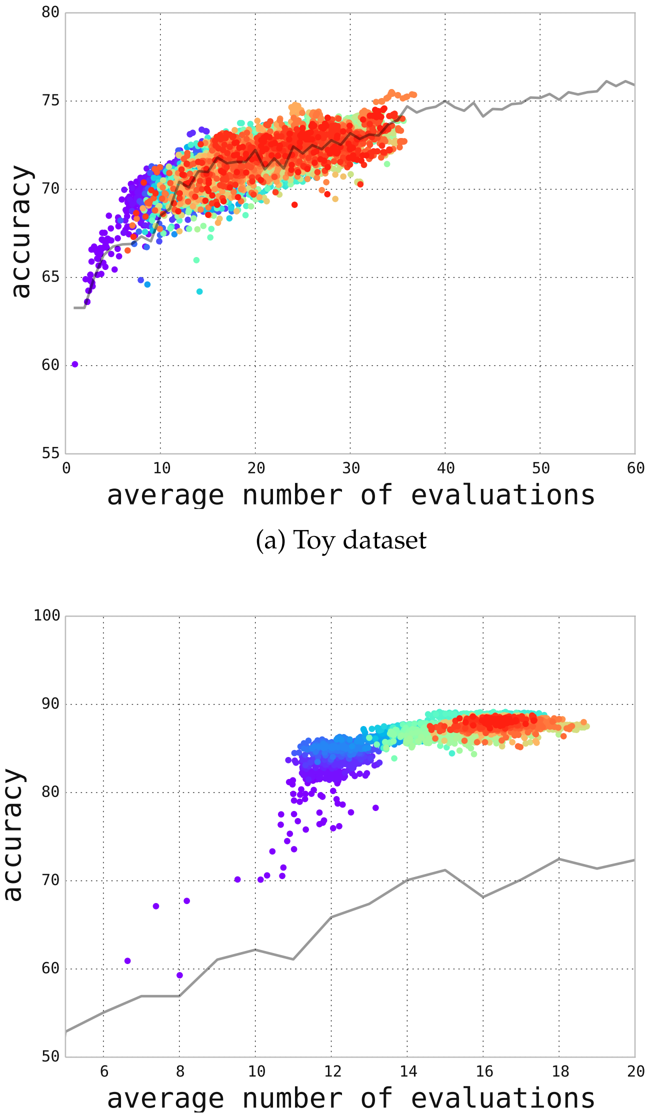


(b) Pendigits

Figure 3.3 – The Pareto front: The scatter points represent different policy evaluations and the continuous line represents the accuracy of AdaBoost.MH for the corresponding complexity. The colors simply allow us to see how the learning evolves: The first episodes correspond to the blue points and subsequent episodes turn progressively into the red.

$$\mathbf{s}_t = (\underbrace{j}_{base\ classifier\ index} , \underbrace{(\ell_1^{(j)}, \ell_2^{(j)})}_{winning\ labels} , \underbrace{\Delta^{(j)}(\mathbf{x})}_{score\ difference}) \tag{3.1.7}$$

Experiments



(a) Toy dataset

(b) Pendigits

Figure 3.4 – The Pareto front with the raw state space

With the exact same parameters as in the previous experiments, MDDAG with this new state space manages to combine more base classifiers, achieving a better accuracy (Figure 3.4).

The Toy dataset with DECISIONSTUMP

Interestingly, when ADABOOST.MH with the DECISIONSTUMP algorithm overfits the Toy dataset, MDDAG acts as a regularizer, outperforming the best validated ADABOOST.MH (Figure 3.5).

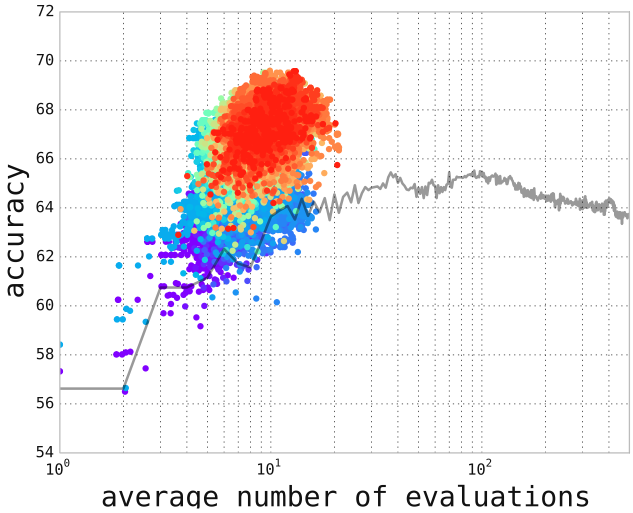


Figure 3.5 – The Pareto front of the Toy dataset with SINGLESTUMP

3.1.6 The learning algorithm

Different learning algorithms in reinforcement learning output agents with different complexities. Learning the policy of the agent with a classifier, known as Classification-based Policy Iteration, would yield a classifier which might be as costly as the final classifier, if not costlier. The same statement holds for Fitted-Q Iteration techniques, where the Q function is approximated through regression, such as with random regression trees (Ernst et al., 2005) or Neural Networks (Riedmiller, 2005). Even though these two family of techniques exhibit satisfying empirical results, they can only be applied when no strong constraints are put on the overall complexity of the classifier. As the complexity of the agent is extremely important, we restrict to that of a table look-up, thus, we only consider Q-learning methods.

We use the tabular version of Q-LEARNING for the following reasons:

- The horizon is finite, thus we can learn individual simple Q functions for each step (Busoniu et al., 2010),

- the complexity of a table look-up is low enough not to overcome the overall complexity of the final classifier.
- With our state representation (equation 3.1.7), we end up with only one continuous variables which can be discretized without blowing the state space.

In Q-LEARNING, the update rule after taking an action a_t in a state $\mathbf{s}_t$ is

$$Q(\mathbf{s}_t, a_t) \leftarrow Q(\mathbf{s}_t, a_t) + \alpha\Big[r_{t+1} + \max_a Q(s_{s+1}, a) - Q(\mathbf{s}_t, a_t)\Big]$$

In our case, the classifiers are processed sequentially during an episode, thus every time t corresponds to a unique classifier j that we have reached with the two top scored labels ℓ_1, ℓ_2 and their score difference Δ, so following the state representation described in equation 3.1.7 and assuming a binning function

$$bin(\Delta) \in \mathcal{N}$$

the update rule after taking an action a_t at a given classifier j is

$$\begin{aligned}&Q_{j,\ell_1^{(j)},\ell_2^{(j)}}(n^{(j)}, a_j) \leftarrow \\ &\quad Q_{j,\ell_1^{(j)},\ell_2^{(j)}}(n^{(j)}, a_j) + \alpha\Big[r_{j+1} + \max_a Q_{j+1,\ell_1^{(j+1)},\ell_2^{(j+1)}}(n^{(j+1)}, a) - Q_{j,\ell_1^{(j)},\ell_2^{(j)}}(n^{(j)}, a_j)\Big]\end{aligned}$$

where $n^{(j)} = bin(\Delta^{(j)})$, $n^{(j+1)} = bin(\Delta^{(j+1)})$, and r_{j+1} is the immediate reward after taking the action a_j.

Instead of discretizing the unique continuous variable, one can also parametrize it. A very common approach is to use a linear approximation with Gaussian radial basis functions (RBF), as defined in equation 3.1.8 and depicted in Figure 3.6.

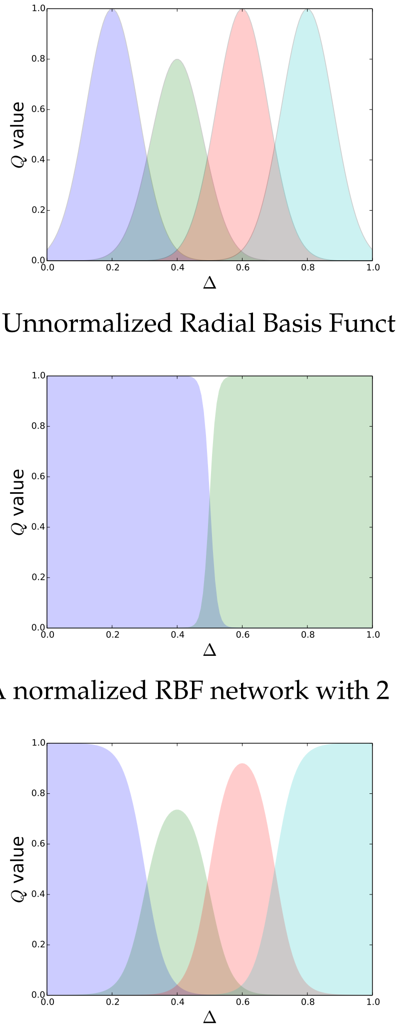
$$k_g(\Delta) = \exp\left(-\frac{||\Delta - c_g||^2}{2\sigma_g^2}\right) \tag{3.1.8}$$

Because this function approximation is local, it is preferable to normalize it as in equation 3.1.9 in order to have a better generalization to unknown regions in the score space (Morimoto and Doya, 1998).

$$b_g(\Delta) = \frac{k_g(\Delta)}{\sum_{l=1}^{G} a_l(\Delta)} \tag{3.1.9}$$

The Q value for a given classifier j and a given couple of winning labels, ℓ_1 and ℓ_2 would be the following function of the score difference Δ

$$Q_{j,\ell_1^{(j)},\ell_2^{(j)}}(\Delta, a_j) = \sum_{g=1}^{G} w_g b_g(\Delta) \tag{3.1.10}$$



(a) Unnormalized Radial Basis Functions

(b) A normalized RBF network with 2 units.

(c) When the two central units are added, the shapes of the previous units also change, due to the normalization.

Figure 3.6 – Examples of unnormalized and normalized Radial Basis Function

Still following the approach of Morimoto and Doya (1998), one can add the RBF units as we explore new score regions for the different classifiers, instead of setting a fixed number of units during the whole episode.

Starting the episode with no RBF, it consists of adding a new unit whenever the activation of the current units falls below a minimum threshold k_{min} and the error on the Q value (i.e, the difference between the current Q value and the immediate reward) is above a second threshold e_{max}. Formally these two criteria are

- $\max_g k_g(\Delta) < k_{min}$ and
- $|Q_{j,\ell_1^{(j)},\ell_2^{(j)}}(\Delta^{(j)}, a_j) - r_{j+1}| > e_{max}$

The new unit k_g is initialized with a center $c_g = \Delta^{(j)}$, a coefficient $w_g = r_{j+1}$ and a pre-defined standard deviation $\sigma_g = \sigma_{init}$.

The Q-LEARNING update is done then with a gradient descent (equation 3.1.11) that not only modifies the coefficients of the units but it can also modify their center and shape as well. Note however that the learning rates for the center and the variance, η_c and η_σ respectively, must be very small in practice to avoid too big steps and the resulting oscillation.

Let

$$err = Q_{j,\ell_1^{(j)},\ell_2^{(j)}}(\Delta^{(j)}, a_j) - r_{j+1}$$

for $g = 1 \dots G$, the update rules are

$$\Delta c_g = -\eta_c \times err \times w_g \times \frac{\partial b_g}{\partial c_g} \tag{3.1.11a}$$

$$= -\eta_c \times err \times w_g \times (b_g(\Delta^{(j)}) - 1) \times b_g(\Delta^{(j)}) \times \frac{\Delta^{(j)} - c_g}{\sigma_g^2} \tag{3.1.11b}$$

$$\Delta \sigma_g = -\eta_\sigma \times err \times w_g \times \frac{\partial b_g}{\partial \sigma_g} \tag{3.1.11c}$$

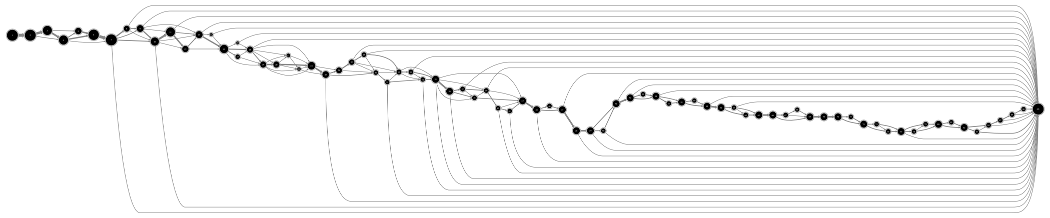
$$= -\eta_\sigma \times err \times w_g \times (b_g(\Delta^{(j)}) - 1) \times b_g(\Delta^{(j)}) \times \frac{(\Delta^{(j)} - c_g)^2}{\sigma_g^3} \tag{3.1.11d}$$

$$\Delta w_g = -\eta_w \times err \times w_g \times \frac{\partial b_g}{\partial w_g} \tag{3.1.11e}$$

$$= -\eta_w \times err \times w_g \times b_g(\Delta^{(j)}) \tag{3.1.11f}$$

3.1.7 Visualizing the final classifier

The final classifier is sequential and linear but if we draw the *paths* that link the evaluated classifiers for each observation individually, we end up with a Directed Acyclic Graph (a DAG, thus the name of the approach) that allow to visualize the instance-dependent aspect of the classifier in a particularly clear way.[4]

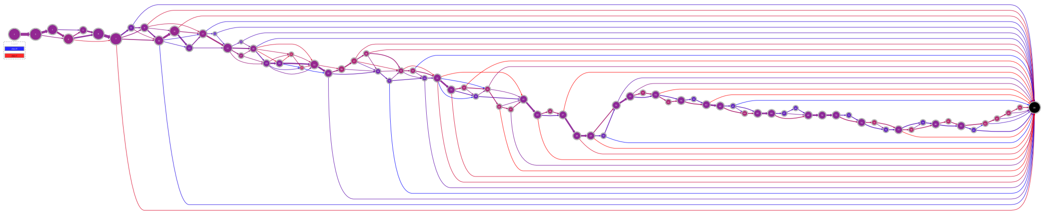


The figure above represents the different paths that are taken by the observations once the learning is done. The node radii and the edge widths are drawn proportionally to the number of observations that traverse them. Thus, we can clearly see how the final classifier is sparse: most of the observations take short paths *in fine*.

[4] The fine details of these graphs can only be seen when blowing up the pdf on screen.

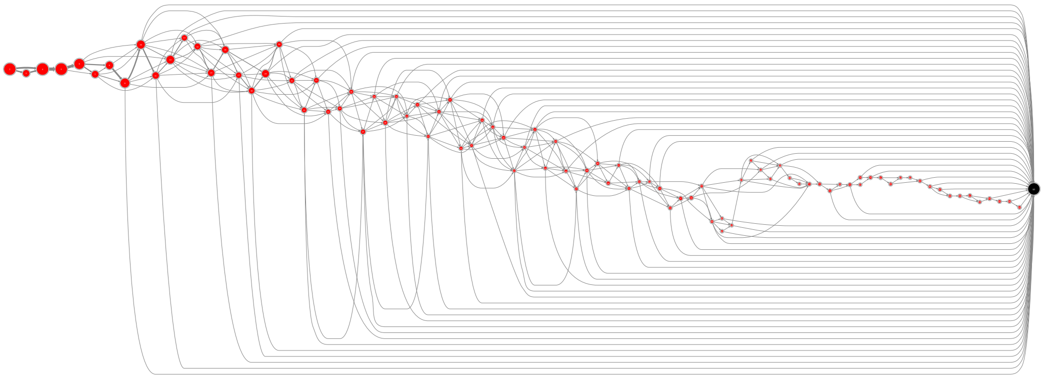
Even though the last figure is a only a conceptual depiction of the structure of MDDAG, it nevertheless gives a strong intuition about the type of classifiers that is learned. If MDDAG had to be compared with trees, the main difference would that each node of the graph in MDDAG can have multiple thresholds and not only one as in trees and that these thresholds depend on the cumulative score of the instance that traverses the graph, instead of its feature values.

In the next figure, we depict each pair of classes with two colors, blue and red. The node and edge are colored by mixing these two colors according to the class proportions of the observations that reached them.

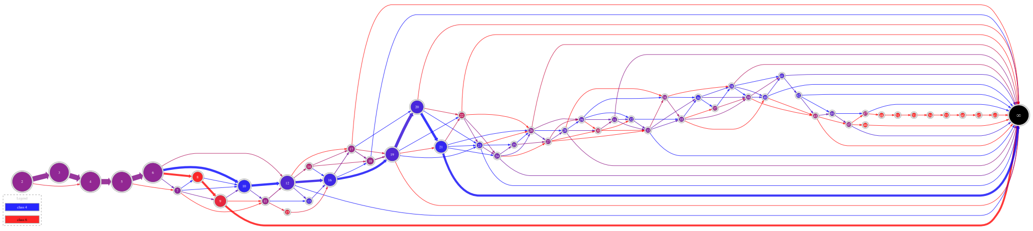


It is interesting to note how the observations of different classes tend to take different paths.

We plot the same DAG for the Pendigits dataset and we obtain the following representation



The last DAG only depicts the paths that are taken only by the 4's and 6's.



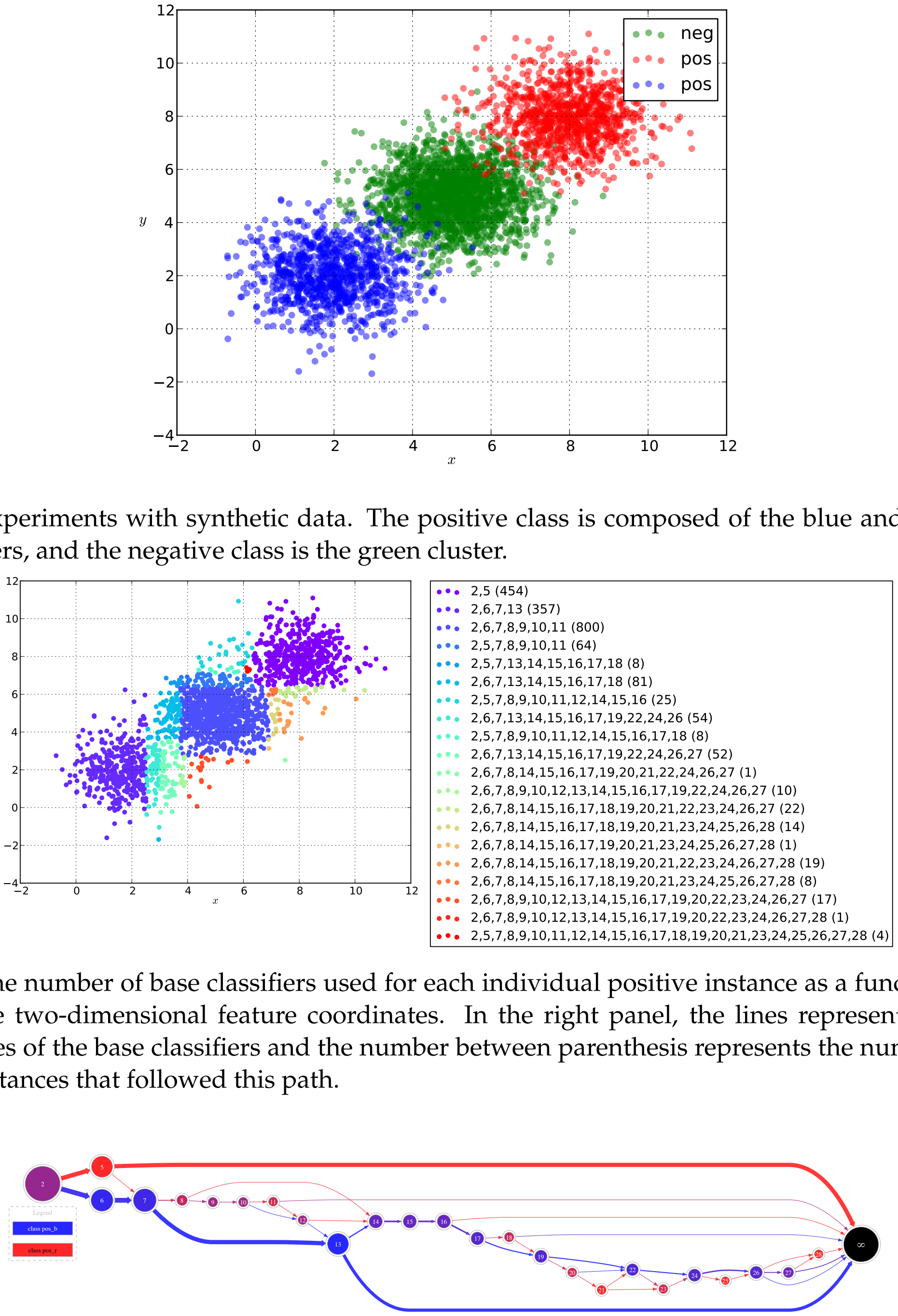
3.2 Unsupervised side-effects

We show two toy examples to illustrate how MDDAG can discover structure in the input data (Benbouzid et al., 2011). In Section 3.2.1 we first verify the sparsity and heterogeneity hypotheses on a synthetic example. In Section 3.2.2 we use an MNIST subproblem to show "path-wise" clustering.

3.2.1 Synthetic data

The goal of this experiment is 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 3.7a). 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.

We ran MDDAG with on 1000 decision stumps learned by ADABOOST.MH. In Figure 3.7b 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 fewer base classifiers it needs for classification. Figure 3.7c shows the actual DAG learned for the positive class and confirms our second hypothesis: base classifiers are used selectively, depending on whether the positive instance is in the blue or red cluster. As in the previous section, 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.



(a) Experiments with synthetic data. 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. In the right panel, the lines represent the indices of the base classifiers and the number between parenthesis represents the number of instances that followed this path.

(c) 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.

3.2.2 MNIST example

We ran MDDAG on 300 Haar stumps trained on 2s and 4s against 6s and 9s. Figure 3.8 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 3.3 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.

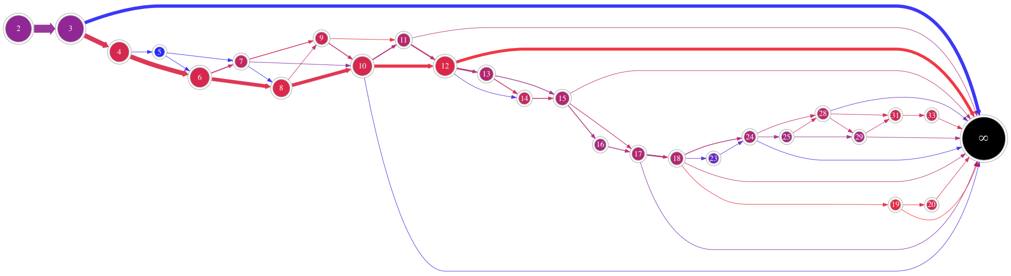


Figure 3.8 – The decision DAG for the 2-4 class. Colors represent sub-class probabilities (blue is 2 and red is 4) and the node sizes and arrow widths represent the number of instances in the states and following the actions, respectively.

3.3 Discussions

3.3.1 The action space

The action space influences both the shape of the final classifier and the complexity of the agent. If the actions were restricted to EVAL and QUIT (without the SKIP action), we would have categorized MDDAG as a cascade algorithm and an *embedded cascade* in particular (2.3.2). However, having the possibility to skip a classifier complexifies the space of the final instance-dependent sparse classifier in addition to showing better performance.

On the other end of the spectrum, allowing the agent to select any classifier at any time would indeed release us from the sorting assumption on the classifiers but

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 3.3 – The paths followed by more than 6 test instances, the corresponding average images, and the number of instances.

it would also necessitate a more complex agent to learn. Not only it is counter-intuitive in Reinforcement Learning to define an action space that grows with the input data (here the input classifiers) but the resulting agent would be too complex to satisfy our complexity criterion, especially if it has to be trained with non-tabular learning algorithms.

However, we can draw a continuum between no skipping at all (the cascade-like case) and the possibility to jump to any classifier by allowing a number $S < N$ of skips. For instance, if $S = 2$, we allow the agent to skip one or two classifiers in one single action, in addition to its the evaluation and quit actions.

3.3.2 The influence of the order of the classifiers

In order to assess the influence of the order of the classifiers in MDDAG, we apply random permutations to the sequence of classifiers before the learning of the MDP and compare the results with the default order output by AdaBoost.

Figure 3.9 provides the empirical hint that the order of the classifiers might influence the average number of evaluated classifiers but not necessarily the accuracy of the final classifier. In other words it just necessitates more classifiers to reach the same accuracy. This might indicate a problem of convergence of the MDP since the learner starts from a *worse* point in the parameter space of the policy.

3.3.3 The evaluation cost of the agent

The policy of the agent consists of a greedy selection of the action with the highest value for a given state. Thus, the complexity of the agent is that of a look-up table. With a naive implementation, one can use an ordered hash table storing the keys in a binary search trees. The states could then be fetched in logarithmic time, however, as the state space can be deconstructed into ordered subparts, it is possible to reduce the table look up to a constant time. In fact, the first subpart of the state space is the index j of the current classifier which is fully ordered. The second subpart consists of the all the possible pairs of labels $(\ell_1^{(j)}, \ell_2^{(j)})$ and it can also be ordered once for all. The third and final part corresponds to the bin of the current score $bin(\Delta^{(j)})$ which can also be accessed in constant time.

3.4 Experiments

We conducted a series of experiments in order to show the applicability of MDDAG. In the next plots, we compare the accuracy of MDDAG with that of ADABOOST.MH. The plots represent the Pareto front between the accuracy and the complexity of the overall classifier. The complexity is measured in terms of the average number of evaluated base classifiers. We mainly adopt this complexity measure because it is independent of the hardware on which the classifiers are run. In the case of ADABOOST.MH, it simply corresponds to the learning curve as the algorithm is iterative and adds one base classifier at a time. For MDDAG, the number of evaluated base classifiers is averaged across the test instances. Note that in all the plots (except for the Viola-Jones dataset), the average number of evaluations is in log scale. Also, the right-most point in ADABOOST.MH curves represents the best validated error of the algorithm, corresponding to the optimal number of base classifiers for the corresponding dataset (except for the Viola-Jones dataset).

We use 8 datasets from the UCI repository[5]. In all the experiments, the hyperparameters of ADABOOST.MH (the number of iterations and the number of nodes in the decision trees of the base learners) are validated using a 5-fold cross validation (except when the dataset already comes with a validation set). Then MDDAG is trained on the validated base classifiers with different values for β (controlling the

[5] http://archive.ics.uci.edu/ml/datasets.html

accuracy/complexity trade-off), namely −0.01, −0.001, and −0.0001. The parameters of the MDP were fixed throughout the experimentation as we noticed that the learning is quite stable with respect to them. The eligibility trace λ is set to 0.93, the learning rate α is set to 0.005. Note that in order to converge theoretically, α must decrease so that

$$\sum_{k=1}^{\infty} \alpha_k = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

however, we found in practice that fixing α provides better performance. All the plots represent the performance on a held-out test set. The solid line represents AdaBoost and the scatter plot represents MDDAG. The different points in the scatter plots result from evaluating the learned policy every 10000 episodes.

We kept the experimentation simple in order to show that MDDAG is almost a turn-key algorithm and that the additional hyperparameters of the MDP are not an obstacle to its effective application. Also, although it acts as a post-processing after the learning of AdaBoost, MDDAG does not need a separate dataset for its training. This is an appealing aspect when dealing with small datasets.

From the following experiments, it appears that MDDAG provides promising results. The resulting sparse classifier is consistently better than AdaBoost in small complexity regions. It is even competitive with the full AdaBoost classifier (represented by the end point in AdaBoost curve). Furthermore, it can sometimes regularize AdaBoost and improve its performance by preventing overfitting, as we already saw with the Toy dataset in Section 3.1.5 and here with the Arcene dataset.

3.4.1 The Adult dataset

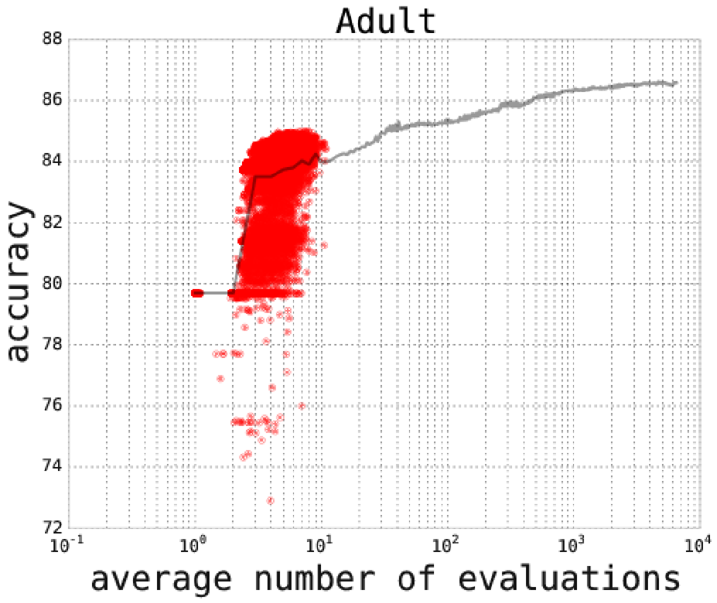
Source: http://archive.ics.uci.edu/ml/datasets/Adult

Description: Predict whether income exceeds $50 000 a year based on census data. Also known as "Census Income" dataset.

Data characteristics:

Number of training instances	30162
Number of test instances	15060
Number of features	105
Number of classes	2

Pareto front:



3.4.2 The Arcene dataset

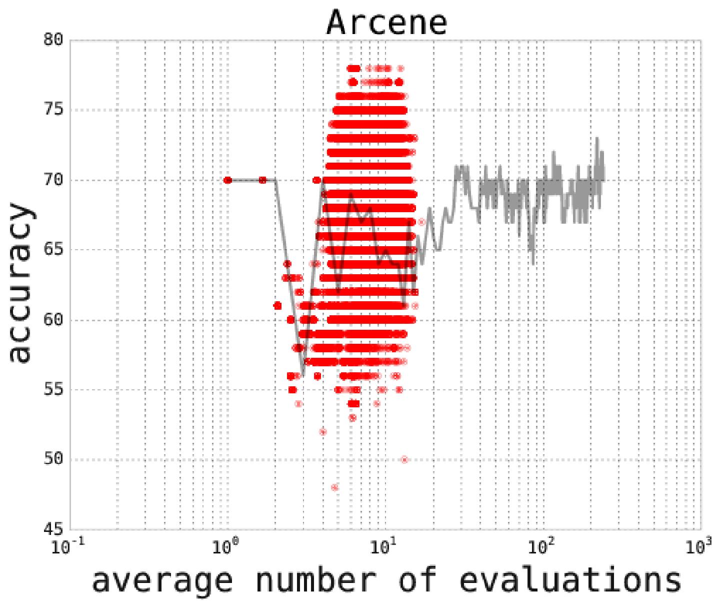
Source: http://archive.ics.uci.edu/ml/datasets/Arcene

Description: ARCENE's task is to distinguish cancer versus normal patterns from mass-spectrometric data. This is a two-class classification problem with continuous input variables. This dataset is one of 5 datasets of the NIPS 2003 feature selection challenge.

Data characteristics:

Number of instances	900
Number of features	10000
Number of classes	2

Pareto front:



Remarque: The Arcene dataset is mainly meant for feature selection challenges. We plot our results on this dataset in order to show the regularization capabilities of MDDAG and not to compete with specialized feature selections algorithms.

3.4.3 The Balance Scale dataset

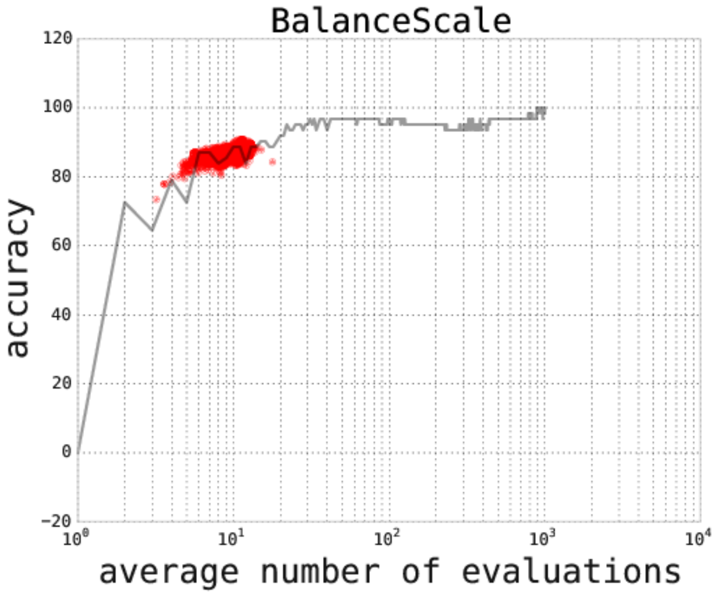
Source: http://archive.ics.uci.edu/ml/datasets/Balance+Scale

Description: This data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight, and the right distance. The correct way to find the class is the greater of (left-distance × left-weight) and (right-distance × right-weight). If they are equal, it is balanced.

Data characteristics:

Number of instances	625
Number of features	4
Number of classes	3

Pareto front:



3.4.4 The Gisette dataset

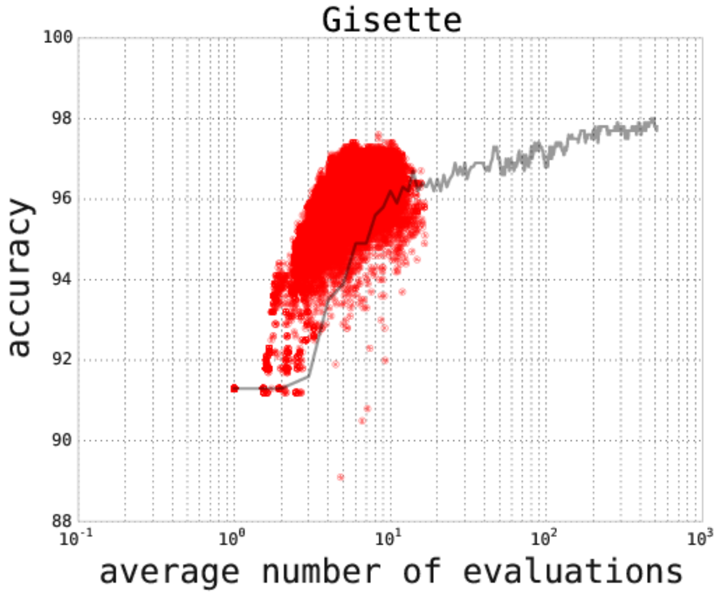
Source: http://archive.ics.uci.edu/ml/datasets/Gisette

Description: GISETTE is a handwritten digit recognition problem. The problem is to separate the highly confusible digits ′4′ and ′9′. This dataset is one of five datasets of the NIPS 2003 feature selection challenge.

Data characteristics:

Number of training instances	6000
Number of validation instances	1000
Number of test instances	6500
Number of features	5000
Number of classes	2

Pareto front:



3.4.5 The Landsat Satellite dataset

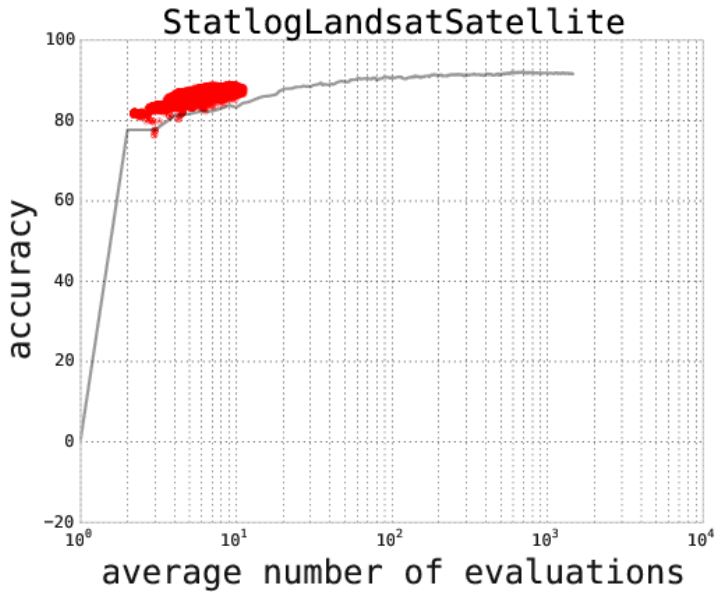
Source: http://archive.ics.uci.edu/ml/datasets/Statlog+%28Landsat+Satellite%29

Description: Multi-spectral values of pixels in 3x3 neighbourhoods in a satellite image, and the classification associated with the central pixel in each neighbourhood.

Data characteristics:

Number of training instances	4435
Number of test instances	2000
Number of features	36
Number of classes	7

Pareto front:



3.4.6 The Pendigits dataset

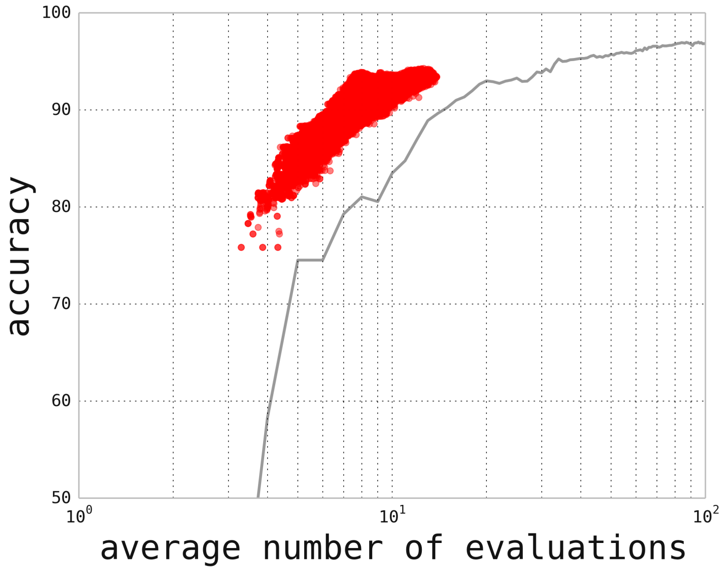
Source: http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits

Description: Digit database of 250 samples from 44 writers.

Data characteristics:

Number of training instances	7494
Number of test instances	3498
Number of features	16
Number of classes	10

Pareto front:



3.4.7 The Viola-Jones dataset

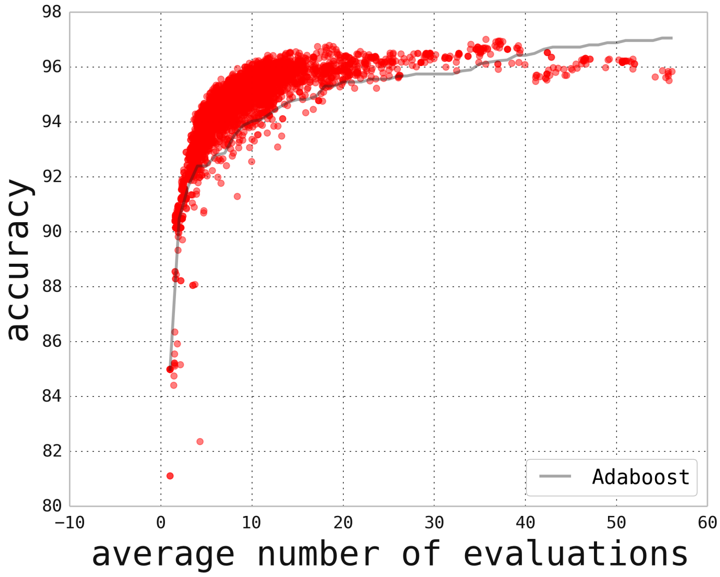
Source: Not published anymore.

Description: Face detection dataset of 24x24 images.

Data characteristics:

Number of training instances	6394
Number of test instances	6394
Number of features	576
Number of classes	2

Pareto front:



3.5 Prediction as a sequential process

In Chapter 2, some of the works we reviewed addressed the problem of classification with sequential models, mainly with cascades architectures. In this section, another type of sequential models is reviewed where the decisions are explicitly modeled, learned, and no longer limited to simple threshold adjustments. These types of approaches have known a rise of interest during the last two years, which tends to confirm the main thesis of this dissertation about the benefits of such modeling. However, it is striking to see, through the relatively few works that have been published, the variety of problems and applications that can be addressed with sequentiality and the direct benefits of translating prediction into decision making.

If our motivation was mainly driven by fast and budgeted classification, in the line of cascade classifiers, we also find that structured prediction and feature selection have been at the origin of a number of works that can be related to ours. Maes et al. (2009) apply reinforcement learning (RL) algorithms to solve sequence labeling problems and tree transformation in structured documents, which they introduce as a new type of structured prediction problem. They mainly define a state space that contains the current observation as well as the partial outputs, and an action space that translates the step-wise structured prediction outputs, hence they cast the original supervised learning problem into a reinforcement learning task. Also in sequence labeling and structured prediction, Daumé III et al. (2009) borrow RL concepts in order to build an algorithm, named SEARN, that takes a binary cost-sensitive learning algorithm (called policy in the original context) and embed it into a meta-learning procedure that iterates between a search phase and a learning phase. The search phase is related to the structured learning problem; it is used to create a cost sensitive dataset, on which a classifier is trained and used to extrapolate previously trained classifiers. Another work, by Weiss et al. (2013); Weiss and Taskar (2013) tackles the problem of combining different models for structured prediction under budget constraints. Directly inspired by cascade classifiers, they sequentially evaluate the models and choose whether to evaluate the next model in the list by explicitly learning the *value* of evaluating the next model given the previously evaluated ones. Also inspired by cascade classifiers but allowing a multi-class classification at each stage, (Trapeznikov and Saligrama, 2013) propose an approach that assumes an ordered sequence of stages wherein each stage allows to use one more feature, usually more informative, than its predecessor. They derive a stage-wise risk minimization resulting from a dynamic programming formulation of the problem and train a classifier for each stage, choosing whether to classify the current observation immediately or to delay it for further stages.

Unlike the previously mentioned works that use both supervised and reinforcement learning, Dulac-Arnold et al. (2011a, 2012) propose to fully formalize the classification problem as a reinforcement learning task in order to obtain a sequential instance-dependent sparsity[6] (Section 3.1.2). The authors learn a linear classifier that, given an observation, sequentially selects the features to evaluate. Thus, the classification consists in taking actions that correspond either to a new feature to evaluate or to one of the class labels, ending the classification with that label as an answer. The authors define a state space composed of the feature values of the current observation as well as a binary vector that indicates which features have been evaluated at the current time. The function is set so that every feature selected is penalized and overall, the learning of the agent corresponds to optimizing a classification loss function, penalized by an L^0 norm. The authors also propose an adaptation of this framework to tackle diverse applications such as structured learning and cost-sensitive classification. Also motivated by feature selection, (Rückstieß et al., 2011) propose a very similar approach for instance-dependent feature selection. They formalize the sequential problem with a Partially Observable MDP (POMDP) in which the belief over the states is deterministically maintained by an overall classifier. Gaudel and Sebag (2010) also use reinforcement learning in order to overcome the intractable problem of feature selection, the final classifier, however, is not instance-dependent.

Computer vision also has been tackled with explicit sequential models. Gao and Koller (2011a) assume a set of trained classifiers and learn the *value* of each classifier based on the information gain it provides penalized by its complexity cost. The authors also mention that a penalized classification loss can be used equivalently. During the classification, the classifiers are picked greedily, i.e, based on the one with the highest value and once a classifier is evaluated, its output is used to update a belief over the classes using locally weighted regression. In order to ac-

[6] Coined in the original paper as *datum-wise sparse classification model*

celerate the classification, the authors also propose to stop the classification early whenever the relative information between the current and the previous posterior estimation is small enough or if the margin information does not change beyond a certain threshold. They also prune the class space and the classifier space as the learning proceeds in order to economize the value computation for the least relevant classifiers and the least likely classes. Arguing that the greedy selection of the classifiers used in Gao and Koller (2011a) offers room for improvement, Karayev et al. (2012), formalize the multi-class object detection problem with RL techniques, in a setup that similar to that of Dulac-Arnold et al. (2012). The authors address the problem of large multi-class problems and aim at providing an anytime algorithm. To this end, they incorporate the time into the reward function.

As a hint towards the fact that sequential models are not necessarily restricted to RL methods, Larochelle and Hinton (2010) train a controller that collect glimpses in an image, indicating the next location to "look at", and classify the glimpses with a small sequential classifier. Both the controller and the classifier are learned with a Restricted Boltzmann Machine that include third-order connections between the visible units, the hidden units and the location of glimpses.

Sequential models were also successfully in other problems such as information extraction (Kanani and McCallum, 2012) and parser design Neu and Szepesvári (2009). Noticeably, these approaches offer a "natural" way to deal with these applications, as they are themselves intrinsically sequential.

Retrospectively, most of the works that explicitly model the classification (or the prediction in general) as a decision making process share a number of common points. In particular, one recurrent aspect is the learning of a *score* function that allows to estimate the best action to take during the prediction process. This score function is very similar, conceptually, to the value of a state or a state-action pair in reinforcement learning (when it is not explicitly defined as such). It is not surprising then that the gist of a broad family of RL methods is also used when supervised learning is cast as making sequential decisions.

The different approaches also diverge in many aspects. Besides the diverse aforementioned motivations, we can in fact distinguish some axes of investigations, along which one can make choices in order to devise a learning method for sequential predictors: some works rely more or less heavily on the use of reinforcement learning when formalizing the original problem, others only employ RL concepts but still use supervised learning during the decision making. As for choosing the learning algorithm, there are a plethora of solutions in the reinforcement learning literature that can be used that, in this particular context, might have different advantages and disadvantages. The choice usually depends on the application and the type of constraints that it involves. The learning also affects the policy that is output, some works adopt myopic policies for choosing the actions and others take the overall horizon into account. Finally, the implementation of a sequential learning approach strongly depends on the choice of the state space, on which the decisions are based, as well as the possible actions that the classifier (or the agent in general) can take in a given state. The different approaches are described in the next sections and summarized in Table 3.4.

3.5.1 The paradigm continuum

In our work (Benbouzid et al., 2011, 2012b), the algorithm takes an input sequence of classifiers and learns an overall sparse classifier that combines them. Similarly, some works first employ supervised learning in order to extract models from the data and then apply reinforcement learning as a post-processing in order to achieve their targeted goal (Rückstieß et al., 2011; Karayev et al., 2012).

On the other hand, it has been shown that the problem of classification can be fully formalized within a reinforcement learning framework, in the sense that the learning agent can have access to the "raw" feature space of the observations without resorting to any type of feature mapping such as the use of pre-trained classifiers. The agent is thus responsible for both querying the data features and assigning labels to the instances. This approach, albeit more elegant and more flexible regarding the different kinds of applications it can tackle, yields a more complex problem to solve and an inference complexity that grows with the dimensionality of the data and the number of classes (Dulac-Arnold et al., 2011a, 2012).

Reinforcement learning methods have sometimes inspired supervised learning approaches that tackle the decision making through multi-class classification, thus only *simulating* the sequential concepts of RL. Trapeznikov and Saligrama (2013) explicitly train classifiers to learn the rejection binary decision of each stage. The objective function is defined recursively across the stages and takes into account the cost of acquiring a new feature when delaying the classification to a further stage. For combining models, Weiss et al. (2013); Weiss and Taskar (2013) learn a value function that explicitly trades off the accuracy gain in using a new model with the cost that it incurs.

3.5.2 The learning method

When learning the decision making is not tackled as supervised learning (Trapeznikov and Saligrama, 2013), one can a use a plethora of reinforcement learning methods for either estimating the state or action-state values, or to directly learn the decision policy. Rollout-based approaches for example have been proved to be efficient both in RL problems in general and in sequential prediction in particular. Rollouts empirically estimate a given policy or the value of a state by running the policy from that state multiple times and then average the cumulated rewards received along the trajectories. Rückstieß et al. (2011) use Fitted-Q-Iteration (Ernst et al., 2005) that use rollouts in order to approximate the Q function with regression. (Dulac-Arnold et al., 2011a, 2012) use Rollouts Classification

Policy Iteration (RCPI) (Lagoudakis and Parr, 2003) to iteratively improve the policy without necessarily estimating the value of each state. The main drawback of rollout-based approaches lies mainly in the complexity of the learning. Other approaches use imitation learning (He et al., 2012) or inverse reinforcement learning (Neu and Szepesvári, 2009).

3.5.3 Myopic vs non myopic

The type of decisions made at each step is very important, some works rely on myopic greedy decisions (Ji and Carin, 2007; Gao and Koller, 2011a) which already show the benefits of using sequential models. In these approaches, a classifier or feature, is selected based on its individual gain with respect to the objective function, however, it has been shown that incorporating the idea of "looking ahead" in the decision making, taking into account the distribution of future actions, improves overall the performance (Karayev et al., 2012).

3.5.4 The features used for the actions

The information on which the sequential decisions are based, let us coarsely call it the state space, can also vary. In our case, we take into account the output of the base classifiers and in order to further improve the performance, we summarize the different outputs by their overall sum, which corresponds to the classification score. Gao and Koller (2011a) and Rückstieß et al. (2011) also assume pretrained classifiers and only consider their output. On the other hand, (Dulac-Arnold et al., 2012) include the observation in the state space that they augment with boolean variables in order to keep track of the evaluated features. The best choice depends on the application at hands and its constraints. In our case, we were motivated by fast classification, hence, mapping the instances with classifiers allows to gain some information and to ease the sequential learning problem. Additionally, different types of heterogeneous classifiers can be used, such as some of the cost-sensitive classifiers mentioned in Chapter 2.

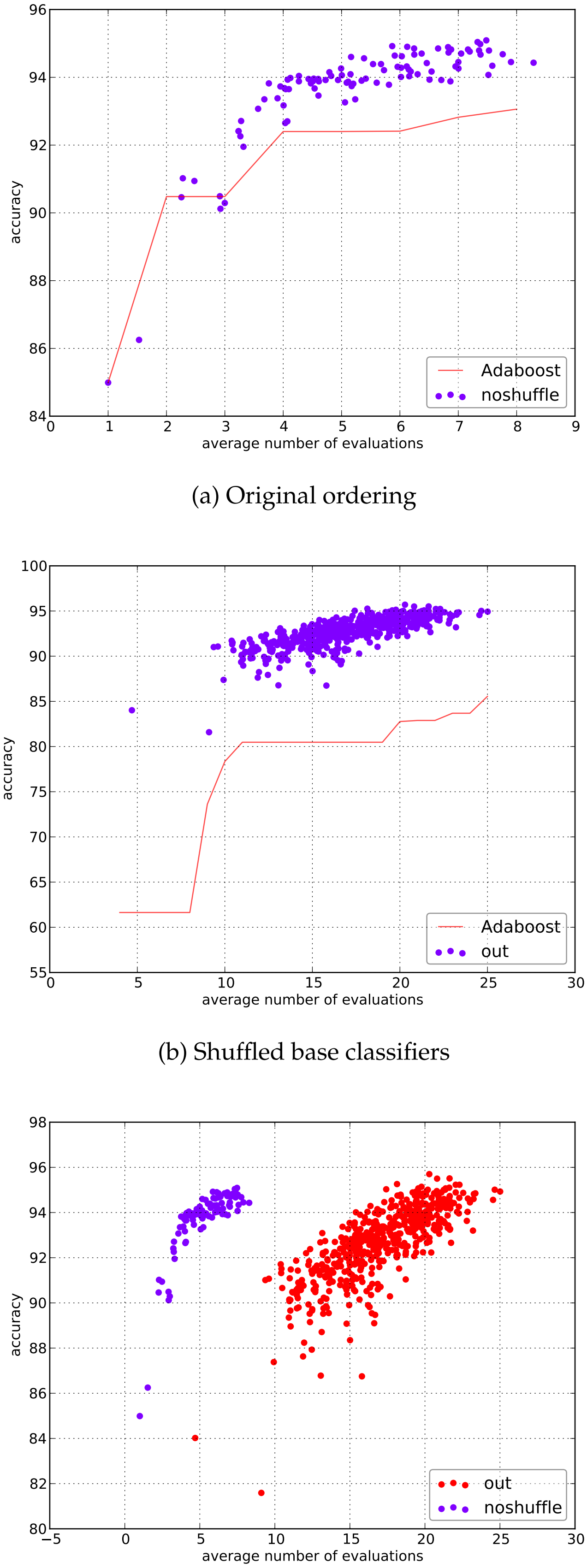
3.5.5 The type of actions

Finally, the different approaches also differ in the choice of actions the agent is given. In this regard, we can distinguish three types of solutions proposed. Some works stick to the cascade architecture and only allow the instance to travel from a stage/classifier to its successor (Trapeznikov and Saligrama, 2013). Other works, mainly in feature selection, allow to select at any time any feature or classifier that has not been evaluated yet (Rückstieß et al., 2011; Dulac-Arnold et al., 2011b). This approach has the benefit of being less restricted but, as noticed by Dulac-Arnold et al. (2011b), the classifier might overfit trying to select the strictly appropriate subset of features for each observation. In order to avoid the overfitting problem, the authors constrain the order in which the features are selected to be the same. This order is determined during the learning. Our approach that impose an order on the base classifiers but allows to skip them can be considered as an intermediate solution, in particular for fast classification problems where the complexity of the decision maker is crucial.

3.6 Conclusion and perspective

In this chapter, we introduced MDDAG as a novel way to tackle real-time and budgeted classification. By formalizing the problem with Reinforcement Learning, we could achieve satisfying classification performance while considerably reducing the amount of computation during the test-time. We also reviewed other works that similarly learn sequential predictors for different tasks and saw how both the initial motivation and the domain-specific constraints impact on the formulation and the choices of the learning algorithms employed.

In the next chapter, we show that our approach can be easily adapted to very complex situations where the classification is subject to constraints that can not be satisfied with classical machine learning methods. We apply MDDAG to our initial motivating problem, the particle physics experiments, wherein the features can have different acquisition costs and, furthermore, exhibit different types of cost calculations depending on the history of the evaluated features within the same instance but also across groups of instances that share some feature computations.



(a) Original ordering

(b) Shuffled base classifiers

(c) Comparison of the original ordering and the shuffled case

Figure 3.9 – The influence of the order of the classifiers.

Variation	Representative papers
Motivation	Fast classification (Póczos et al., 2009; Benbouzid et al., 2012b)
	Feature selection (Gaudel and Sebag, 2010; Rückstieß et al., 2011; Dulac-Arnold et al., 2011b)
	Structured learning Neu and Szepesvári (2009); Daumé III et al. (2009); Maes et al. (2009); Weiss et al. (2013)
	Cost-sensitive learning (Greiner, 2002; Ji and Carin, 2007)
	Multi-class budgeted classification (Karayev et al., 2012; Trapeznikov and Saligrama, 2013)
Learning paradigm	Supervised learning (Daumé III et al., 2009; Larochelle and Hinton, 2010; Weiss et al., 2013; Weiss and Taskar, 2013; Trapeznikov and Saligrama, 2013)
	Reinforcement learning (Ji and Carin, 2007; Dulac-Arnold et al., 2011a, 2012)
	Mixed formalization (Benbouzid et al., 2011, 2012b; Rückstieß et al., 2011; Karayev et al., 2012).
Decision type	Myopic (Ji and Carin, 2007; Gao and Koller, 2011a)
	Non-myopic (Karayev et al., 2012; Trapeznikov and Saligrama, 2013)
RL Learning method	Q-learning (Benbouzid et al., 2011, 2012b)
	Fitted-Q-Iteration Rückstieß et al. (2011)
	RCPI (Dulac-Arnold et al., 2011a, 2012)
	Imitation Learning (He et al., 2012)
	Inverse RL Neu and Szepesvári (2009)
State space	Full observation information (Dulac-Arnold et al., 2011a, 2012)
	Outcome of classifiers Gao and Koller (2011a); Rückstieß et al. (2011); Benbouzid et al. (2012b)
Action space	Features and class labels (Rückstieß et al., 2011; Dulac-Arnold et al., 2011b)
	Ordered (Trapeznikov and Saligrama, 2013)
	Ordered with skipping (Benbouzid et al., 2011, 2012b)

Table 3.4 – The different type of sequential approaches.

CHAPTER 4

Trigger design in the LHCb experiment

Contents

In this chapter, we apply MDDAG to data that belong to the LHCb experiment. In this particle physics experiment, an on-line fast classifier, called trigger, classifies the events that the particle collisions generate in order to discriminate what is called the background, i.e, uninteresting events, from events that are likely to contain the targeted particle decay that is to be studied. From a machine learning learning point of view, the problem has some very interesting particularities. Firstly, the studied phenomenon is a rare event and most of the data that arrive to the trigger actually belongs to the background category, which allows us to draw parallels with the object detection in images problems. Secondly, the different attributes that are used during the classification necessitate an acquisition cost and, more particularly, this cost is computed in a non trivial way, depending not only on the feature requested but also on other features that are related from a physics point of view. Hence, we adapt MDDAG to manage this cost calculation through simple modification of the state space, showing at the same time the flexibility of MDDAG when it comes to real applications.

4.1 Background

Modern particle physics is a branch of physics that studies the elementary components of the matter. If the elementary particles that form the matter, such as the quarks, can not be seen through microscopes because of their small size – even smaller than the smallest wavelength of the visible light, they can however be investigated by accelerating other particles, providing them with enough speed and energy in order to collide and collapse into elementary particles, the trace of which can be then measured. This is, quickly enunciated, the principle behind particle colliders.

The Large Hadron Collider (LHC) is a 27 km wide ring-shaped particle collider located in CERN on the France-Swiss border. Hadrons are a family of composite particles made of two or three quarks held together by the strong force, such as protons and neutrons. The LHC accelerates two beams of protons with opposite directions so as to create collisions between the particles. The collisions happen at a rate of 600 million per second and generate 1 Petabyte of data per second.

4.1.1 The LHCb experiment

The LHC contains four experiments spread around the ring, at points designed to make the collisions occur the most frequently. The LHCb is one of them. It has been designed to study the physics of two quarks *flavors*, namely, the *beauty*, or **b** quark and the *charm* quark, also known as **c** quark. These two elementary particles have the advantage of decaying relatively slowly, easing their identification, and they are expected to be particularly affected by what is known as the CP violation: each of the **b** and **c** quarks behave differently from their corresponding antiparticle, $\bar{\mathbf{b}}$ and $\bar{\mathbf{c}}$ respectively, when decaying from a parent hadron, violating the CP-symmetry principle.

The particles that produce a **b** quark when decaying are called B mesons. Similarly, D mesons produce **c** quarks. Since these mesons are relatively slow to decay, the location whereby the decay occurs can be recovered with a high precision, this location is called the *primary vertex*. Determining the primary vertex happens in the *vertex detector* and is essential for discriminating events that generated a **b** or **c** quark from the ones that did not.

4.1.2 The LHCb trigger

The data arrives at the LHCb detector at a rate of 16MHz. Since it is impossible to store all the events on disk, an on-line classification must filter the events and let the most likely to be interesting be stored. This selection is done in the LHCb trigger in two steps, similarly to a two-stage cascade. The first stage, implemented electronically, reduces the event rate to 1MHz. This operation takes 4μs. The second stage, called the High Level Trigger, reduces the event rate to 2kHz so that 200 megabyte can be stored every second. The typical time frame at which the second stage must discard an event is about 30ms (Aaij et al., 2013). Our aim in fact is to design the High Level Trigger by applying MDDAG and to be able to classify an event in the order of 30ms.

4.2 The LHCb data

4.2.1 The D decay

The dataset we use in this section focuses on the D meson decay. In particular, a first decay occurs in which a D^* produces a pair of D^0s and a charged pion. The D^0 meson, with a relatively long lifetime, flies a certain distance away from the primary vertex and then decays into n other charged particles (kaons or pions) producing n tracks. The decay, known as an *n-body* decay, produces either 2, 3 or 4 particles in the case of the D^0 meson.

4.2.2 Data description

The data is generated by simulation and have been privately provided by a member of the LHCb collaboration. It consists of a 4 class classification problem where one of the classes, named (*bkgd*), constitutes the background to be discarded and the other three classes constitute three different types of signal that are to be detected. The signal classes, named *2body*, *3body* and *4body*, correspond to different decay channels for the D^0 particle, corresponding respectively to 2, 3, and 4 body decay types. The features are described in Table 4.1.

The number of instances in the dataset reflects, to some extent, the class proportions in nature so the background class is several orders of magnitude more prevalent than the signal classes. The class disproportionality is particularly important when computing the overall classification cost as we describe it further. Furthermore, instances that belong to the same class and which correspond to the same particle decay are grouped in *bags*, also named *events*. The reason behind is that when a particle decays and produces multiple tracks, the features are computed on combinations of two of these tracks, hence, each combination is represented as an instance in the dataset. In particle physics, the individual instances are called *candidates*. Henceforth, when taking the bags into account, $\mathbf{x}_{ij}$ will denote the j^{th} instance (or candidate) of the i^{th} bag (or event). Likewise, $\mathbf{y}_{ij} = \mathbf{y}_i$ will denote the corresponding label vector, keeping the same semantic as described in Section 1.3.

Grouping the instances into bags is very similar to what is known as *Multiple-Instance* Learning (MIL), introduced by Dietterich et al. (1997). In a MIL setup, the examples also come grouped in bags but, unlike our setup, one only has access to the bag labels and ignores the individual labels within each bag. The first example of MIL problems described in Dietterich et al. (1997) was that of predicting drug molecules activity. The goal was to predict whether a drug molecule would bind to a target protein, however, the molecules might come in different shapes and the only information available is that inside a positive bag, there was at least one positive instance corresponding to a shape that binds well. The negative bags were consequently defined as containing strictly no positive instance. MIL approaches were later successfully applied to other problems as well such as object detection in images wherein positive bags are defined as containing at least one target image (Viola et al., 2006).

The similarity thus between the MIL setup and ours lies in that a bag (or an event) is classified as positive (i.e, one of the signals) if at least one of its instances is classified positive. Otherwise, the bag is classified as negative. However, the most important consequence in grouping the instances, for our problem in particular, lies in the way classification cost is computed.

4.2.3 The feature costs

As described in Table 4.1, the data features consist of measurements and computations related to different particles. Because of the architecture of the LHCb detector, some features are computed immediately for all the instances within a bag and others are acquired on demand. All the feature costs are expressed in terms of computational time, also, we distinguish three non mutually exclusive types of cost, namely

- immediate cost,
- value-dependent cost,
- and bag-dependent cost.

Features that incur an immediate cost are computed in every case, before the classification starts. In the dataset, they correspond to what is computed at the level of the vertex detector:

- D0_VTX_FD
- PiS_IP
- D0C_1_IP
- D0C_2_IP

Particle	Feature name	Description
D^0	D0M	The invariant mass of the particle.
	D0Tau	The particle life time.
	D0_VTX_DF	The distance from the primary vertex.
D^*	DstM	The invariant mass of the particle.
Slow Pion	PiS_IP	Impact parameter wrt. the primary vertex.
	PiS_PT	Transverse momentum.
	PiS_IPC	The χ^2 fit of the impact parameter.
	PiS_TFC	The χ^2 fit of the Kalman track fit.
D^0 first child	D0C_1_IP	Impact parameter wrt. the primary vertex.
	D0C_1_PT	Transverse momentum.
	D0C_1_IPC	The χ^2 fit of the impact parameter.
	D0C_1_TFC	The χ^2 fit of the Kalman track fit.
D^0 second child	D0C_2_IP	Impact parameter wrt. the primary vertex.
	D0C_2_PT	Transverse momentum.
	D0C_2_IPC	The χ^2 fit of the impact parameter.
	D0C_2_TFC	The χ^2 fit of the Kalman track fit.

Table 4.1 – Description of the dataset features.

and they cost **4ms** to be computed altogether and for all the instances within a bag.

For every feature that corresponds to a transverse momentum (D0C_1_PT, D0C_2_PT, PiS_PT), the acquisition cost depends on the actual value of the feature, they are the *value-dependent cost* category. In the case of this dataset, whenever the momentum is higher than 1200 MeV, these features cost **0.5ms**, otherwise, the computation is more expensive and it costs **1.5ms**.

The last category concerns features that are computed only once for a subset of the instances within a bag. This comes from the fact that the candidates belong to the same event and some of them have tracks in common, thus, they also share the computation of the corresponding features. It is important to note that, once a bag-dependent feature is computed, it is cached so that all the subsequent requests of that feature incur no cost for the related candidates. In this dataset, this category concerns all the features that do not incur an immediate cost.

Finally, there is another level of dependencies among the features. Some features necessitate other features to be computed beforehand, which creates a network of dependencies among the features, depicted in Figure 4.1.

Note that the real classification cost involves the feature acquisition but also the base classifier cost and the cost of final linear combination of the base classifiers. The last two costs are in the order of nanoseconds, since the base classifiers used consist of decision stumps or two-nodes decision trees, therefore, we only consider the feature acquisition cost in our experiments.

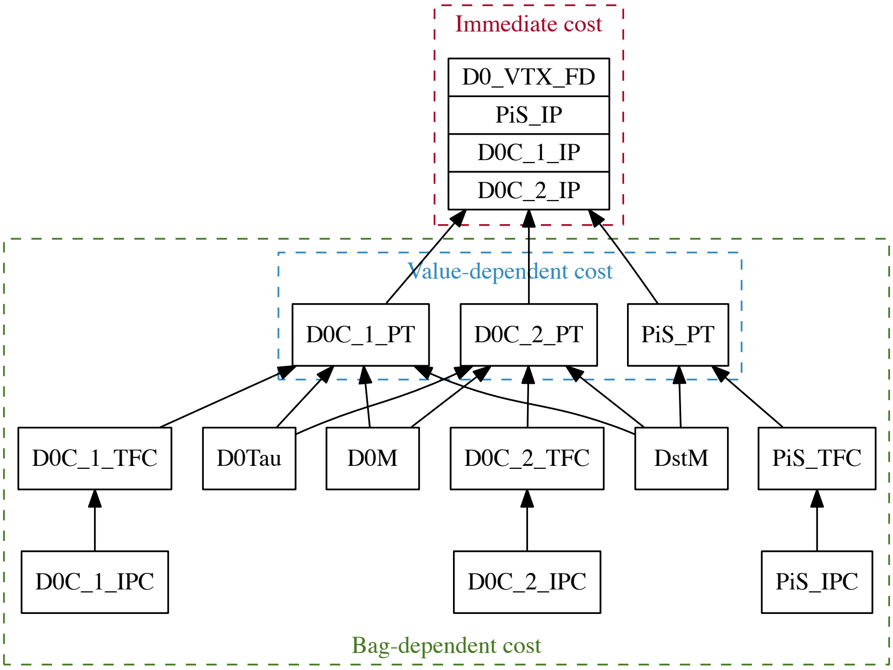


Figure 4.1 – The dependency graph of the feature cost calculation.

4.2.4 The data filtering trick

In order to let MDDAG "focus" on the most difficult instances, we first filter the data set and discard a subset that is classified as background with a high probability. This filtering only concerns the training phase and not the test phase, in which we use the entire dataset. To this end, we train AdaBoost.MH using 100 decision trees, each of which contains 3 decision nodes, however, we limit the features to the four that have an immediate cost and that are computed anyway. We obtain a vector-valued discriminant function, $\mathbf{f} \in \mathbb{R}^4$ that provides a classification score for each of the 4 classes.

	Number of instances		Number of bags	
Class	**Entire**	**Post-filtering**	**Entire**	**Post-filtering**
2body	3347	3347	3347	3347
3body	9799	9792	3935	3935
4body	18706	18685	3889	3889
bkgd	8077679	639309	94555	56784
Total	8109531	671133	105726	67955

Table 4.2 – The properties of the illustrative datasets.

$$\mathbf{f}(\mathbf{x}) = \sum_{j=1}^{100} \mathbf{h}_j(\mathbf{x})$$

and its prediction for the class index of $\mathbf{x}$ is

$$\widehat{\ell} = \arg\max_{\ell} f_\ell(\mathbf{x}).$$

We test the data with this classifier and retain only what is not classified as background. In order not to discard signal instances, we bias the score of the background class negatively. Table 4.2 describes the initial as well as the filtered data.

Conceptually, it is similar to manually designing a two-stage cascade but, in fact, we only use the second stage, i.e, MDDAG, in the final classification and totally throw the first stage away. As shown further, this speeds up the learning of MDDAG without impacting the overall performance. This "trick" is in fact only a speed-up that corresponds to sampling of the observations during the learning of the MDP, giving more chances to the difficult instances to be selected.

4.2.5 MDDAG training

We adapt MDDAG in order to accommodate the feature costs of this dataset. We only make two modifications to the setup presented in Chapter 3. First, after each Eval action, the agent receives a reward that directly translates the computation time of features used in the corresponding base classifier. This reward takes into account the whole dependency graph illustrated in Figure 4.1 so that when a feature is acquired, all of its dependencies are also considered as computed.

The other modification comes from the fact that, during an episode, if the agent selects an already computed feature, it must not incur a new cost. Thus, in order to keep track of the computed features, we augment the state space with a boolean variable c, that informs the agent whether the feature used in the current base classifier has already been acquired. If the base classifier employs more than one feature (as for trees), we simply consider that the boolean value changes whenever *all* the corresponding features have been acquired. Another solution would have been to augment the state space with as much boolean variables as there are features, but it would cause a combinatorial explosion of the state space. Our setup only doubles the number of states in the state space and can be seen as a compromise that keeps our initial tractability. By extending the state space defined in Equation 3.1.7, the new state space is then

$$\mathbf{s}_t = \Big(\underbrace{j}_{base\ classifier\ index}, \underbrace{c}_{Feature\ evaluation\ boolean}, \underbrace{(\ell_1^{(j)}, \ell_2^{(j)})}_{winning\ labels}, \underbrace{\Delta^{(j)}(\mathbf{x})}_{score\ difference} \Big)$$

4.2.6 Results

First, we compare the new setup with the classical one that does not take the feature costs into account. We use the same set of hyperparameters and compare the evolution of the cost during the training of the MDP. The plots in Figure 4.2 show that MDDAG uses more costly base classifiers during the exploration (the oscillation effects comes from the discrete nature of the cost additions). Moreover, the final cost is lower in the cost-sensitive learning.

Validation of the hyperparameters

All the learning was done by splitting the data described in Table 4.2 into training, validation and test sets. The following hyperparameters were then validated,

- The base learning : Decision Stump, Decision Trees.
- The number of nodes in the case of Decision Trees.
- The complexity-accuracy trade-off parameter β.

For the MDP hyperparameters, i.e, the learning rate α, the exploration rate ε, the eligibility trace λ, they are first validated and then fixed once for all the other hyperparameters.

Figure 4.3 shows the Pareto front between the accuracy of the classifier and its average evaluation cost. Each red dot in the scatter plot represents a policy evaluated by MDDAG. The red solid line represents a convex hull over the dots.

We take one of the dots in Figure 4.3, and show its detailed performance. Tables 4.3 and 4.4 show the confusion matrices on the test set, the results are shown on

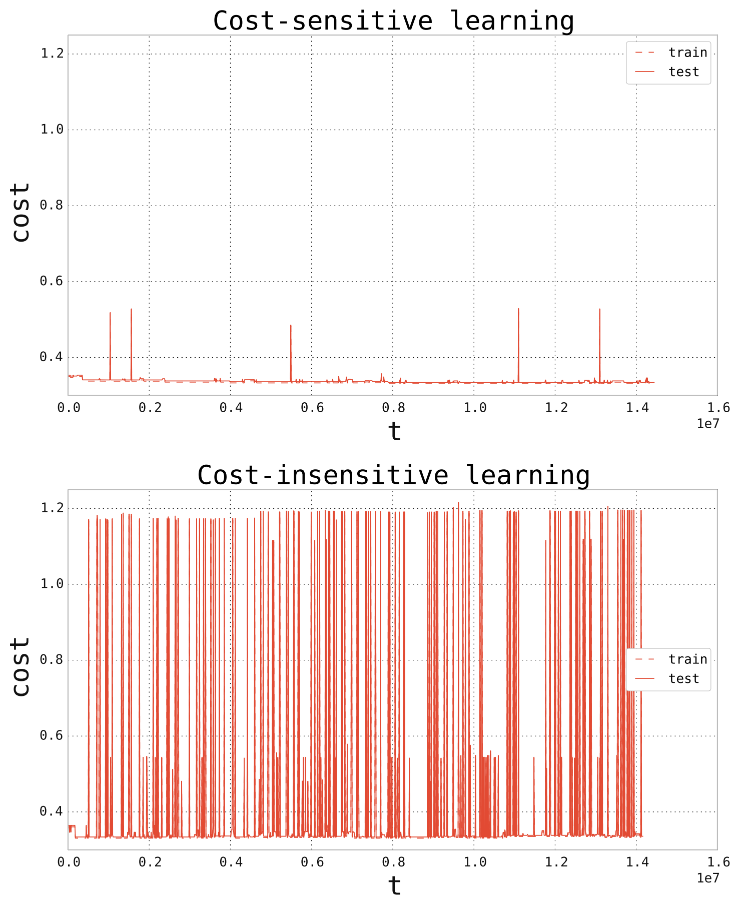


Figure 4.2 – Comparing the new cost-sensitive setup (top) with the classical cost-insensitive setup (bottom). On the y-axis, the overall classifier cost. On the x-axis, the episode number (t). In the classical setting, the learning does not take into account the features costs, which results in high cost "spikes" during the exploration. It also shows that the problem is non trivial: when the learning only minimizes the classification error, the resulting classifier ends up using costly features. The cost-sensitive setup, on the other hands, keeps the overall cost low and orient the exploration towards the usage of cheap features.

the instance and bag level respectively. The prediction of a bag corresponds to the prediction of its instance with the highest score, i.e,

$$\hat{\ell}(\mathbf{x}_i) = \arg\max_{\ell} \left(\mathbf{max}_j\, \mathbf{f}(\mathbf{x}_{ij}) \right)$$

with

$$\mathbf{max}_j\, \mathbf{f}(\mathbf{x}_{ij}) = \left(\max_j f_1(\mathbf{x}_{ij}), \ldots, \max_j f_K(\mathbf{x}_{ij}) \right) \in \mathbb{R}^K.$$

The bag score is

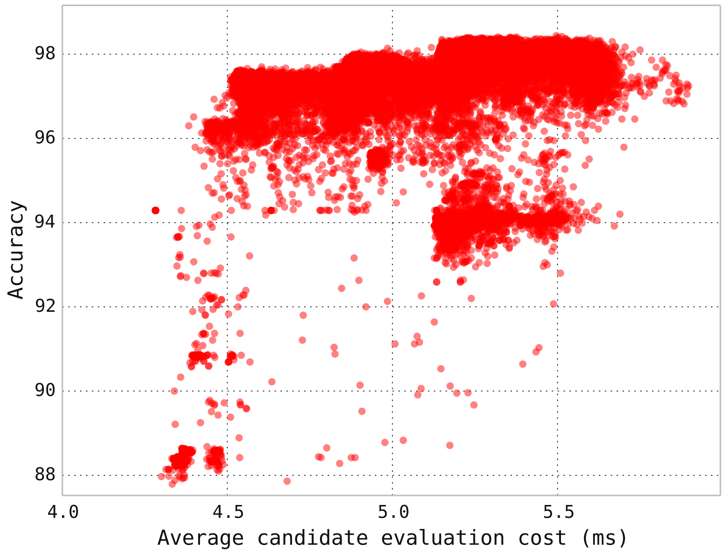


Figure 4.3 – The Pareto front of the trigger. Red dots are individual policies, most of them correspond to exploration policies. The accuracy is expressed in percent and the evaluation cost in millisecond.

$$\mathbf{f}(\mathbf{x}_i) = \max_j \mathbf{f}(\mathbf{x}_{ij})$$

Noticeably, only a small fraction of the background is classified as signal, which is one of the most appealing aspect when designing a trigger. However, the 3body class bags tend to be classified as 4body. In our present case, it is not a major issue since the function of a trigger is to discard most of the background. Table 4.5 shows the average classification costs for the instances and the bags each class.

Plotting the ROC curve

Since the problem is a 4-class classification, we first convert it into a virtual binary classification problem in order to obtain a unique score for every bag and, hence, draw the ROC curve. We first merge the three signal classes, 2body, 3body, and 4body under one unique *signal* label, st. for a bag $\mathbf{x}_i$,

$$f_{signal}(\mathbf{x}_i) = \max\left(f_{\ell_{2body}}(\mathbf{x}_i), f_{\ell_{3body}}(\mathbf{x}_i), f_{\ell_{4body}}(\mathbf{x}_i)\right)$$

The unique score f_{unique}, defined as

$$f_{unique}(\mathbf{x}_i) = f_{signal}(\mathbf{x}_i) - f_{background}(\mathbf{x}_i)$$

		Predicted label				
		2body	3body	4body	bkgd	
Real label	2body	1636	7	4	27	Instances
	3body	22	1980	2683	207	
	4body	16	795	8240	339	
	bkgd	255	961	21019	4001962	
	2body	0.9773	0.0042	0.0024	0.0161	%
	3body	0.0045	0.4047	0.5484	0.0423	
	4body	0.0017	0.0847	0.8775	0.0361	
	bkgd	0.0001	0.0002	0.0052	0.9945	

Table 4.3 – The confusion matrix, at the instance level, on the test set.

		Predicted label				
		2body	3body	4body	bkgd	
Real label	2body	1636	7	4	27	Bags
	3body	8	895	956	109	
	4body	3	140	1663	139	
	bkgd	0	0	2	47275	
	2body	0.9773	0.0042	0.0024	0.0161	%
	3body	0.0041	0.4548	0.4858	0.0554	
	4body	0.0015	0.0720	0.8550	0.0715	
	bkgd	0.0000	0.0000	0.0000	1.0000	

Table 4.4 – The confusion matrix, at the bag level, on the test set.

is thresholded in order to obtain the ROC curve in Figure 4.4.

The intelligibility of the classifier

MDDAG allows us to see exactly what base classifiers were used for a given instance. In the context of particle physics, this intelligibility can be helpful when doing analysis. Figures 4.5, 4.6 and 4.7 represent the DAG visualization of the classifier selected in the previous section.

	Instances	Bags
2body	8.7763	8.7763
3body	8.1334	14.2746
4body	6.4882	16.0126
bkgd	4.3742	35.8483
Average	4.3854	33.4581

Table 4.5 – The average classification costs in milliseconds.

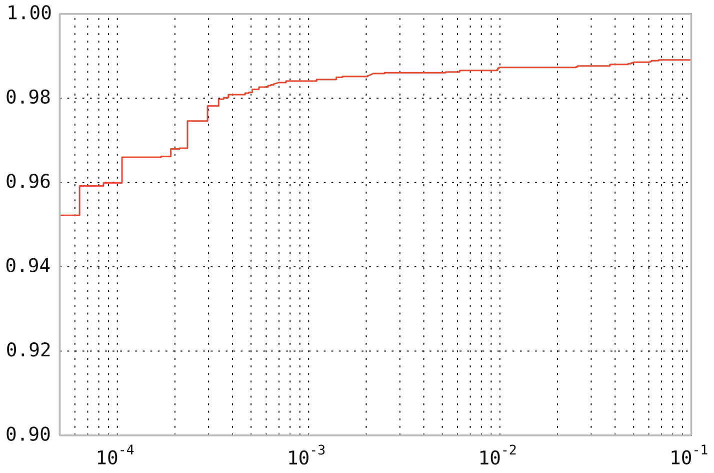


Figure 4.4 – The ROC curve of the trigger. The three signal classes are merged into one class that form with the bkgd class a virtual binary problem.

4.3 Conclusion

In this chapter, we introduced a new type of budgeted learning problems that can be encountered in particle physics experiments. The specificity of these problems lies in the complex computation of the classification costs that can be value-dependent or bag-dependent. We also showed that MDDAG can successfully accommodate complex feature cost schemata, with inter-feature dependencies. Through simple modifications of the state space and the rewards, we could include in the algorithm the entire logic for computing value-dependent and bag-dependent cost. To our knowledge, there are no turn-key methods that can be applied in a such a straightforward way to complex real applications in budgeted classification.

Further engineering steps and research questions

As a next engineering step before actually implementing MDDAG as a trigger, we can think of producing smaller decision DAGs wherein each node would use one and only one feature. This way, the cost management policy would become more explicit and the resulting classifier would potentially exhibit more intelligibility. This can be achieved by grouping all the base classifiers by feature and consider each group as an atomic base classifier, before the actual MDP training. For two-nodes trees, the grouping can be done by every couple of features, which would only increase the number of nodes in the final DAG.

Furthermore, one could reduce the number of used base classifiers by binning the feature values. Classifiers depending on feature values that fall into the same bin can be summed up simply by adding their coefficients. Also, if we form groups of base classifiers, nothing prevents us from, in turn, defining a sub-state space within each group in order to obtain a sequential instance-dependent sparsity inside each group of base classifiers.

These are engineering considerations that show that MDDAG, by using base classifiers, is flexible enough to be further "tuned" and adapted to real applications. However, grouping the base classifiers can also bring interesting research questions about how to hierarchically define a state-space so that subgroups of base classifiers are recursively sparsified. This could be, in fact, a research direction towards scaling MDDAG to a broader range of classification problems, that go beyond fast classification. Another line of research is also to derive a batch version of the algorithm. This could for example allow to tackle multiple-instance problems through sequential models.

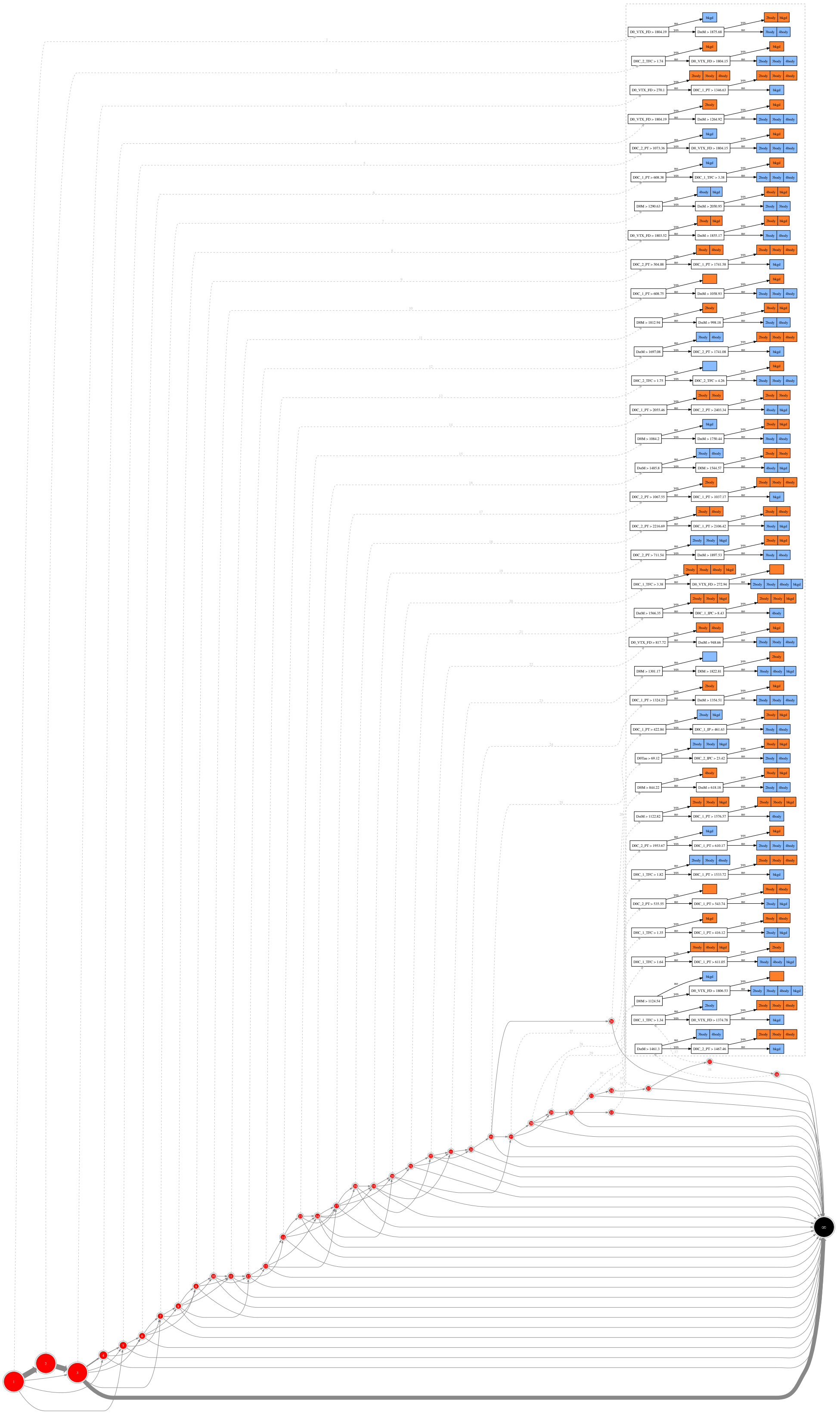


Figure 4.5 – A visualization of the selected MDDAG classifier. The red nodes represent 2-node trees, which are represented in the upper right corner. Every red node is linked to its corresponding tree with a dashed line. As for the previous visualizations, the node radii and the edge widths are proportional to the number of instances that traverse the corresponding element.

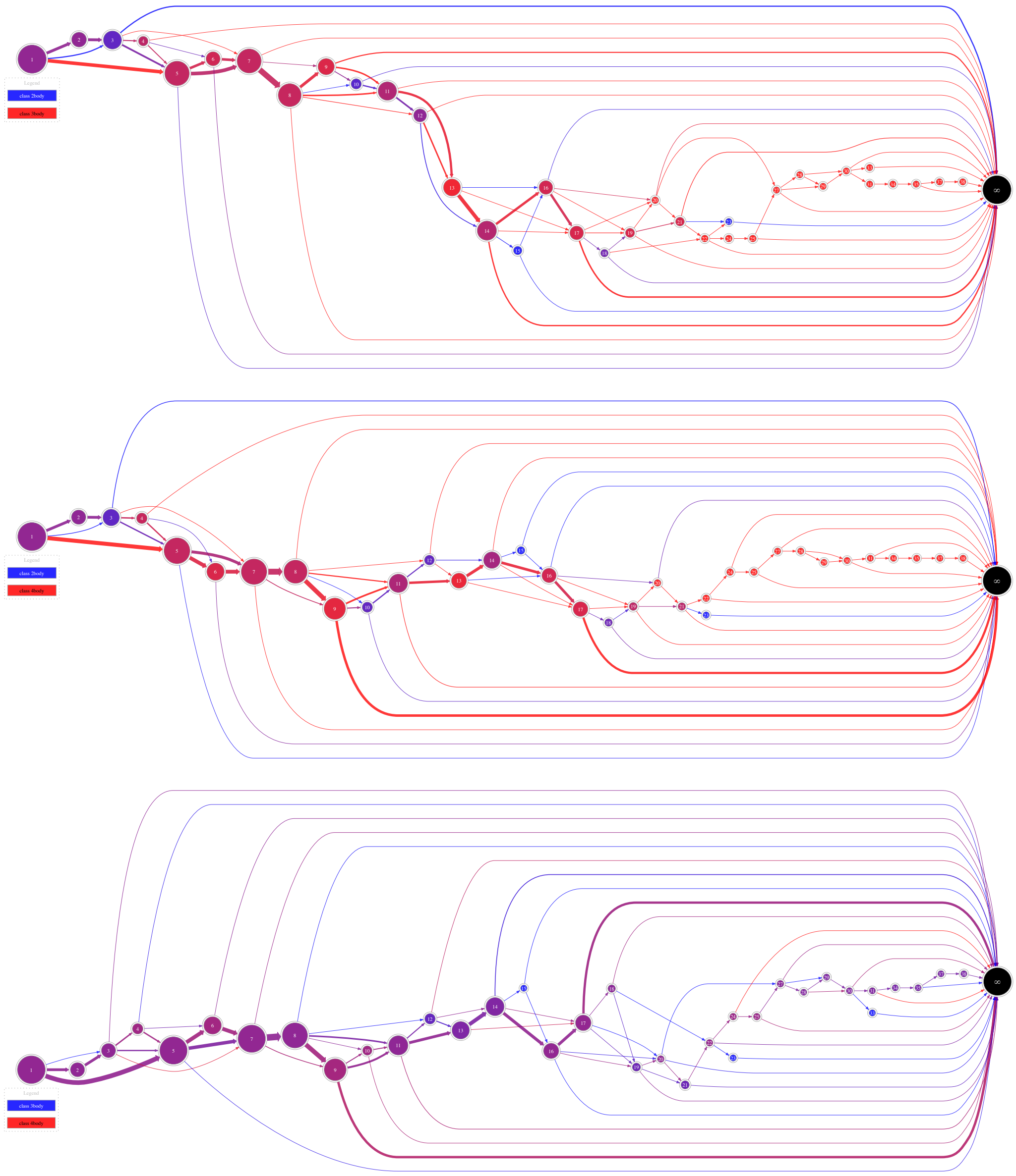


Figure 4.6 – MDDAG: The 3 signal classes visualized pair-wise. The colors represent the class-wise proportions.

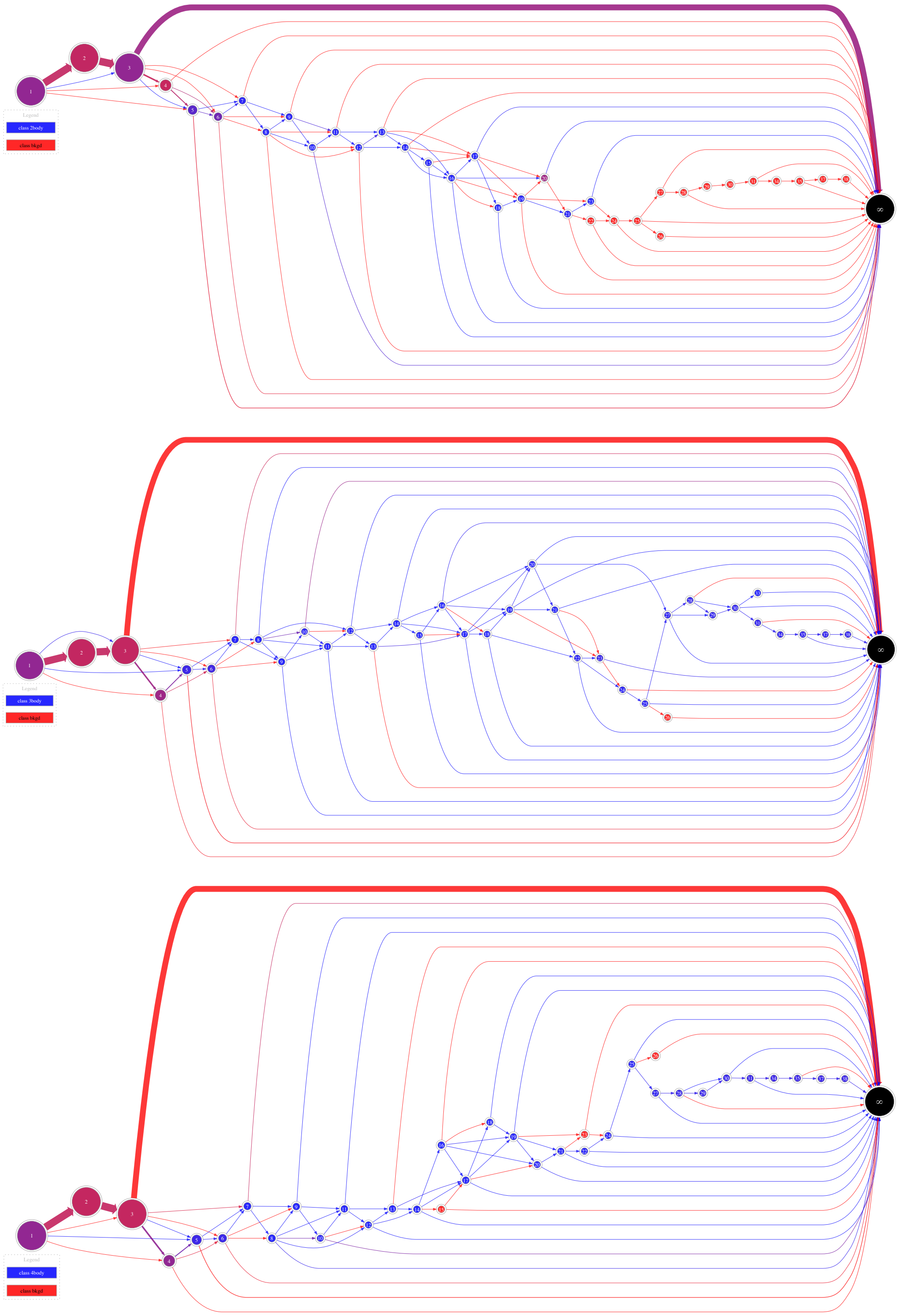


Figure 4.7 – MDDAG: The 3 signal classes visualized along with the background class. The colors represent the class-wise proportions.

CHAPTER 5

Conclusion

5.1 Epilogue

In this dissertation, we have introduced a novel way to design fast classifiers. The proposed approach benefits both from supervised learning in the training of accurate classifiers and reinforcement learning in the flexibility to accommodate real application constraints. The use of boosted base classifiers, even though not necessary at a conceptual level, shows in practice the benefits of providing relevant classifiers that have low computational costs and that can provide a reliable classification when combined together. Furthermore, the pre-training of the base classifiers simplifies the subsequent reinforcement learning problem and can be seen as a help in the exploration / exploitation learning of the MDP.

Our approach can be related to many domains. By our initial motivation, we naturally position MDDAG among the fast cascade classifiers that also share with MDDAG the sequential aspect of evaluating boosted classifiers. Because it produces sparse final classifiers, it is also linked to feature selection, and in particular, because it takes the different feature costs into account, MDDAG can be also related to *active feature acquisition* and budgeted learning.

It appears to us that designing sequential predictors is a particularly *natural* way of conceiving supervised learning in the sense that, for living beings in nature, learning often occurs in step-wise episodes and during the inference, not all the cognitive resources are used but only what is necessary to process a punctual task. There is also evidence that for classifying hierarchical objects, the human brain does have different answer delays depending on the semantic distance of the objects in the hierarchy. In this regard, predicting with sequential models mimics the non-monolithic aspect of the brain when classifying objects. It has been logical for what is considered today as "stantard" classification algorithms in machine learning to be first formalized as monolithic procedures that output "one-shot" answers. Obtaining good generalizations properties is, in and of itself, far from being a trivial problem, however, we saw in this thesis and through the different works that we reviewed that "unrolling" the prediction into a sequence of decision steps is an important step to make in order to open machine learning to a broader range of applications that will, in turn, certainly lead to new research questions.

The main difficulty of learning sequential models lies, at it is recurrent in machine learning, in the curse of dimensionality. In fact, adding a temporal dimension to the selection of the feature or the classifiers results in a complex combinatorial problem. It is perhaps the reason why reinforcement learning algorithms have been so successful at solving these problems since they explicitly trade off the exploitation of the relevant features / classifiers with the exploration of this complex space.

The new question is then what are the most appropriate reinforcement learning methods for sequential prediction tasks and in particular, can we device tailored approaches that are more specific to these types of problems. In fact, reinforcement learning algorithms were generally designed for other types of problems and with other type of constraints. Some of these constraints are simply absent from prediction tasks which makes the prediction an easier problem and, conversely, prediction tasks introduce new challenges:

- The state space space is usually intractable due to the number of continuous variables. Value estimation becomes then difficult and consequently, identifying good actions is also difficult.
- Some actions can have *meaning* in certain applications where the intelligibility is important.
- On the other hand, the environment is usually deterministic and in, some cases, completely known which can lead to analytical solutions as with dynamic programming.
- When the environment is deterministic, new exploration strategies related to the application domain can appear to more suitable.
- Finally, prediction tasks usually correspond to a finite horizon sequential problem.

5.2 Future work

Many directions can be investigated from now, both in the specific framework of MDDAG and the general concept of prediction with sequential models. MDDAG is now restricted to fast and budgeted classifier where the average number of evaluated base classifiers is relatively low. The same sparsity questions however can be tackled for more complex classifiers. In fact, some applications require a number of base classifiers that is in the order of tens of thousands in order to compete with the most accurate classifiers (eg. Kégl and Busa-Fekete (2009) for the digit recognition and Busa-Fekete et al. (2011, 2013) for ranking problems) so, without going down to the level of tens of base classifiers, is it still possible to reduce the overall complexity by an important factor? Such a requirement is a complex combinatorial problem for which one can not afford keeping simple controllers.

Another question related to MDDAG concerns the path space that the observations traverse. As we saw in Section 3.1.7, there is an unsupervised side effect that takes place when learning the agent. Similar examples tend to follow similar paths even when the class information is not given during the learning, forming a relatively simple encoding of the instances. It is interesting to investigate from a information theoretic point of view the gain that this graph shape offers, whether this encoding can be made more complex and to which extent it can be exploited during the classification.

If the path that the examples take conveys some new information, one can think for example of using MDDAG as a base classifier within an ensemble model. This can also be a way to scale the algorithm to other types of problems where the budget is not strictly bounded. Historically, this happened for decision trees as they are already full classification algorithms, but can also be used to capture variable correlations as base classifiers before being boosted or bagged.

Another direction would be to derive a batch version of MDDAG, which could then be adapted to problems where the grouping of observations is necessary, such as in Multiple Instance Learning.

On a more generic level, we think that one of the main upcoming challenges of machine learning would be to devise a unified framework for sequential prediction in order to foster the investigation of new dedicated learning methods as well as new learning formalisms. Orthogonally to deep learning approaches that are showing that feature abstraction is an important component of learning, we think that incorporating sequentiality into the prediction can broaden the *palette* of applications of machine learning methods. Again, behaviorist psychology and cognitive science appear to be a tremendous source of inspiration for computer scientists.

Bibliography

Aaij, R., Albrecht, J., Alessio, F., Amato, S., Aslanides, E., Belyaev, I., van Beuzekom, M., Bonaccorsi, E., Bonnefoy, R., Brarda, L., et al. (2013). The LHCb trigger and its performance in 2011. *Journal of Instrumentation*, 8(04):P04022. (Cited on page 93.)

Bache, K. and Lichman, M. (2013). UCI machine learning repository. (Cited on page 53.)

Benbouzid, D., Busa-Fekete, R., Casagrande, N., Collin, F.-D., and Kégl, B. (2012a). MultiBoost: a multi-purpose boosting package. *Journal of Machine Learning Research*, 13:549–553. (Cited on pages 28 and 51.)

Benbouzid, D., Busa-Fekete, R., and Kégl, B. (2011). MDDAG: learning deep decision DAGs in a Markov decision process setup. In *NIPS'11 workshop on Deep Learning and Unsupervised Feature Learning*. (Cited on pages 70, 85 and 89.)

Benbouzid, D., Busa-Fekete, R., and Kégl, B. (2012b). Fast classification using sparse decision DAGs. In *Proceedings of the 29th International Conference on Machine Learning*. (Cited on pages 48, 85 and 89.)

Bourdev, L. and Brandt, J. (2005). Robust object detection via soft cascade. In *Conference on Computer Vision and Pattern Recognition*, volume 2, pages 236–243. IEEE Computer Society. (Cited on pages 28, 38, 39 and 40.)

Bradley, J. and Schapire, R. (2008). FilterBoost: Regression and classification on large datasets. In *Advances in Neural Information Processing Systems*, volume 20. The MIT Press. (Cited on page 28.)

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). Classification and regression trees. wadsworth & brooks. *Monterey, CA*. (Cited on page 44.)

Brubaker, S., Mullin, M., and Rehg, J. (2006). Towards optimal training of cascaded detectors. *Computer Vision–ECCV 2006*, pages 325–337. (Cited on page 40.)

Brubaker, S. C., Wu, J., Sun, J., Mullin, M. D., and Rehg, J. M. (2008). On the design of cascades of boosted ensembles for face detection. *International Journal of Computer Vision*, 77(1-3):65–86. (Cited on page 40.)

Busa-Fekete, R., Kégl, B., Éltető, T., and Szarvas, G. (2011). A robust ranking methodology based on diverse calibration of AdaBoost. In *European Conference on Machine Learning*, volume LNCS, 6911, pages 263–279. (Cited on page 108.)

Busa-Fekete, R., Kégl, B., Éltető, T., and Szarvas, G. (2013). Tune and mix: learning to rank using ensembles of calibrated multi-class classifiers. *Machine Learning Journal*, 93(2):261–292. (Cited on page 108.)

Busoniu, L., Babuska, R., De Schutter, B., and Ernst, D. (2010). *Reinforcement learning and dynamic programming using function approximators*. CRC Press. (Cited on page 65.)

Cambazoglu, B. B., Zaragoza, H., Chapelle, O., Chen, J., Liao, C., Zheng, Z., and Degenhardt, J. (2010). Early exit optimizations for additive machine learned ranking systems. In *Proceedings of the third ACM International Conference on Web Search and Data Mining*, pages 411–420. (Cited on page 18.)

Chen, M., Xu, Z., Weinberger, K., Chapelle, O., and Kedem, D. (2012). Classifier cascade for minimizing feature evaluation cost. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)*. (Cited on page 44.)

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297. (Cited on page 44.)

Cowan, N. (2001). The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and brain sciences*, 24(1):87–114. (Cited on page 12.)

Daumé III, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine Learning*, 75(3):297–325. (Cited on pages 82 and 89.)

Dietterich, T. G., Lathrop, R. H., and Lozano-Pérez, T. (1997). Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1):31–71. (Cited on pages 93 and 94.)

Dulac-Arnold, G., Denoyer, L., Preux, P., and Gallinari, P. (2011a). Datum-wise classification: a sequential approach to sparsity. In *Machine Learning and Knowledge Discovery in Databases*, pages 375–390. Springer. (Cited on pages 83, 85 and 89.)

Dulac-Arnold, G., Denoyer, L., Preux, P., and Gallinari, P. (2011b). Datum-wise classification: A sequential approach to sparsity. In *European Conference on Machine Learning*. (Cited on pages 86 and 89.)

Dulac-Arnold, G., Denoyer, L., Preux, P., and Gallinari, P. (2012). Sequential approaches for learning datum-wise sparse representations. *Machine Learning*, pages 1–36. (Cited on pages 83, 84, 85, 86 and 89.)

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6. (Cited on pages 65 and 85.)

Fan, W., Stolfo, S. J., Zhang, J., and Chan, P. K. (1999). Adacost: misclassification cost-sensitive boosting. In *ICML*, pages 97–105. Citeseer. (Cited on page 37.)

Freund, Y. and Mason, L. (1999). The alternating decision tree learning algorithm. In *Proceedings of the 16th International Conference on Machine Learning*, pages 124–133. (Cited on page 43.)

Freund, Y. and Schapire, R. (2012). *Boosting: Foundations and Algorithms*. MIT Press, Cambridge, MA. (Cited on pages 24 and 25.)

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139. (Cited on pages 24 and 27.)

Friedman, J., Hastie, T., and Tibshirani, R. (1998). Additive logistic regression: a statistical view of boosting. Technical report, Dept. of Statistics, Stanford University. (Cited on page 59.)

Gao, T. and Koller, D. (2011a). Active classification based on value of classifier. In *NIPS*, pages 1062–1070. (Cited on pages 83, 84, 86 and 89.)

Gao, T. and Koller, D. (2011b). Multiclass boosting with hinge loss based on output coding. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 569–576. (Cited on page 51.)

Gaudel, R. and Sebag, M. (2010). Feature selection as a one-player game. In *International Conference on Machine Learning (ICML'10)*. (Cited on pages 83 and 89.)

Gligorov, V. and Williams, M. (2012). Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree. Technical report, arXiv:1210.6861. (Cited on page 24.)

Greiner, R. (2002). Learning cost-sensitive active classifiers. *Artificial Intelligence*, 139(2):137–174. (Cited on page 89.)

Grubb, A. and Bagnell, J. (2012). Speedboost: Anytime prediction with uniform near-optimality. In *AISTATS*, volume 15, pages 458–466. (Cited on pages 43 and 44.)

Guyon, I. (2003). Design of experiments of the nips 2003 variable selection benchmark. In *NIPS 2003 workshop on feature extraction and feature selection*. (Cited on page 53.)

He, H., Daumé III, H., and Eisner, J. (2012). Cost-sensitive dynamic feature selection. In *ICML Workshop on Inferning: Interactions between Inference and Learning*. 6 pages. (Cited on pages 86 and 89.)

Hoecker, A., Speckmayer, P., Stelzer, J., Therhaag, J., von Toerne, E., Voss, H., Backes, M., Carli, T., Cohen, O., Christov, A., et al. (2007). Tmva-toolkit for multivariate data analysis. *arXiv preprint physics/0703039*. (Cited on page 12.)

Hou, X., Liu, C., and Tan, T. (2006). Learning boosted asymmetric classifiers for object detection. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 330–338. IEEE. (Cited on page 37.)

Huang, C., Ai, H., Li, Y., and Lao, S. (2005). Vector boosting for rotation invariant multi-view face detection. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 446–453. IEEE. (Cited on page 42.)

Ji, S. and Carin, L. (2007). Cost-sensitive feature acquisition and classification. *Pattern Recognition*, 40(5):1474–1485. (Cited on pages 86 and 89.)

Jones, M. and Viola, P. (2003). Fast multi-view face detection. *Mitsubishi Electric Research Lab TR-20003-96*, 3. (Cited on page 42.)

Kanani, P. H. and McCallum, A. (2012). Selecting actions for resource-bounded information extraction using reinforcement learning. In *WSDM*, pages 253–262. (Cited on page 84.)

Karayev, S., Baumgartner, T., Fritz, M., and Darrell, T. (2012). Timely object recognition. *machine translation*, 1:C2. (Cited on pages 84, 85, 86 and 89.)

Kearns, M. and Valiant, L. (1994). Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95. (Cited on page 23.)

Kearns, M. and Vazirani, U. V. (1994). *An introduction to computational learning theory*. The MIT Press. (Cited on page 23.)

Kégl, B. and Busa-Fekete, R. (2009). Boosting products of base classifiers. In *International Conference on Machine Learning*, volume 26, pages 497–504, Montreal, Canada. (Cited on pages 29 and 108.)

Kim, T., Budvytis, I., and Cipolla, R. (2011). Making a shallow network deep: Conversion of a boosting classifier into a decision tree by boolean optimisation. *International Journal of Computer Vision*, pages 1–13. (Cited on page 43.)

Lagoudakis, M. and Parr, R. (2003). Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th International Conference on Machine Learning*, pages 424–431. (Cited on page 86.)

Larochelle, H. and Hinton, G. (2010). Learning to combine foveal glimpses with a third-order Boltzmann machine. In *Advances in Neural Information Processing Systems 23*, pages 1243–1251. MIT Press. (Cited on pages 84 and 89.)

LeCun, Y. and Cortes, C. (1998). Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*. (Cited on page 49.)

Lefakis, L. and Fleuret, F. (2010). Joint cascade optimization using a product of boosted classifiers. In *Advances in neural information processing systems*, pages 1315–1323. (Cited on page 41.)

Li, S. and Zhang, Z. (2004). Floatboost learning and statistical face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1112–1123. (Cited on page 42.)

Li, S., Zhu, L., Zhang, Z., Blake, A., Zhang, H., and Shum, H. (2006). Statistical learning of multi-view face detection. *Computer Vision ECCV 2002*, pages 117–121. (Cited on page 36.)

Li, S. Z., Zhang, Z., Shum, H.-Y., and Zhang, H. (2002). Floatboost learning for classification. In *Advances in Neural Information Processing Systems*, pages 993–1000. (Cited on page 37.)

Lienhart, R., Kuranov, A., and Pisarevsky, V. (2003). Empirical analysis of detection cascades of boosted classifiers for rapid object detection. *Pattern Recognition*, pages 297–304. (Cited on page 37.)

Lienhart, R. and Maydt, J. (2002). An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages I–900. IEEE. (Cited on page 36.)

Lin, Y. and Liu, T. (2005). Robust face detection with multi-class boosting. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 680–687. IEEE. (Cited on page 42.)

Maes, F., Denoyer, L., and Gallinari, P. (2009). Structured prediction with reinforcement learning. *Machine Learning Journal*, 77(2-3):271–301. (Cited on pages 82 and 89.)

Masnadi-Shirazi, H. and Vasconcelos, N. (2011). Cost-sensitive boosting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(2):294–309. (Cited on page 37.)

Mason, L., Bartlett, P., Baxter, J., and Frean, M. (2000). Boosting algorithms as gradient descent. In *Advances in Neural Information Processing Systems*, volume 12, pages 512–518. The MIT Press. (Cited on page 25.)

Meir, R. and Rätsch, G. (2003). An introduction to boosting and leveraging. *Advanced lectures on machine learning*, pages 118–183. (Cited on page 24.)

Miller, G. A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81. (Cited on page 12.)

Mitchell, T. M. (2006). *The discipline of machine learning*. Carnegie Mellon University, School of Computer Science, Machine Learning Department. (Cited on page 13.)

Morimoto, J. and Doya, K. (1998). Reinforcement learning of dynamic motor sequence: Learning to stand up. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 3, pages 1721–1726. IEEE. (Cited on pages 66 and 67.)

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, volume 2011. (Cited on page 49.)

Neu, G. and Szepesvári, C. (2009). Training parsers by inverse reinforcement learning. *Machine Learning*, 77(2-3):303–337. (Cited on pages 84, 86 and 89.)

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830. (Cited on page 53.)

Pham, M. and Cham, T. (2007). Online learning asymmetric boosted classifiers for object detection. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE. (Cited on page 37.)

Póczos, B., Abbasi-Yadkori, Y., Szepesvári, C., Greiner, R., and Sturtevant, N. (2009). Learning when to stop thinking and do something! In *Proceedings of the 26th International Conference on Machine Learning*, pages 825–832. (Cited on pages 40 and 89.)

Reyzin, L. (2011). Boosting on a budget: Sampling for feature-efficient prediction. ICML. (Cited on page 43.)

Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. *Machine Learning: ECML 2005*, pages 317–328. (Cited on page 65.)

Roth, D., Yang, M.-H., and Ahuja, N. (2000). A snow-based face detector. *Urbana*, 51:61801. (Cited on page 18.)

Rückstieß, T., Osendorfer, C., and van der Smagt, P. (2011). Sequential feature selection for classification. *AI 2011: Advances in Artificial Intelligence*, pages 132–141. (Cited on pages 83, 85, 86 and 89.)

Russell, S. and Norvig, P. (2009). *Artificial intelligence: a modern approach*. Prentice Hall. (Cited on page 11.)

Saberian, M. and Vasconcelos, N. (2010). Boosting classifier cascades. In *Advances in Neural Information Processing Systems 23*, pages 2047–2055. MIT Press. (Cited on pages 40 and 41.)

Schapire, R. E. (2003). The boosting approach to machine learning: An overview. *LECTURE NOTES IN STATISTICS-NEW YORK-SPRINGER VERLAG-*, pages 149–172. (Cited on page 24.)

Schapire, R. E. and Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336. (Cited on pages 27, 55 and 59.)

Schneiderman, H. (2004). Feature-centric evaluation for efficient cascaded object detection. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–29. IEEE. (Cited on page 18.)

Schneiderman, H. and Kanade, T. (2000). A statistical method for 3d object detection applied to faces and cars. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 746–751. IEEE. (Cited on page 18.)

Sun, P. and Zhou, J. (2013). Saving evaluation time for the decision function in boosting: Representation and reordering base learner. In *International Conference on Machine Learning (ICML)*. (Cited on pages 38 and 40.)

Sun, Y., Wong, A., and Wang, Y. (2005). Parameter inference of cost-sensitive boosting algorithms. In *Proceedings of the 4th international conference on Machine Learning and Data Mining in Pattern Recognition*, pages 21–30. Springer-Verlag. (Cited on page 37.)

Sung, K.-K. and Poggio, T. (1998). Example-based learning for view-based human face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(1):39–51. (Cited on page 41.)

Ting, K. M. (2000). A comparative study of cost-sensitive boosting algorithms. In *In Proceedings of the 17th International Conference on Machine Learning*. Citeseer. (Cited on page 37.)

Trapeznikov, K. and Saligrama, V. (2013). Supervised sequential classification under budget constraints. (Cited on pages 83, 85, 86 and 89.)

Tu, Z. (2005). Probabilistic boosting-tree: Learning discriminative models for classification, recognition, and clustering. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1589–1596. IEEE. (Cited on page 43.)

Turney, P. (1995). Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research (JAIR)*, 2. (Cited on page 44.)

Viola, P. and Jones, M. (2001). Fast and robust classification using asymmetric adaboost and a detector cascade. *Proc. of NIPS01*. (Cited on page 37.)

Viola, P. and Jones, M. (2004). Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154. (Cited on pages 18, 28, 31, 35, 39 and 40.)

Viola, P., Jones, M. J., and Snow, D. (2005). Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161. (Cited on page 36.)

Viola, P., Platt, J., and C., Z. (2006). Multiple instance boosting for object detection. In *Advances in Neural Information Processing Systems*, volume 18, pages 1417–1424. (Cited on page 94.)

Weiss, D., Sapp, B., and Taskar, B. (2012). Structured prediction cascades. *arXiv preprint arXiv:1208.3279*. (Cited on page 18.)

Weiss, D., Sapp, B., and Taskar, B. (2013). Dynamic structured model selection. ICCV. (Cited on pages 83, 85 and 89.)

Weiss, D. J. and Taskar, B. (2013). Learning adaptive value of information for structured prediction. In *Advances in Neural Information Processing Systems*, pages 953–961. (Cited on pages 83, 85 and 89.)

Wu, B., Ai, H., Huang, C., and Lao, S. (2004). Fast rotation invariant multi-view face detection based on real adaboost. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 79–84. Ieee. (Cited on pages 37, 38, 40, 41 and 42.)

Xiao, R., Zhu, L., and Zhang, H. J. (2003). Boosting chain learning for object detection. In *Ninth IEEE International Conference on Computer Vision*, volume 9, pages 709–715. (Cited on pages 37, 38 and 40.)

Xu, Z., Kusner, M., Huang, G., and Weinberger, K. Q. (2013). Anytime representation learning. In Dasgupta, S. and Mcallester, D., editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1076–1084. JMLR Workshop and Conference Proceedings. (Cited on page 44.)

Xu, Z., Kusner, M. J., Weinberger, K. Q., and Chen, M. (2012a). Cost-sensitive tree of classifiers. *arXiv preprint arXiv:1210.2771*. (Cited on page 44.)

Xu, Z., Weinberger, K., and Chapelle, O. (2012b). The greedy miser: Learning under test-time budgets. *Arxiv preprint arXiv:1206.6451*. (Cited on page 44.)

Zehnder, P., Meier, E., and Gool, L. (2008). An efficient shared multi-class detection cascade. In *British Machine Vision Conf.* (Cited on page 42.)

Zhang, C. and Viola, P. (2011). Multiple-instance pruning for learning efficient cascade detectors. US Patent 8,010,471. (Cited on page 39.)

Zhang, C. and Zhang, Z. (2009). Winner-take-all multiple category boosting for multi-view face detection. Technical report, Citeseer. (Cited on page 42.)

Zhang, C. and Zhang, Z. (2010). A survey of recent advances in face detection. Technical report, Microsoft Research. (Cited on page 36.)

Abstract

Classification in machine learning has been extensively studied during the past decades. Many solutions have been proposed to output accurate classifiers and to obtain statistical grantees on the unseen observations. However, when machine learning algorithms meet concrete industrial or scientific applications, new computational criteria appear to be as important to satisfy as those of classification accuracy. In particular, when the output classifier must comply with a computational budget needed to obtain the features that are evaluated at test time, we talk about "budgeted" learning. The features can have different acquisition costs and, often, the most discriminative features are the costlier. Medical diagnosis and web-page ranking, for instance, are typical applications of budgeted learning. In the former, the goal is to limit the number of medical tests evaluate for patients, and in the latter, the ranker has limited time to order documents before the user goes away.

This thesis introduces a new way of tackling classification in general and budgeted learning problems in particular, through a novel framework lying in the intersection of supervised learning and decision theory. We cast the classification problem as a sequential decision making procedure and show that this framework yields fast and accurate classifiers. Unlike classical classification algorithms that output a "one-shot" answer, we show that considering the classification as a series of small steps wherein the information is gathered sequentially also provides a flexible framework that allows to accommodate different types of budget constraints in a "natural" way. In particular, we apply our method to a novel type of budgeted learning problems motivated by particle physics experiments. The particularity of this problem lies in atypical budget constraints and complex cost calculation schemata where the calculation of the different features depends on many factors. We also review similar sequential approaches that have recently known a particular interest and provide a global perspective on this new paradigm.