Microsoft Research technical report TR-2011-114

Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning

A. Criminisi[1], J. Shotton[2] and E. Konukoglu[3]

[1] *Microsoft Research Ltd, 7 J J Thomson Ave, Cambridge, CB3 0FB, UK*
[2] *Microsoft Research Ltd, 7 J J Thomson Ave, Cambridge, CB3 0FB, UK*
[3] *Microsoft Research Ltd, 7 J J Thomson Ave, Cambridge, CB3 0FB, UK*

Abstract

This paper presents a unified, efficient model of random decision forests which can be applied to a number of machine learning, computer vision and medical image analysis tasks.

Our model extends existing forest-based techniques as it unifies classification, regression, density estimation, manifold learning, semi-supervised learning and active learning under the same decision forest framework. This means that the core implementation needs be written and optimized only once, and can then be applied to many diverse tasks. The proposed model may be used both in a generative or discriminative way and may be applied to discrete or continuous, labelled or unlabelled data.

The main contributions of this paper are: 1) proposing a single, probabilistic and efficient model for a variety of learning tasks; 2) demonstrating margin-maximizing properties of classification forests; 3) introducing density forests for learning accurate probability density functions; 4) proposing efficient algorithms for sampling from the forest generative model; 5) introducing manifold forests for non-linear embedding and dimensionality reduction; 6) proposing new and efficient forest-

based algorithms for transductive and active learning. We discuss how alternatives such as random ferns and extremely randomized trees stem from our more general model.

This paper is directed at both students who wish to learn the basics of decision forests, as well as researchers interested in our new contributions. It presents both fundamental and novel concepts in a structured way, with many illustrative examples and real-world applications. Thorough comparisons with state of the art algorithms such as support vector machines, boosting and Gaussian processes are presented and relative advantages and disadvantages discussed. The many synthetic examples and existing commercial applications demonstrate the validity of the proposed model and its flexibility.

Contents

1

Overview and scope

This document presents a unified, efficient model of random decision forests which can be used in a number of applications such as scene recognition from photographs, object recognition in images, automatic diagnosis from radiological scans and semantic text parsing. Such applications have traditionally been addressed by different, supervised or unsupervised machine learning techniques.

In this paper, diverse learning tasks such as regression, classification and semi-supervised learning are explained as instances of the same general decision forest model. This unified framework then leads to novel uses of forests, *e.g.* in density estimation and manifold learning. The corresponding inference algorithm can be implemented and optimized only once, with relatively small changes allowing us to address different tasks.

This paper is directed at engineers and PhD students who wish to learn the basics of decision forests as well as more senior researchers interested in the new research contributions.

We begin by presenting a roughly chronological, non-exhaustive survey of decision trees and forests, and their use in the past two decades. Further references will be available in the relevant chapters.

1.1 A brief literature survey

One of the seminal works on decision tress is the Classification and Regression Trees (CART) book of Breiman et al. [12], where the authors describe the basics of decision trees and their use for both classification and regression. However, training optimal decision trees from data has been a long standing problem, for which one of the most popular algorithms is "C4.5" of Quinlan [72].

In this early work trees are used as individual entities. However, recently it has emerged how using an ensemble of learners (*e.g.* weak classifiers) yields greater accuracy and generalization.[1] One of the earliest references to ensemble methods is in the boosting algorithm of Schapire [78], where the author discusses how iterative re-weighting of training data can be used to build accurate "strong" classifiers as linear combination of many "weak" ones.

A random decision forest is instead an ensemble of randomly trained decision trees. Decision forests seem to have been introduced for the first time in the work of T. K. Ho for handwritten digit recognition [45]. In that work the author discusses tree training via randomized feature selection; a very popular choice nowadays. All tree outputs are fused together by averaging their class posteriors. In subsequent work [46] forests are shown to yield superior generalization to both boosting and pruned C4.5-trained trees on some tasks. The author also shows comparisons between different split functions in the tree nodes. A further application of randomized trees to digit and shape recognition is reported in [3].

Breiman's work in [10, 11] further consolidated the random forest model. However, the author introduces a different way of injecting randomness in the forest by randomly sampling the labelled training data ("bagging"). The author also describes techniques for predicting the forest test error based on measures of tree strength and correlation.

In computer vision, ensemble methods became popular with the seminal face and pedestrian detection papers of Viola and Jones [99, 98]. Recent years have seen an explosion of forest-based techniques in

[1] Depending on perspective trees can be seen as weak or strong classifiers [102].

the machine learning, vision and medical imaging literature [9, 15, 24, 29, 33, 35, 52, 53, 54, 56, 58, 59, 60, 61, 65, 70, 80, 83, 88, 102]. Decision forests compare favourably with respect to other techniques [15] and have lead to one of the biggest success stories of computer vision in recent years: the Microsoft Kinect for XBox 360 [37, 82, 100].

1.2 Outline

The document is organized as a tutorial, with different chapters for different tasks and structured references within. It was compiled in preparation for the homonymous tutorial presented at the International Conference on Computer Vision (ICCV) held in Barcelona in 2011. Corresponding slides and demo videos may be downloaded from [1].

A new, unified model of decision forests is presented in chapter 2. Later chapters show instantiations of such model to specific tasks such as classification (chapter 3) and regression (chapter 4). Chapter 5 introduces, for the first time, the use of forests as density estimators. The corresponding *generative* model gives rise to novel manifold forests (chapter 6) and semi-supervised forests (chapter 7). Next, we present details of the general forest model and associated training and testing algorithms.

2

The random decision forest model

Problems related to the automatic or semi-automatic analysis of complex data such as text, photographs, videos and n-dimensional medical images can be categorized into a relatively small set of prototypical machine learning tasks. For instance:

- Recognizing the type (or category) of a scene captured in a photograph can be cast as *classification*, where the output is a discrete, categorical label (*e.g.* a beach scene, a cityscape, indoor *etc.*).
- Predicting the price of a house as a function of its distance from a good school may be cast as a *regression* problem. In this case the desired output is a continuous variable.
- Detecting abnormalities in a medical scan can be achieved by evaluating the scan under a learned probability *density* function for scans of healthy individuals.
- Capturing the intrinsic variability of size and shape of patients brains in magnetic resonance images may be cast as *manifold learning*.
- Interactive image segmentation may be cast as a *semi-*

supervised problem, where the user's brush strokes define labelled data and the rest of image pixels provide already available unlabelled data.

- Learning a general rule for detecting tumors in images using minimal amount of manual annotations is an *active learning* task, where expensive expert annotations can be optimally acquired in the most economical fashion.

Despite the recent popularity of decision forests their application, has been confined mostly to classification tasks. This chapter presents a unified model of decision forests which can be used to tackle *all* the common learning tasks outlined above: classification, regression, density estimation, manifold learning, semi-supervised learning and active learning.

This unification yields both theoretical and practical advantages. In fact, we show how multiple prototypical machine learning problems can be all mapped onto the same general model by means of different parameter settings. A practical advantage is that one can implement and optimize the associated inference algorithm only once and then apply it, with relatively small modifications, in many tasks. As it will become clearer later our model can deal with both labelled and unlabelled data, with discrete and continuous output.

Before delving into the model description we need to introduce the general mathematical notation and formalism. Subsequent chapters will make clear which components need be adapted and how for each specific task.

2.1 Background and notation

2.1.1 Decision tree basics

Decision trees have been around for a number of years [12, 72]. Their recent revival is due to the discovery that ensembles of slightly different trees tend to produce much higher accuracy on previously unseen data, a phenomenon known as generalization [3, 11, 45]. Ensembles of trees will be discussed extensively throughout this document. But let us focus first on individual trees.

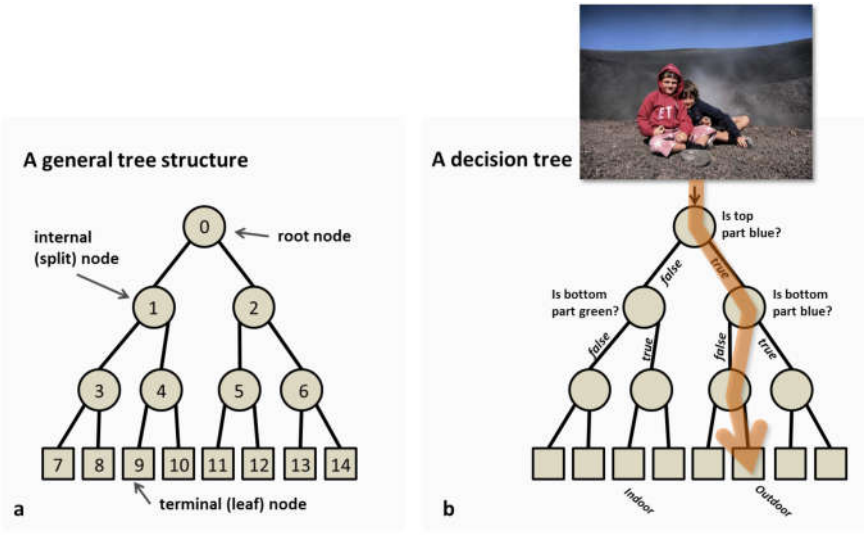

Fig. 2.1: **Decision tree. (a)** A tree is a set of nodes and edges organized in a hierarchical fashion. In contrast to a graph, in a tree there are no loops. Internal nodes are denoted with circles and terminal nodes with squares. **(b)** A decision tree is a tree where each split node stores a test function to be applied to the incoming data. Each leaf stores the final answer (predictor). This figure shows an illustrative decision tree used to figure out whether a photo represents and indoor or outdoor scene.

A tree is a collection of nodes and edges organized in a hierarchical structure (fig. 2.1a). Nodes are divided into internal (or split) nodes and terminal (or leaf) nodes. We denote internal nodes with circles and terminal ones with squares. All nodes have exactly one incoming edge. Thus, in contrast to graphs a tree does not contain loops. Also, in this document we focus only on binary trees where each internal node has exactly two outgoing edges.

A *decision* tree is a tree used for making decisions. For instance, imagine we have a photograph and we need to construct an algorithm for figuring out whether it represents an indoor scene or an outdoor one. We can start by looking at the top part of the image. If it is blue then that probably corresponds to a sky region. However, if also the

bottom part of the photo is blue then perhaps it is an indoor scene and we are looking at a blue wall. All the questions/tests which help our decision making can be organized hierarchically, in a decision tree structure where each internal node is associated with one such test. We can imagine the image being injected at the root node, and a test being applied to it (see fig. 2.1b). Based on the result of the test the image data is then sent to the left or right child. There a new test is applied and so on until the data reaches a leaf. The leaf contains the answer (*e.g.* "outdoor"). Key to a decision tree is to establish all the test functions associated to each internal node and also the decision-making predictors associated with each leaf.

A decision tree can be interpreted as a technique for splitting complex problems into a hierarchy of simpler ones. It is a hierarchical piecewise model. Its parameters (*i.e.* all node tests parameters, the leaves parameters etc.) could be selected by hand for simple problems. In more complex problems (such as vision related ones) the tree structure and parameters are learned automatically from training data. Next we introduce some notation which will help us formalize these concepts.

2.1.2 Mathematical notation

We denote vectors with boldface lowercase symbols (*e.g.* $\mathbf{v}$), matrices with teletype uppercase letters (*e.g.* $\mathtt{M}$) and sets in calligraphic notation (*e.g.* $\mathcal{S}$).

A generic data point is denoted by a vector $\mathbf{v} = (x_1, x_2, \cdots, x_d) \in \mathbb{R}^d$. Its components x_i represent some scalar feature responses. Such features are kept general here as they depend on the specific application at hand. For instance, in a computer vision application $\mathbf{v}$ may represent the responses of a chosen filter bank at a particular pixel location. See fig. 2.2a for an illustration.

The feature dimensionality d may be very large or even infinite in practice. However, in general it is not necessary to compute all d dimensions of $\mathbf{v}$ ahead of time, but only on a as-needed basis. As it will be clearer later, often it is advantageous to think of features as being randomly sampled from the set of all possible features, with a function $\boldsymbol{\phi}(\mathbf{v})$ selecting a subset of features of interest. More formally,

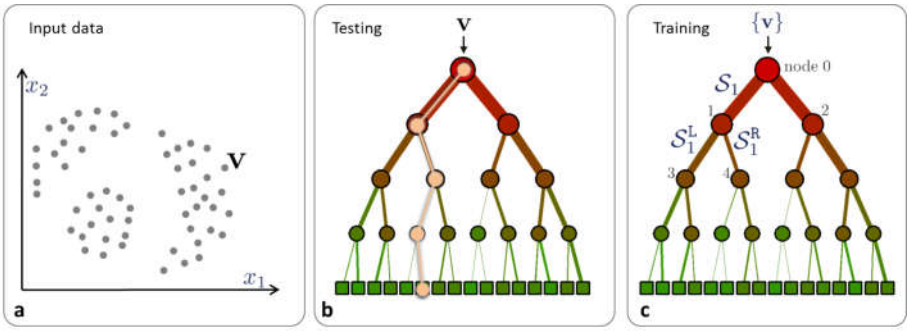

Fig. 2.2: **Basic notation. (a)** Input data is represented as a collection of points in the d-dimensional space defined by their feature responses (2D in this example). **(b)** A decision tree is a hierarchical structure of connected nodes. During testing, a split (internal) node applies a test to the input data $\mathbf{v}$ and sends it to the appropriate child. The process is repeated until a leaf (terminal) node is reached (beige path). **(c)** Training a decision tree involves sending all training data $\{\mathbf{v}\}$ into the tree and optimizing the parameters of the split nodes so as to optimize a chosen energy function. See text for details.

$\boldsymbol{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$, with $d' << d$.

2.1.3 Training and testing decision trees

At a high level, the functioning of decision trees can be separated into an off-line phase (training) and an on-line one (testing).

Tree testing (runtime). Given a previously unseen data point $\mathbf{v}$ a decision tree hierarchically applies a number of predefined tests (see fig. 2.2b). Starting at the root, each split node applies its associated split function to $\mathbf{v}$. Depending on the result of the *binary* test the data is sent to the right or left child.[1] This process is repeated until the data point reaches a leaf node.

[1] In this work we focus only on binary decision trees because they are simpler than n-ary ones. In our experiments we have not found big accuracy differences when using non binary trees.

Usually the leaf nodes contain a predictor (*e.g.* a classifier, or a regressor) which associates an output (*e.g.* a class label) to the input $\mathbf{v}$. In the case of forests many tree predictors are combined together (in ways which will be described later) to form a single forest prediction.

Tree training (off-line). The off-line, training phase is in charge of optimizing parameters of the split functions associated with all the internal nodes, as well as the leaf predictors.

When discussing tree training it is convenient to think of subsets of training points associated with different tree branches. For instance $\mathcal{S}_1$ denotes the subset of training points reaching node 1 (nodes are numbered in breadth-first order starting from 0 for the root fig. 2.2c); and $\mathcal{S}_1^{\mathtt{L}}$, $\mathcal{S}_1^{\mathtt{R}}$ denote the subsets going to the left and to the right children of node 1, respectively. In binary trees the following properties apply $\mathcal{S}_j = \mathcal{S}_j^{\mathtt{L}} \cup \mathcal{S}_j^{\mathtt{R}}$, $\mathcal{S}_j^{\mathtt{L}} \cap \mathcal{S}_j^{\mathtt{R}} = \emptyset$, $\mathcal{S}_j^{\mathtt{L}} = \mathcal{S}_{2j+1}$ and $\mathcal{S}_j^{\mathtt{R}} = \mathcal{S}_{2j+2}$ for each split node j.

Given a training set $\mathcal{S}_0$ of data points $\{\mathbf{v}\}$ and the associated ground truth labels the tree parameters are chosen so as to minimize a chosen energy function (discussed later). Various predefined stopping criteria (discussed later) are applied to decide when to stop growing the various tree branches. In our figures the edge thickness is proportional to the number of training points going through them. The node and edge colours denote some measure of information, such as purity or entropy, which depends on the specific task at hand (*e.g.* classification or regression).

In the case of a forest with T trees the training process is typically repeated independently for each tree. Note also that randomness is only injected during the training process, with testing being completely deterministic once the trees are fixed.

2.1.4 Entropy and information gain

Before discussing details about tree training it is important to familiarize ourselves with the concepts of entropy and information gain. These concepts are usually discussed in information theory or probability courses and are illustrated with toy examples in fig. 2.3 and

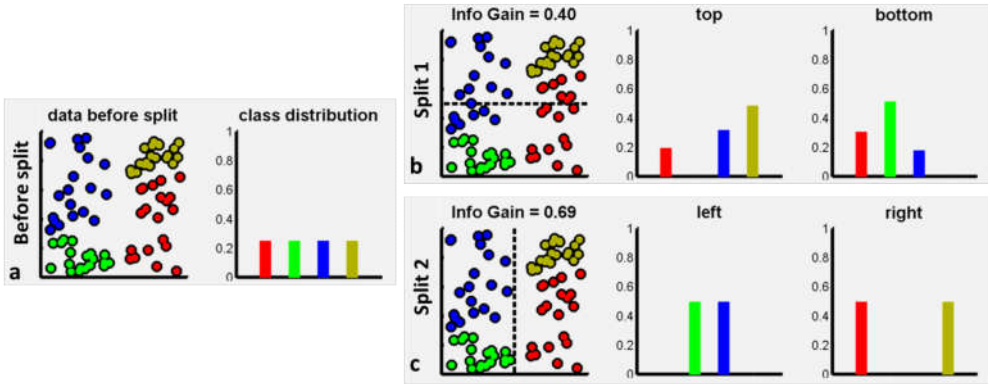

Fig. 2.3: **Information gain for discrete, non-parametric distributions. (a)** Dataset $\mathcal{S}$ before a split. **(b)** After a horizontal split. **(c)** After a vertical split.

fig. 2.4.

Figure 2.3a shows a number of data points on a 2D space. Different colours indicate different classes/groups of points. In fig. 2.3a the distribution over classes is uniform because we have exactly the same number of points in each class. If we split the data horizontally (as shown in fig. 2.3b) this produces two sets of data. Each set is associated with a lower entropy (higher information, peakier histograms). The gain of information achieved by splitting the data is computed as

$$I = H(\mathcal{S}) - \sum_{i \in \{1,2\}} \frac{|\mathcal{S}^i|}{|\mathcal{S}|} H(\mathcal{S}^i)$$

with the Shannon entropy defined mathematically as: $H(\mathcal{S}) = -\sum_{c \in \mathcal{C}} p(c) \log(p(c))$. In our example a horizontal split does not separate the data well, and yields an information gain of $I = 0.4$. When using a vertical split (such as the one in fig. 2.3c) we achieve better class separation, corresponding to lower entropy of the two resulting sets and a higher information gain ($I = 0.69$). This simple example shows how we can use information gain to select the split which produces the highest information (or confidence) in the final distributions. This concept is at the basis of the forest training algorithm.

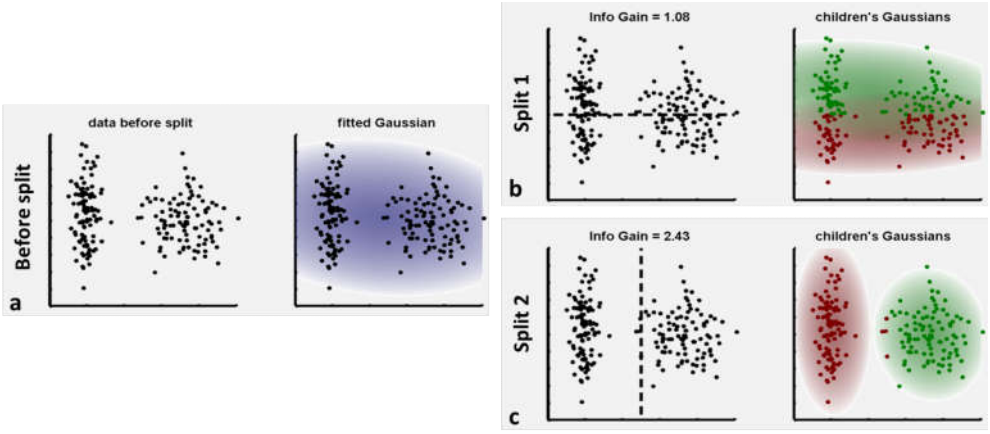

Fig. 2.4: **Information gain for continuous, parametric densities.** **(a)** Dataset $\mathcal{S}$ before a split. **(b)** After a horizontal split. **(c)** After a vertical split.

The previous example has focused on discrete, categorical distributions. But entropy and information gain can also be defined for continuous distributions. In fact, for instance, the differential entropy of a d-variate Gaussian density is defined as.

$$H(\mathcal{S}) = \frac{1}{2} \log \left((2\pi e)^d |\Lambda(\mathcal{S})| \right)$$

An example is shown in fig. 2.4. In fig. 2.4a we have a set $\mathcal{S}$ of *unlabelled* data points. Fitting a Gaussian to the entire initial set $\mathcal{S}$ produces the density shown in blue. Splitting the data horizontally (fig. 2.4b) produces two largely overlapping Gaussians (in red and green). The large overlap indicates a suboptimal separation and is associated with a relatively low information gain ($I = 1.08$). Splitting the data points vertically (fig. 2.4c) yields better separated, peakier Gaussians, with a correspondingly higher value of information gain ($I = 2.43$). The fact that the information gain measure can be defined flexibly, for discrete and continuous distributions, for supervised and unsupervised data is a useful property that is exploited here to construct a unified forest framework to address many diverse tasks.

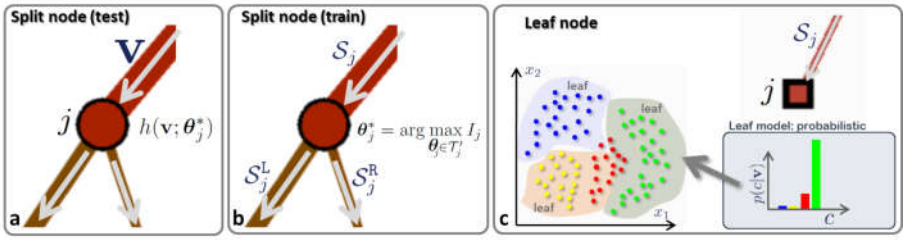

Fig. 2.5: **Split and leaf nodes. (a)** Split node (testing). A split node is associated with a weak learner (or split function, or test function). **(b)** Split node (training). Training the parameters $\boldsymbol{\theta}_j$ of node j involves optimizing a chosen objective function (maximizing the information gain I_j in this example). **(c)** A leaf node is associated with a predictor model. For example, in classification we may wish to estimate the conditional $p(c|\mathbf{v})$ with $c \in \{c_k\}$ indicating a class index.

2.2 The decision forest model

A random decision forest is an ensemble of randomly trained decision trees. The forest model is characterized by a number of components. For instance, we need to choose a family of split functions (also referred to as "weak learners" for consistency with the literature). Similarly, we must select the type of leaf predictor. The randomness model also has great influence on the workings of the forest. This section discusses each component one at a time.

2.2.1 The weak learner model

Each split node j is associated with a binary split function

$$h(\mathbf{v}, \boldsymbol{\theta}_j) \in \{0, 1\}, \tag{2.1}$$

with *e.g.* 0 indicating "false" and 1 indicating "true". The data arriving at the split node is sent to its left or right child node according to the result of the test (see fig.2.5a). The weak learner model is characterized by its parameters $\boldsymbol{\theta} = (\boldsymbol{\phi}, \boldsymbol{\psi}, \boldsymbol{\tau})$ where $\boldsymbol{\psi}$ defines the geometric primitive used to separate the data (*e.g.* an axis-aligned hyperplane, an oblique hyperplane [43, 58], a general surface *etc.*). The parameter vector $\boldsymbol{\tau}$

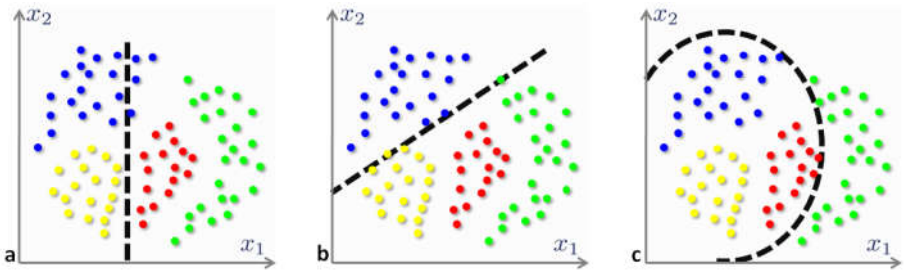

Fig. 2.6: **Example weak learners. (a)** Axis-aligned hyperplane. **(b)** General oriented hyperplane. **(c)** Quadratic (conic in 2D). For ease of visualization here we have $\mathbf{v} = (x_1\ x_2) \in \mathbb{R}^2$ and $\boldsymbol{\phi}(\mathbf{v}) = (x_1\ x_2\ 1)$ in homogeneous coordinates. In general data points $\mathbf{v}$ may have a much higher dimensionality and $\boldsymbol{\phi}$ still a dimensionality of ≤ 2.

captures thresholds for the inequalities used in the binary test. The filter function $\boldsymbol{\phi}$ selects some features of choice out of the entire vector $\mathbf{v}$. All these parameters will be optimized at each split node. Figure 2.6 illustrates a few possible weak learner models, for example:

Linear data separation. In our model linear weak learners are defined as

$$h(\mathbf{v}, \boldsymbol{\theta}_j) = [\tau_1 > \boldsymbol{\phi}(\mathbf{v}) \cdot \boldsymbol{\psi} > \tau_2]\,, \tag{2.2}$$

where [.] is the indicator function[2]. For instance, in the 2D example in fig. 2.6b $\boldsymbol{\phi}(\mathbf{v}) = (x_1\ x_2\ 1)^\top$, and $\boldsymbol{\psi} \in \mathbb{R}^3$ denotes a generic line in homogeneous coordinates. In (2.2) setting $\tau_1 = \infty$ or $\tau_2 = -\infty$ corresponds to using a single-inequality splitting function. Another special case of this weak learner model is one where the line $\boldsymbol{\psi}$ is aligned with one of the axes of the feature space (*e.g.* $\boldsymbol{\psi} = (1\ 0\ \psi_3)$ or $\boldsymbol{\psi} = (0\ 1\ \psi_3)$, as in fig. 2.6a). Axis-aligned weak learners are often used in the boosting literature and they are referred to as *stumps* [98].

[2] Returns 1 if the argument is true and 0 if it is false.

Nonlinear data separation. More complex weak learners are obtained by replacing hyperplanes with higher degree of freedom surfaces. For instance, in 2D one could use conic sections as

$$h(\mathbf{v}, \boldsymbol{\theta}_j) = \left[\tau_1 > \boldsymbol{\phi}^\top(\mathbf{v})\ \boldsymbol{\psi}\ \boldsymbol{\phi}(\mathbf{v}) > \tau_2\right] \tag{2.3}$$

with $\boldsymbol{\psi} \in \mathbb{R}^{3\times 3}$ a matrix representing the conic section in homogeneous coordinates.

Note that low-dimensional weak learners of this type can be used even for data that originally resides in a very high dimensional space ($d >> 2$). In fact, the selector function $\boldsymbol{\phi}_j$ can select a different, small set of features (*e.g.* just one or two) and they can be different for different nodes.

As shown later, the number of degrees of freedom of the weak learner influences heavily the forest generalization properties.

2.2.2 The training objective function

During training, the optimal parameters $\boldsymbol{\theta}_j^*$ of the j^{th} split node need to be computed. This is done here by maximizing an information gain objective function:

$$\boldsymbol{\theta}_j^* = \arg\max_{\boldsymbol{\theta}_j} I_j \tag{2.4}$$

with

$$I_j = I(\mathcal{S}_j, \mathcal{S}_j^{\mathtt{L}}, \mathcal{S}_j^{\mathtt{R}}, \boldsymbol{\theta}_j). \tag{2.5}$$

The symbols $\mathcal{S}_j, \mathcal{S}_j^{\mathtt{L}}, \mathcal{S}_j^{\mathtt{R}}$ denote the sets of training points before and after the split (see fig. 2.2b and fig. 2.5b). Equation (2.5) is of an abstract form here. Its precise definition depends on the task at hand (*e.g.* supervised or not, continuous or discrete output) as will be shown in later chapters.

Node optimization. The maximization operation in (2.4) can be achieved simply as an exhaustive search operation. Often, finding the optimal values of the $\boldsymbol{\tau}$ thresholds may be obtained efficiently by means of integral histograms.

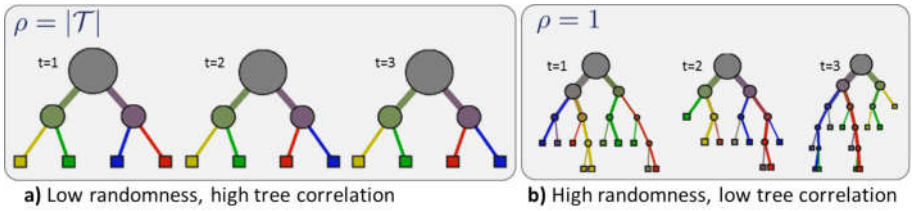

Fig. 2.7: **Controlling the amount of randomness and tree correlation. (a)** Large values of ρ correspond to little randomness and thus large tree correlation. In this case the forest behaves very much as if it was made of a single tree. **(b)** Small values of ρ correspond to large randomness in the training process. Thus the forest component trees are all very different from one another.

2.2.3 The randomness model

A key aspect of decision forests is the fact that its component trees are all randomly different from one another. This leads to de-correlation between the individual tree predictions and, in turn, to improved generalization. Forest randomness also helps achieve high robustness with respect to noisy data.

Randomness is injected into the trees during the training phase. Two of the most popular ways of doing so are:

- random training data set sampling [11] (*e.g.* bagging), and
- randomized node optimization [46].

These two techniques are not mutually exclusive and could be used together. However, in this paper we focus on the second alternative which: i) enables us to train each tree on the totality of training data, and ii) yields margin-maximization properties (details in chapter 3). On the other hand, bagging yields greater training efficiency.

Randomized node optimization. If $\mathcal{T}$ is the entire set of all possible parameters $\boldsymbol{\theta}$ then when training the j^{th} node we only make available a small subset $\mathcal{T}_j \subset \mathcal{T}$ of such values. Thus under the randomness

model training a tree is achieved by optimizing each split node j by

$$\boldsymbol{\theta}_j^* = \arg\max_{\boldsymbol{\theta}_j \in \mathcal{T}_j} I_j. \tag{2.6}$$

The amount of randomness is controlled by the ratio $|\mathcal{T}_j|/|\mathcal{T}|$. Note that in some cases we may have $|\mathcal{T}| = \infty$. At this point it is convenient to introduce a parameter $\rho = |\mathcal{T}_j|$. The parameter $\rho = 1, \ldots, |\mathcal{T}|$ controls the degree of randomness in a forest and (usually) its value is fixed for all nodes in all trees. For $\rho = |\mathcal{T}|$ all trees in the forests are identical to one another and there is no randomness in the system (fig. 2.7a). Vice-versa, for $\rho = 1$ we get maximum randomness and uncorrelated trees (fig. 2.7b).

2.2.4 The leaf prediction model

During training, information that is useful for prediction in testing will be learned for all leaf nodes. In the case of classification each leaf may store the empirical distribution over the classes associated to the subset of training data that has reached that leaf. The probabilistic leaf predictor model for the t^{th} tree is then

$$p_t(c|\mathbf{v}) \tag{2.7}$$

with $c \in \{c_k\}$ indexing the class (see fig. 2.5c). In regression instead, the output is a continuous variable and thus the leaf predictor model may be a posterior over the desired continuous variable. In more conventional decision trees [12] the leaf output was not probabilistic, but rather a point estimate, *e.g.* $c^* = \arg\max_c p_t(c|\mathbf{v})$. Forest-based probabilistic regression was introduced in [24] and it will be discussed in detail in chapter 4.

2.2.5 The ensemble model

In a forest with T trees we have $t \in \{1, \cdots, T\}$. All trees are trained independently (and possibly in parallel). During testing, each test point $\mathbf{v}$ is simultaneously pushed through all trees (starting at the root) until it reaches the corresponding leaves. Tree testing can also often be done in parallel, thus achieving high computational efficiency on modern

parallel CPU or GPU hardware (see [80] for GPU-based classification). Combining all tree predictions into a single forest prediction may be done by a simple averaging operation [11]. For instance, in classification

$$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} p_t(c|\mathbf{v}). \tag{2.8}$$

Alternatively one could also multiply the tree output together (though the trees are not statistically independent)

$$p(c|\mathbf{v}) = \frac{1}{Z}\prod_{t=1}^{T} p_t(c|\mathbf{v}) \tag{2.9}$$

with the partition function Z ensuring probabilistic normalization

Figure 2.8 illustrates tree output fusion for a regression example. Imagine that we have trained a regression forest with $T = 4$ trees to predict a "dependent" continuous output y. [3] For a test data point $\mathbf{v}$ we get the corresponding tree posteriors $p_t(y|\mathbf{v})$, with $t = \{1, \cdots, 4\}$. As illustrated some trees produce peakier (more confident) predictions than others. Both the averaging and the product operations produce combined distributions (shown in black) which are heavily influenced by the most confident, most informative trees. Therefore, such simple operations have the effect of selecting (softly) the more confident trees out of the forest. This selection is carried out on a leaf-by-leaf level and the more confident trees may be different for different leaves. Averaging many tree posteriors also has the advantage of reducing the effect of possibly noisy tree contributions. In general, the product based ensemble model may be less robust to noise. Alternative ensemble models are possible, where for instance one may choose to select individual trees in a hard way.

2.2.6 Stopping criteria

Other important choices are to do with when to stop growing individual tree branches. For instance, it is common to stop the tree when a maximum number of levels D has been reached. Alternatively, one

[3] Probabilistic regression forests will be described in detail in chapter 4.

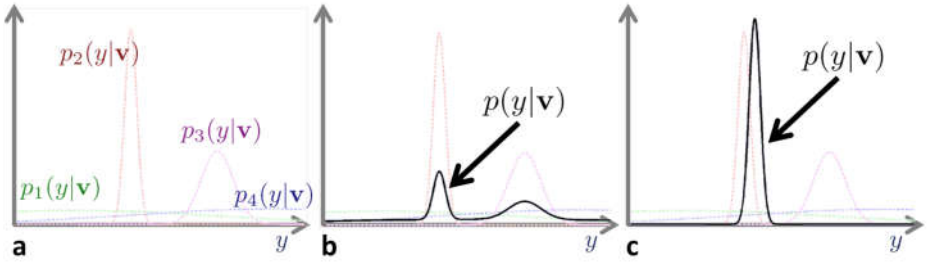

Fig. 2.8: **Ensemble model. (a)** The posteriors of four different trees (shown with different colours). Some correspond to higher confidence than others. **(b)** An ensemble posterior $p(y|\mathbf{v})$ obtained by averaging all tree posteriors. **(c)** The ensemble posterior $p(y|\mathbf{v})$ obtained as product of all tree posteriors. Both in (b) and (c) the ensemble output is influenced more by the more informative trees.

can impose a minimum information gain. Tree growing may also be stopped when a node contains less that a defined number of training points. Avoiding growing full trees has repeatedly been demonstrated to have positive effects in terms of generalization. In this work we avoid further post-hoc operations such as tree pruning [42] to keep the training process as simple as possible.

2.2.7 Summary of key model parameters

In summary, the parameters that most influence the behaviour of a decision forest are:

- The forest size T;
- The maximum allowed tree depth D;
- The amount of randomness (controlled by ρ) and its type;
- The choice of weak learner model;
- The training objective function;
- The choice of features in practical applications.

Those choices directly affect the forest predictive accuracy, the *accuracy of its confidence*, its generalization and its computational efficiency.

For instance, several papers have pointed out how the testing accuracy increases monotonically with the forest size T [24, 83, 102]. It is also known that very deep trees can lead to overfitting, although using very large amounts of training data mitigates this problem [82]. In his seminal work Breiman [11] has also shown the importance of randomness and its effect on tree correlation. Chapter 3 will show how the choice of randomness model directly influences a classification forest's generalization. A less studied issue is how the weak learners influence the forest's accuracy and its estimated uncertainty. To this end, the next chapters will show the effect of ρ on the forest behaviour with some simple toy examples and compare the results with existing alternatives.

Now we have defined our generic decision forest model. Next we discuss its specializations for the different tasks of interest. The explanations will be accompanied by a number of synthetic examples in the hope of increasing clarity of exposition and helping understand the forests' general properties. Real-world applications will also be presented and discussed.

3

Classification forests

This chapter discusses the most common use of decision forests, *i.e.* classification. The goal here is to automatically associate an input data point $\mathbf{v}$ with a discrete class $c \in \{c_k\}$. Classification forests enjoy a number of useful properties:

- they naturally handle problems with more than two classes;
- they provide a probabilistic output;
- they generalize well to previously unseen data;
- they are efficient thanks to their parallelism and reduced set of tests per data point.

In addition to these known properties this chapter also shows that:

- under certain conditions classification forests exhibit margin-maximizing behaviour, and
- the quality of the posterior can be controlled via the choice of the specific weak learner.

We begin with an overview of general classification methods and then show how to specialize the generic forest model presented in the previous chapter for the classification task.

3.1 Classification algorithms in the literature

One of the most widely used classifiers is the support vector machine (SVM) [97] whose popularity is due to the fact that in binary classification problems (only two target classes) it guarantees maximum-margin separation. In turn, this property yields good generalization with relatively little training data.

Another popular technique is boosting [32] which builds strong classifiers as linear combination of many weak classifiers. A boosted classifier is trained iteratively, where at each iteration the training examples for which the classifier works less well are "boosted" by increasing their associated training weight. Cascaded boosting was used in [98] for efficient face detection and localization in images, a task nowadays handled even by entry-level digital cameras and webcams.

Despite the success of SVMs and boosting, these techniques do not extend naturally to multiple class problems [20, 94]. In principle, classification trees and forests work, unmodified with any number of classes. For instance, they have been tested on ~ 20 classes in [83] and ~ 30 classes in [82].

Abundant literature has shown the advantage of fusing together multiple simple learners of different types [87, 95, 102, 105]. Classification forests represent a simple, yet effective way of combining randomly trained classification trees. A thorough comparison of forests with respect to other binary classification algorithms has been presented in [15]. In average, classification forests have shown good generalization, even in problems with high dimensionality. Classification forests have also been employed successfully in a number of practical applications [23, 54, 74, 83, 100].

3.2 Specializing the decision forest model for classification

This section specializes the generic model introduced in chapter 2 for use in classification.

Problem statement. The classification task may be summarized as follows:

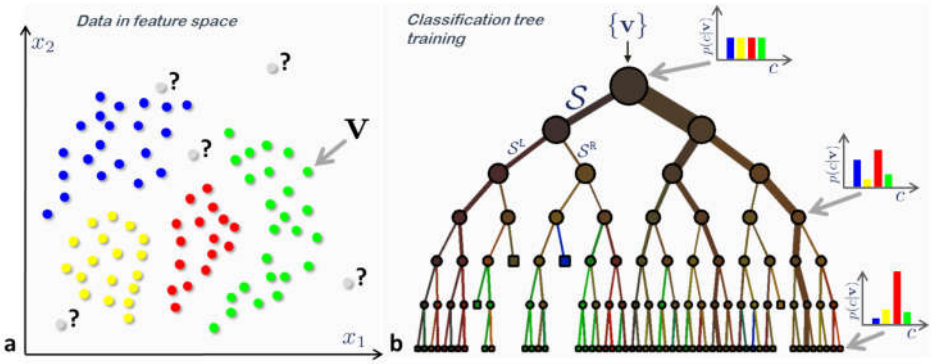

Fig. 3.1: **Classification: training data and tree training. (a)** Input data points. The ground-truth label of training points is denoted with different colours. Grey circles indicate unlabelled, previously unseen test data. **(b)** A binary classification tree. During training a set of labelled training points $\{\mathbf{v}\}$ is used to optimize the parameters of the tree. In a classification tree the entropy of the class distributions associated with different nodes decreases (the confidence increases) when going from the root towards the leaves.

Given a labelled training set learn a general mapping which associates previously unseen test data with their correct classes.

The need for a general rule that can be applied to "not-yet-available" test data is typical of *inductive* tasks.[1] In classification the desired output is of discrete, categorical, unordered type. Consequently, so is the nature of the training labels. In fig. 3.1a data points are denoted with circles, with different colours indicating different training labels. Testing points (not available during training) are indicated in grey.

More formally, during testing we are given an input test data $\mathbf{v}$ and we wish to infer a class label c such that $c \in \mathcal{C}$, with $\mathcal{C} = \{c_k\}$. More generally we wish to compute the whole distribution $p(c|\mathbf{v})$. As

[1] As opposed to *transductive* tasks. The distinction will become clearer later.

usual the input is represented as a multi-dimensional vector of feature responses $\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$. Training happens by optimizing an energy over a training set $\mathcal{S}_0$ of data and associated ground-truth labels. Next we specify the precise nature of this energy.

The training objective function. Forest training happens by optimizing the parameters of the weak learner at each split node j via:

$$\boldsymbol{\theta}_j^* = \arg\max_{\boldsymbol{\theta}_j \in \mathcal{T}_j} I_j. \tag{3.1}$$

For classification the objective function I_j takes the form of a classical information gain defined for discrete distributions:

$$I_j = H(\mathcal{S}_j) - \sum_{i \in \{L,R\}} \frac{|\mathcal{S}_j^i|}{|\mathcal{S}_j|} H(\mathcal{S}_j^i)$$

with i indexing the two child nodes. The entropy for a generic set $\mathcal{S}$ of training points is defined as:

$$H(\mathcal{S}) = -\sum_{c \in \mathcal{C}} p(c) \log p(c)$$

where $p(c)$ is calculated as normalized empirical histogram of labels corresponding to the training points in $\mathcal{S}$. As illustrated in fig. 3.1b training a classification tree by maximizing the information gain has the tendency to produce trees where the entropy of the class distributions associated with the nodes decreases (the prediction confidence increases) when going from the root towards the leaves. In turn, this yields increasing confidence of prediction.

Although the information gain is a very popular choice of objective function it is not the only one. However, as shown in later chapters, using an information-gain-like objective function aids unification of diverse tasks under the same forest framework.

Randomness. In (3.1) randomness is injected via randomized node optimization, with as before $\rho = |\mathcal{T}_j|$ indicating the amount of randomness. For instance, before starting training node j we can randomly

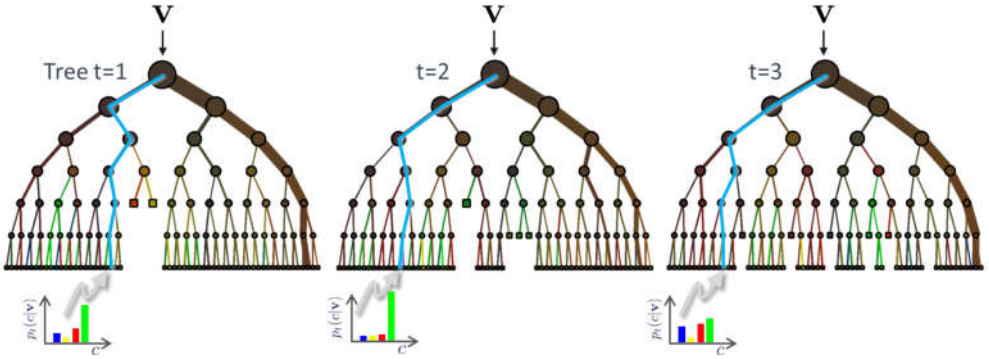

Fig. 3.2: **Classification forest testing.** During testing the same unlabelled test input data $\mathbf{v}$ is pushed through each component tree. At each internal node a test is applied and the data point sent to the appropriate child. The process is repeated until a leaf is reached. At the leaf the stored posterior $p_t(c|\mathbf{v})$ is read off. The forest class posterior $p(c|\mathbf{v})$ is simply the average of all tree posteriors.

sample $\rho = 1000$ parameter values out of possibly billions or even infinite possibilities. It is important to point out that it is not necessary to have the entire set $\mathcal{T}$ pre-computed and stored. We can generate each random subset $\mathcal{T}_j$ as needed before starting training the corresponding node.

The leaf and ensemble prediction models. Classification forests produce probabilistic output as they return not just a single class point prediction but an entire class distribution. In fact, during testing, each tree leaf yields the posterior $p_t(c|\mathbf{v})$ and the forest output is simply:

$$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v}).$$

This is illustrated with a small, three-tree forest in fig. 3.2.

The choices made above in terms of the form of the objective function and that of the prediction model characterize a classification forest. In later chapter we will discuss how different choices lead to different

models. Next, we discuss the effect of model parameters and important properties of classification forests.

3.3 Effect of model parameters

This section studies the effect of the forest model parameters on classification accuracy and generalization. We use many illustrative, synthetic examples designed to bring to life different properties. Finally, section 3.6 demonstrates such properties on a real-world, commercial application.

3.3.1 The effect of the forest size on generalization

Figure 3.3 shows a first synthetic example. Training points belonging to two different classes (shown in yellow and red) are randomly drawn from two well separated Gaussian distributions (fig. 3.3a). The points are represented as 2-vectors, where each dimension represents a different feature.

A forest of shallow trees ($D = 2$) and varying size T is trained on those points. In this example simple axis-aligned weak learners are used. In such degenerate trees there is only one split node, the root itself (fig. 3.3b). The trees are all randomly different from one another and each defines a slightly different partition of the data. In this simple (linearly separable) example each tree defines a "perfect" partition since the training data is separated perfectly. However, the partitions themselves are still randomly different from one another.

Figure 3.3c shows the testing classification posteriors evaluated for all non-training points across a square portion of the feature space (the white testing pixels in fig. 3.3a). In this visualization the colour associated with each test point is a linear combination of the colours (red and yellow) corresponding to the two classes; where the mixing weights are proportional to the posterior itself. Thus, intermediate, mixed colours (orange in this case) correspond to regions of high uncertainty and low predictive confidence.

We observe that each single tree produces *over-confident* predictions (sharp probabilities in fig. 3.3c$_1$). This is undesirable. In fact, intuitively one would expect the confidence of classification to be reduced for test

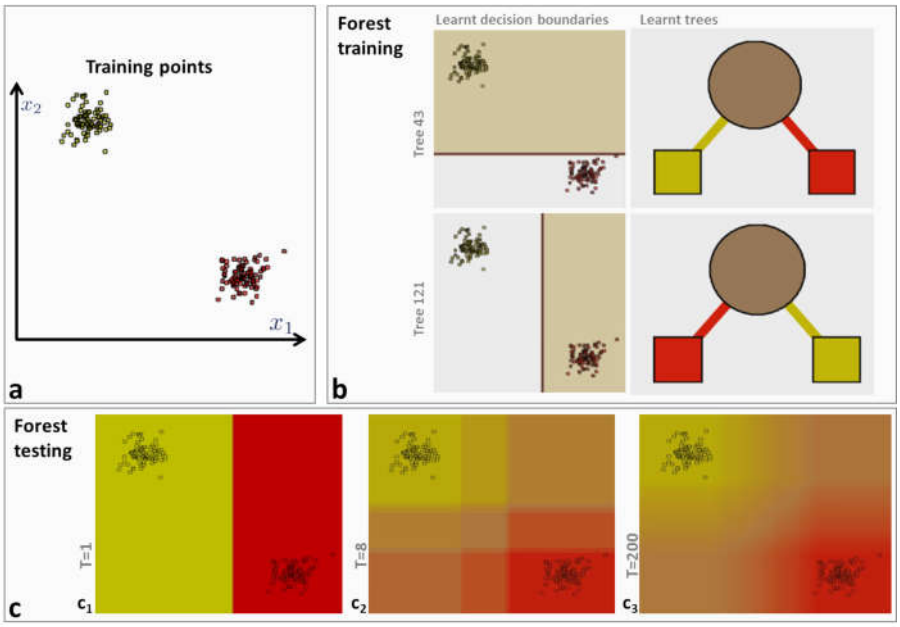

Fig. 3.3: **A first classification forest and the effect of forest size T. (a)** Training points belonging to two classes. **(b)** Different training trees produce different partitions and thus different leaf predictors. The colour of tree nodes and edges indicates the class probability of training points going through them. **(c)** In testing, increasing the forest size T produces smoother class posteriors. All experiments were run with $D = 2$ and axis-aligned weak learners. See text for details.

data which is "different" than the training data. The larger the difference, the larger the uncertainty. Thanks to all trees being different from one another, increasing the forest size from $T = 1$ to $T = 200$ produces much smoother posteriors (fig. 3.3c$_3$). Now we observe higher confidence near the training points and lower confidence away from training regions of space; an indication of good generalization behaviour.

For few trees (*e.g.* $T = 8$) the forest posterior shows clear box-like artifacts. This is due to the use of an axis-aligned weak learner model. Such artifacts yield low quality confidence estimates (especially

when extrapolating away from training regions) and ultimately imperfect generalization. Therefore, in the remainder of this paper we will always keep an eye on the *accuracy of the uncertainty* as this is key for inductive generalization away from (possibly little) training data. The relationship between quality of uncertainty and maximum-margin classification will be studied in section 3.4.

3.3.2 Multiple classes and training noise

One major advantage of decision forests over *e.g.* support vector machines and boosting is that the same classification model can handle both binary and multi-class problems. This is illustrated in fig. 3.4 with both two- and four-class examples, and different levels of noise in the training data.

The top row of the figure shows the input training points (two classes in fig. 3.4a and four classes in figs. 3.4b,c). The middle row shows corresponding testing class posteriors. the bottom row shows entropies associated to each pixel. Note how points in between spiral arms or farther away from training points are (correctly) associated with larger uncertainty (orange pixels in fig. 3.4a' and grey-ish ones in figs. 3.4b',c').

In this case we have employed a richer *conic section* weak learner model which removes the blocky artifacts observed in the previous example and yields smoother posteriors. Notice for instance in fig. 3.4b' how the curve separating the red and the green spiral arms is nicely continued away from training points (with increasing uncertainty).

As expected, if the noise in the position of training points increases (*cf* fig. 3.4b and 3.4 c) then training points for different classes are more intermingled with one another. This yields a larger overall uncertainty in the testing posterior (captured by less saturated colours in fig. 3.4c'). Next we delve further into the issue of training noise and mixed or "sloppy" training data.

3.3.3 "Sloppy" labels and the effect of the tree depth

The experiment in fig. 3.5 illustrates the behaviour of classification forests on a four-class training set where there is both mixing of la-

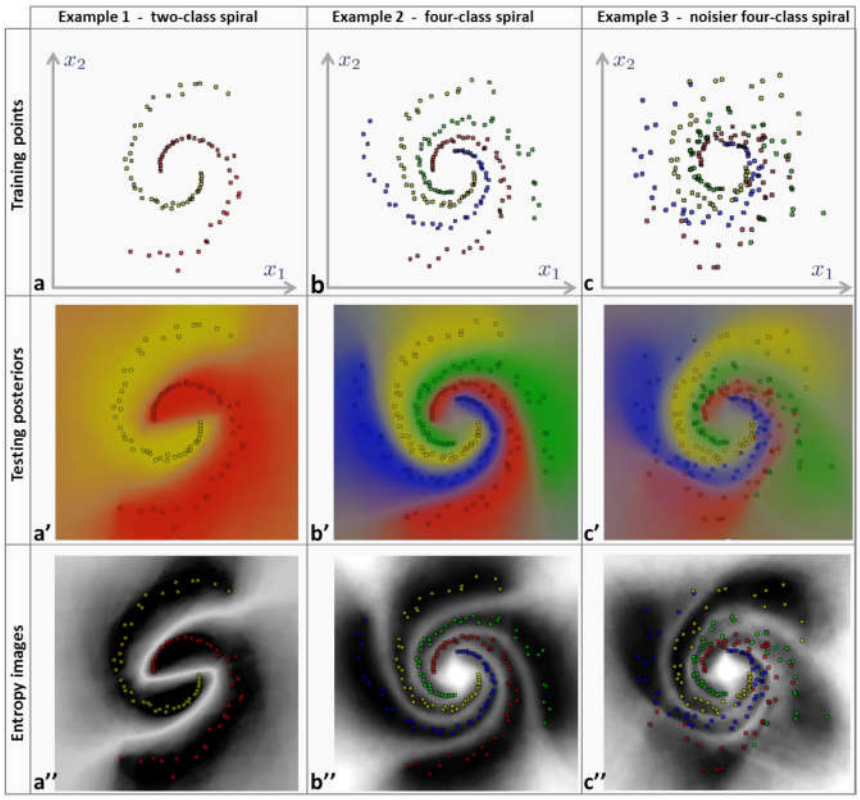

Fig. 3.4: **The effect of multiple classes and noise in training data. (a,b,c)** Training points for three different experiments: 2-class spiral, 4-class spiral and another 4-class spiral with noisier point positions, respectively. **(a',b',c')** Corresponding testing posteriors. **(a",b",c")** Corresponding entropy images (brighter for larger entropy). The classification forest can handle both binary as well as multi-class problems. With larger training noise the classification uncertainty increases (less saturated colours in c' and less sharp entropy in c"). All experiments in this figure were run with $T = 200$, $D = 6$, and a conic-section weak-learner model.

bels (in feature space) and large gaps. Here three different forests have been trained with the same number of trees $T = 200$ and varying maximum depth D. We observe that as the tree depth increases the overall prediction confidence also increases. Furthermore, in large gaps (*e.g.* between red and blue regions), the optimal separating surface tends to

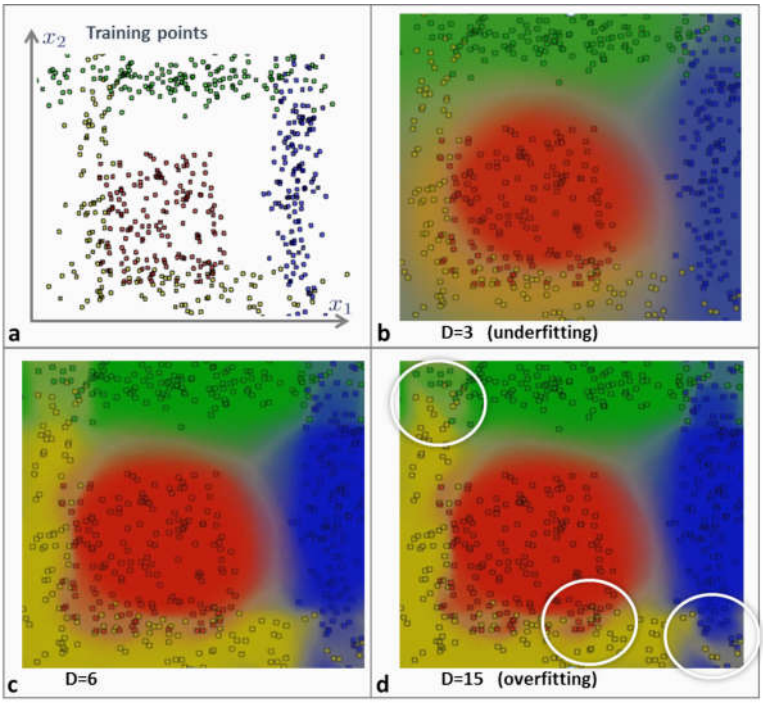

Fig. 3.5: **The effect of tree depth.** A four-class problem with both mixing of training labels and large gaps. **(a)** Training points. **(b,c,d)** Testing posteriors for different tree depths. All experiments were run with $T = 200$ and a conic weak-learner model. The tree depth is a crucial parameter in avoiding under- or over-fitting.

be placed roughly in the middle of the gap.[2]

Finally, we notice that a large value of D ($D = 15$ in the example) tends to produce *overfitting*, *i.e.* the posterior tends to split off isolated clusters of noisy training data (denoted with white circles in the figure). In fact, the maximum tree depth parameter D controls the amount of overfitting. By the same token, too shallow trees produce washed-out, low-confidence posteriors. Thus, while using multiple

[2] This effect will be analyzed further in the next section.

trees alleviates the overfitting problem of individual trees, it does not cure it completely. In practice one has to be very careful to select the most appropriate value of D as its optimal value is a function of the problem complexity.

3.3.4 The effect of the weak learner

Another important issue that has perhaps been a little overlooked in the literature is the effect of a particular choice of weak learner model on the forest behaviour.

Figure 3.6 illustrates this point. We are given a single set of training points arranged in four spirals, one for each class. Six different forests have been trained on the same training data, for 2 different values of tree depth and 3 different weak learners. The 2×3 arrangement of images shows the output test posterior for varying D (in different rows) and varying weak learner model (in different columns). All experiments are conducted with a very large number of trees ($T = 400$) to remove the effect of forest size and reach close to the maximum possible smoothness under the model.

This experiment confirms again that increasing D increases the confidence of the output (for fixed weak learner). This is illustrated by the more intense colours going from top row to the bottom row. Furthermore we observe that the choice of weak learner model has a large impact on the test posterior and the quality of its confidence. The axis-aligned model may still separate the training data well, but produces large blocky artifacts in the test regions. This tends to indicate bad generalization. The oriented line model [43, 58] is a clear improvement, and better still is the non-linear model as it extrapolates the shape of the spiral arms in a more naturally curved manner.

On the flip side, of course, we should also consider the fact that axis-aligned tests are extremely efficient to compute. So the choice of the specific weak learner has to be based on considerations of both accuracy and efficiency and depends on the specific application at hand. Next we study the effect of randomness by running exactly the same experiment but with a much larger amount of training randomness.

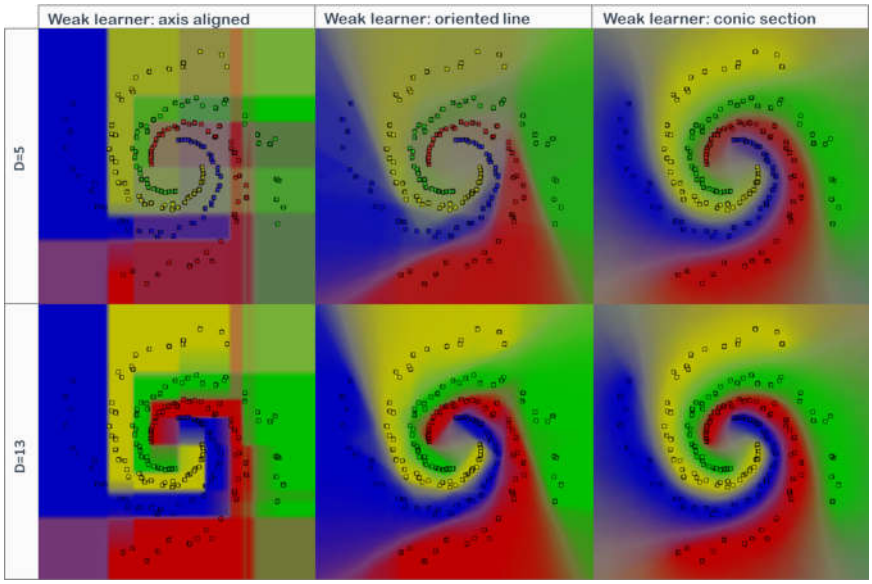

Fig. 3.6: **The effect of weak learner model.** The same set of 4-class training data is used to train 6 different forests, for 2 different values of D and 3 different weak learners. For fixed weak learner deeper trees produce larger confidence. For constant D non-linear weak learners produce the best results. In fact, an axis-aligned weak learner model produces blocky artifacts while the curvilinear model tends to extrapolate the shape of the spiral arms in a more natural way. Training has been achieved with $\rho = 500$ for all split nodes. The forest size is kept fixed at $T = 400$.

3.3.5 The effect of randomness

Figure 3.7 shows the same experiment as in fig. 3.6 with the only difference that now $\rho = 5$ as opposed to $\rho = 500$. Thus, much fewer parameter values were made available to each node during training. This increases the randomness of each tree and reduces their correlation.

Larger randomness helps reduce a little the blocky artifacts of the axis-aligned weak-learner as it produces more rounded decision boundaries (first column in fig. 3.7). Furthermore, larger randomness yields a

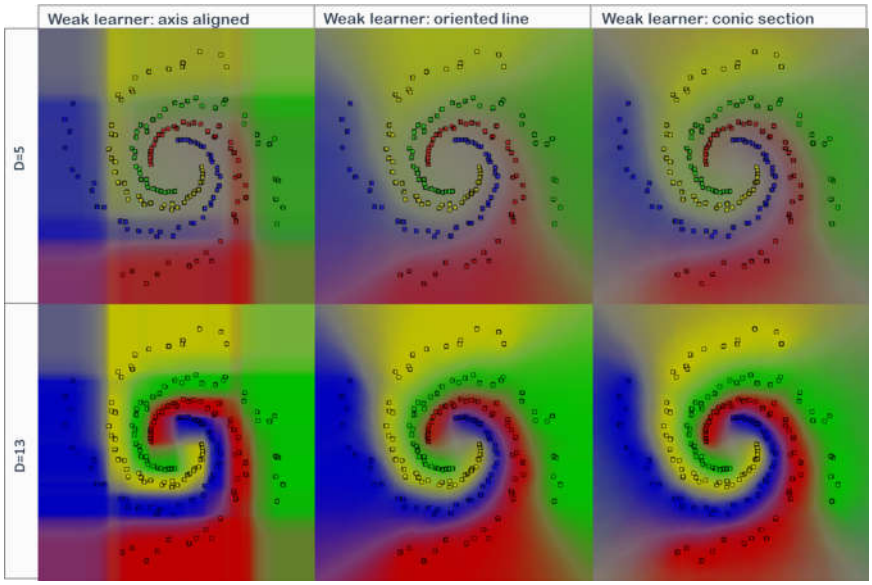

Fig. 3.7: **The effect of randomness.** The same set of 4-class training data is used to train 6 different forests, for 2 different values of D and 3 different weak learners. This experiment is identical to that in fig. 3.6 except that we have used much more training randomness. In fact $\rho = 5$ for all split nodes. The forest size is kept fixed at $T = 400$. More randomness reduces the artifacts of the axis-aligned weak learner a little, as well as reducing overall prediction confidence too. See text for details.

much lower overall confidence, especially noticeable in shallower trees (washed out colours in the top row).

A disadvantage of the more complex weak learners is that they are associated to a larger parameters space. Thus finding discriminative sets of parameter values may be time consuming. However, in this toy example the more complex conic section learner model works well for deeper trees ($D = 13$) even for small values of ρ (large randomness). The results reported here are only indicative. In fact, which specific weak learner to use depends on considerations of efficiency as well as accuracy and it is application dependent. Many more examples, ani-

mations and demo videos are available at [1].

Next, we move on to show further properties of classification forests. Specifically, we demonstrate how under certain conditions forests exhibit margin-maximizing capabilities.

3.4 Maximum-margin properties

The hallmark of support vector machines is their ability to separate data belonging to different classes via a margin-maximizing surface. This, in turn, yields good generalization even with relatively little training data. This section shows how this important property is replicated in random classification forests and under which conditions. Margin maximizing properties of random forests were discussed in [52]. Here we show a different, simpler formulation, analyze the conditions that lead to margin maximization, and discuss how this property is affected by different choices of model parameters.

Imagine we are given a linearly separable 2-class training data set such as that shown in fig. 3.8a. For simplicity here we assume $d = 2$ (only two features describe each data point), an axis-aligned weak learner model and $D = 2$ (trees are simple binary stumps). As usual randomness is injected via randomized node optimization (section 2.2.3).

When training the root node of the first tree, if we use enough candidate features/parameters (*i.e.* $|\mathcal{T}_0|$ is large) the selected separating line tends to be placed somewhere within the gap (see fig. 3.8a) so as to separate the training data perfectly (maximum information gain). Any position within the gap is associated with exactly the same, maximum information gain. Thus, a collection of randomly trained trees produces a set of separating lines randomly placed within the gap (an effect already observed in fig. 3.3b).

If the candidate separating lines are sampled from a uniform distribution (as is usually the case) then this would yield forest class posteriors that vary within the gap as a linear ramp, as shown in fig. 3.8b,c. If we are interested in a hard separation then the optimal separating surface (assuming equal loss) is such that the posteriors for the two classes are identical. This corresponds to a line placed right in the mid-

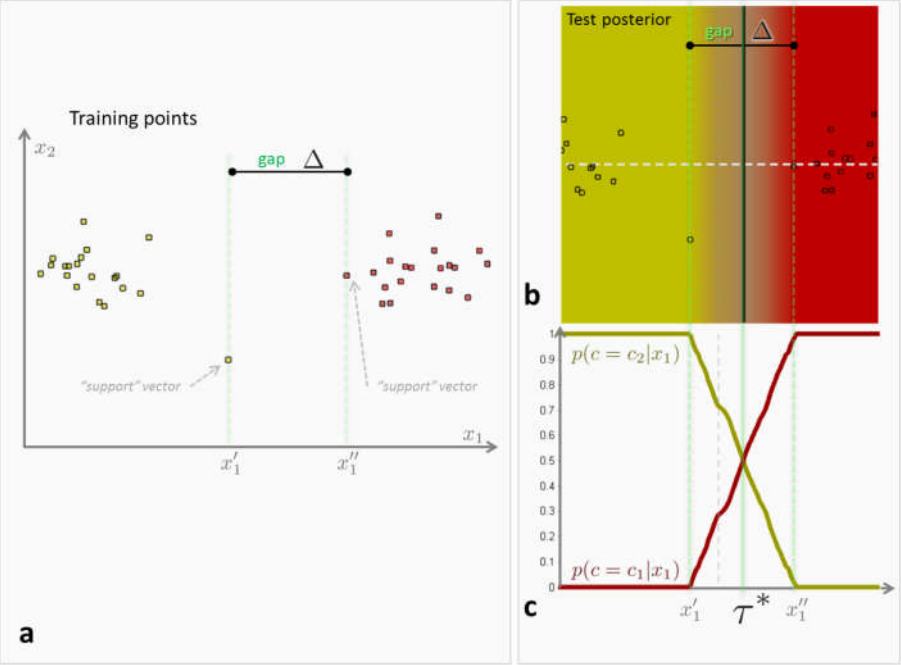

Fig. 3.8: **Forest's maximum-margin properties. (a)** Input 2-class training points. They are separated by a gap of dimension Δ. **(b)** Forest posterior. Note that all of the uncertainty band resides within the gap. **(c)** Cross-sections of class posteriors along the horizontal, white dashed line in (b). Within the gap the class posteriors are linear functions of x_1. Since they have to sum to 1 they meet right in the middle of the gap. In these experiments we use $\rho = 500, D = 2, T = 500$ and axis aligned weak learners.

dle of the gap, *i.e.* the maximum-margin solution. Next, we describe the same concepts more formally.

We are given the two-class training points in fig. 3.8a. In this simple example the training data is not only linearly separable, but it is perfectly separable via vertical stumps on x_1. So we constrain our weak learners to be vertical lines only, *i.e.*

$$h(\mathbf{v}, \boldsymbol{\theta}_j) = [\phi(\mathbf{v}) > \tau] \quad \text{with} \quad \phi(\mathbf{v}) = x_1.$$

Under these conditions we can define the *gap* Δ as $\Delta = x_1'' - x_1'$, with x_1' and x_1'' corresponding to the first feature of the two "support vectors"[3], *i.e.* the yellow point with largest x_1 and the red point with smallest x_1. For a fixed x_2 the classification forest produces the posterior $p(c|x_1)$ for the two classes c_1 and c_2. The optimal separating line (vertical) is at position τ^* such that

$$\tau^* = \arg\min_{\tau} |p(c = c_1|x_1 = \tau) - p(c = c_2|x_1 = \tau)|.$$

We make the additional assumption that when training a node its available test parameters (in this case just τ) are sampled from a uniform distribution, then the forest posteriors behave linearly within the gap region, *i.e.*

$$\lim_{\rho \to |\mathcal{T}|, T \to \infty} p(c = c_1|x_1) = \frac{x_1 - x_1'}{\Delta} \quad \forall x_1 \in [x_1', x_1''].$$

(see fig. 3.8b,c). Consequently, since $\sum_{c \in \{c_1, c_2\}} p(c|x_1) = 1$ we have

$$\lim_{\rho \to |\mathcal{T}|, T \to \infty} \tau^* = x_1' + \Delta/2.$$

which shows that the optimal separation is placed right in the middle of the gap. This demonstrates the forest's margin-maximization properties for this simple example.

Note that each individual tree is *not* guaranteed to produce maximum-margin separation; it is instead the combination of multiple trees that at the limit $T \to \infty$ produces the desired max-margin behaviour. In practice it suffices to have T and ρ "large enough". Furthermore, as observed earlier, for perfectly separable data each tree produces over-confident posteriors. Once again, their combination in a forest yields fully probabilistic and smooth posteriors (in contrast to SVM).

The simple mathematical derivation above provides us with some intuition on how model choices such as the amount of randomness or the type of weak learner affect the placement of the forest's separating surface. The next sections should clarify these concepts further.

[3] analogous to support vectors in SVM.

3.4.1 The effect of randomness on optimal separation

The experiment in fig. 3.8 has used a large value of ρ ($\rho \to |\mathcal{T}|$, little randomness, large tree correlation) to make sure that each tree decision boundary fell within the gap. When using more randomness (smaller ρ) then the individual trees are not guaranteed to split the data perfectly and thus they may yield a sub-optimal information gain. In turn, this yields a lower confidence in the posterior. Now, the locus of points where $p(c = c_1|x_1) = p(c = c_2|x_1)$ is no longer placed right in the middle of the gap. This is shown in the experiment in fig. 3.9 where we can observe that by increasing the randomness (decreasing ρ) we obtain smoother and more spread-out posteriors. The optimal separating surface is less sharply defined. The effect of individual training points is weaker as compared to the entire mass of training data; and in fact, it is no longer possible to identify individual support vectors. This may be advantageous in the presence of "sloppy" or inaccurate training data.

The role of the parameter ρ is very similar to that of "slack" variables in SVM [97]. In SVM the slack variables control the influence of individual support vectors versus the rest of training data. Appropriate values of slack variables yield higher robustness with respect to training noise.

3.4.2 Influence of the weak learner model

Figure 3.10 shows how more complex weak learners affects the shape and orientation of the optimal, hard classification surface (as well as the uncertain region, in orange). Once again, the position and orientation of the separation boundary is more or less sensitive to individual training points depending on the value of ρ. Little randomness produces a behaviour closer to that of support vector machines.

In classification forests, using linear weak-learners still produces (in general) globally non-linear classification (see the black curves in fig. 3.9c and fig. 3.10b). This is due to the fact that multiple simple linear split nodes are organized in a hierarchical fashion.

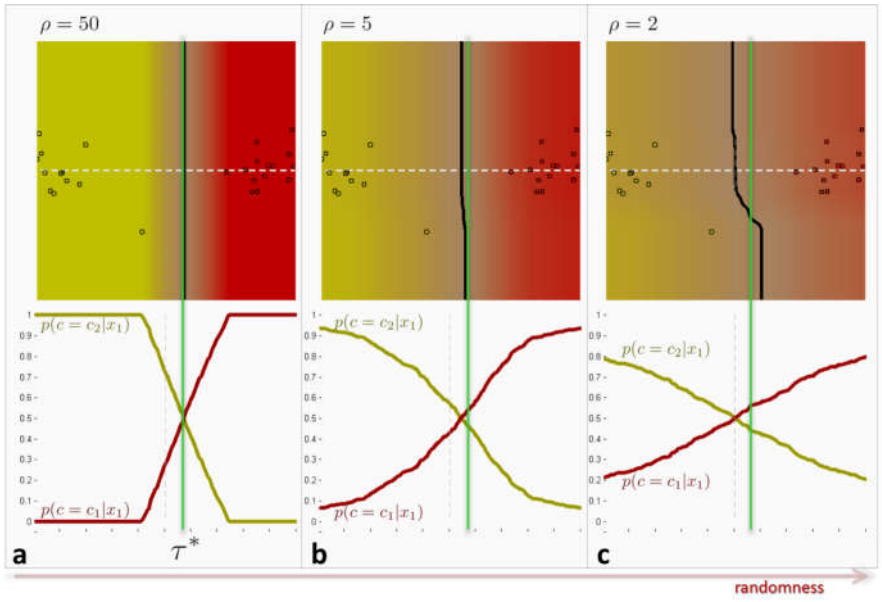

Fig. 3.9: **The effect of randomness on the forest margin. (a)** Forest posterior for $\rho = 50$ (small randomness). **(b)** Forest posterior for $\rho = 5$. **(c)** Forest posterior for $\rho = 2$ (highest randomness). These experiments have used $D = 2, T = 400$ and axis-aligned weak learners. The bottom row shows 1D posteriors computed along the white dashed line. Increasing randomness produces less well defined separating surfaces. The optimal separating surface, *i.e.* the loci of points where the class posteriors are equal (shown in black) moves towards the left of the margin-maximizing line (shown in green in all three experiments). As randomness increases individual training points have less influence on the separating surface.

3.4.3 Max-margin in multiple classes

Since classification forests can naturally apply to more than 2 classes how does this affect their maximum-margin properties? We illustrate this point with a multi-class synthetic example. In fig. 3.11a we have a linearly separable four-class training set. On it we have trained two

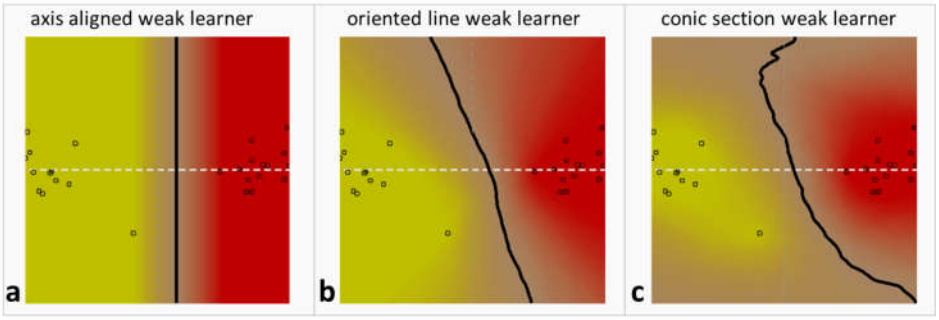

Fig. 3.10: **The effect of the weak learner on forest margin. (a)** Forest posterior for axis aligned weak learners. **(b)** Forest posterior for oriented line weak learners. **(c)** Forest posterior for conic section weak learners. In these experiments we have used $\rho = 50, D = 2, T = 500$. The choice of weak learner affects the optimal, hard separating surface (in black). Individual training points influence the surface differently depending on the amount of randomness in the forest.

forests with $|\mathcal{T}_j| = 50, D = 3, T = 400$. The only difference between the two forests is the fact that the first one uses an oriented line weak learner and the second a conic weak learner. Figures 3.11b,c show the corresponding testing posteriors. As usual grey pixels indicate regions of higher posterior entropy and lower confidence. They roughly delineate the four optimal hard classification regions. Note that in both cases their boundaries are roughly placed half-way between neighbouring classes. As in the 2-class case the influence of individual training points is dictated by the randomness parameter ρ.

Finally, when comparing fig. 3.11c and fig. 3.11b we notice that for conic learners the shape of the uncertainty region evolves in a curved fashion when moving away from training data.

3.4.4 The effect of the randomness model

This section shows a direct comparison between the randomized node optimization and the bagging model.

In bagging randomness is injected by randomly sampling different

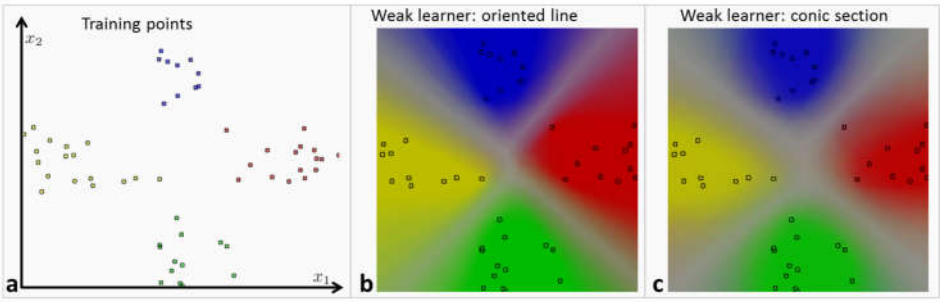

Fig. 3.11: **Forest's max-margin properties for multiple classes.** **(a)** Input four-class training points. **(b)** Forest posterior for oriented line weak learners. **(c)** Forest posterior for conic section weak learners. Regions of high entropy are shown as grey bands and correspond to loci of optimal separation. In these experiments we have used the following parameter settings $\rho = 50, D = 3, T = 400$.

subsets of training data. So, each tree sees a different training subset. Its node parameters are then fully optimized on this set. This means that specific "support vectors" may not be available in some of the trees. The posterior associated with those trees will then tend to move the optimal separating surface away from the maximum-margin one.

This is illustrated in fig. 3.12 where we have trained two forests with $\rho = 500, D = 2, T = 400$ and two different randomness models. The forest tested in fig. 3.12a uses randomized node optimization (RNO). The one in fig. 3.12b uses bagging (randomly selecting 50% training data with replacement) on exactly the same training data. In bagging, when training a node, there may be a whole range of values of a certain parameter which yield maximum information gain (*e.g.* the range $[\tau_1', \tau_1'']$ for the threshold τ_1). In such a case we could decide to always select one value out of the range (*e.g.* τ_1'). But this would probably be an unfair comparison. Thus we chose to randomly select a parameter value uniformly within that range. In effect here we are combining bagging and random node optimization together. The effect is shown in fig. 3.12b. In both cases we have used a large value of ρ to make sure that each tree achieves decent optimality in parameter selection.

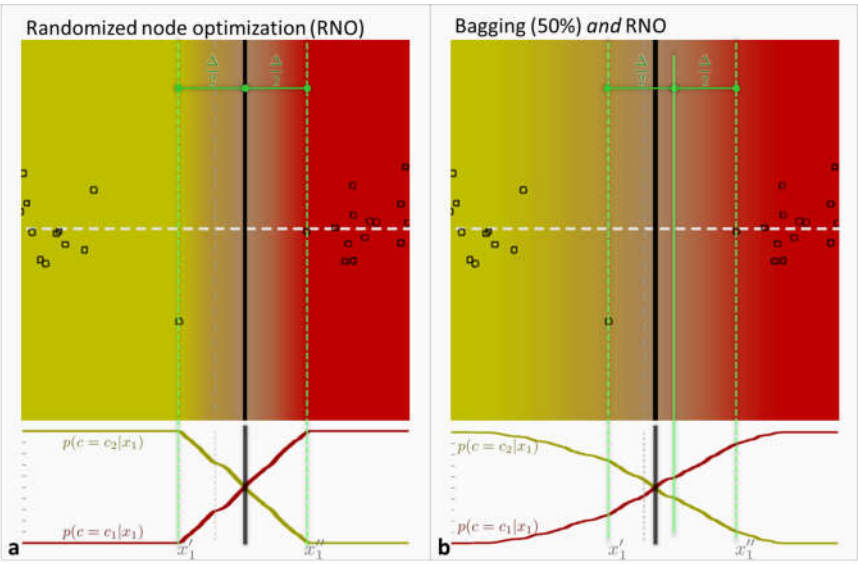

Fig. 3.12: **Max-margin: bagging v randomized node optimization. (a)** Posterior for forest trained with randomized node optimization. **(b)** Posterior for forest trained with bagging. In bagging, for each tree we use 50% random selection of training data with replacement. Loci of optimal separation are shown as black lines. In these experiments we use $\rho = 500, D = 2, T = 400$ and axis-aligned weak learners. Areas of high entropy have been shown strongly grey to highlight the separating surfaces.

We observe that the introduction of training set randomization leads to smoother posteriors whose optimal boundary (shown as a vertical black line) does *not* coincide with the maximum margin (green, solid line). Of course this behaviour is controlled by how much (training set) randomness we inject in the system. If we were to take all training data then we would reproduce a max-margin behaviour (but it would not be bagging). One advantage of bagging is increased training speed (due to reduced training set size). More experiments and comparisons are available in [1]. In the rest of the paper we use the RNO randomness model because it allows us to use all available training data and en-

ables us to control the maximum-margin behaviour simply, by means of changing ρ.

3.5 Comparisons with alternative algorithms

This section compares classification forests to existing state-of-the art algorithms.

3.5.1 Comparison with boosting

Figure 3.13 shows a comparison between classification forests and ModestBoost on two synthetic experiments.[4] Here, for both algorithm we use shallow tree stumps ($D = 2$) with axis-aligned split functions as this is what is conventionally used in boosting [99].

The first column presents the soft testing posteriors of the classification forest. The third column presents a visualization of the real-valued output of the boosted strong classifier, while the second column shows the more conventional, thresholded boosting output. The figure illustrates the superiority of the forest in terms of the additional uncertainty encoded in its posterior. Although both algorithms separate the training data perfectly, the boosting binary output is overly confident, thus potentially causing incorrect classification of previously unseen testing points. Using the real valued boosted output (third column) as a proxy for uncertainty does not seem to produce intuitively meaningful confidence results in these experiments. In fact, in some cases (experiment 1) there is not much difference between the thresholded and real-valued boosting outputs. This is due to the fact that all boosting's weak learners are identical to one another, in this case. The training procedure of the boosting algorithm tested here does not encourage diversity of weak learners in cases where the data can be easily separated by a single stump. Alternative boosting technique may produce better behaviour.

[4] Boosting results are obtained via the publically available Matlab toolbox in `http://graphics.cs.msu.ru/ru/science/research/machinelearning/adaboosttoolbox`

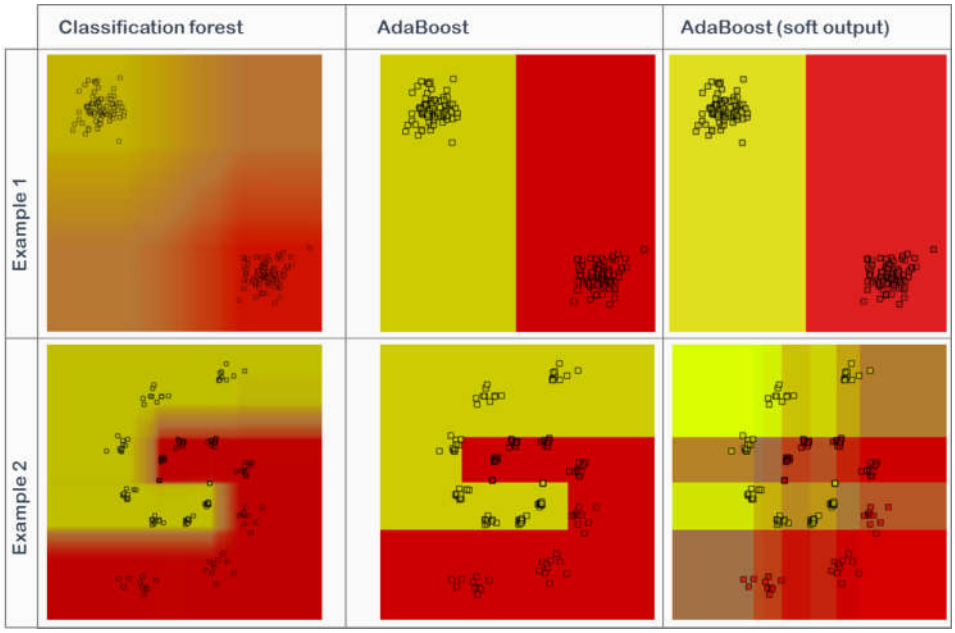

Fig. 3.13: **Comparison between classification forests and boosting** on two examples. Forests produce a smooth, probabilistic output. High uncertainty is associated with regions between different classes or away from training data. boosting produces a hard output. Interpreting the output of a boosted strong classifier as real valued does not seem to produce intuitively meaningful confidence. The forest parameters are: $D = 2$, $T = 200$, and we use axis-aligned weak learners. Boosting was also run with 200 axis-aligned stumps and the remaining parameters optimized to achieve best results.

3.5.2 Comparison with support vector machines

Figure 3.14 illustrates a comparison between classification forests and conventional support vector machines[5] on three different four-class training sets. In all examples the four classes are nicely separable

[5] SVM experiments are obtained via the publically available code in http://asi.insa-rouen.fr/enseignants/ arakotom/toolbox/index.html. For multi-class experiments we run one-v-all SVM.

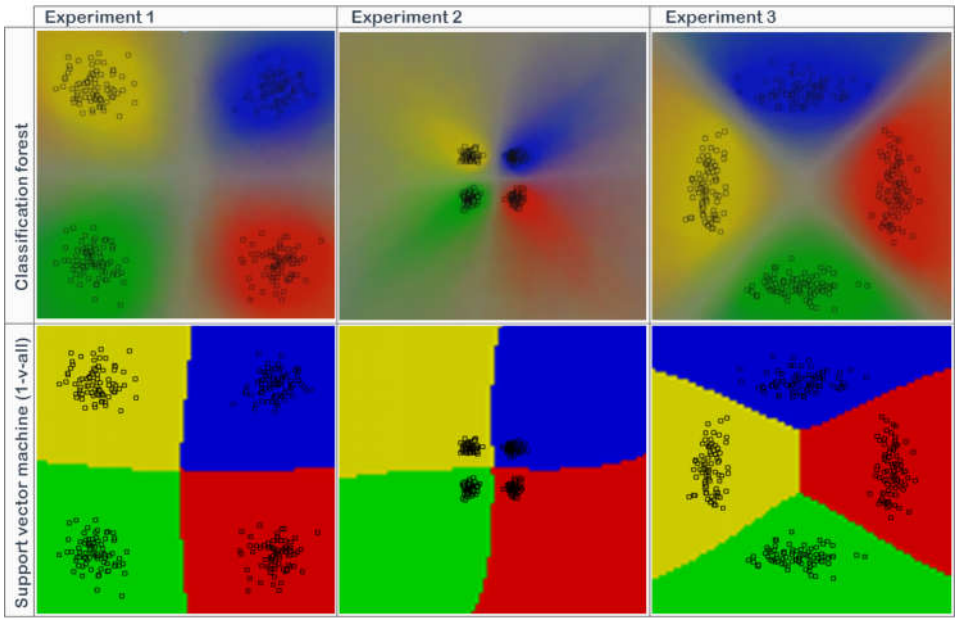

Fig. 3.14: **Comparison between classification forests and support vector machines.** All forest experiments were run with $D = 3$, $T = 200$ and conic weak learner. The SVM parameters were optimized to achieve best results.

and both forests and SVMs achieve good separation results. However, forests also produce uncertainty information. Probabilistic SVM counterparts such as the relevance vector machine [93] do produce confidence output but at the expense of further computation.

The role of good confidence estimation is particularly evident in fig. 3.14b where we can see how the uncertainty increases as we move away from the training data. The exact shape of the confidence region is dictated strongly by the choice of the weak learner model (conic section in this case), and a simple axis-aligned weak learner would produce inferior results. In contrast, the SVM classifier assigns a hard output class value to each pixel, with equal confidence.

Unlike forests, SVMs were born as two-class classifiers, although

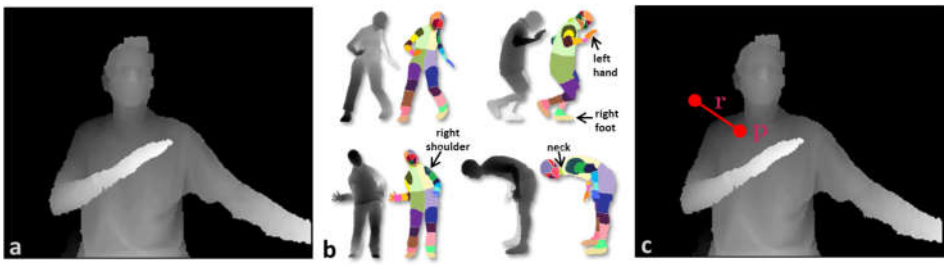

Fig. 3.15: **Classification forests in Microsoft Kinect for XBox 360. (a)** An input frame as acquired by the Kinect depth camera. **(b)** Synthetically generated ground-truth labeling of 31 different body parts [82]. **(c)** One of the many features of a "reference" point $\mathbf{p}$. Given $\mathbf{p}$ computing the feature amounts to looking up the depth at a "probe" position $\mathbf{p} + \mathbf{r}$ and comparing it with the depth of $\mathbf{p}$.

recently they have been adapted to work with multiple classes. Figure 3.14c shows how the sequentiality of the one-v-all SVM approach may lead to asymmetries which are not really justified by the training data.

3.6 Human body tracking in Microsoft Kinect for XBox 360

This section describes the application of classification forests for the real-time tracking of humans, as employed in the Microsoft Kinect gaming system [100]. Here we present a summary of the algorithm in [82] and show how the forest employed within is readily interpreted as an instantiation of our generic decision forest model.

Given a depth image such as the one shown in fig. 3.15a we wish to say which body part each pixel belongs to. This is a typical job for a classification forest. In this application there are thirtyone different body part classes: $c \in \{\texttt{left hand}, \texttt{right hand}, \texttt{head}, \texttt{l. shoulder}, \texttt{r. shoulder}, \cdots\}$. The unit of computation is a single pixel in position $\mathbf{p} \in \mathbb{R}^2$ and with associated feature vector $\mathbf{v}(\mathbf{p}) \in \mathbb{R}^d$.

During testing, given a pixel $\mathbf{p}$ in a previously unseen test image we

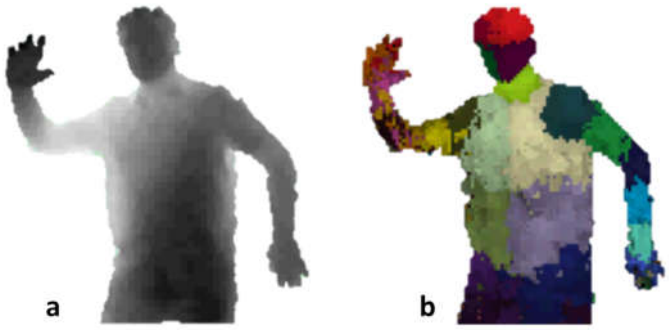

Fig. 3.16: **Classification forests in Kinect for XBox 360. (a)** An input depth frame with background removed. **(b)** The body part classification posterior. Different colours corresponding to different body parts, out of 31 different classes.

wish to estimate the posterior $p(c|\mathbf{v})$. Visual features are simple depth comparisons between pairs of pixel locations. So, for pixel $\mathbf{p}$ its feature vector $\mathbf{v} = (x_1, \ldots, x_i, \ldots, x_d) \in \mathbb{R}^d$ is a collection of depth differences:

$$x_i = J(\mathbf{p}) - J\left(\mathbf{p} + \frac{\mathbf{r}_i}{J(\mathbf{p})}\right) \tag{3.2}$$

where $J(.)$ denotes a pixel depth in mm (distance from camera plane). The 2D vector $\mathbf{r}_i$ denotes a displacement from the reference point $\mathbf{p}$ (see fig. 3.15c). Since for each pixel we can look around at an infinite number of possible displacements ($\forall\, \mathbf{r} \in \mathbb{R}^2$) we have $d = \infty$.

During training we are given a large number of pixel-wise labelled training image pairs as in fig 3.15b. Training happens by maximizing the information gain for discrete distributions (3.1). For a split node j its parameters are

$$\boldsymbol{\theta}_j = (\mathbf{r}_j, \tau_j)$$

with $\mathbf{r}_j$ a randomly chosen displacement. The quantity τ_j is a learned scalar threshold. If $d = \infty$ then also the whole set of possible split parameters has infinite cardinality, *i.e.* $|\mathcal{T}| = \infty$.

An axis-aligned weak learner model is used here with the node split

function as follows

$$h(\mathbf{v}, \boldsymbol{\theta}_j) = [\phi(\mathbf{v}, \mathbf{r}_j) > \tau_j] \,.$$

As usual, the selector function ϕ takes the entire feature vector $\mathbf{v}$ and returns the single feature response (3.2) corresponding to the chosen displacement $\mathbf{r}_j$. In practice, when training a split node j we first randomly generate a set of parameters $\mathcal{T}_j$ and then maximize the information gain by exhaustive search. Therefore we never need to compute the entire infinite set $\mathcal{T}$.

Now we have defined all model parameters for the specific application at hand. Some example results are shown in fig. 3.16; with many more shown in the original paper [82]. Now that we know how this application relates to the more abstract description of the classification forest model it would be interesting to see how the results change, *e.g.* when changing the weak learner model, or the amount of randomness *etc.* However, this investigation is beyond the scope of this paper.

Moving on from classification, the next chapter addresses a closely related problem, that of probabilistic, non-linear regression. Interestingly, regression forests have very recently been used for skeletal joint prediction in Kinect images [37].