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.

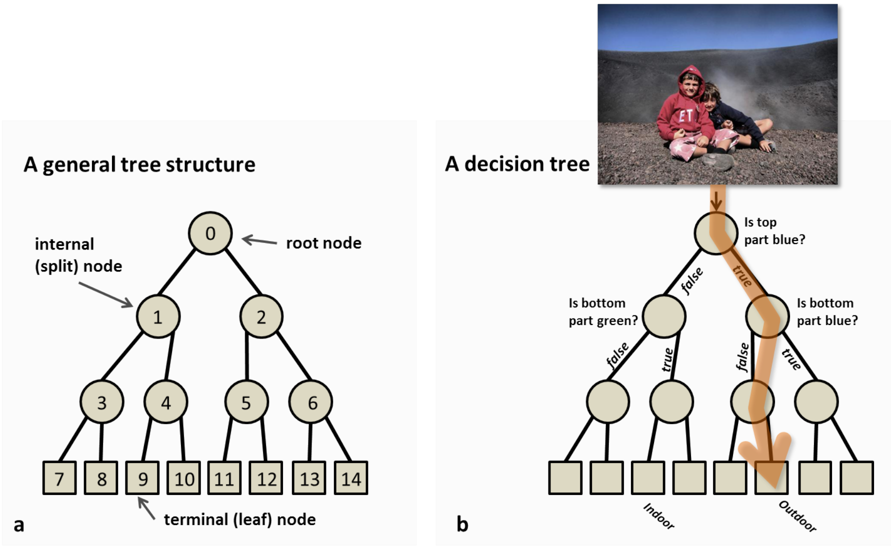


Fig. 2.1: **Decision tree. (a)** A tree is a set of nodes and edges organized in a hierarchical fashion. In contrast to a graph, in a tree there are no loops. Internal nodes are denoted with circles and terminal nodes with squares. **(b)** A decision tree is a tree where each split node stores a test 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,

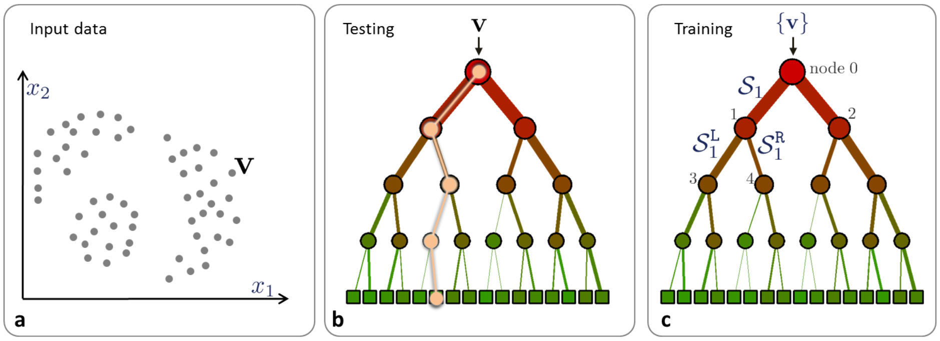


Fig. 2.2: **Basic notation. (a)** Input data is represented as a collection of points in the d-dimensional space defined by their feature responses (2D in this example). **(b)** A decision tree is a hierarchical structure of connected nodes. During testing, a split (internal) node applies a test to the 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

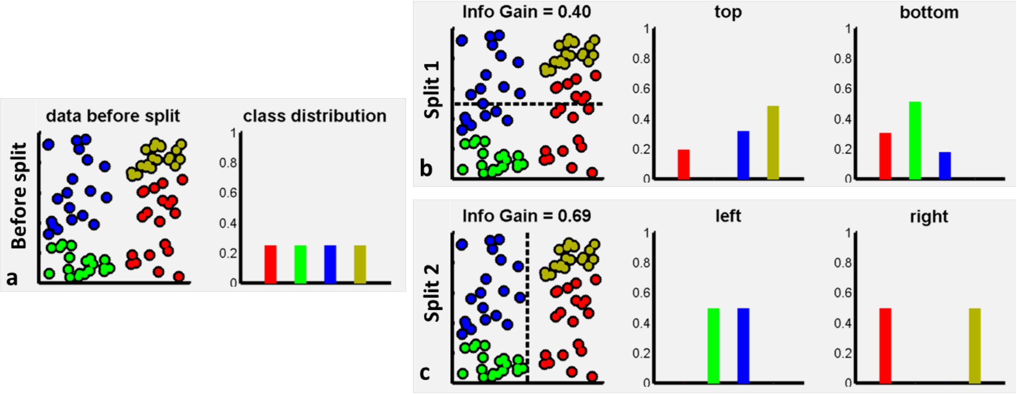


Fig. 2.3: **Information gain for discrete, non-parametric distributions. (a)** Dataset $\mathcal{S}$ before a split. **(b)** After a horizontal split. **(c)** After a vertical split.

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.

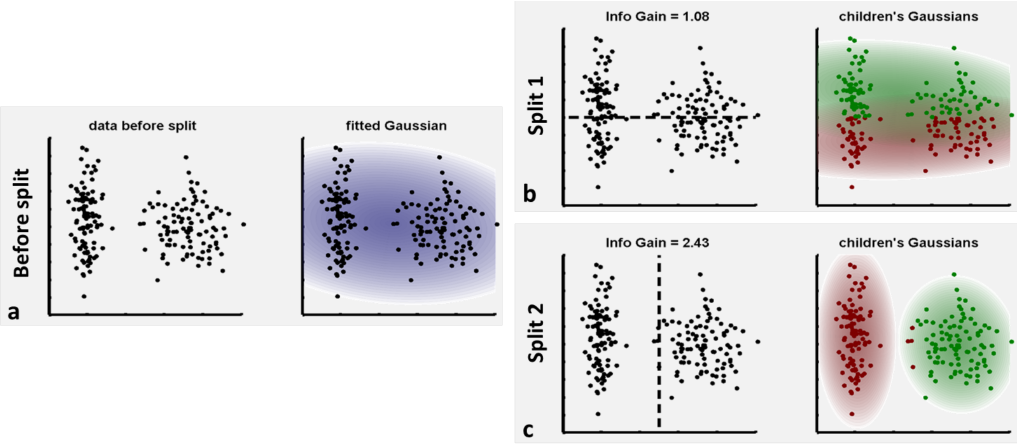


Fig. 2.4: **Information gain for continuous, parametric densities.** **(a)** Dataset $\mathcal{S}$ before a split. **(b)** After a horizontal split. **(c)** After a vertical split.

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.

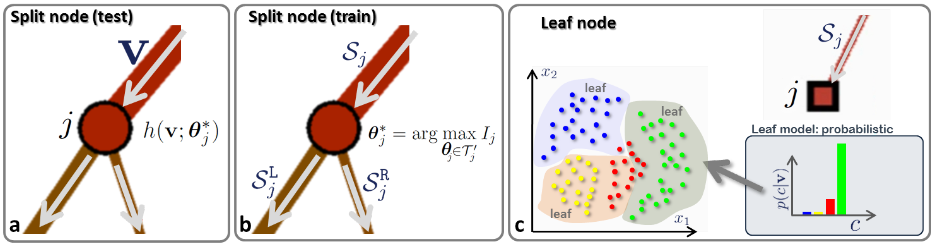

Split node (test)
V
j
$h(\mathbf{v};\boldsymbol{\theta}_j^*)$
a
Split node (train)
$\mathcal{S}_j$
$\boldsymbol{\theta}_j^* = \arg\max_{\boldsymbol{\theta}_j \in \mathcal{T}_j'} I_j$
$\mathcal{S}_j^{\mathrm{L}}$
$\mathcal{S}_j^{\mathrm{R}}$
b
Leaf node
$\mathcal{S}_j$
j
x_2
leaf
leaf
leaf
x_1
Leaf model: probabilistic
$p(c|\mathbf{v})$
c
c

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 $\boldsymbol{\tau}$

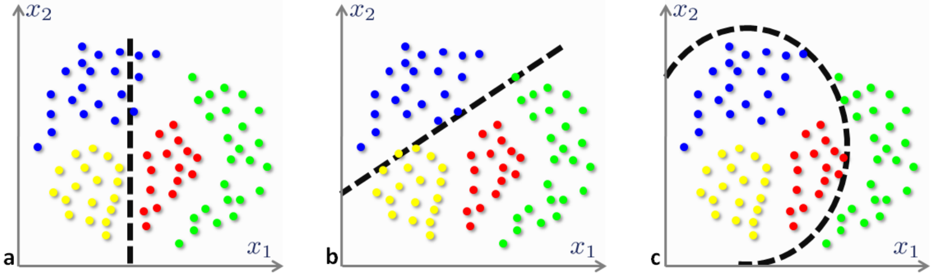


Fig. 2.6: **Example weak learners. (a)** Axis-aligned hyperplane. **(b)** General oriented hyperplane. **(c)** Quadratic (conic in 2D). For ease of visualization here we have $\mathbf{v} = (x_1\ x_2) \in \mathbb{R}^2$ and $\boldsymbol{\phi}(\mathbf{v}) = (x_1\ x_2\ 1)$ in homogeneous coordinates. In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

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

The symbols $\mathcal{S}_j, \mathcal{S}^{\mathtt{L}}_j, \mathcal{S}^{\mathtt{R}}_j$ 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.

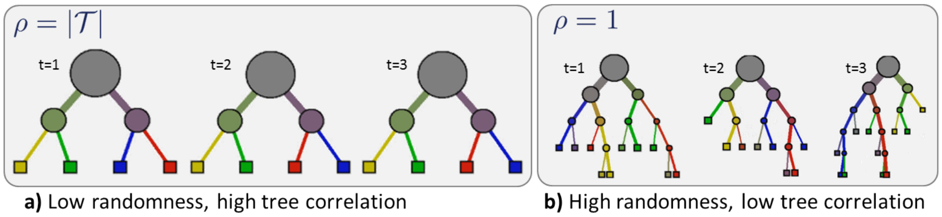

$\rho = |\mathcal{T}|$
t=1
t=2
t=3
$\rho = 1$
t=1
t=2
t=3
a) Low randomness, high tree correlation
b) High randomness, low tree correlation

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.

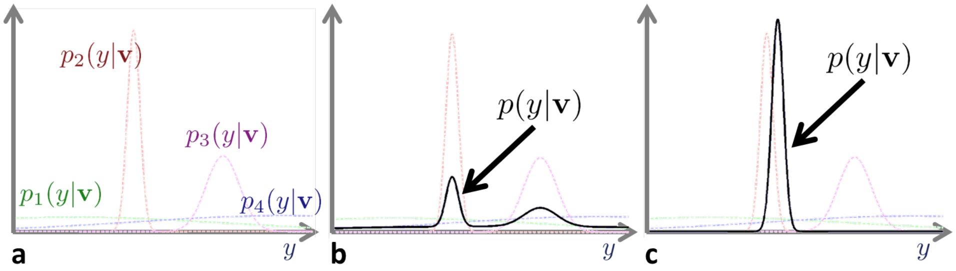


Fig. 2.8: **Ensemble model. (a)** The posteriors of four different trees (shown with different colours). Some correspond to higher confidence than others. **(b)** An ensemble posterior $p(y|\mathbf{v})$ obtained by averaging all tree posteriors. **(c)** The ensemble posterior $p(y|\mathbf{v})$ obtained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

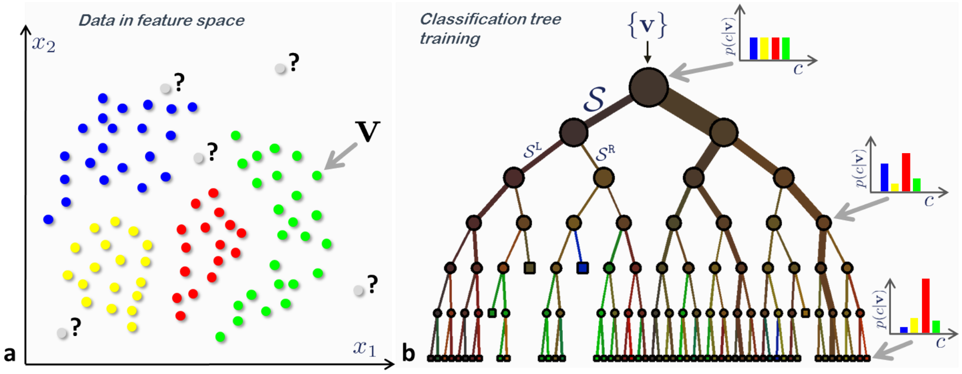


Fig. 3.1: **Classification: training data and tree training.** **(a)** Input data points. The ground-truth label of training points is denoted with different colours. Grey circles indicate unlabelled, previously unseen test data. **(b)** A binary classification tree. During training a set of labelled 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

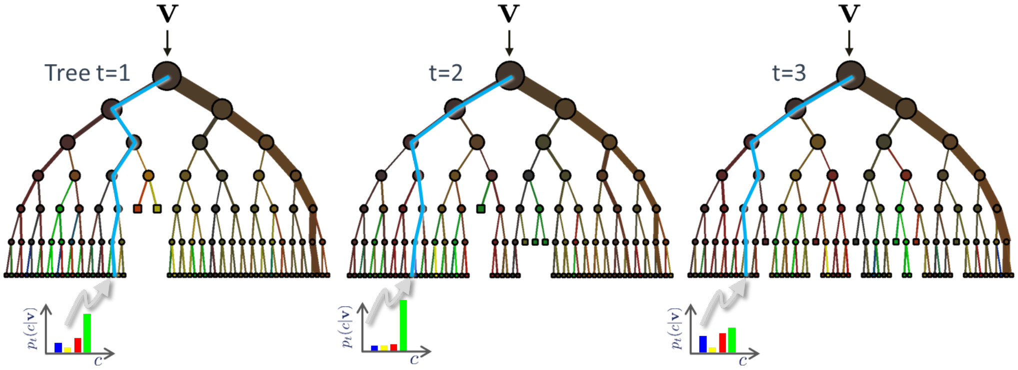


Fig. 3.2: **Classification forest testing.** During testing the same unlabelled test input data $\mathbf{v}$ is pushed through each component tree. At each internal node a test is applied and the data point sent to the appropriate child. The process is repeated until a leaf is reached. At the leaf the 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

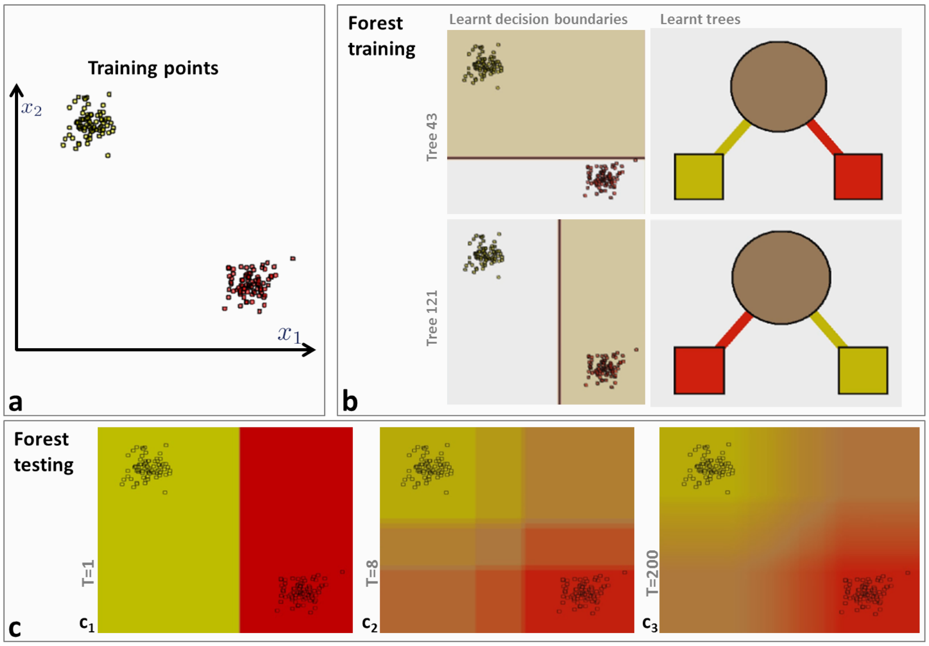


Fig. 3.3: **A first classification forest and the effect of forest size T.** (**a**) Training points belonging to two classes. (**b**) Different training trees produce different partitions and thus different leaf predictors. The colour of tree nodes and edges indicates the class probability of training 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-

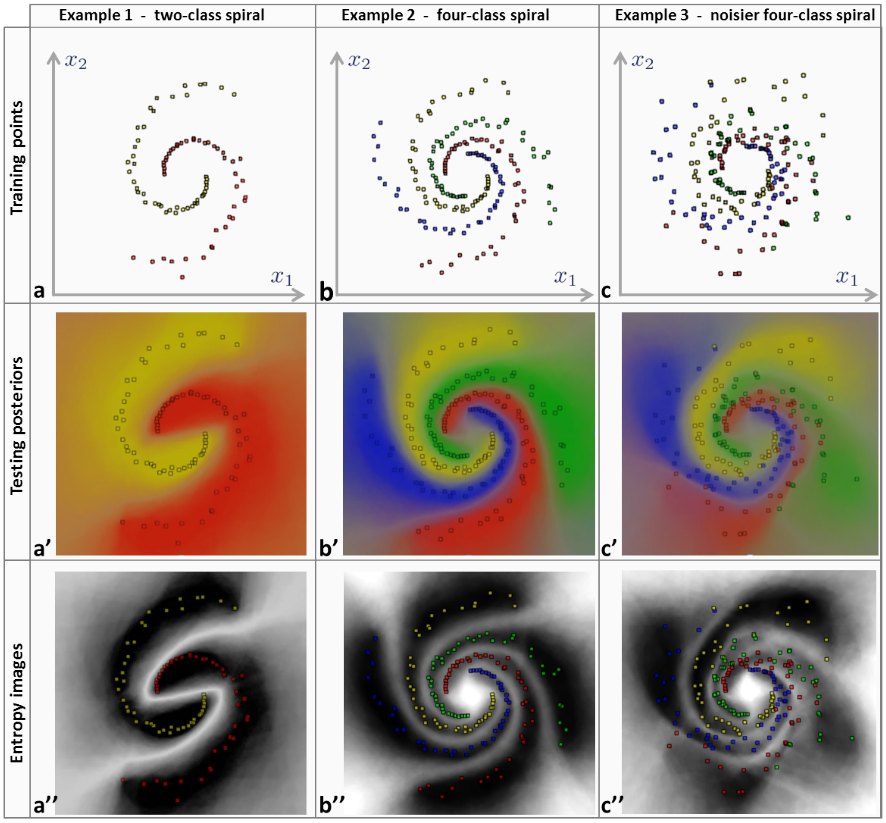


Fig. 3.4: **The effect of multiple classes and noise in training data. (a,b,c)** Training points for three different experiments: 2-class spiral, 4-class spiral and another 4-class spiral with noisier point positions, respectively. **(a',b',c')** Corresponding testing posteriors. **(a",b",c")** Corresponding 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

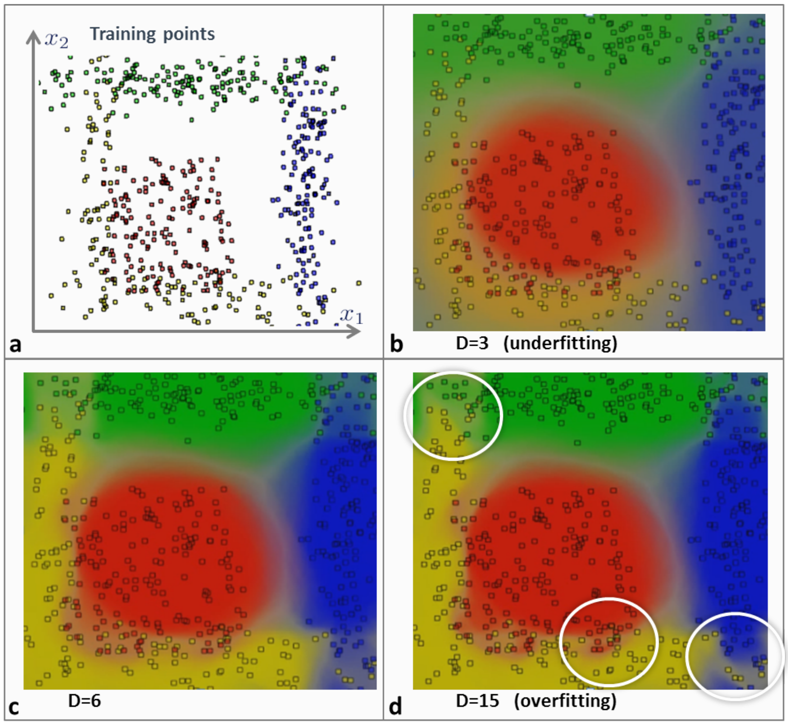


Fig. 3.5: **The effect of tree depth.** A four-class problem with both mixing of training labels and large gaps. **(a)** Training points. **(b,c,d)** Testing posteriors for different tree depths. All experiments were run with $T = 200$ and a conic weak-learner model. The tree depth is a crucial parameter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

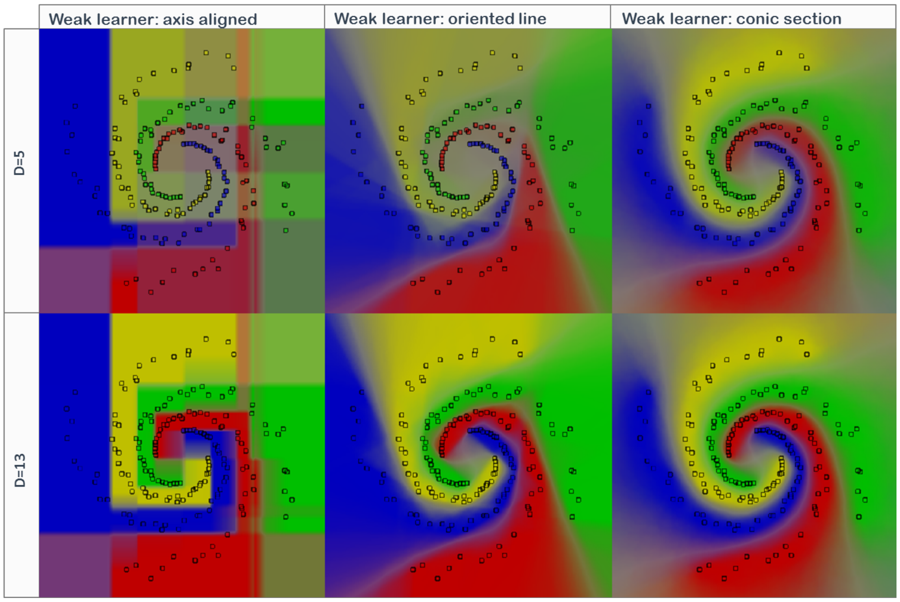


Fig. 3.6: **The effect of weak learner model.** The same set of 4-class training data is used to train 6 different forests, for 2 different values of D and 3 different weak learners. For fixed weak learner deeper trees produce larger confidence. For constant D non-linear weak learners produce the best 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

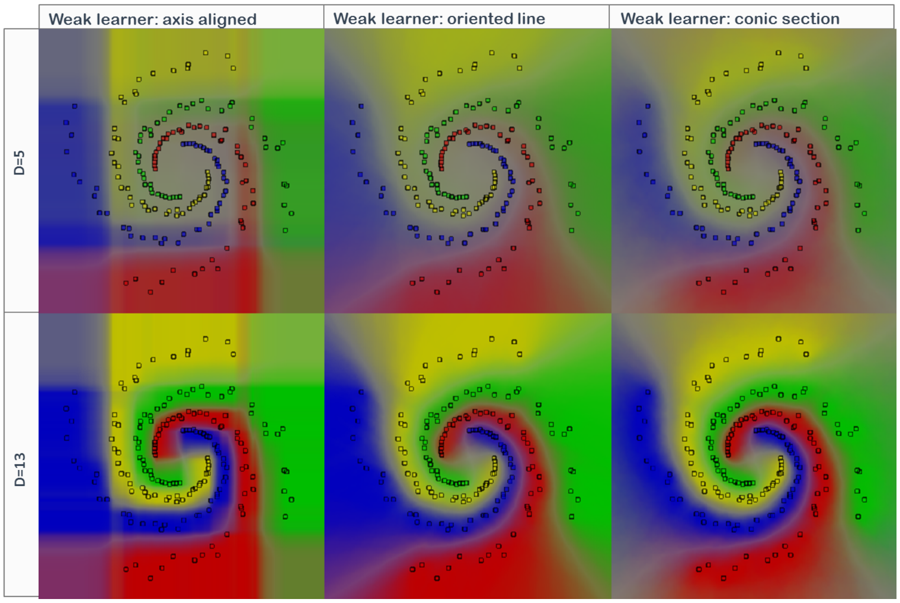


Fig. 3.7: **The effect of randomness.** The same set of 4-class training data is used to train 6 different forests, for 2 different values of D and 3 different weak learners. This experiment is identical to that in fig. 3.6 except that we have used much more training randomness. In fact $\rho = 5$ for 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, animations 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-

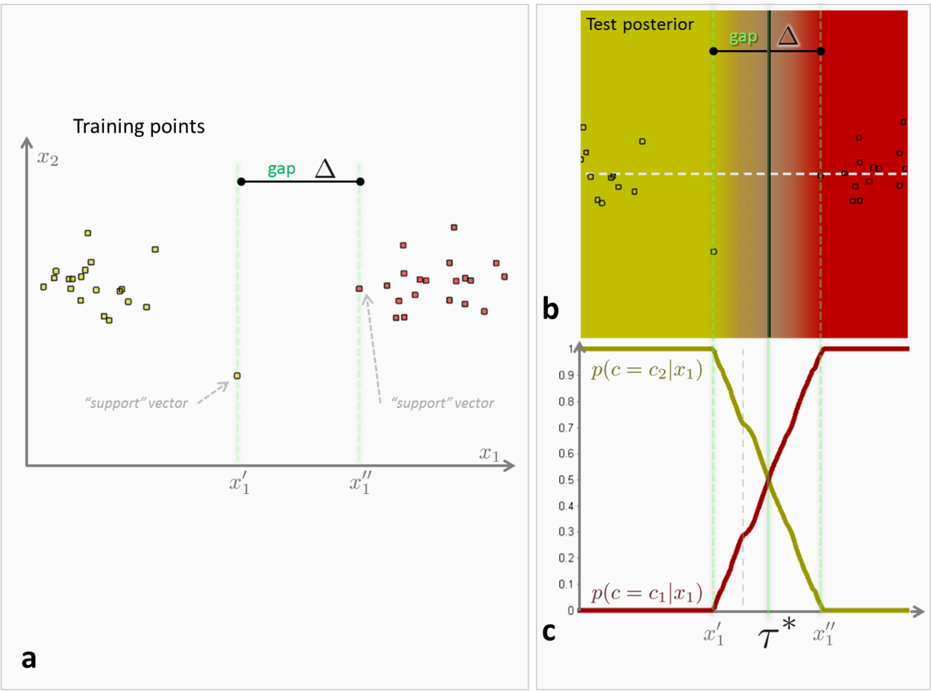


Fig. 3.8: **Forest's maximum-margin properties. (a)** Input 2-class training points. They are separated by a gap of dimension Δ. **(b)** Forest posterior. Note that all of the uncertainty band resides within the gap. **(c)** Cross-sections of class posteriors along the horizontal, white dashed line in (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.

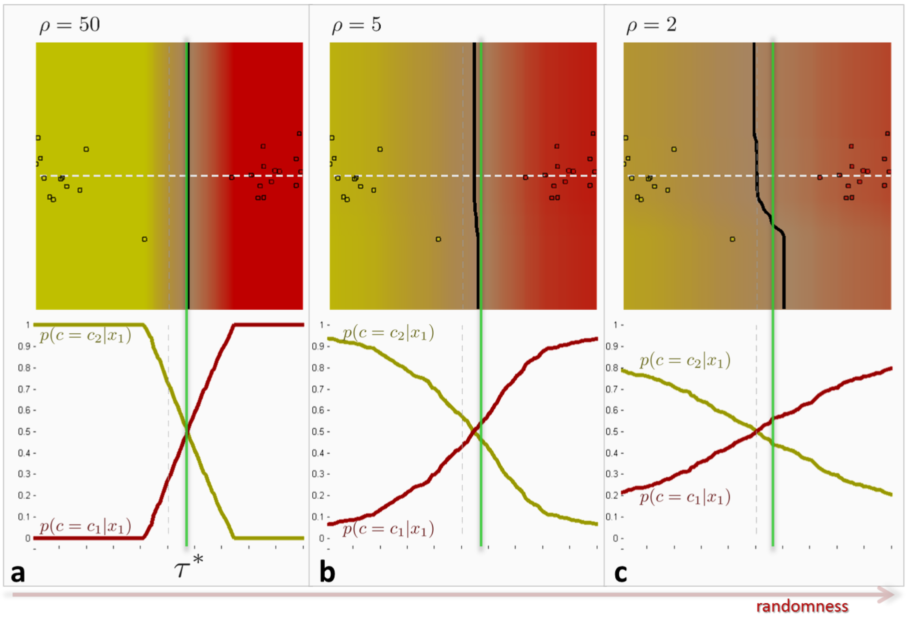


Fig. 3.9: **The effect of randomness on the forest margin. (a)** Forest posterior for $\rho = 50$ (small randomness). **(b)** Forest posterior for $\rho = 5$. **(c)** Forest posterior for $\rho = 2$ (highest randomness). These experiments have used $D = 2, T = 400$ and axis-aligned weak learners. The 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

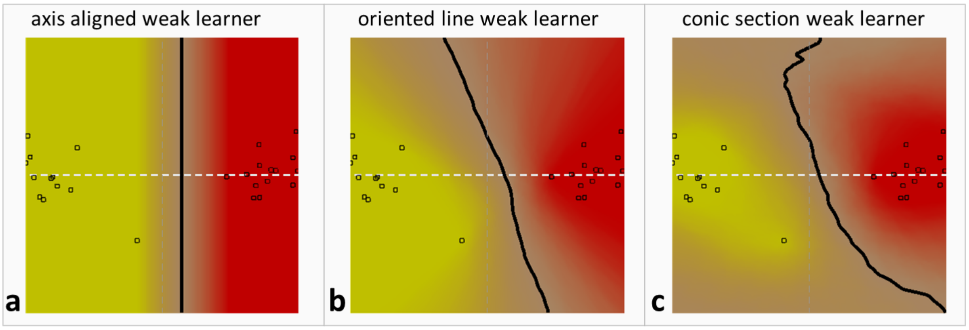


Fig. 3.10: **The effect of the weak learner on forest margin. (a)** Forest posterior for axis aligned weak learners. **(b)** Forest posterior for oriented line weak learners. **(c)** Forest posterior for conic section weak learners. In these experiments we have used $\rho = 50, D = 2, T = 500$. The choice 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

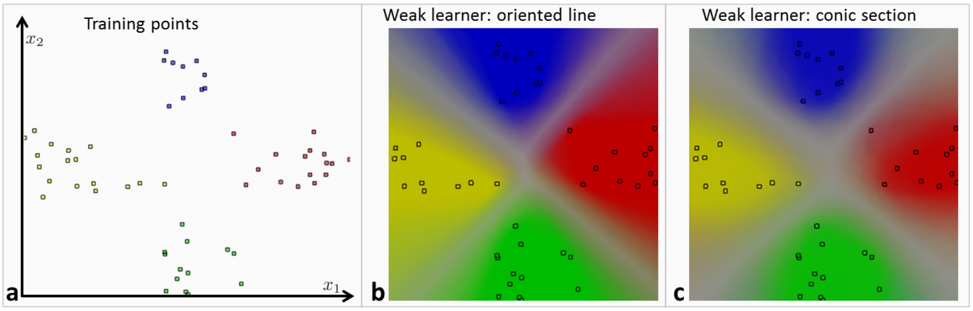


Fig. 3.11: **Forest's max-margin properties for multiple classes. (a)** Input four-class training points. **(b)** Forest posterior for oriented line weak learners. **(c)** Forest posterior for conic section weak learners. Regions of high entropy are shown as grey bands and correspond to loci of optimal 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.

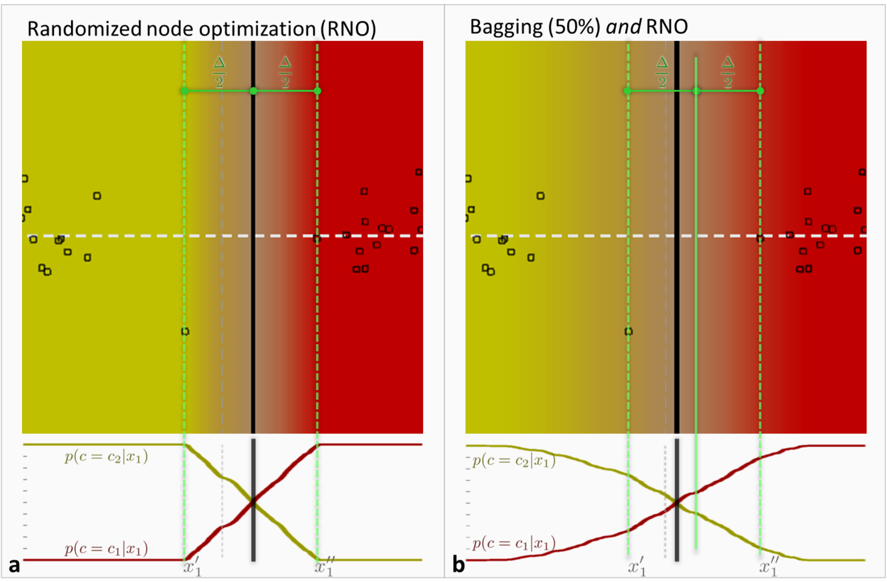


Fig. 3.12: **Max-margin: bagging v randomized node optimization. (a)** Posterior for forest trained with randomized node optimization. **(b)** Posterior for forest trained with bagging. In bagging, for each tree we use 50% random selection of training data with replacement. Loci of optimal separation 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 enables 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`

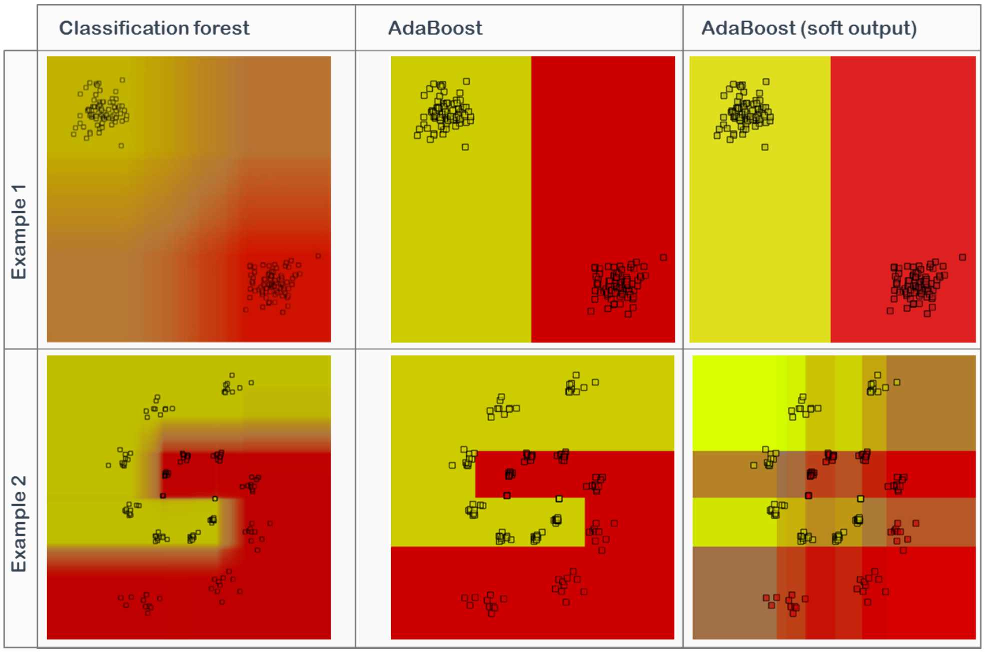


Fig. 3.13: **Comparison between classification forests and boosting** on two examples. Forests produce a smooth, probabilistic output. High uncertainty is associated with regions between different classes or away from training data. boosting produces a hard output. Interpreting the output of a boosted 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.

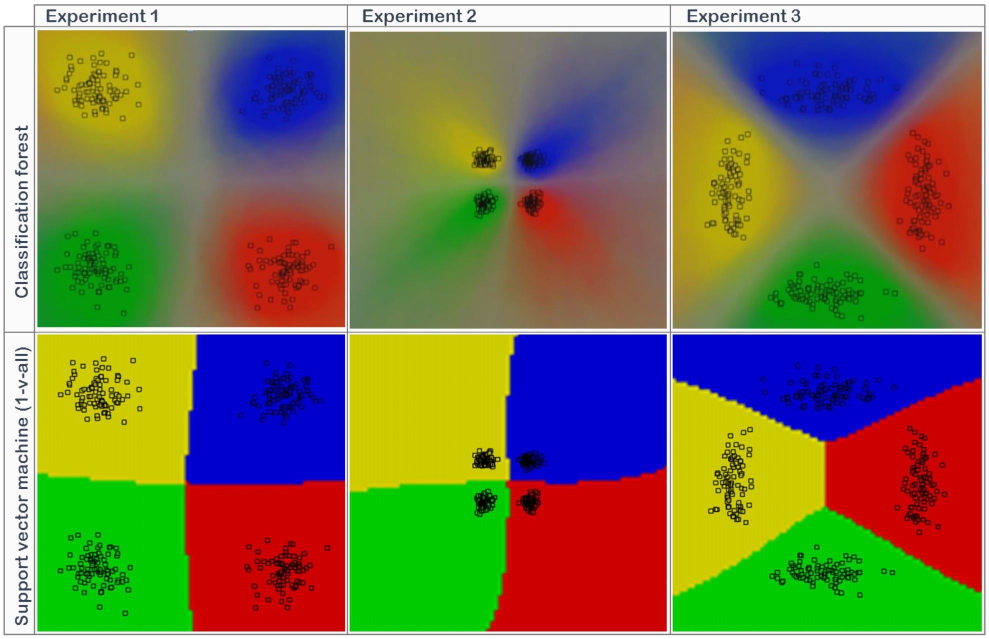


Fig. 3.14: **Comparison between classification forests and support vector machines.** All forest experiments were run with $D = 3$, $T = 200$ and conic weak learner. The SVM parameters were optimized to achieve best results.

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

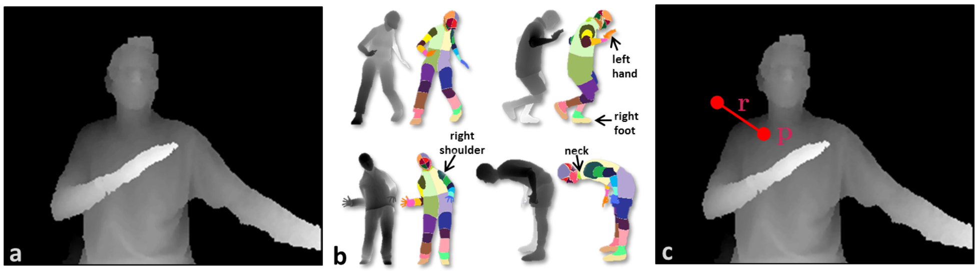


Fig. 3.15: **Classification forests in Microsoft Kinect for XBox 360.** **(a)** An input frame as acquired by the Kinect depth camera. **(b)** Synthetically generated ground-truth labeling of 31 different body parts [82]. **(c)** One of the many features of a "reference" point $\mathbf{p}$. Given $\mathbf{p}$ 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

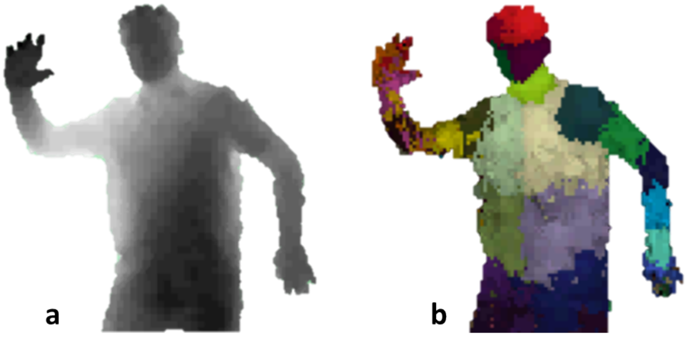


Fig. 3.16: **Classification forests in Kinect for XBox 360. (a)** An input depth frame with background removed. **(b)** The body part classification posterior. Different colours corresponding to different body parts, out of 31 different classes.

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].

4

Regression forests

This chapter discusses the use of random decision forests for the estimation of continuous variables.

Regression forests are used for the non-linear regression of dependent variables given independent input. Both input and output may be multi-dimensional. The output can be a point estimate or a full probability density function.

Regression forests are less popular than their classification counterpart. The main difference is that the output label to be associated with an input data is continuous. Therefore, the training labels are continuous. Consequently the objective function has to be adapted appropriately. Regression forests share many of the advantages of classification forests such as efficiency and flexibility.

As with the other chapters we start with a brief literature survey of linear and non-linear regression techniques, then we describe the regression forest model and finally we demonstrate its properties with examples and comparisons.

4.1 Nonlinear regression in the literature

Given a set of noisy input data and associated continuous measurements, least squares techniques [7] (closely related to principal component analysis [48]) can be used to fit a linear regressor which minimizes some error computed over all training points. Under this model, given a new test input the corresponding output can be efficiently estimated. The limitation of this model is in its linear nature, when we know that most natural phenomena have non-linear behaviour [79]. Another well known issue with linear regression techniques is their sensitivity to input noise.

In geometric computer vision, a popular technique for achieving robust regression via randomization is RANSAC [30, 41]. For instance the estimation of multi-view epipolar geometry and image registration transformations can be achieved in this way [41]. One disadvantage of conventional RANSAC is that its output is non probabilistic. As will be clearer later, regression forests may be thought of as an extension of RANSAC, with little RANSAC regressors for each leaf node.

In machine learning, the success of support vector classification has encouraged the development of support vector regression (SVR [51, 86]). Similar to RANSAC, SVR can deal successfully with large amounts of noise. In Bayesian machine learning Gaussian processes [5, 73] have enjoyed much success due to their simplicity, elegance and their rigorous uncertainty modeling.

Although (non-probabilistic) regression forests were described in [11] they have only recently started to be used in computer vision and medical image analysis [24, 29, 37, 49, 59]. Next, we discuss how to specialize the generic forest model described in chapter 2 to do probabilistic, nonlinear regression efficiently. Many synthetic experiments, commercial applications and comparisons with existing algorithms will validate the regression forest model.

4.2 Specializing the decision forest model for regression

The regression task can be summarized as follows:

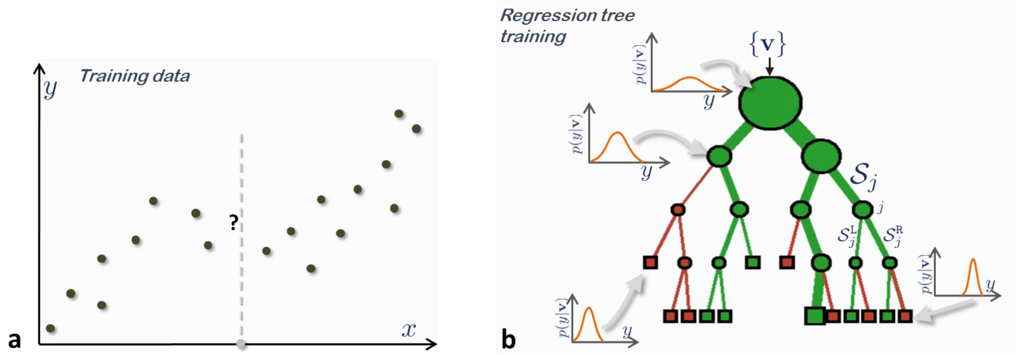


Fig. 4.1: **Regression: training data and tree training. (a)** Training data points are shown as dark circles. The associated ground truth label is denoted by their position along the y coordinate. The input feature space here is one-dimensional in this example ($\mathbf{v} = (x)$). x is the independent input and y is the dependent variable. A previously unseen test input is indicated with a light gray circle. **(b)** A binary regression tree. During training a set of labelled training points $\{\mathbf{v}\}$ is used to optimize the parameters of the tree. In a regression tree the entropy of the continuous densities associated with different nodes decreases (their confidence increases) when going from the root towards the leaves.

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

Like classification the regression task is inductive, with the main difference being the continuous nature of the output. Figure 4.1a provides an illustrative example of training data and associated continuous ground-truth labels. A previously unseen test input (unavailable during training) is shown as a light grey circle on the x axis.

Formally, given a multi-variate input $\mathbf{v}$ we wish to associate a continuous multi-variate label $\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^n$. More generally, we wish to estimate the probability density function $p(\mathbf{y}|\mathbf{v})$. As usual the

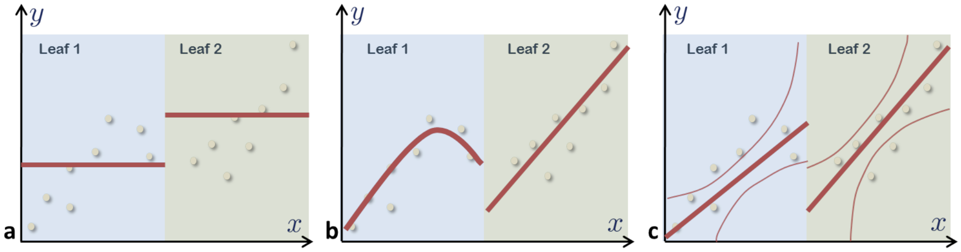


Fig. 4.2: **Example predictor models.** Different possible predictor models. **(a)** Constant. **(b)** Polynomial and linear. **(c)** Probabilistic-linear. The conditional distribution $p(y|x)$ is returned in the latter.

input is represented as a multi-dimensional feature response vector $\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$.

Why regression forests? A regression forest is a collection of randomly trained regression trees (fig. 4.3). Just like in classification it can be shown that a forest generalizes better than a single over-trained tree.

A regression tree (fig. 4.1b) splits a complex nonlinear regression problem into a set of smaller problems which can be more easily handled by simpler models (*e.g.* linear ones; see also fig.4.2). Next we specify the precise nature of each model component.

The prediction model. The first job of a decision tree is to decide which branch to direct the incoming data to. But when the data reaches a terminal node then that leaf needs to make a prediction.

The actual form of the prediction depends on the prediction model. In classification we have used the pre-stored empirical class posterior as model. In regression forests we have a few alternatives, as illustrated in fig. 4.2. For instance we could use a polynomial function of a subspace of the input $\mathbf{v}$. In the low dimensional example in the figure a generic polynomial model corresponds to $y(x) = \sum_{i=0}^{n} w_i x^i$. This simple model also captures the linear and constant models (see fig. 4.2a,b).

In this paper we are interested in output confidence as well as its

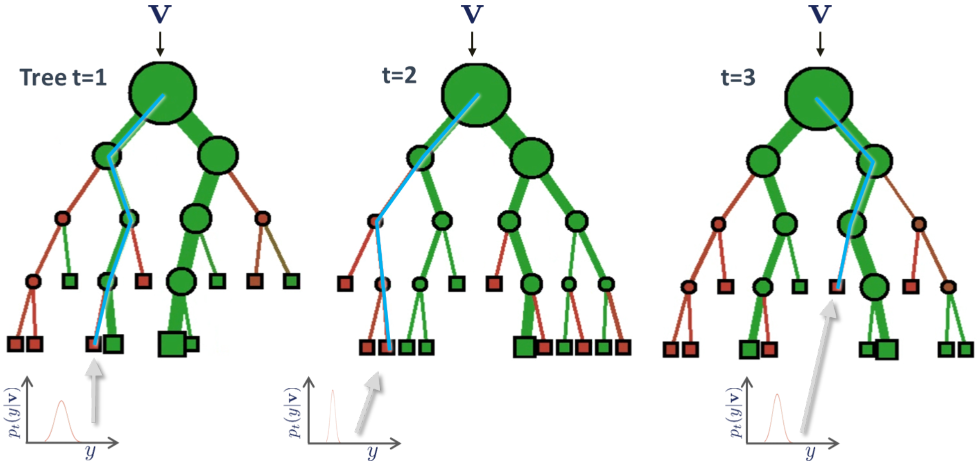


Fig. 4.3: **Regression forest: the ensemble model.** The regression forest posterior is simply the average of all individual tree posteriors $p(\mathbf{y}|\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} p_t(\mathbf{y}|\mathbf{v})$.

actual value. Thus for prediction we can use a probability density function over the continuous variable $\mathbf{y}$. So, given the t^{th} tree in a forest and an input point $\mathbf{v}$, the associated leaf output takes the form $p_t(\mathbf{y}|\mathbf{v})$. In the low-dimensional example in fig. 4.2c we assume an underlying linear model of type $y = w_0 + w_1 x$ and each leaf yields the conditional $p(y|x)$.

The ensemble model. Just like in classification, the forest output is the average of all tree outputs (fig. 4.3):

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

A practical justification for this model was presented in section 2.2.5.

Randomness model. Like in classification here we use a randomized node optimization model. Therefore, the amount of randomness is controlled during training by the parameter $\rho = |\mathcal{T}_j|$. The random subsets of split parameters $\mathcal{T}_j$ can be generated on the fly when training the j^{th} node.

The training objective function. Forest training happens by optimizing an energy over a training set $\mathcal{S}_0$ of data and associated continuous labels. Training a split node j happens by optimizing the parameters of its weak learner:

$$\boldsymbol{\theta}_j^* = \arg \max_{\boldsymbol{\theta}_j \in \mathcal{T}_j} I_j. \tag{4.1}$$

Now, the main difference between classification and regression forest is in the form of the objective function I_j.

In [12] regression trees are trained by minimizing a least-squares or least-absolute error function. Here, for consistency with our general forest model we employ a continuous formulation of information gain. Appendix A illustrates how information theoretical derivations lead to the following definition of information gain:

$$I_j = \sum_{\mathbf{v} \in \mathcal{S}_j} \log \left(|\Lambda_{\mathbf{y}}(\mathbf{v})| \right) - \sum_{\mathtt{i} \in \{\mathtt{L},\mathtt{R}\}} \left(\sum_{\mathbf{v} \in \mathcal{S}_j^{\mathtt{i}}} \log \left(|\Lambda_{\mathbf{y}}(\mathbf{v})| \right) \right) \tag{4.2}$$

with $\Lambda_{\mathbf{y}}$ the conditional covariance matrix computed from probabilistic linear fitting (see also fig. 4.4). $\mathcal{S}_j$ indicates the set of training data arriving at node j, and $\mathcal{S}_j^{\mathtt{L}}$, $\mathcal{S}_j^{\mathtt{R}}$ the left and right split sets. Note that (4.2) is valid only for the case of a probabilistic-linear prediction model (fig. 4.2).

By comparison, the error or fit objective function used in [12] (for single-variate output y) is:

$$\sum_{\mathbf{v} \in \mathcal{S}_j} \left(y - \overline{y}_j \right)^2 - \sum_{\mathtt{i} \in \{\mathtt{L},\mathtt{R}\}} \left(\sum_{\mathbf{v} \in \mathcal{S}_j^{\mathtt{i}}} \left(y - \overline{y}_j \right)^2 \right), \tag{4.3}$$

with $\overline{y}_j$ indicating the mean value of y for all training points reaching the j^{th} node. Note that (4.3) is closely related to (4.2) but limited to constant predictors. Also, in [12] the author is only interested in a point estimate of y rather than a fully probabilistic output. Furthermore, using an information theoretic formulation allows us to unify

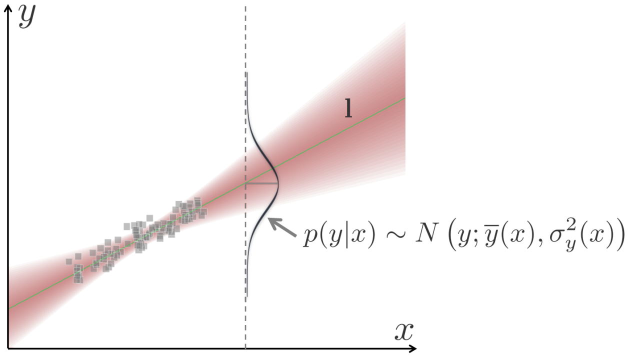


Fig. 4.4: **Probabilistic line fitting.** Given a set of training points we can fit a line $\mathbf{l}$ to them, *e.g.* by least squares or RANSAC. In this example $\mathbf{l} \in \mathbb{R}^2$. Matrix perturbation theory (see appendix A) enables us to estimate a probabilistic model of $\mathbf{l}$ from where we can derive $p(y|x)$ (modelled here as a Gaussian). Training a regression tree involves minimizing the uncertainty of the prediction $p(y|x)$ over the training set. Therefore, the training objective is a function of σ_y^2 evaluated at the training points.

different tasks within the same, general probabilistic forest model. To fully characterize our regression forest model we still need to decide how to split the data arriving at an internal node.

The weak learner model. As usual, the data arriving at a split node j is separated into its left or right children (see fig. 4.1b) according to a binary weak learner stored in an internal node, of the following general form:

$$h(\mathbf{v}, \boldsymbol{\theta}_j) \in \{0, 1\}, \tag{4.4}$$

with 0 indicating "false" (go left) and 1 indicating "true" (go right). Like in classification here we consider three types of weak learners: (i) axis-aligned, ii) oriented hyperplane, (iii) quadratic (see fig. 4.5 for an illustration on 2D→1D regression). Many additional weak learner models may be considered.

Next, a number of experiments will illustrate how regression forests

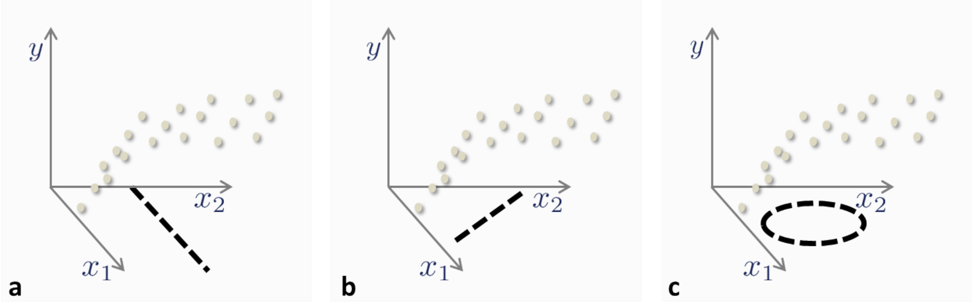


Fig. 4.5: **Example weak learners.** The (x_1, x_2) plane represents the $d-$dimensional input domain (independent). The y space represents the $n-$dimensional continuous output (dependent). The example types of weak learner are like in classification **(a)** Axis-aligned hyperplane. **(b)** General oriented hyperplane. **(c)** Quadratic (corresponding to a conic section in 2D). Further weak learners may be considered.

work in practice and the effect of different model choices on their output.

4.3 Effect of model parameters

This section discusses the effect of model choices such as: tree depth, forest size and weak learner model on the forest behaviour.

4.3.1 The effect of the forest size

Figure 4.6 shows a first, simple example. We are given the training points shown in fig. 4.6a. We can think of those as being randomly drawn from two segments with different orientations. Each point has a 1-dimensional input feature x and a corresponding scalar, continuous output label y.

A forest of shallow trees ($D = 2$) and varying size T is trained on those points. We use axis-aligned weak learners, and probabilistic-linear predictor models. The trained trees (fig. 4.6b) are all slightly different from each other as they produce different leaf models (fig. 4.6b). During training, as expected each leaf model produces smaller uncertainty near

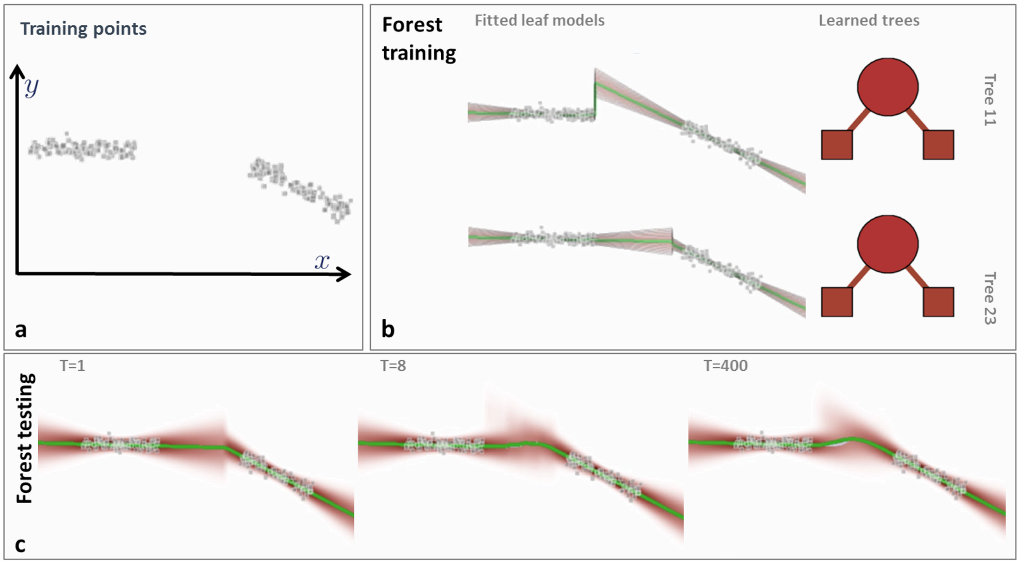


Fig. 4.6: **A first regression forest and the effect of its size T.** (**a**) Training points. (**b**) Two different shallow trained trees ($D = 2$) split the data into two portions and produce different piece-wise probabilistic-linear predictions. (**c**) Testing posteriors evaluated for all values of x and increasing number of trees. The green curve denotes the conditional mean $\mathcal{E}[y|x] = \int y \cdot p(y|x)\ dy$. The mean curve corresponding to a single tree ($T = 1$) shows a sharp change of direction in the gap. Increasing the forest size produces smoother class posteriors $p(y|x)$ and smoother mean curves in the interpolated region. All examples have been run with $D = 2$, axis-aligned weak learners and probabilistic-linear prediction models.

the training points and larger away from them. In the gap the actual split happens in different places along the x axis for different trees.

The bottom row (fig. 4.6c) shows the regression posteriors evaluated for *all* positions along the x axis. For each x position we plot the entire distribution $p(y|x)$, where darker red indicates larger values of the posterior. Thus, very compact, dark pixels correspond to high prediction confidence.

Note how a single tree produces a sharp change in direction of the mean prediction $\overline{y}(x) = \mathcal{E}[y|x] = \int y \cdot p(y|x)\ dy$ (shown in green) in the large gap between the training clusters. But as the number of trees increases both the prediction mean and its uncertainty become smoother. Thus smoothness of the interpolation is controlled here simply by the parameter T. We can also observe how the uncertainty increases as we move away from the training data (both in the interpolated gap and in the extrapolated regions).

4.3.2 The effect of the tree depth

Figure 4.7 shows the effect of varying the maximum allowed tree depth D on the same training set as in fig.4.6. A regression forest with $D = 1$ (top row in figure) corresponds to conventional linear regression (with additional confidence estimation). In this case the training data is more complex than a single line and thus such a degenerate forest under-fits. In contrast, a forest of depth $D = 5$ (bottom row in figure) yields overfitting. This is highlighted in the figure by the high-frequency variations in the prediction confidence and the mean $\overline{y}(x)$.

4.3.3 Spatial smoothness and testing uncertainty

Figure 4.8 shows four more experiments. The mean prediction curve $\overline{y}(x)$ is plotted in green and the mode $\hat{y}(x) = \arg\max_y p(y|x)$ is shown in grey. These experiments highlight the smooth interpolating behaviour of the mean prediction in contrast to the more jagged nature of the mode.[1] The uncertainty increases away from training data. Finally, notice how in the gaps the regression forest can correctly capture multi-modal posteriors. This is highlighted by the difference between mode and mean predictions. In all experiments we used a probabilistic-linear predictor with axis-aligned weak learner, $T = 400$ and $D = 7$. Many more examples, animations and videos are available at [1].

[1] The smoothness of the mean curve is a function of T. The larger the forest size the smoother the mean prediction curve.

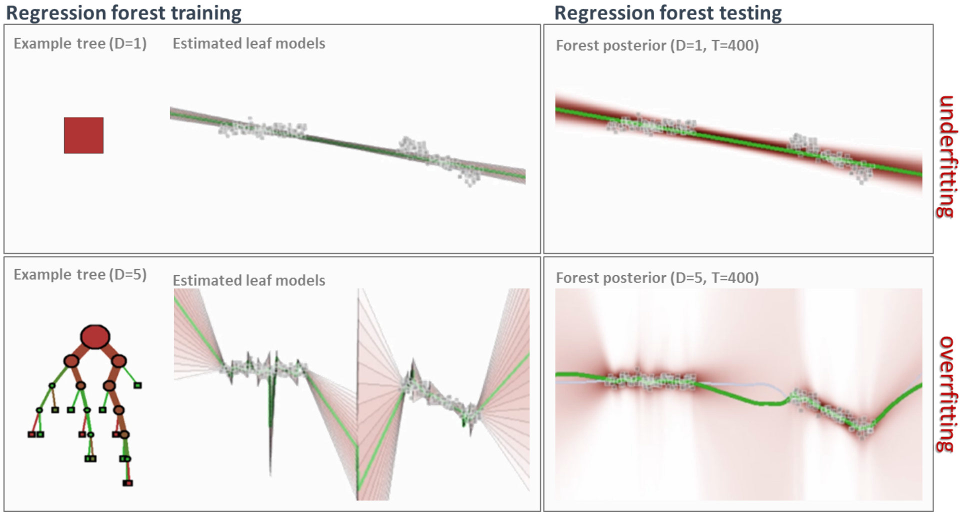


Fig. 4.7: **The effect of tree depth. (Top row)** Regression forest trained with $D = 1$. Trees are degenerate (each tree corresponds only to their root node). This corresponds to conventional linear regression. In this case the data is more complex than a single linear model and thus this forest under-fits. **(Bottom row)** Regression forest trained with $D = 5$. Much deeper trees produce the opposite effect, *i.e.* over-fitting. This is evident in the high-frequency, spiky nature of the testing posterior. In both experiments we use $T = 400$, axis-aligned weak learners and probabilistic-linear prediction models.

4.4 Comparison with alternative algorithms

The previous sections have introduced the probabilistic regression forest model and discussed some of its properties. This section shows a comparison between forests and allegedly the most common probabilistic regression technique, Gaussian processes [73].

4.4.1 Comparison with Gaussian processes

The hallmark of Gaussian processes is their ability to model uncertainty in regression problems. Here we compare regression forests with

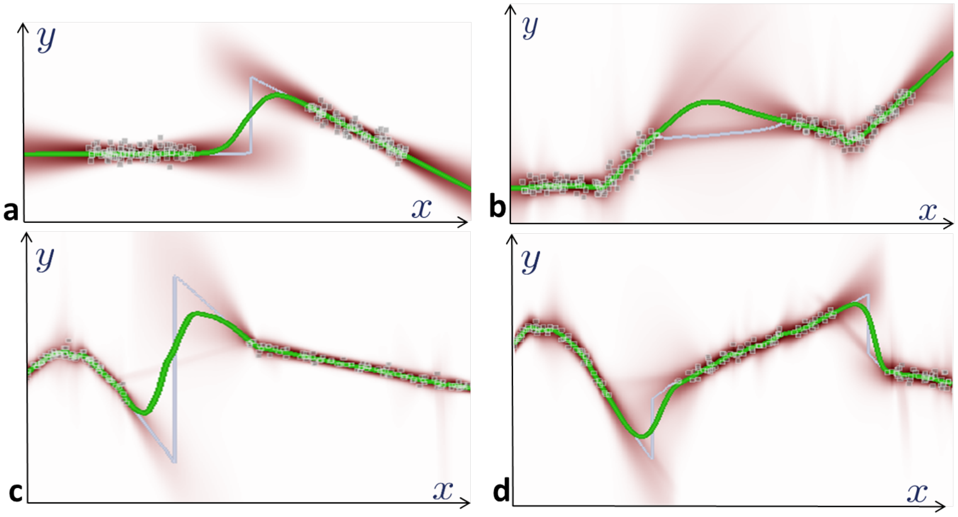


Fig. 4.8: **Spatial smoothness, multi-modal posteriors and testing uncertainty.** Four more regression experiments. The squares indicate labelled training data. The green curve is the estimated conditional mean $\overline{y}(x) = E[y|x] = \int y \cdot p(y|x)\, dy$ and the grey curve the estimated mode $\hat{y}(x) = \arg\max_y p(y|x)$. Note the smooth interpolating behaviour of the mean over large gaps and increased uncertainty away from training data. The forest is capable of capturing multi-modal behaviour in the gaps. See text for details.

Gaussian Processes on a few representative examples.[2]

In figure 4.9 we compare the two regression models on three different training sets. In the first experiment the training data points are simply organized along a line segment. In the other two experiments the training data is a little more complex with large gaps. We wish to investigate the nature of the interpolation and its confidence in those gaps. The 2×3 table of images show posteriors corresponding to the 3 different training sets (columns) and 2 models (rows).

[2] The Gaussian process results in this section were obtained with the "Gaussian Process Regression and Classification Toolbox version 3.1", publically available at `http://www.gaussianprocess.org/gpml/code/matlab/doc`.

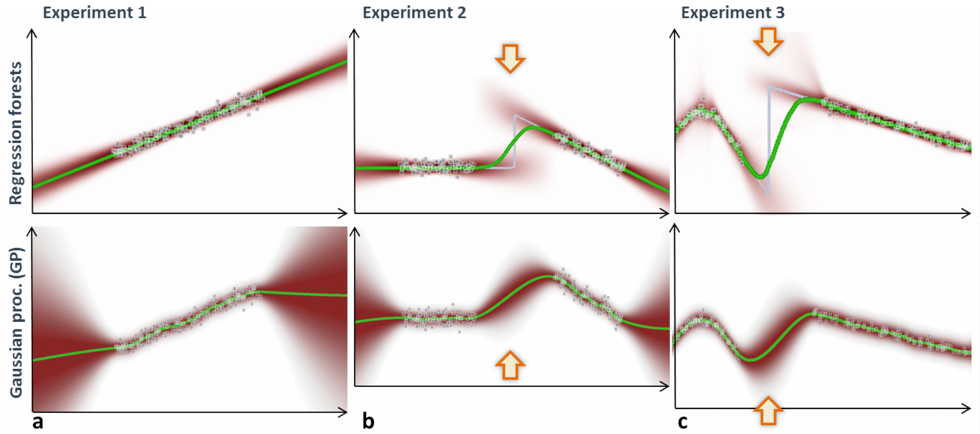


Fig. 4.9: **Comparing regression forests with Gaussian processes. (a,b,c)** Three training datasets and the corresponding testing posteriors overlaid on top. In both the forest and the GP model uncertainties increase as we move away from training data. However, the actual shape of the posterior is different. **(b,c)** Large gaps in the training data are filled in both models with similarly smooth mean predictions (green curves). However, the regression forest manages to capture the bi-modal nature of the distributions, while the GP model produces intrinsically uni-modal Gaussian predictions.

Gaussian processes are well known for how they model increasing uncertainty with increasing distance from training points. The bottom row illustrates this point very clearly. Both in extrapolated and interpolated regions the associated uncertainty increases smoothly. The Gaussian process mean prediction (green curve) is also smooth and well behaved.

Similar behaviour can be observed for the regression forest too (top row). As observed also in previous examples the confidence of the prediction decreases with distance from training points. The specific shape in which the uncertainty region evolves is a direct consequence of the particular prediction model used (linear here). One striking difference between the forest and the GP model though is illustrated in

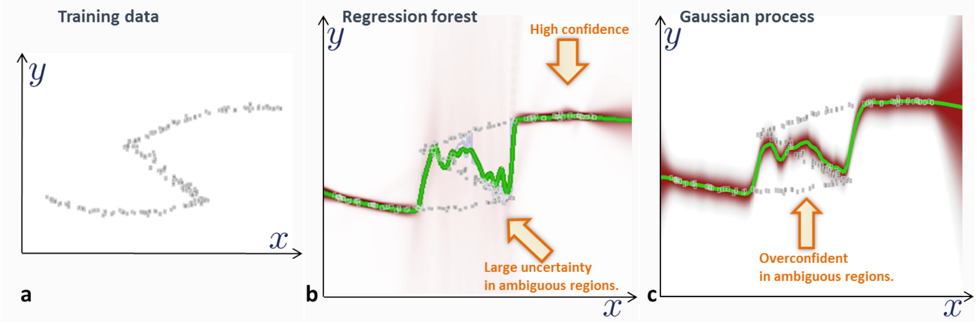


Fig. 4.10: **Comparing forests and GP on ambiguous training data. (a)** Input labelled training points. The data is ambiguous because a given input x may correspond to multiple values of y. **(b)** The posterior $p(y|x)$ computed via random regression forest. The middle (ambiguous) region remains associated with high uncertainty (in grey). **(c)** The posterior computed via Gaussian Processes. Conventional GP models do not seem flexible enough to capture spatially varying noise in training points. This yields an over-confident prediction in the central region. In all these experiments the GP parameters have been automatically optimized for optimal results, using the provided Matlab code.

figs. 4.9b,c. There, we can observe how the forest can capture bi-modal distributions in the gaps (see orange arrows). Due to their piece-wise nature the regression forest seems more apt at capturing multi-modal behaviour in testing regions and thus modeling intrinsic ambiguity (different y values may be associated with the same x input). In contrast, the posterior of a Gaussian process is by construction a (uni-modal) Gaussian, which may be a limitation in some applications. The same uni-modal limitation also applies to the recent “relevance voxel machine” technique in [76].

This difference between the two models in the presence of ambiguities is tested further in fig. 4.10. Here the training data itself is arranged in an ambiguous way, as a “non-function” relation (see also [63] for computer vision examples). For the same value of x there may be multiple training points with different values of y.

The corresponding testing posteriors are shown for the two models in fig. 4.10b and fig. 4.10c, respectively. In this case neither model can model the central, ambiguous region correctly. However, notice how although the mean curves are very similar to one another, the uncertainty is completely different. The Gaussian process yields a largely over-confident prediction in the ambiguous region; while the forest correctly yields a very large uncertainty. it may be possible to think of improving the forest output *e.g.* by using a mixture of probabilistic-linear predictors at each leaf (as opposed to a single line). Later chapters will show how a tighter, more informative prediction can be obtained in this case, using density forests.

4.5 Semantic parsing of 3D computed tomography scans

This section describes a practical application of regression forest which is now part of the commercial product Microsoft Amalga Unified Intelligence System.[3]

Given a 3D Computed Tomography (CT) image we wish to automatically detect the presence/absence of a certain anatomical structure, and localize it in the image (see fig. 4.11). This is useful for *e.g.* (i) the efficient retrieval of selected portions of patients scans through low bandwidth networks, (ii) tracking patients' radiation dose over time, (iii) the efficient, semantic navigation and browsing of n-dimensional medical images, (iv) hyper-linking regions of text in radiological reports with the corresponding regions in medical images, and (v) assisting the image registration in longitudinal studies [50]. Details of the algorithm can be found in [24]. Here we give a very brief summary of this algorithm to show how it stems naturally from the general model of regression forests presented here.

In a given volumetric image the position of each voxel is denoted with a 3-vector $\mathbf{p} = (x\,y\,z)$. For each organ of interest we wish to estimate the position of a 3D axis-aligned bounding box tightly placed to contain the organ. The box is represented as a 6-vector containing the absolute coordinates (in mm) of the corresponding walls: $\mathbf{b} = \left(b^{\mathtt{L}}, b^{\mathtt{R}}, b^{\mathtt{H}}, b^{\mathtt{F}}, b^{\mathtt{A}}, b^{\mathtt{P}}\right) \in \mathbb{R}^6$ (see fig. 4.12a). For simplicity here we

[3] http://en.wikipedia.org/wiki/Microsoft_Amalga.

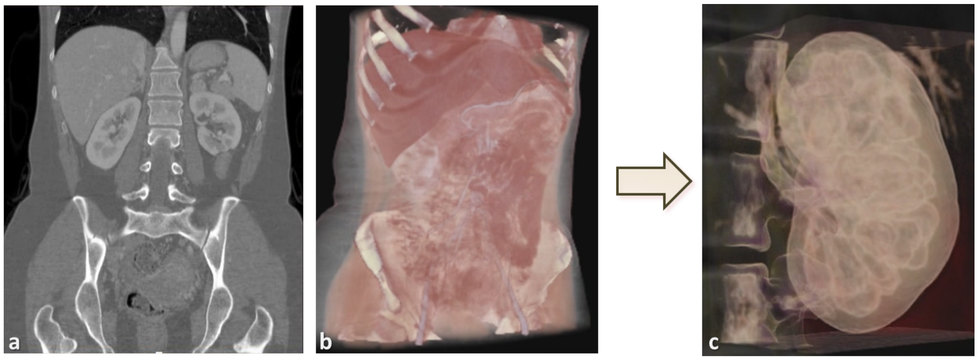


Fig. 4.11: **Automatic localization of anatomy in 3D Computed Tomography images. (a)** A coronal slice (frontal view) from a test 3D CT patient's scan. **(b)** Volumetric rendering of the scan to aid visualization. **(c)** Automatically localized left kidney using regression forest. Simultaneous localization of 25 different anatomical structures takes $\sim 4s$ on a single core of a standard desktop machine, with a localization accuracy of ~ 1.5cm. See [24] for algorithmic details.

focus on a single organ of interest.[4]

The continuous nature of the output suggests casting this task as a regression problem. Inspired by the work in [33] here we allow each voxel to vote (probabilistically) for the positions of all six walls. So, during testing, each voxel $\mathbf{p}$ in a CT image votes for where it thinks *e.g.* the left kidney should be. The votes take the form of relative displacement vectors

$$\mathbf{d}(\mathbf{p}) = \left(d^{\mathtt{L}}(\mathbf{p}), d^{\mathtt{R}}(\mathbf{p}), d^{\mathtt{A}}(\mathbf{p}), d^{\mathtt{P}}(\mathbf{p}), d^{\mathtt{H}}(\mathbf{p}), d^{\mathtt{F}}(\mathbf{p})\right) \in \mathbb{R}^6$$

(see fig. 4.12b). The $\mathtt{L}, \mathtt{R}, \mathtt{A}, \mathtt{P}, \mathtt{H}, \mathtt{F}$ symbols are conventional radiological notation and indicate the left, right, anterior, posterior, head and foot directions of the 3D volumetric scan. Some voxels have more influence (because associated with more confident localization predictions) and some less influence on the final prediction. The voxels relative weights are estimated probabilistically via a regression forest.

[4] A more general parametrization is given in [24].

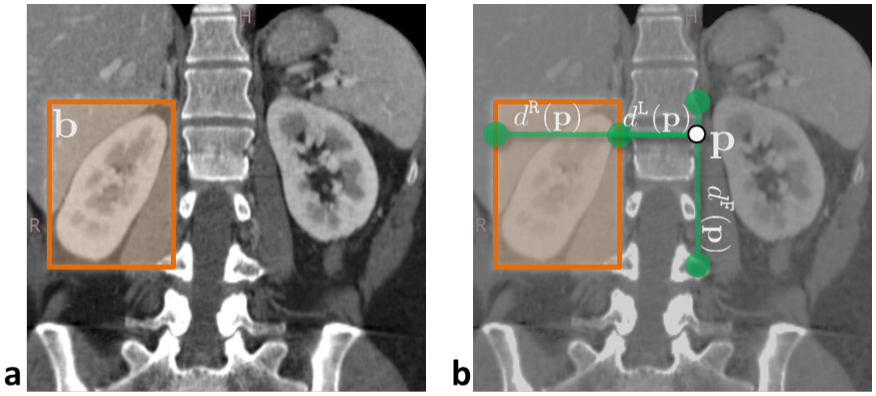

a
b
H
R
b
H
R
$d^{\mathrm{R}}(\mathbf{p})$
$d^{\mathrm{L}}(\mathbf{p})$
p
$d^{\mathrm{F}}(\mathbf{p})$

Fig. 4.12: **Automatic localization of anatomy in 3D CT images. (a)** A coronal view of the abdomen of a patient in a CT scan. The bounding box of the right kidney is shown in orange. **(b)** Each voxel **p** in the volume votes for the position of the six walls of the box via the relative displacements $d^{\mathrm{R}}(\mathbf{p})$, $d^{\mathrm{L}}(\mathbf{p})$, and so on.

For a voxel **p** its feature vector $\mathbf{v}(\mathbf{p}) = (x_1, \ldots, x_i, \ldots, x_d) \in \mathbb{R}^d$ is a collection of differences:

$$x_i = \frac{1}{|\mathbf{B}_i|} \sum_{\mathbf{q} \in \mathbf{B}_i} J(\mathbf{q}). \tag{4.5}$$

where $J(\mathbf{p})$ denotes the density of the tissue in an element of volume at position **p** as measured by the CT scanner (in calibrated Hounsfield Units). The 3D feature box **B** (not to be confused with the output organ bounding box) is displaced from the reference point **p** (see fig. 4.13a). Since for each reference pixel **p** we can look at an infinite number of possible feature boxes ($\forall\ \mathbf{B} \in \mathbb{R}^6$) we have $d = \infty$.

During training we are given a database of CT scans which have been manually labelled with 3D boxes around organs of interest. A regression forest is trained to learn the association of voxel features and bounding box location. Training is achieved by maximizing a continuous information gain as in (4.1). Assuming multivariate Gaussian distributions at the nodes yields the already known form of continuous

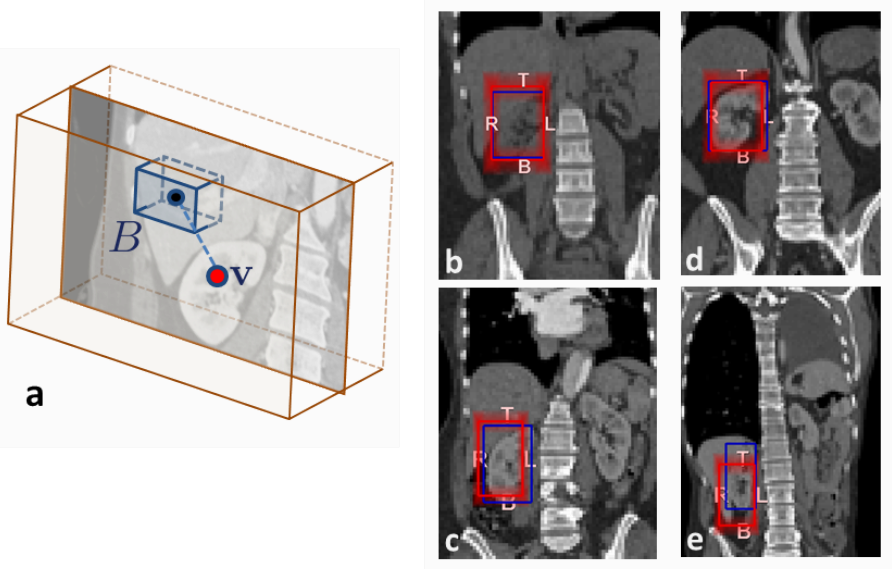


Fig. 4.13: **Features and results. (a)** Feature responses are defined via integral images in displaced 3D boxes, denoted with **B**. **(b,c,d,e)** Some results on four different test patients. The right kidney (red box) is correctly localized in all scans. The corresponding ground-truth is shown with a blue box. Note the variability in position, shape and appearance of the kidney, as well as larger scale variations in patient's body, size, shape and possible anomalies such as the missing left lung, in (e).

information gain:

$$I_j = \log|\Lambda(\mathcal{S}_j)| - \sum_{i \in \{\texttt{L},\texttt{R}\}} \frac{|\mathcal{S}_j^i|}{|\mathcal{S}_j|} \log|\Lambda(\mathcal{S}_j^i)| \tag{4.6}$$

with $\Lambda(\mathcal{S}_j)$ the 6×6 covariance matrix of the relative displacement vector $\mathbf{d}(\mathbf{p})$ computed for all points $\mathbf{p} \in \mathcal{S}_j$. Note that here as a prediction model we are using a multivariate, probabilistic-*constant* model rather than the more general probabilistic-linear one used in the earlier examples. Using the objective function (4.6) encourages the forest to cluster voxels together so as to ensure small determinant of prediction covariances, *i.e.* highly peaked and confident location predictions. In this application, the parameters of a split node j are

$$\boldsymbol{\theta}_j = (\mathbf{B}_j, \tau_j) \in \mathbb{R}^7,$$

with $\mathbf{B}_j$ the "probe" feature box, and τ_j a scalar parameter. Here we use an axis-aligned weak learner model

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

with $\boldsymbol{\phi}(\mathbf{v}, \mathbf{B}_j) = x_j$. The leaf nodes are associated with multivariate-Gaussians as their predictor model. The parameters of such Gaussians are learned during training from all the relative displacements arriving at the leaf.

During testing all voxels of a previously unseen test volume are pushed through all trees in the regression forest until they reach their leaves, and the corresponding Gaussian predictions for the relative displacements are read off. Finally, posteriors over relative displacements are mapped to posteriors over absolute positions [24].

Figure 4.13 shows some illustrative results on the localization of the right kidney in 2D coronal slices. In fig. 4.13e the results are relatively robust to the large anomaly (missing left lung). Results on 3D detections are shown in fig. 4.11b with many more available in the original paper.

An important advantage of decision forests (compared to *e.g.* neural networks) is their interpretability. In fact, in a forest it is possible to look at individual nodes and make sense of what has been learned and why. When using a regression forest for anatomy localization the various tree nodes represent clusters of points. Each cluster predicts the location of a certain organ with more or less confidence. So, we can think of the nodes associated with higher prediction confidence as automatically discovered salient anatomical landmarks. Figure 4.14 shows some such landmark regions when localizing kidneys in a 3D CT scan. More specifically, given a trained regression tree and an input volume, we select one or two leaf nodes with high prediction confidence for a chosen organ class (*e.g.* `l. kidney`). Then, for each sample arriving at the selected leaf nodes, we shade in green the cuboidal regions of the input volume that were used during evaluation of the parent nodes'

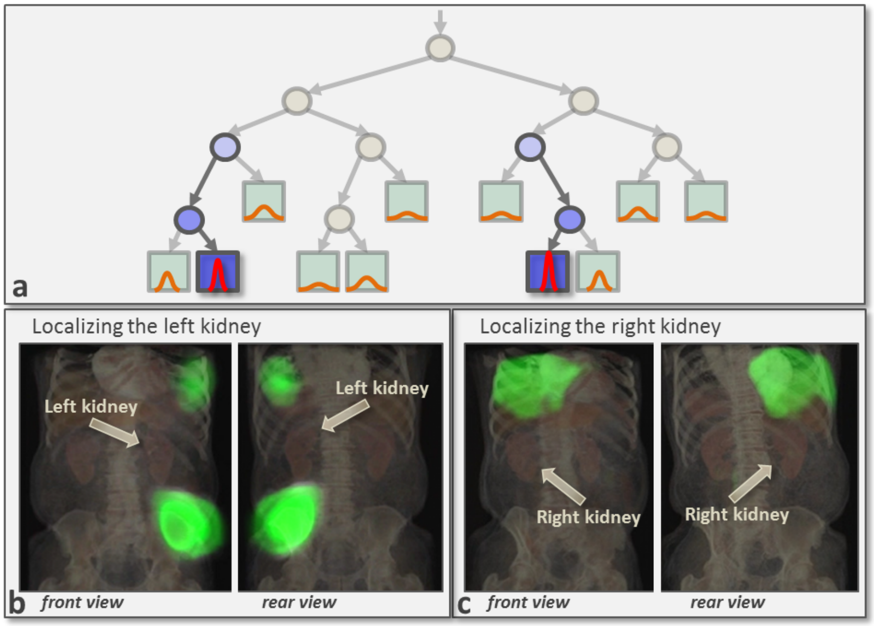


Fig. 4.14: **Automatic discovery of salient anatomical landmarks.** **(a)** Leaves associated with the most peaked densities correspond to clusters of points which predict organ locations with high confidence. **(b)** A 3D rendering of a CT scan and (in green) landmarks automatically selected as salient predictors of the position of the left kidneys. **(c)** Same as in (b) but for the right kidney.

feature tests. Thus, the green regions represent some of the anatomical locations that were used to estimate the location of the chosen organ. In this example, the bottom of the left lung and the top of the left pelvis are used to predict the position of the left kidney. Similarly, the bottom of the right lung is used to localize the right kidney. Such regions correspond to meaningful, visually distinct, anatomical landmarks that have been computed without any manual tagging.

Recently, regression forests were used for anatomy localization in the more challenging full-body, magnetic resonance images [68]. See also [38, 76] for alternative techniques for regressing regions of interest in brain MR images with localization of anatomically salient voxels. The interested reader is invited to browse the *InnerEye* project page [2] for further examples and applications of regression forests to medical image analysis.

5

Density forests

Chapters 3 and 4 have discussed the use of decision forests in supervised tasks, *i.e.* when *labelled* training data is available. In contrast, this chapter discusses the use of forests in unlabelled scenarios.

For instance, one important task is that of discovering the intrinsic nature and structure of large sets of unlabelled data. This task can be tackled via another probabilistic model, density forest. Density forests are explained here as an instantiation of our more abstract decision forest model (described in chapter 2). Given some observed unlabelled data which we assume has been generated from a probabilistic density function we wish to estimate the unobserved underlying generative model itself. More formally, one wishes to learn the density $p(\mathbf{v})$ which has generated the data.

The problem of density estimation is closely related to that of data clustering. Although much research has gone in tree-based clustering algorithms, to our knowledge this is the first time that ensembles of randomized trees are used for density estimation.

We begin with a very brief literature survey, then we show how to adapt the generic forest model to the density estimation task and then discuss advantages and disadvantages of density forests in comparison with alternative techniques.

5.1 Literature on density estimation

The literature on density estimation is vast. Here we discuss only a few representative papers.

Density estimation is closely related to the problem of data clustering, for which an ubiquitous algorithm is k-means [55]. A very successful probabilistic density model is the Gaussian mixture model (GMM), where complex distributions can be approximated via a collection of simple (multivariate) Gaussian components. Typically, the parameters of a Gaussian mixture are estimated via the well known Expectation Maximization algorithm [5]. EM can be thought of as a probabilistic variant of k-means.

Popular, non-parametric density estimation techniques are kernel-based algorithms such as the Parzen-Rosenblatt windows estimator [67]. The advantage of kernel-based estimation over *e.g.* more crude histogram-based techniques is in the added smoothness of the reconstruction which can be controlled by the kernel parameters. Closely related is the k-nearest neighbour density estimation algorithm [5].

In Breiman's seminal work on forests the author mentions using forests for clustering unsupervised data [11]. However, he does it via classification, by introducing dummy additional classes. In contrast, here we use a well defined information gain-based optimization, which fits well within our unified forest model. Forest-based data clustering has been discussed in [61, 83] for computer vision applications.

For further reading on general density estimation techniques the reader is invited to explore the following material [5, 84].

5.2 Specializing the forest model for density estimation

This section specializes the generic forest model introduced in chapter 2 for use in density estimation.

Problem statement. The density estimation task can be summarized as follows:

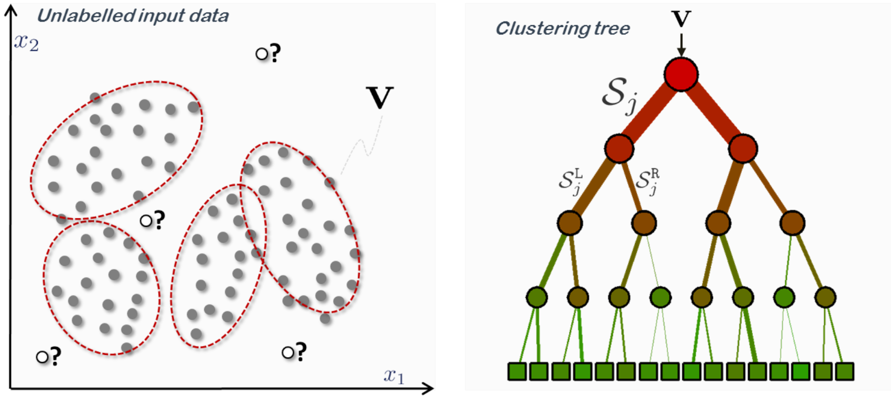


Fig. 5.1: **Input data and density forest training. (a)** Unlabelled data points using for training a density forest are shown as dark circles. White circles indicate previously unseen test data. **(b)** Density forests are ensembles of clustering trees.

> *Given a set of unlabelled observations we wish to estimate the probability density function from which such data has been generated.*

Each input data point $\mathbf{v}$ is represented as usual as a multi-dimensional feature response vector $\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$. The desired output is the entire probability density function $p(\mathbf{v}) \geq 0 \;\; s.t. \int p(\mathbf{v}) d\mathbf{v} = 1$, for any generic input $\mathbf{v}$. An explanatory illustration is shown in fig. 5.1a. Unlabelled training data points are denoted with dark circles, while white circles indicate previously unseen test data.

What are density forests? A density forest is a collection of randomly trained clustering trees (fig. 5.1b). The tree leaves contain simple prediction models such as Gaussians. So, loosely speaking a density forest can be thought of as a generalization of Gaussian mixture models (GMM) with two differences: (i) multiple hard clustered data partitions are created, one by each tree. This is in contrast to the single "soft" clustering generated by the EM algorithm, (ii) the forest posterior is a combination of tree posteriors. So, each input data point is explained by multiple clusters (one per tree). This is in contrast to the single linear combination of Gaussians in a GMM.

These concepts will become clearer later. Next, we delve into a detailed description of the model components, starting with the objective function.

The training objective function. Given a collection of unlabelled points $\{\mathbf{v}\}$ we train each individual tree in the forest independently and if possible in parallel. As usual we employ randomized node optimization. Thus, optimizing the j^{th} split node is done as the following maximization:

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

with the generic information gain I_j defined as:

$$I_j = H(\mathcal{S}_j) - \sum_{i=\mathtt{L},\mathtt{R}} \frac{|\mathcal{S}_j^i|}{|\mathcal{S}_j|} H(\mathcal{S}_j^i) \tag{5.1}$$

In order to fully specify the density model we still need to define the exact form of the entropy $H(\mathcal{S})$ of a set of training points $\mathcal{S}$. Unlike classification and regression, here the are no ground-truth labels. Thus, we need to define an *unsupervised* entropy, *i.e.* one which applies to unlabelled data. As with a GMM, we use the working assumption of multi-variate Gaussian distributions at the nodes. Then, the differential (continuous) entropy of an $d-$variate Gaussian can be shown to be

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

(with Λ the associated $d \times d$ covariance matrix). Consequently, the information gain in (5.1) reduces to

$$I_j = \log(|\Lambda(\mathcal{S}_j)|) - \sum_{i \in \{\mathtt{L},\mathtt{R}\}} \frac{|\mathcal{S}_j^i|}{|\mathcal{S}_j|} \log \left(|\Lambda(\mathcal{S}_j^i)| \right) \tag{5.2}$$

with $|\cdot|$ indicating a determinant for matrix arguments, or cardinality for set arguments.

Motivation. For a set of data points in feature space, the determinant of the covariance matrix is a function of the volume of the ellipsoid corresponding to that cluster. Therefore, by maximizing (5.2) the tree training procedure tends to split the original dataset $\mathcal{S}_0$ into a number of compact clusters. The centres of those clusters tends to be placed in areas of high data density, while the separating surfaces are placed along regions of low density. In (5.2), weighting by the cardinality of children sets avoids splitting off degenerate, single-point clusters.

Finally, our derivation of density-based information gain in (5.2) builds upon an assumption of Gaussian distribution at the nodes. Of course, this is not realistic as real data may be distributed in much more complex ways. However, this assumption is useful in practice as it yields a simple and efficient objective function. Furthermore, the hierarchical nature of the trees allows us to construct very complex distributions by mixing the individual Gaussians associated at the leaves. Alternative measures of "cluster compactness" may also be employed.

The prediction model. The set of leaves in the t^{th} tree in a forest defines a partition of the data such that

$$l(\mathbf{v}) : \mathbb{R}^d \rightarrow \mathcal{L} \subset \mathbb{N}$$

where $l(\mathbf{v})$ denotes the leaf reached (deterministically) by the input point $\mathbf{v}$, and $\mathcal{L}$ the set of all leaves in a given tree (the tree index t is not shown here to avoid cluttering the notation). The statistics of all training points arriving at each leaf node are summarized by a single multi-variate Gaussian distribution $\mathcal{N}(\mathbf{v}; \boldsymbol{\mu}_{l(\mathbf{v})}, \Lambda_{l(\mathbf{v})})$. Then, the output of the t^{th} tree is:

$$p_t(\mathbf{v}) = \frac{\pi_{l(\mathbf{v})}}{Z_t} \mathcal{N}(\mathbf{v}; \boldsymbol{\mu}_{l(\mathbf{v})}, \Lambda_{l(\mathbf{v})}). \tag{5.3}$$

The vector $\boldsymbol{\mu}_l$ denotes the mean of all points reaching the leaf l and Λ_l the associated covariance matrix. The scalar π_l is the proportion of all training points that reach the leaf l, *i.e.* $\pi_l = \frac{|\mathcal{S}_l|}{\mathcal{S}_0}$. Thus (5.3) defines a piece-wise Gaussian density (see fig. 5.2 for an illustration).

Partition function. Note that in (5.3) each Gaussian is truncated by the boundaries of the partition cell associated with the corresponding

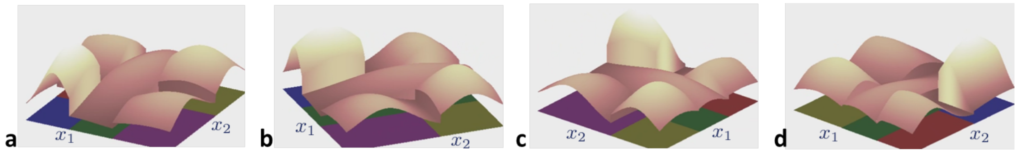


Fig. 5.2: **A tree density is piece-wise Gaussian. (a,b,c,d)** Different views of a tree density $p_t(\mathbf{v})$ defined over an illustrative 2D feature space. Each individual Gaussian component is defined over a bounded domain. See text for details.

leaf (see fig. 5.2). Thus, in order to ensure probabilistic normalization we need to incorporate the partition function Z_t, which is defined as follows:

$$Z_t = \int_{\mathbf{v}} \left(\sum_l \pi_l \, \mathcal{N}(\mathbf{v}; \boldsymbol{\mu}_l, \Lambda_l) \; p(l|\mathbf{v}) \right) \, d\mathbf{v}. \tag{5.4}$$

However, in a density forest each data point reaches exactly *one* terminal node. Thus, the conditional $p(l|\mathbf{v})$ is a delta function $p(l|\mathbf{v}) = [\mathbf{v} \in l(\mathbf{v})]$ and consequently (5.4) becomes

$$Z_t = \int_{\mathbf{v}} \pi_{l(\mathbf{v})} \, \mathcal{N}(\mathbf{v}; \boldsymbol{\mu}_{l(\mathbf{v})}, \Lambda_{l(\mathbf{v})}) \, d\mathbf{v}. \tag{5.5}$$

As it is often the case when dealing with generative models, computing Z_t in high dimensions may be challenging.

In the case of axis-aligned weak learners it is possible to compute the partition function via the cumulative multivariate normal distribution function. In fact, the partition function Z_t is the sum of all the volumes subtended by each Gaussian cropped by its associated partition cell (cuboidal in shape, see fig. 5.2). Unfortunately, the cumulative multivariate normal does not have a close form solution. However, approximating its functional form has is a well researched problem and a number of good numerical approximations exist [39, 71].

For more complex weak-learners it may be easier to approximate Z_t by numerical integration, *i.e.*

$$Z_t \approx \Delta \cdot \sum_i \pi_{l(\mathbf{v}_i)} \, \mathcal{N}(\mathbf{v}_i; \boldsymbol{\mu}_{l(\mathbf{v}_i)}, \Lambda_{l(\mathbf{v}_i)}),$$

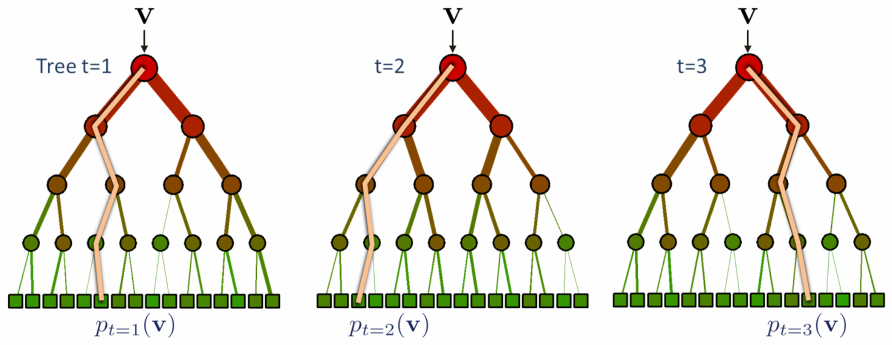


Fig. 5.3: **Density forest: the ensemble model.** A density forest is a collection of clustering trees trained on unlabelled data. The tree density is the Gaussian associated with the leaf reached by the input test point: $p_t(\mathbf{v}) = \frac{\pi_{l(\mathbf{v})}}{Z_t}\mathcal{N}\left(\mathbf{v}; \boldsymbol{\mu}_{l(\mathbf{v})}, \Lambda_{l(\mathbf{v})}\right)$. The forest density is the average of all tree densities: $p(\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} p_t(\mathbf{v})$.

with the points $\mathbf{v}_i$ generated on a finite regular grid with spacing Δ (where Δ represents a length, area, volume etc. depending on the dimensionality of the domain). Smaller grid cells yield more accurate approximations of the partition function at a greater computational cost. Recent, Monte Carlo-based techniques for approximating the partition function are also a possibility [64, 85]. Note that estimating the partition function is necessary only at training time. One may also think of using density forests with a predictor model other than Gaussian.

The ensemble model. The forest density is given by the average of all tree densities

$$p(\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} p_t(\mathbf{v}), \tag{5.6}$$

as illustrated in fig. 5.3.

Discussion. There are similarities and differences between the probabilistic density model defined above and a conventional Gaussian mixture model. For instance, both models are built upon Gaussian components. However, given a single tree an input point $\mathbf{v}$ belongs *deterministically* to only one of its leaves, and thus only one domain-bounded Gaussian component. In a forest with T trees a point $\mathbf{v}$ belongs to T components, one per tree. The ensemble model (5.6) induces a uniform "mixing" across the different trees. The benefits of such forest-based mixture model will become clearer in the next section. The parameters of a GMM are typically learned via Expectation Maximization (EM). In contrast, the parameters of a density forest are learned via a hierarchical information gain maximization criterion. Both algorithms may suffer from local minima.

5.3 Effect of model parameters

This section studies the effect of the forest model parameters on the accuracy of density estimation. We use many illustrative, synthetic examples, designed to bring to life different properties, advantages and disadvantages of density forests compared to alternative techniques. We begin by investigating the effect of two of the most important parameters: the tree depth D and the forest size T.

5.3.1 The effect of tree depth

Figure 5.4 presents first density forest results. Figure 5.4a shows some unlabelled points used to train the forest. The points are randomly drawn from two 2D Gaussian distributions.

Three different density forests have been trained on the same input set with $T = 200$ and varying tree depth D. In all cases the weak learner model was of the axis-aligned type. Trees of depth 2 (stumps) produce a binary partition of the training data which, in this simple example, produce perfect separation. As usual the trees are all slightly different from one another, corresponding to different decision boundaries (not shown in the figure). In all cases each leaf is associated with a bounded Gaussian distribution learned from the training points arriving at the leaf itself. We can observe that deeper trees (*e.g.* for $D = 5$) tend to create further splits and smaller Gaussians, leading to over-fitting on this simple dataset. Deeper trees tend to "fit to the noise" of the training data, rather than capture the underlying nature of the data.

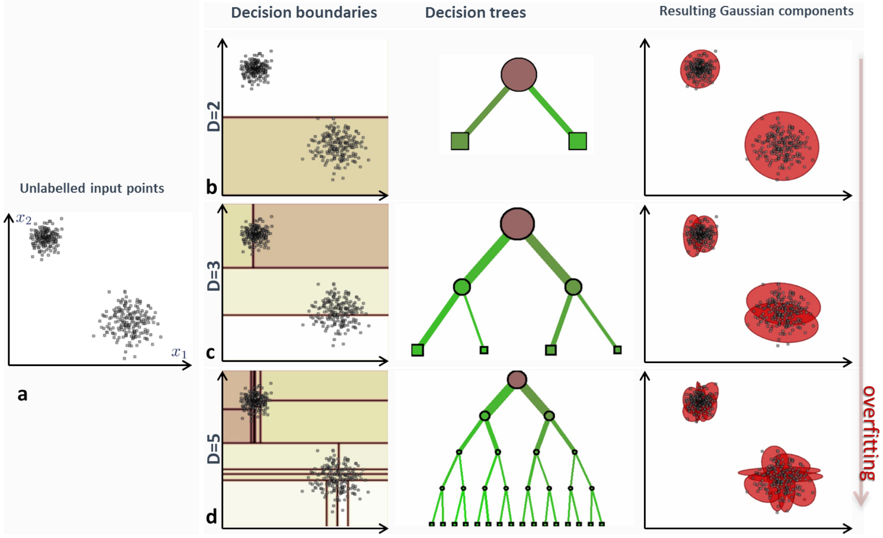


Fig. 5.4: **The effect of tree depth on density. (a)** Input unlabelled data points in a 2D feature space. **(b,c,d)** Individual trees out of three density forests trained on the same dataset, for different tree depths D. A forest with unnecessarily deep trees tends to fit to the training noise, thus producing very small, high-frequency bumps in the density.

In this simple example $D = 2$ (top row) produces the best results.

5.3.2 The effect of forest size

Figure 5.5 shows the output of six density forests trained on the input data in fig. 5.4a for two different values of T and three values of D. The images visualize the output density $p(\mathbf{v})$ computed for all points in a square subset of the feature space. Dark pixels indicate low values and bright pixels high values of density.

We observe that even if individual trees heavily over-fit (*e.g.* for $D = 6$), the addition of further trees tends to produce smoother densities. This is thanks to the randomness of each tree density estimation and reinforces once more the benefits of a forest ensemble model. The tendency of larger forests to produce better generalization has been

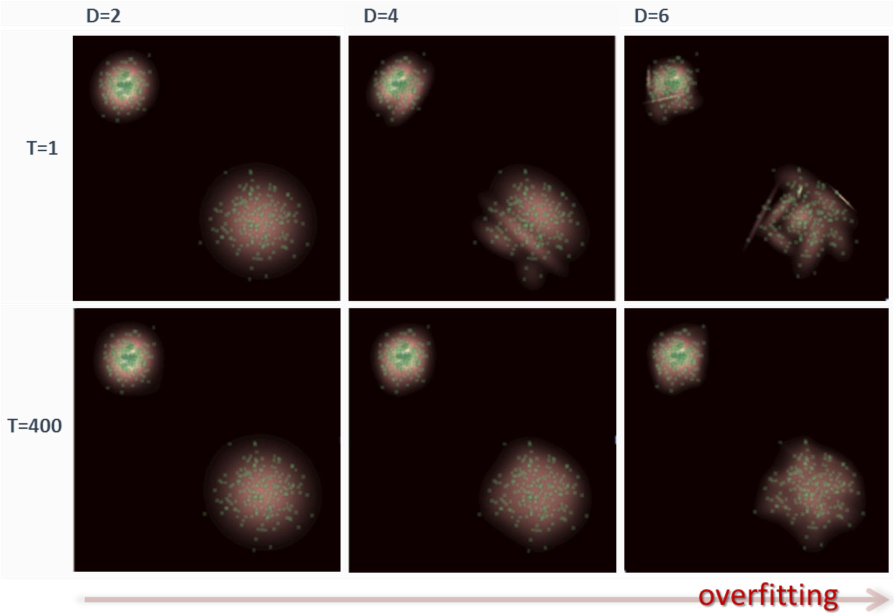


Fig. 5.5: **The effect of forest size on density.** Densities $p(\mathbf{v})$ for six density forests trained on the same unlabelled dataset for varying T and D. Increasing the forest size T *always* improves the smoothness of the density and the forest generalization, even for deep trees.

observed also for classification and regression and it is an important characteristic of forests. Since increasing T always produces better results (at an increased computational cost) in practical applications we can just set T to a "sufficiently large" value, without worrying too much about optimizing its value.

5.3.3 More complex examples

A more complex example is shown in fig. 5.6. The noisy input data is organized in the shape of a four-arm spiral (fig. 5.6a). Three density forests are trained on the same dataset with $T = 200$ and varying depth D. The corresponding densities are shown in fig. 5.6b,c,d. Here, due to the greater complexity of the input data distribution shallower

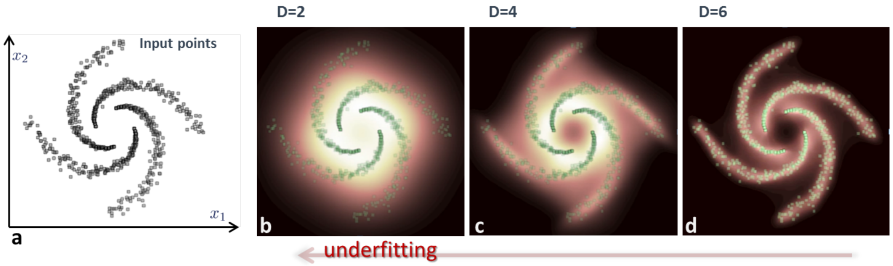


Fig. 5.6: **Density forest applied to a spiral data distribution. (a)** Input unlabelled data points in their 2D feature space. **(b,c,d)** Forest densities for different tree depths D. The original training points are overlaid in green. The complex distribution of input data is captured correctly by a deeper forest, *e.g.* $D = 6$, while shallower trees produce under-fitted, overly smooth densities.

trees yield under-fitting, *i.e.* overly smooth and detail-lacking density estimates. In this example good results are obtained for $D = 6$ as the density nicely captures the individuality of the four spiral arms while avoiding fitting to high frequency noise. Just like in classification and regression here too the parameter D can be used to set a compromise between smoothness of output and the ability to capture structural details.

So far we have described the density forest model and studied some of its properties on synthetic examples. Next we study density forests in comparison to alternative algorithms.

5.4 Comparison with alternative algorithms

This section discusses advantages and disadvantages of density forests as compared to the most common parametric and non-parametric density estimation techniques.

5.4.1 Comparison with non-parametric estimators

Figure 5.7 shows a comparison between forest density, Parzen window estimation and k-nearest neighbour density estimation. The comparison is run on the same three datasets of input points. In the first experiments points are randomly drawn from a five-Gaussian mixture. In the second they are arranged along an "S" shape and in the third they are arranged along four short spiral arms. Comparison between the forest densities in fig. 5.7b and the corresponding non-parametric densities in fig. 5.7c,d clearly shows much smoother results for the forest output. Both the Parzen and the nearest neighbour estimators produce artifacts due to hard choices of *e.g.* the Parzen window bandwidth or the number k of neighbours. Using heavily optimized single trees would also produce artifacts. However, the use of many trees in the forest yields the observed smoothness.

A quantitative assessment of the density forest model is presented at the end of this chapter. Next, we compare (qualitatively) density forests with variants of the Gaussian mixture model.

5.4.2 Comparison with GMM EM

Figure 5.8 shows density estimates produced by forests in comparison to various GMM-based densities for the same input datasets as in fig. 5.7a. Figure 5.7b shows the (visually) best results obtained with a GMM, using EM for its parameter estimation [5]. We can observe that on the simpler 5-component dataset (experiment 1) the two models work equally well. However, the "S" and spiral-shaped examples show very distinct blob-like artifacts when using the GMM model. One may argue that this is due to the use of too few components. So we increased their number k and the corresponding densities are shown in fig. 5.7c. Artifacts still persist. Some of them are due to the fact that the greedy EM optimization gets stuck in local minima. So, a further alternative to improve the GMM results is to add randomness. In fig. 5.7c, for each example we have trained 400 GMM-EM models (trained with 400 random initializations, a common way of injecting randomness in GMM training) and averaged together their output to produce a single density (as shown in the figure). The added randomness produces benefits in terms of smoothness, but the forest densities are still slightly superior, especially for the spiral dataset.

In summary, our synthetic experiments confirm that the use of ran-

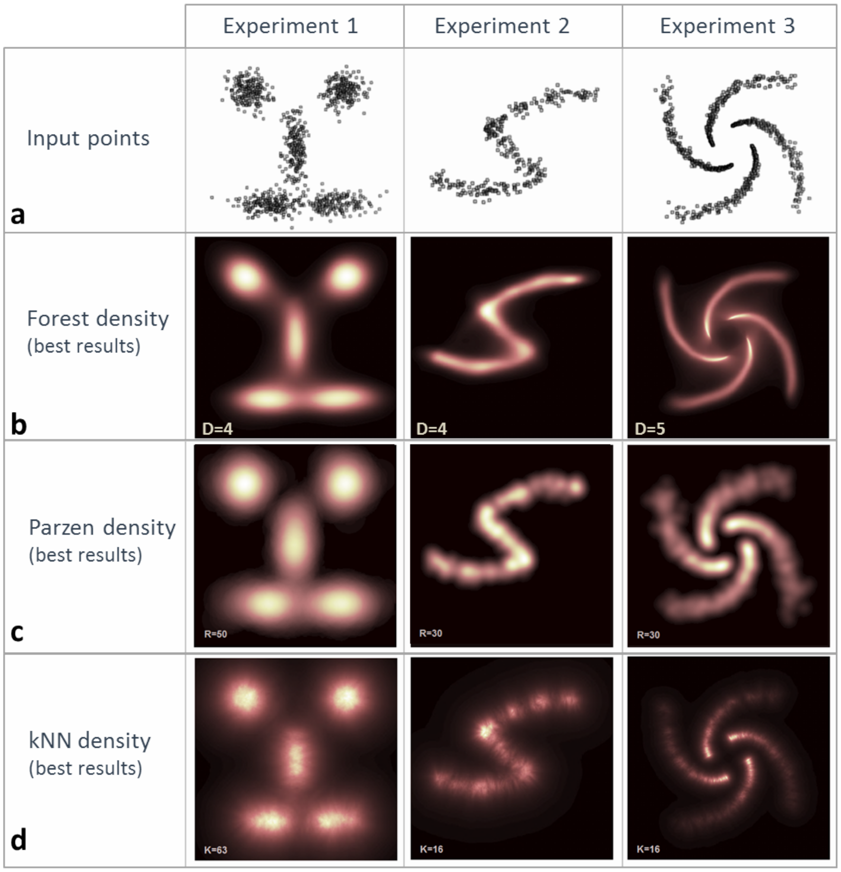


Fig. 5.7: **Comparison between density forests and non parametric estimators. (a)** Input unlabelled points for three different experiments. **(b)** Forest-based densities. Forests were computed with $T = 200$ and varying depth D. **(c)** Parzen window densities (with Gaussian kernel). **(d)** K-nearest neighbour densities. In all cases parameters were optimized to achieve the best possible results. Notice the abundant artifacts in (c) and (d) as compared to the smoother forest estimates in (b).

domness (either in a forest model or in a Gaussian mixture model) yields improved results. Possible issues with EM getting stuck in local minima produce artifacts which appear to be mitigated in the forest model. Let us now look at differences in terms of computational cost.

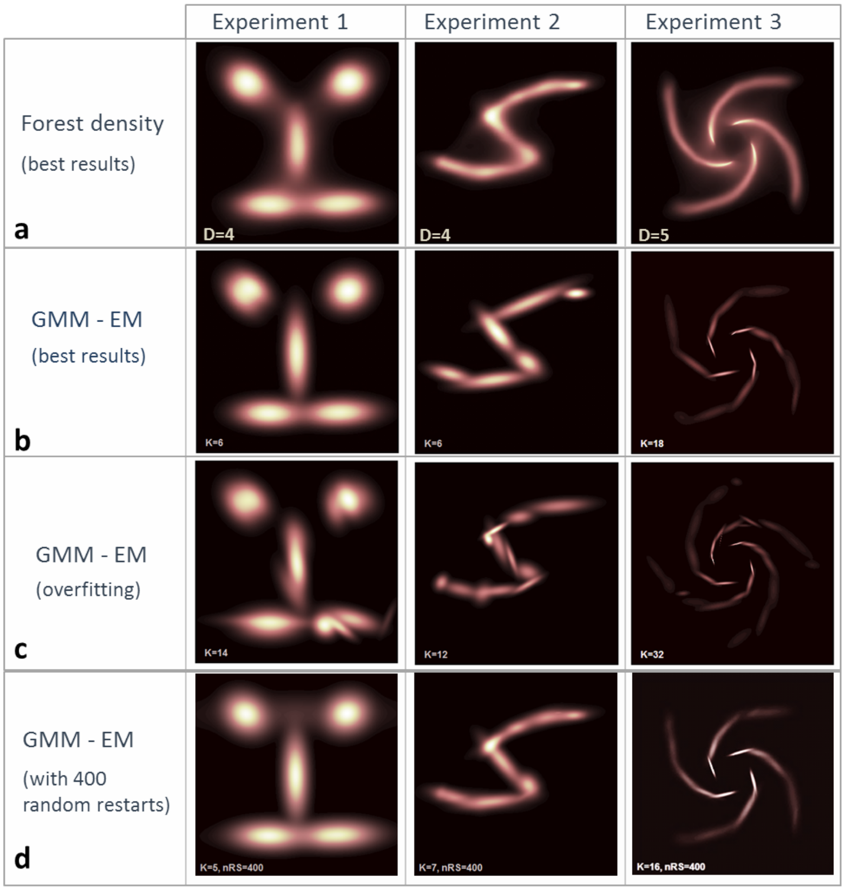


Fig. 5.8: **Comparison with GMM EM (a)** Forest-based densities. Forests were computed with $T = 200$ and optimized depth D. **(b)** GMM density with a relatively small number of Gaussian components. The model parameters are learned via EM. **(c)** GMM density with a larger number of Gaussian components. Increasing the components does not remove the blob-like artifacts. **(d)** GMM density with multiple (400) random re-initializations of EM. Adding randomness to the EM algorithm improves the smoothness of the output density considerably. The results in (a) are still visually smoother.

Comparing computational complexity. Given an input test point $\mathbf{v}$ evaluating $p(\mathbf{v})$ under a random-restart GMM model has cost

$$R \times T \times G, \tag{5.7}$$

with R the number of random restarts (the number of trained GMM models in the set), T the number of Gaussian components and G the cost of evaluating $\mathbf{v}$ under each individual Gaussian.

Similarly, estimating $p(\mathbf{v})$ under a density forest with T trees of maximum depth D has cost

$$T \times G \; + \; T \times D \times B, \tag{5.8}$$

with B the cost of a binary test at a split node.

The cost in (5.8) is an upper bound because the average length of a generic root-leaf path is less than D nodes. Depending on the application, the binary tests can be extremely efficient to compute[1], thus we may be able to ignore the second term in (5.8). In this case the cost of testing a density forest becomes comparable to that of a conventional, single GMM (with T components).

Comparing training costs between the two models is a little harder because it involves the number of EM iterations (in the GMM model) and the value of ρ (in the forest). In practical applications (especially real-time ones) minimizing the testing time is more important than reducing the training time. Finally, testing of both GMM as well as density forests can be easily parallelized.

5.5 Sampling from the generative model

The density distribution $p(\mathbf{v})$ we learn from the unlabelled input data represents a probabilistic generative model. In this section we describe an algorithm for the efficient sampling of random data under the learned model. The sampling algorithm uses the structure of the forest itself (for efficiency) and proceeds as described in algorithm 5.1. See also fig. 5.9 for an accompanying illustration.

In this algorithm for each sample a random path from a tree root to one of its leaves is randomly generated and then a feature vector randomly generated from the associated Gaussian. Thus, drawing one random sample involves generating at most D random numbers from uniform distributions plus sampling a d-dimensional vector from

[1] A weak learner binary stump is applied usually only to a small, selected subset of features $\boldsymbol{\phi}(\mathbf{v})$ and thus it can be computed very efficiently.

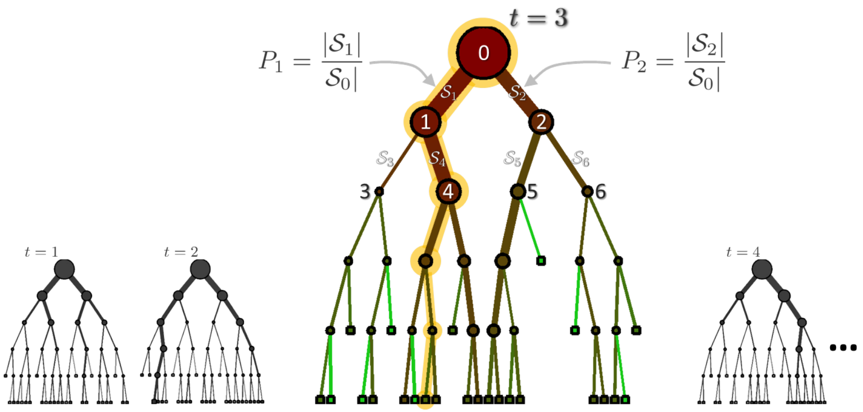


Fig. 5.9: **Drawing random samples from the generative density model.** Given a trained density forest we can generate random samples by: i) selecting one of the component trees, ii) randomly navigating down to a leaf and, iii) drawing a sample from the associated Gaussian. The precise algorithmic steps are listed in algorithm 5.1.

Given a density forest with T trees:

(1) Draw uniformly a random tree index $t \in \{1, , T\}$ to select a single tree in the forest.

(2) Descend the tree

 (a) Starting at the root node, for each split node randomly generate the child index with probability proportional to the number of training points in edge (proportional to the edge thickness in fig. 5.9);

 (b) Repeat step 2 until a leaf is reached.

(3) At the leaf draw a random sample from the *domain bounded* Gaussian stored at that leaf.

Algorithm 5.1: **Sampling from the density forest model.**

a Gaussian.

An equivalent and slightly faster version of the sampling algorithm is obtained by compounding all the probabilities associated with individual edges at different levels together as probabilities associated with

Given a set of R GMMs learned with random restarts:

(1) Draw uniformly a GMM index $r \in \{1, , R\}$ to select a single GMM in the set.

(2) Select one Gaussian component by randomly drawing in proportion to the associated priors.

(3) Draw a random sample from the selected Gaussian component.

Algorithm 5.2: **Sampling from a random-restart GMM.**

the leaves only. Thus, the tree traversal step (step 2 in algorithm 5.2) is replaced by direct random selection of one of the leaves.

Efficiency. The cost of randomly drawing N samples under the forest model is

$$N \times (1 + 1) \times J \; + \; N \times G$$

with J the cost (almost negligible) of randomly generating an integer number and G the cost of drawing a d-dimensional vector from a Gaussian distribution.

For comparison, sampling from a random-restart GMM is illustrated in the algorithm 5.2. The cost of drawing samples under a GMM model is also

$$N \times (1 + 1) \times J \; + \; N \times G$$

It is interesting to see how although the two algorithms are built upon different data structures, their steps are very similar. Their theoretical complexity is the same.

In summary, despite the added richness in the hierarchical structure of the density forest its sampling complexity is very much comparable to that of a random-restart GMM.

Results. Figure 5.10 shows results of sampling $10,000$ random points from density forest trained on five different input datasets. The top row of the figure shows the densities on a 2D feature space. The bottom row shows (with small red dots) random points drawn from the corresponding forests via the algorithm described in algorithm 5.1. Such

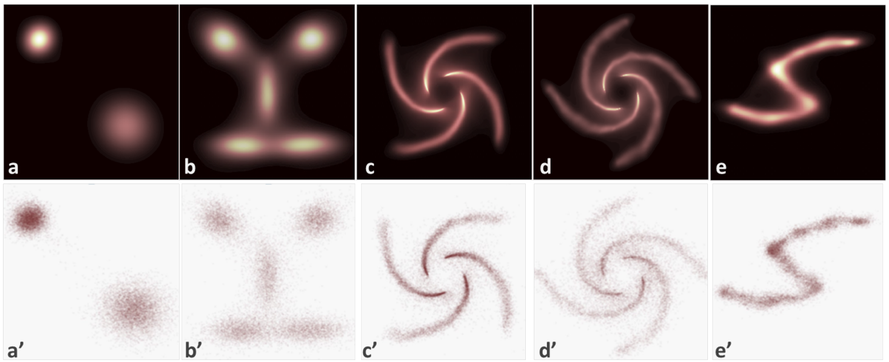


Fig. 5.10: **Sampling results (Top row)** Densities learned from hundreds of training points, via density forests. **(Bottom row)** Random points generated from the learned forests. We draw 10,000 random points per experiment (different experiments in different columns).

a simple algorithm produces good results both for simpler, Gaussian-mixture distributions (figs. 5.10a,b) as well as more complex densities like spirals and other convolved shapes (figs. 5.10c,d,e).

5.6 Dealing with non-function relations

Chapter 4 concluded by showing shortcomings of regression forests trained on inherently ambiguous training data, *i.e.* data such that for a given value of x there may be multiple corresponding values of y (a *relation* as opposed to a *function*). This section shows how better predictions may be achieved in ambiguous settings by means of density forests.

5.6.1 Regression from density

In fig. 4.10b a regression forest was trained on ambiguous training data. The corresponding regression posterior $p(y|x)$ yielded a very large uncertainty in the ambiguous, central region. However, despite its inherent ambiguity, the training data shows some interesting, multi-modal structure that if modelled properly could increase the prediction confidence

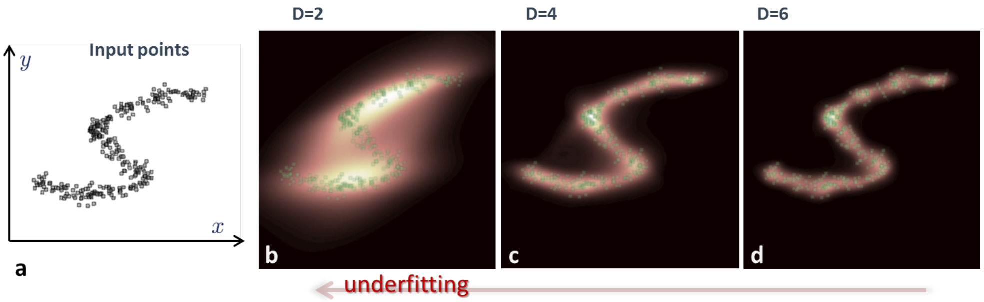


Fig. 5.11: **Training density forest on a "non-function" dataset.** **(a)** Input *unlabelled* training points on a 2D space. **(b,c,d)** Three density forests are trained on such data, and the corresponding densities shown in the figures. Dark pixels correspond to small density and vice-versa. The original points are overlaid in green. Visually reasonable results are obtained in this dataset for $D = 4$.

(see also [63]).

We repeat a variant of this experiment in fig. 5.11. However, this time a *density* forest is trained on the "S-shaped" training set. In contrast to the regression approach in chapter 4, here the data points are represented as pairs (x, y), with both dimensions treated equally as *input features*. Thus, now the data is thought of as *unlabelled*. Then, the joint generative density function $p(x, y)$ is estimated from the data. The density forest for this 2D dataset remains defined as

$$p(x, y) = \frac{1}{T}\sum_{t=1}^{T} p_t(x, y)$$

with t indexing the trees. Individual tree densities are

$$p_t(x, y) = \frac{\pi_l}{Z_t}\mathcal{N}\left((x, y); \boldsymbol{\mu}_l, \Lambda_l\right),$$

where $l = l(x, y)$ denotes the leaf reached by the point (x, y). For each leaf l in the t^{th} tree we have $\pi_l = |\mathcal{S}_l|/|\mathcal{S}_0|$, the mean $\boldsymbol{\mu}_l = (\mu_x, \mu_y)$ and the covariance

$$\Lambda_l = \begin{pmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 \\ \sigma_{xy}^2 & \sigma_{yy}^2 \end{pmatrix}.$$

In fig. 5.11 we observe that a forest with $D = 4$ produces a visually smooth, artifact-free density. Shallower or deeper trees produce under-fitting and over-fitting, respectively. Now, for a previously unseen, input test point $\mathbf{v} = (x, y)$ we can compute its probability $p(\mathbf{v})$. However, in regression, at test time we only know the independent variable x, and its associated y is unknown (y is the quantity we wish to regress/estimate). Next we show how we can exploit the known generative density $p(x, y)$ to predict the regression conditional $p(y|x)$.

Figure 5.12a shows the training points and an input value for the independent variable $x = x^*$. Given the trained density forest and x^* we wish to estimate the conditional $p(y|x = x^*)$. For this problem we make the further assumption that the forest has been trained with axis-aligned weak learners. Therefore, some split nodes act *only* on the x coordinate (namely x-nodes) and others *only* on the y coordinate (namely y-nodes). Figure 5.12b illustrates this point. When testing a tree on the input $x = x^*$ the y-nodes cannot apply the associated split function (since the value of y is unknown). In those cases the data point is sent to *both* children. In contrast, the split function associated to the x-nodes is applied as usual and the data sent to the corresponding single child. So, in general multiple leaf nodes will be reached by a single input (see the bifurcating orange paths in fig. 5.12b). As shown in fig. 5.12c this corresponds to selecting multiple, contiguous cells in the partitioned space, so as to cover the entire y range (for a fixed x^*).

So, along the line $x = x^*$ several Gaussians are encountered, one per leaf (see fig. 5.12d and fig. 5.13). Consequently, the tree conditional is piece-wise Gaussian and defined as follows:

$$p_t\left(y|x = x^*\right) = \frac{1}{Z_{t,x^*}} \sum_{l \in \mathcal{L}_{t,x^*}} [y_l^{\mathtt{B}} \le y < y_l^{\mathtt{T}}]\ \pi_l\, \mathcal{N}\left(y; \mu_{y|x,l}, \sigma^2_{y|x,l}\right) \quad (5.9)$$

with the leaf conditional mean $\mu_{y|x,l} = \mu_y + \frac{\sigma^2_{xy}}{\sigma^2_{yy}}(x^* - \mu_x)$ and variance $\sigma^2_{y|x,l} = \sigma^2_{yy} - \frac{\sigma^4_{xy}}{\sigma^2_{xx}}$. In (5.9) $\mathcal{L}_{t,x^*}$ denotes the subset of all leaves in the tree t reached by the input point x^* (three leaves out of four in the example in the figure).

The conditional partition function Z_{t,x^*} ensures normalization, *i.e.*

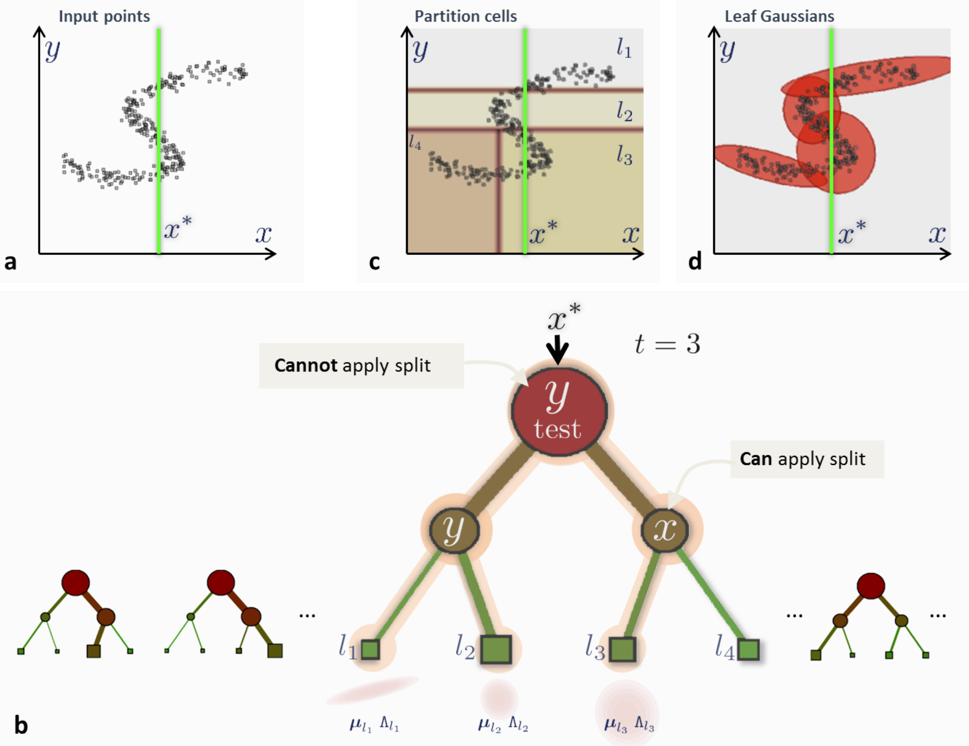


Fig. 5.12: **Regression from density forests.** **(a)** 2D training points are shown in black. The green vertical line denotes the value x^* of the independent variable. We wish to estimate $p(y|x = x^*)$. **(b)** When testing a tree on the input x^* some split nodes cannot apply their associated split function and the data is sent to both children (see orange paths). **(c)** The line $x = x^*$ intersects multiple cells in the partitioned feature space. **(d)** The line $x = x^*$ intersects multiple leaf Gaussians. The conditional output is a combination of those Gaussians.

$\int_y p_t(y|x = x^*)\, dy = 1$, and can be computed as follows:

$$Z_{t,x^*} = \sum_{l \in \mathcal{L}_{t,x^*}} \pi_l \left(\phi_{t,l}(y_l^{\mathsf{T}}|x = x^*) - \phi_{t,l}(y_l^{\mathsf{B}}|x = x^*) \right)$$

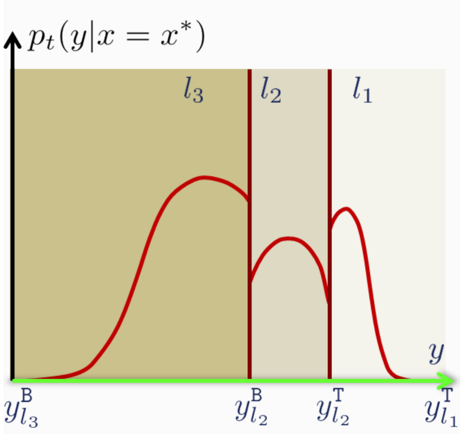


Fig. 5.13: **The tree conditional is a piece-wise Gaussian.** See text for details.

with ϕ denoting the 1D cumulative normal distribution function

$$\phi_{t,l}(y|x=x^*) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{y-\mu_{y|x,l}}{\sqrt{2\sigma^2_{y|x,l}}}\right)\right].$$

Finally, the forest conditional is:

$$p(y|x=x^*) = \frac{1}{T}\sum_{t=1}^{T} p_t(y|x=x^*)$$

Figure 5.14 shows the forest conditional distribution computed for five fixed values of x. When comparing *e.g.* the conditional $p(y|x=x_3)$ in fig. 5.14 with the distribution in 4.10b we see that now the conditional shows three very distinct modes rather than a large, uniformly uninformative mass. Although some ambiguity remains (it is inherent in the training set) now we have a more precise description of such ambiguity.

5.6.2 Sampling from conditional densities

We conclude this chapter by discussing the issue of efficiently drawing random samples from the conditional model $p(y|x)$.

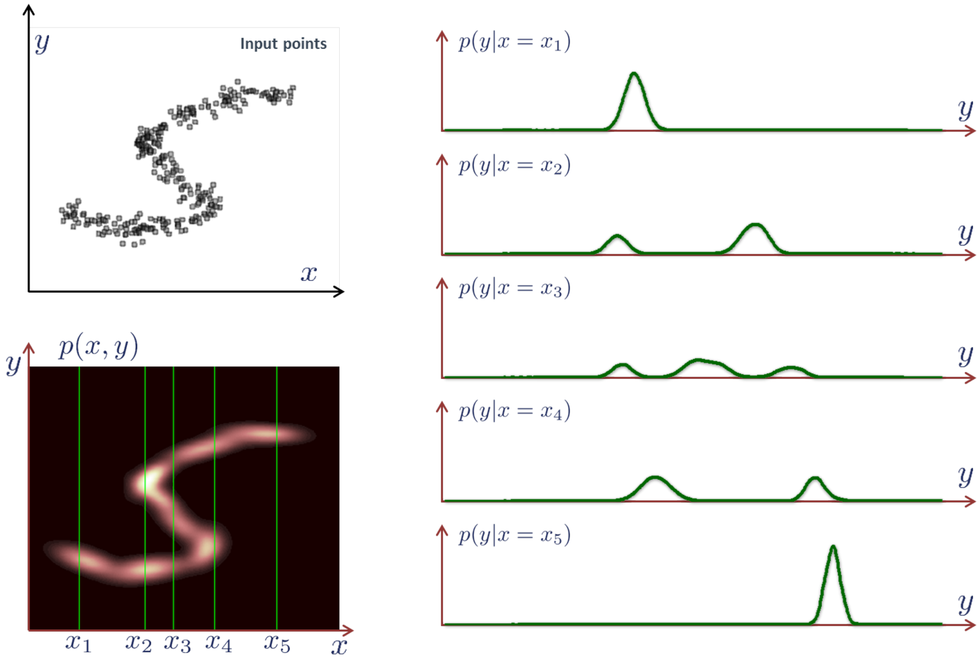


Fig. 5.14: **Regression from density forests.** The conditionals $p(y|x = x_i)$ show multimodal behaviour. This is an improvement compared to regression forests.

Given a fixed and known $x = x^*$ we would like to sample different random values of y distributed according to the conditional $p(y|x = x^*)$. Like in the previous version we assume available a density forest which has been trained with axis-aligned weak learners (fig. 5.15). The necessary steps are described in Algorithm 5.3.

Each iteration of Algorithm 5.3 produces a value y drawn randomly from $p(y|x = x^*)$. Results on our synthetic example are shown in fig. 5.16, for five fixed values of the independent variable x. In fig. 5.16b darker regions indicate overlapping sampled points. Three distinct clusters of points are clearly visible along the $x = x_3$ line, two clusters along the $x = x_2$ and along the $x = x_4$ lines and so on. This algorithm extends to more than two dimensions. As expected, the quality of the sampling depends on the usual parameters such as the tree depth D,

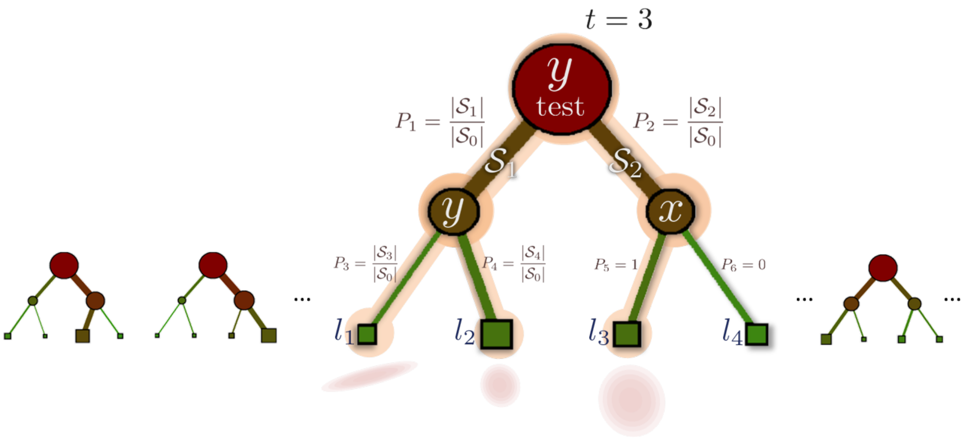

$t = 3$
y
test
$P_1 = \frac{|\mathcal{S}_1|}{|\mathcal{S}_0|}$
$P_2 = \frac{|\mathcal{S}_2|}{|\mathcal{S}_0|}$
$\mathcal{S}_1$
$\mathcal{S}_2$
y
x
$P_3 = \frac{|\mathcal{S}_3|}{|\mathcal{S}_0|}$
$P_4 = \frac{|\mathcal{S}_4|}{|\mathcal{S}_0|}$
$P_5 = 1$
$P_6 = 0$
l_1
l_2
l_3
l_4
...
...
...

Fig. 5.15: **Sampling from conditional model.** Since x is known and y unknown y-nodes cannot apply the associated split function. When sampling from such a tree a child of a y-node is chosen randomly. Instead, the child of an x-node is selected deterministically. See text for details.

the forest size T, the amount of training randomness ρ *etc.*

5.7 Quantitative analysis

This section assesses the accuracy of the density estimation algorithm with respect to ground-truth. Figure 5.17a shows a ground-truth probability density function. The density is represented non-parametrically as a normalized histogram defined over the 2D (x_1, x_2) domain.

Given the ground-truth density we randomly sample $5,000$ points numerically (fig. 5.17b), via the multivariate inverse probability integral transform algorithm [26]. The goal now is as follows: Given the sampled points only, reconstruct a probability density function which is as close as possible to the ground-truth density.

Thus, a density forest is trained using the sampled points alone. No use is made of the ground-truth density in this stage. Given the trained forest we test it on all points in a predefined domain (not just on the training points, fig. 5.17c). Finally, a quantitative comparison between

```
Given a density forest with $T$ trees trained with axis-aligned weak learners and an input value $x = x^*$:
    (1) Sample uniformly $t \in \{1, \cdots, T\}$ to select a tree in the forest.
    (2) Starting at the root node descend the tree by:
          • at $x$-nodes applying the split function and following the corresponding branch.
          • at a $y$-node $j$ random sample one of the two children according to their respective probabilities: $P_{2j+1} = \frac{|\mathcal{S}_{2j+1}|}{|\mathcal{S}_j|}$, $P_{2j+2} = \frac{|\mathcal{S}_{2j+2}|}{|\mathcal{S}_j|}$.
    (3) Repeat step 2 until a (single) leaf is reached.
    (4) At the leaf sample a value $y$ from the *domain bounded* 1D conditional $p(y|x = x^*)$ of the 2D Gaussian stored at that leaf.
```

Algorithm 5.3: **Sampling from conditionals via a forest.**

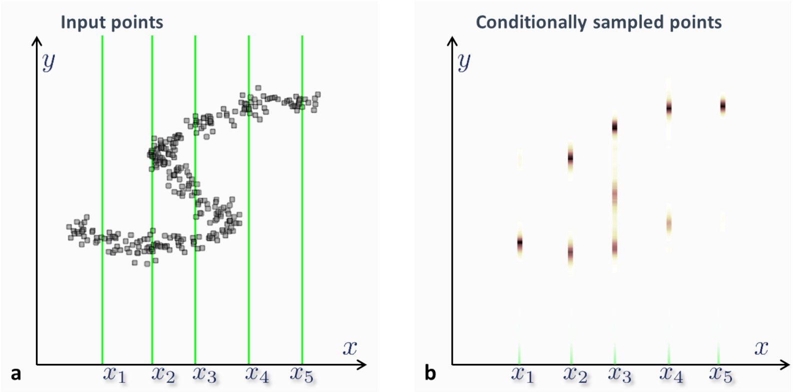
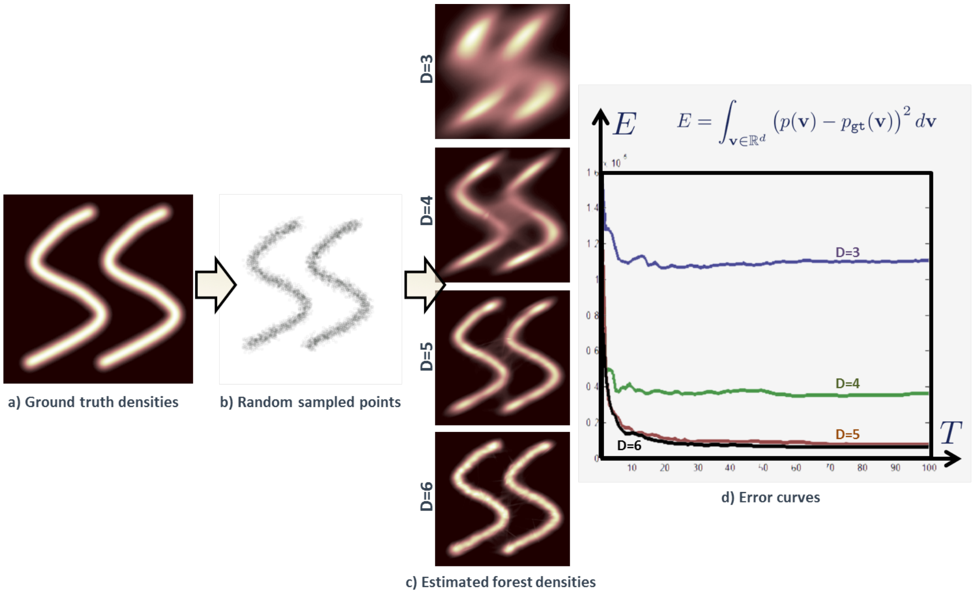


Fig. 5.16: **Results on conditional point sampling.** Tens of thousands of random samples of y are drawn for five fixed positions in x following algorithm 5.3. In (b) the multimodal nature of the underlying conditional becomes apparent from the empirical distribution of the samples.

the estimated density ($p(\mathbf{v})$) and the ground-truth one ($p_{\mathtt{gt}}(\mathbf{v})$) can be carried out. The density reconstruction error is computed here as a



a) Ground truth densities

b) Random sampled points

c) Estimated forest densities

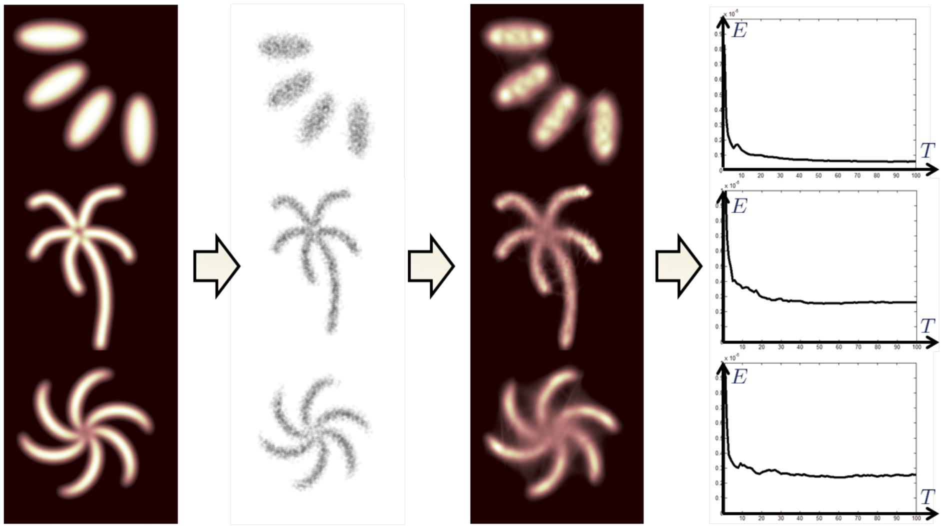
d) Error curves

Fig. 5.17: **Quantitative evaluation of forest density estimation.** **(a)** An input ground-truth density (non-Gaussian in this experiment). **(b)** Thousands of random points drawn randomly from the density. The points are used to train four density forests with different depths. **(c)** During testing the forests are used to estimate density values for all points in a square domain. **(d)** The reconstructed densities are compared with the ground-truth and error curves plotted as a function of the forest size T. As expected, larger forests yield higher accuracy. In these experiments we have used four forests with $T = 100$ trees and $D \in \{3, 4, 5, 6\}$.

sum of squared differences:

$$E = \sum_{\mathbf{v}} \left(p(\mathbf{v}) - p_{\mathtt{gt}}(\mathbf{v})\right)^2 \qquad (5.10)$$

Alternatively one may consider the technique in [90]. Note that due to probabilistic normalization the maximum value of the error in (5.10) is 4. The curves in fig. 5.17d show how the reconstruction error diminishes with increasing forest size and depth. Unsurprisingly, in our



a) Ground truth densities b) Random sampled points c) Estimated forest densities d) Error curves

Fig. 5.18: **Quantitative evaluation, further results. (a)** Input ground-truth densities. **(b)** Thousands of points sampled randomly from the ground-truth densities. **(c)** Densities estimated by the forest. Density values are computed for all points in the domain (not just the training points). **(d)** Error curves as a function of the forest size T. As expected a larger forest yields better accuracy. These results are obtained with $T = 100$ and $D = 5$. Different parameter values and using richer weak learners may improve the accuracy in troublesome regions (*e.g.* at the centre of the spiral arms).

experiments we have observed the overall error to start increasing again after an optimal value of D (suggesting overfitting).

Figure 5.18 shows further quantitative results on more complex examples. In the bottom two examples some difficulties arise in the central part (where the spiral arms converge). This causes larger errors. Using different weak learners (*e.g.* curved surfaces) may produce better results in those troublesome areas.

Density forests are the backbone of manifold learning and semi-supervised learning techniques, described next.

6

Manifold forests

The previous chapter has discussed the use of decision forests for modeling the density of unlabelled data. This has led to an efficient probabilistic generative model which captures the intrinsic structure of the data itself.

This chapter delves further into the issue of learning the structure of high-dimensional data as well as mapping it onto a much lower dimensional space, while preserving relative distances between data points. This task goes under the name of manifold learning and is closely related to dimensionality reduction and embedding.

This task is important because real data is often characterized by a very large number of dimensions. However, a careful inspection often shows a much lower dimensional underlying distribution (*e.g.* on a hyper-plane, or a curved surface *etc.*). So, if we can automatically discover the underlying manifold and "unfold" it, this may lead to easier data interpretation as well as more efficient algorithms for handling such data.

Here we show how decision forests can be used also for manifold learning. Advantages with respect to other techniques include: (i) computational efficiency (due to ease of parallelization of forest-based algorithms), (ii) automatic selection of discriminative features via information-based energy optimization, (iii) being part of a more general forest model and, in turn code re-usability, and (iv) automatic estimation of the optimal dimensionality of the target space (this is in common with other spectral techniques). After a brief literature survey we discuss details of the manifold forest model and then show its properties with examples and experiments.

6.1 Literature on manifold learning

Discovering the intrinsic structure of a dataset (manifold learning) and mapping it onto a lower dimensional representation (dimensionality reduction or embedding) are related problems which have been investigated at length in the literature. The simplest algorithm is "principal component analysis" (PCA) [48]. PCA is based on the computation of directions of maximum data spread. This is obtained simply by eigendecomposition of the data covariance matrix computed in the original space. Therefore, PCA is a linear model and as such has considerable limitations for more realistic problems. A popular, nonlinear technique is "isometric feature mapping" (or IsoMap) [92] which estimates low dimensional embeddings that tend to preserve geodesic distances between point pairs.

Manifold forests build upon "Laplacian eigenmaps" [4] which is a technique derived from spectral graph theory. Laplacian eigenmaps try to preserve *local* pairwise point distances only, with a simple and efficient algorithm. This technique has very close connections with spectral clustering and the normalized cuts image segmentation algorithm in [81]. Recent probabilistic interpretation of spectral dimensionality reduction may be found in [62, 25]. A generative, probabilistic model for learning a latent manifold is discussed in [6].

Manifold learning has recently become popular in the medical image analysis community, *e.g.* for cardiac analysis [103, 28], registration [40] and brain image analysis [34]. A more thorough exploration of the vast literature on manifold learning and dimensionality reduction is beyond the scope of this work. The interested reader is referred to some excellent surveys in [16, 18].

6.2 Specializing the forest model for manifold learning

The idea of using tree-based random space projections for manifold learning is not new [31, 44]. Here we show how a whole *ensemble* of randomized trees can be used for this purpose, and its advantages. We start by specializing the generic forest model (chapter 2) for use in manifold learning.

Problem statement. The manifold learning task is summarized here as follows:

> *Given a set of k unlabelled observations* $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_i, \ldots, \mathbf{v}_k\}$ *with* $\mathbf{v}_i \in \mathbf{R}^d$ *we wish to find a smooth mapping* $\mathbf{f} : \mathbb{R}^d \to\in \mathbb{R}^{d'}$, $\mathbf{f}(\mathbf{v}_i) = \mathbf{v}'_i$ *such that* $d' << d$ *and that preserves the observations' relative geodesic distances.*

As illustrated in fig. 6.1 each input observation $\mathbf{v}$ is represented as a multi-dimensional feature response vector $\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$. The desired output is the mapping function $\mathbf{f}(\cdot)$.

In fig. 6.1a input data points are denoted with circles. They live in a 2D space. We wish to find a function $\mathbf{f}(\cdot)$ which maps those points to their corresponding locations in a lower dimensional space (in the figure, $d' = 1$) such that Euclidean distances in the new space are as close as possible to the geodesic distances in the original space.

What are manifold forests? As mentioned, the manifold learning problem and the density estimation one are closely related. This chapter builds upon density forests, with much of the mathematical modeling borrowed from chapter 5. So, manifold forests are also collections of clustering trees. However, unlike density forests, the manifold forest model requires extra components such as an *affinity model* and an efficient algorithm for estimating the optimal mapping $\mathbf{f}$. Details are described next.

The training objective function. Using randomized node optimization, training happens by maximizing a continuous information

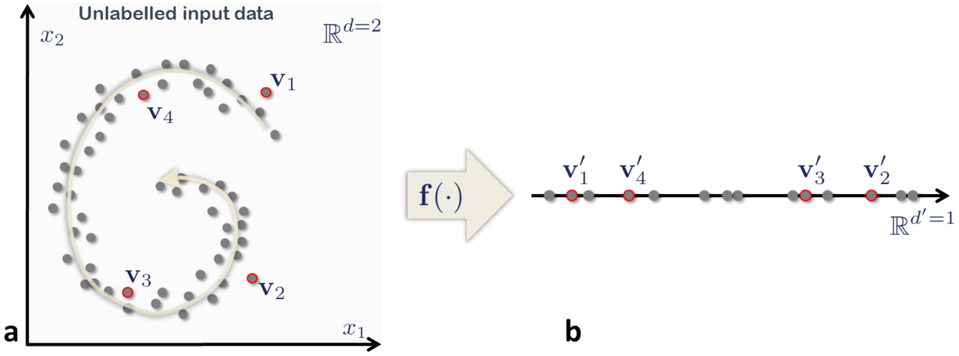


Fig. 6.1: **Manifold learning and dimensionality reduction. (a)** Input, unlabelled data points are denoted with circles. They live in a high-dimensional space (here $d = 2$ for illustration clarity). A red outline highlights some selected points of interest. **(b)** The target space is much lower dimensionality (here $d' = 1$ for illustration). Geodesic distances and ordering are preserved.

gain measure

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

with I_j defined as for density forests:

$$I_j = \log(|\Lambda(\mathcal{S}_j)|) - \sum_{i \in \{L,R\}} \frac{|\mathcal{S}_j^i|}{|\mathcal{S}_j|} \log(|\Lambda(\mathcal{S}_j^i)|). \tag{6.1}$$

The previous chapter has discussed properties and advantages of (6.1).

The predictor model. Like in the density model the statistics of all training points arriving at each leaf node is summarized with a single multi-variate Gaussian:

$$p_t(\mathbf{v}) = \frac{\pi_{l(\mathbf{v})}}{Z_t} \mathcal{N}(\mathbf{v}; \boldsymbol{\mu}_{l(\mathbf{v})}, \Lambda_{l(\mathbf{v})}).$$

The affinity model. Unlike other tasks, in manifold learning we need to estimate some measure of similarity or distance between data points so that we can preserve those inter-point distances after the mapping. When working with complex data in high dimensional spaces it is important for this affinity model to be as efficient as possible. Here we introduce another novel contribution. We use decision forests to define data affinity in a simple and effective way.

As seen in the previous chapter, at its leaves a clustering tree t defines a partition of the input points

$$l(\mathbf{v}) : \mathbb{R}^d \rightarrow \mathcal{L} \subset \mathbb{N}$$

with l a leaf index and $\mathcal{L}$ the set of all leaves in a given tree (the tree index is not shown to avoid cluttering the notation). For a clustering tree t we can compute the $k \times k$ points' affinity matrix $\mathtt{W}^t$ with elements

$$\mathtt{W}^t_{ij} = e^{-D^t(\mathbf{v}_i,\mathbf{v}_j)}. \tag{6.2}$$

The matrix $\mathtt{W}^t$ can be thought of as un-normalized transition probabilities in Markov random walks defined on a fully connected graph (where each data point corresponds to a node). The distance D can be defined in different ways. For example:
Mahalanobis affinity

$$D^t(\mathbf{v}_i,\mathbf{v}_j) = \begin{cases} \mathbf{d}_{ij}^\top \left(\Lambda^t_{l(\mathbf{v}_i)}\right)^{-1} \mathbf{d}_{ij} & \text{if } \ l(\mathbf{v}_i) = l(\mathbf{v}_j) \\ \infty & \text{otherwise} \end{cases} \tag{6.3}$$

Gaussian affinity

$$D^t(\mathbf{v}_i,\mathbf{v}_j) = \begin{cases} \frac{\mathbf{d}_{ij}^\top \mathbf{d}_{ij}}{\epsilon^2} & \text{if } \ l(\mathbf{v}_i) = l(\mathbf{v}_j) \\ \infty & \text{otherwise} \end{cases} \tag{6.4}$$

Binary affinity

$$D^t(\mathbf{v}_i,\mathbf{v}_j) = \begin{cases} 0 & \text{if } \ l(\mathbf{v}_i) = l(\mathbf{v}_j) \\ \infty & \text{otherwise} \end{cases} \tag{6.5}$$

where $\mathbf{d}_{ij} = \mathbf{v}_i - \mathbf{v}_j$, and $\Lambda_{l(\mathbf{v}_i)}$ is the covariance matrix associated with the leaf reached by the point $\mathbf{v}_i$. Note that in all cases it is not necessary to compute the partition function Z_t. More complex probabilistic models of affinity may also be used.

The simplest and most interesting model of affinity in the list above is the binary one. It can be thought of as a special case of the Gaussian model with the length parameter $\epsilon \to \infty$. Thus the binary affinity model is parameter-free. It says that given a tree t and two points $\mathbf{v}_i$ and $\mathbf{v}_j$ we assign perfect affinity (affinity=1, distance=0) to those points if they end up in the same cluster (leaf) and null affinity (infinite distance) otherwise.

The crucial aspect of manifold forests is that information-theoretical objective function maximization leads to a natural definition of point neighborhoods and similarities. In fact, defining appropriate data neighborhoods is an important problem in many manifold learning algorithms as it is crucial for defining good approximations to the pairwise geodesic distances. In data intensive applications using an information-gain objective is more natural than having to design pairwise distances between complex data points.

The ensemble model. In Laplacian eigenmaps [4] constructing an affinity matrix of the type in (6.2) is the first step. Then, spectral analysis of the affinity matrix produces the desired mapping $\mathbf{f}$. However, for a single randomly trained tree its affinity matrix is not going to be representative of the correct pairwise point affinities. This is true especially if binary affinity is employed. However, having a collection of random trees enables us to collect evidence from the entire ensemble. This has the effect of producing a smooth forest affinity matrix even in the presence of a parameter-free binary affinity model. Once again, the use of randomness is key here.

More formally, in a forest of T trees its affinity matrix is defined as:

$$\mathtt{W} = \frac{1}{T}\sum_{t=1}^{T} \mathtt{W}^t. \tag{6.6}$$

In a given tree two points may not belong to the same cluster. In some other tree they do. The averaging operation in (6.6) has the effect of propagating pairwise affinities across the graph of all points.

Having discussed how to use forests for computing the data affinity matrix (*i.e.* building the graph), next we proceed with the actual estimation of the mapping function $\mathbf{f}(\cdot)$. This second part is based on the well known Laplacian eigen-maps technique [4, 62] which we summarize here for convenience.

Estimating the embedding function. A low dimensional embedding is now found by simple linear algebra. Given a graph whose nodes are the input points and its affinity matrix $\mathtt{W}$ we first construct the $k \times k$ normalized graph-Laplacian matrix as:

$$\mathtt{L} = \mathtt{I} - \Upsilon^{-\frac{1}{2}}\mathtt{W}\Upsilon^{-\frac{1}{2}} \tag{6.7}$$

with the normalizing diagonal ("degree") matrix Υ, such that $\Upsilon_{ii} = \sum_j \mathtt{W}_{ij}$ [18]. Now, the nonlinear mapping $\mathbf{f}$ is found via eigen-decomposition of $\mathtt{L}$. Let $\mathbf{e}_0, \mathbf{e}_1, \cdots, \mathbf{e}_{k-1}$ be the solutions of (6.7) in *increasing* order of eigenvalues

$$\begin{aligned} \mathtt{L}\mathbf{e}_0 &= \lambda_0 \mathbf{e}_0 \qquad (6.8) \\ \mathtt{L}\mathbf{e}_1 &= \lambda_1 \mathbf{e}_1 \\ &\cdots \\ \mathtt{L}\mathbf{e}_{k-1} &= \lambda_{k-1} \mathbf{e}_{k-1} \end{aligned}$$

with

$$0 = \lambda_0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{k-1}$$

We ignore the first eigenvector $\mathbf{e}_0$ as it corresponds to a degenerate solution (global translation) and use the next $d' << d$ eigenvectors (from $\mathbf{e}_1$ to $\mathbf{e}_{d'}$) to construct the $k \times d'$ matrix $\mathtt{E}$ as

$$\mathtt{E} = \begin{pmatrix} | & | & | & | & | & | \\ \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_j & \cdots & \mathbf{e}_{d'} \\ | & | & | & | & | & | \end{pmatrix}$$

with $j \in \{1, \cdots, d'\}$ indexing the eigenvectors (represented as column vectors). Finally, mapping a point $\mathbf{v}_i \in \mathbb{R}^d$ onto its corresponding point $\mathbf{v}' \in \mathbb{R}^{d'}$ is done simply by reading the i^{th} row of $\mathtt{E}$:

$$\mathbf{v}'_i = \mathbf{f}(\mathbf{v}_i) = (\mathtt{E}_{i1}, \cdots, \mathtt{E}_{ij}, \cdots, \mathtt{E}_{id'})^\top \tag{6.9}$$

where $i \in \{1, \cdots, k\}$ indexes the individual points. Note that d' must be $< k$ which is easy to achieve as we normally wish to have a small target dimensionality d'. In summary, the embedding function $\mathbf{f}$ remains implicitly defined by its k corresponding point pairs, through the eigenvector matrix $\mathtt{E}$. In contrast to existing techniques, here, we do not need to fine-tune a length parameter or a neighborhood size. In fact, when using the binary affinity model the point neighborhood remains defined automatically by the forest leaves.

Mapping previously unseen points. There may be applications where after having trained the forest on a given training set, further data points become available. In order to map the new points to the corresponding lower dimensional space one may think of retraining the entire manifold forest from scratch. However, a more efficient, approximate technique consists in interpolating the point position given the already available embedding. More formally, given a previously unseen point $\mathbf{v}$ and an already trained manifold forest we wish to find the corresponding point $\mathbf{v}'$ in the low dimensional space. The point $\mathbf{v}'$ may be computed as follows:

$$\mathbf{v}' = \frac{1}{T}\sum_t \frac{1}{\eta^t}\sum_i \left(e^{-D^t(\mathbf{v},\mathbf{v}_i)}\ \mathbf{f}(\mathbf{v}_i)\right)$$

with $\eta^t = \sum_i e^{-D^t(\mathbf{v},\mathbf{v}_i)}$ the normalizing constant and the distance $D^t(\cdot,\cdot)$ computed by testing the existing t^{th} tree on $\mathbf{v}$. This interpolation technique works well for points which are somewhat close to the original training set. Other alternatives are possible.

6.2.1 Properties and advantages.

Let us discuss some properties of manifold forests.

Ensemble clustering for distance estimation. When dealing with complex data (*e.g.* images) defining pairwise distances can be a challenging. Here we avoid that problem since we use directly the pairwise affinities defined by the tree structure itself. This is especially true of the simpler binary affinity model. The trees and their tests are automatically optimized from training data with minimal user input.

As an example, imagine that we have a collection of holiday photos containing images of beaches, forests and cityscapes (see fig. 6.2). Each image defines a data point in a high dimensional space. When training a manifold forest we can imagine that *e.g.* some trees group all beach photos in a cluster, all forest photos in a different leaf and all cityscapes in yet another leaf. A different tree, by using different features may be mixing some of the forest photos with some of the beach ones (*e.g.* because of the many palm trees along the shore), but the cityscape are visually very distinct and will tend to remain (mostly) in a separate cluster. So, forests and beach scenes are more likely to end up in the same leaf while building photos do not tend to mix with other classes (just an example). Therefore, the matrix (6.6) will assign higher affinity (smaller distance) to a forest-beach image pair than to a beach-city pair. This shows how an ensemble of multiple hard clusterings can yield a soft distance measure.

Choosing the feature space. An issue with manifold learning techniques is that often one needs to decide ahead of time how to represent each data point. For instance one has to decide its dimensionality and what features to use. Thinking of the practical computer vision problem of learning manifolds of images the complexity of this problem becomes apparent.

One potential advantage of manifold forests is that we do not need to specify manually the features to use. We can define the generic *family* of features (*e.g.* gradients, Haar wavelent, output of filter banks *etc.*). Then the tree training process will automatically select optimal features and corresponding parameters for each node of the forest, so as to optimize a well defined objective function (a clustering information gain in this case). For instance, in the illustrative example in fig. 6.2 as features we could use averages of pixel colours within rectangles placed within the image frame. Position and size of the rectangles is selected during training. This would allow the system to learn *e.g.* that brown-coloured regions are expected towards the bottom of the image for beach scenes, long vertical edges are expected in cityscapes etc.

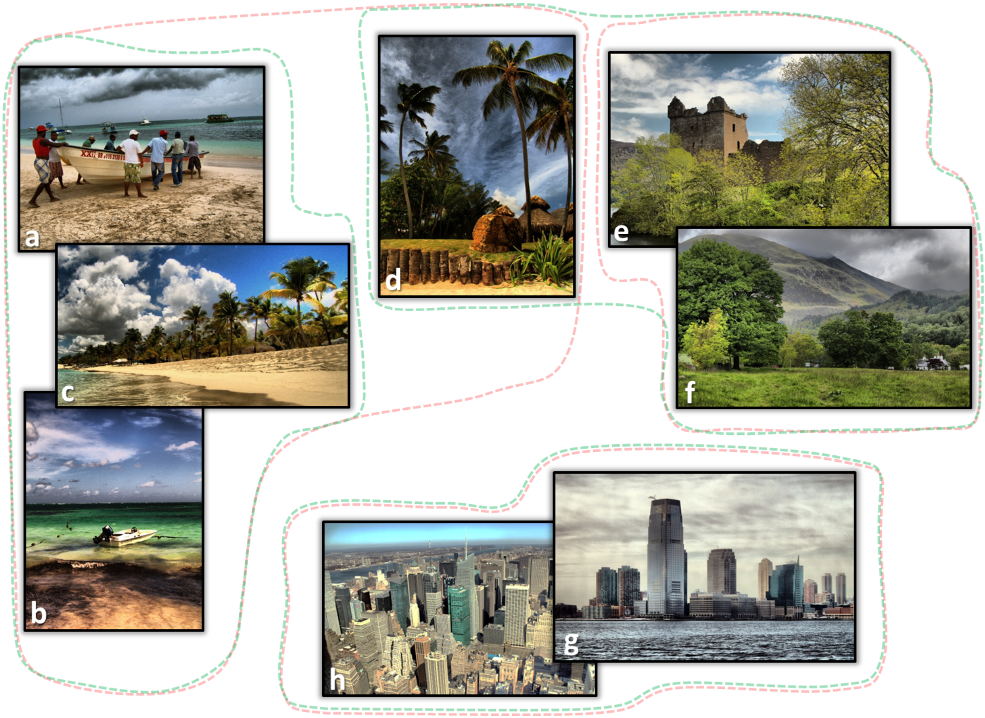


Fig. 6.2: **Image similarity via ensemble clustering.** Different trees (whose leaves are denoted by different colour curves) induce different image partitions. The red tree yields the partition $\{\{a, b, c, d\}, \{e, f\}, \{g, h\}\}$. The green tree yields the partition $\{\{a, b, c\}, \{d, e, f\}, \{g, h\}\}$. The overlap between clusters in different trees is captured mathematically by the forest affinity matrix $\mathtt{W}$. In $\mathtt{W}$ we will have that image e is closer to image c than to image g. Therefore, ensemble-based clustering induces data affinity. See text for details.

Computational efficiency. In this algorithm the bottleneck is the solution of the eigen-system (6.7) which could be slow for a large number of input points k. However, in (6.9) only the $d' << k$ bottom eigenvectors are necessary. This, in conjunction with the fact that the matrix $\mathtt{L}$ is usually very sparse (especially for the binary affinity model) can yield efficient implementations. Please note that only one eigen-system needs be solved, independent from the forest size T. On the other hand all the tree-based affinity matrices $\mathtt{W}^t$ may be computed in parallel.

Estimating the target intrinsic dimensionality. The algorithm above can be applied for any dimensionality d' of the target space. If we do not know d' in advance (*e.g.* from application-specific knowledge) an optimal value can be chosen by looking at the profile of (ordered) eigenvalues λ_j and choosing the minimum number of eigenvalues corresponding to a sharp elbow in such profile [4]. Here we need to make clear that being able to estimate automatically the manifold dimensionality is a property shared with other spectral techniques and is *not* unique to manifold forests. This idea will be tested in some examples at the end of the chapter.

6.3 Experiments and the effect of model parameters

This section presents some experiments and studies the effect of the manifold forest parameters on the accuracy of the estimated non-linear mapping.

6.3.1 The effect of the forest size

We begin by discussing the effect of the forest size parameter T. In a forest of size T each randomly trained clustering tree produces a different, disjoint partition of the data.[1] In the case of a binary affinity model the elements of the affinity matrices $\mathtt{W}^t$ are binary ($\in \{0, 1\}$, either two points belong to the same leaf/cluster or not). A given pair of points will belong to the same cluster (leaf) in some trees and not in others (see fig. 6.3). Via the ensemble model the *forest* affinity matrix $\mathtt{W}$ is much smoother since multiple trees enable different point pairs to exchange information about their relative position. Even if we use the binary affinity case the forest affinity $\mathtt{W}$ is in general *not* binary. Large forests (large values of T) correspond to averaging many tree affinity matrices together. In turn, this produces robust estimation of pairwise

[1] If the input points were reordered correctly for each tree we would obtain an affinity matrix $\mathtt{W}^t$ with block-diagonal structure.

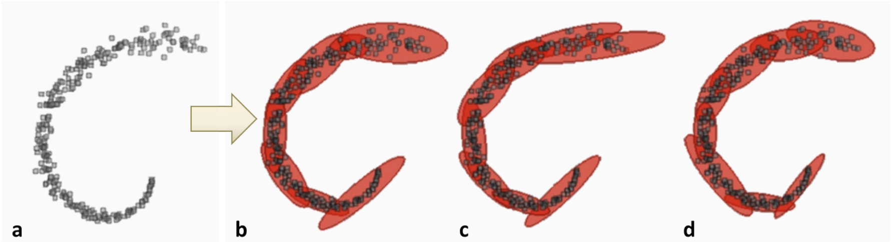


Fig. 6.3: **Different clusterings induced by different trees. (a)** The input data in 2D. **(b,c,d)** Different partitions learned by different random trees in the same manifold forest. A given pair of points will belong to the same cluster (leaf) in some trees and not in others.

affinities even between distant pairs of points.

Figure 6.4 shows two examples of nonlinear dimensionality reduction. In each experiment we are given some noisy, unlabelled 2D points distributed according to some underlying nonlinear 1D manifold. We wish to discover the manifold and map those points onto a 1D real axis while preserving their relative geodesic distances. The figure shows that when using a very small number of trees such mapping does not work well. This is illustrated *e.g.* in fig. 6.4b-leftmost, by the vertical banding artifacts; and in fig. 6.4d-leftmost by the single red colour for all points. However, as the number of trees the affinity matrix W better represents the true pairwise graph affinity. Consequently the colour coding (linearly going from dark blue to dark red) starts to follow correctly the 1D evolution of the points.

6.3.2 The effect of the affinity model

Here we discuss advantages and disadvantages of using different affinity models. Binary affinities (6.5) are extremely fast to compute and avoid the need to define explicit distances between (possibly complex) data points. For example defining a sensible distance metric between images is difficult. With our binary model pairwise affinities are defined implicitly by the hierarchical structure of the trees.

Figure 6.5 compares the binary and Gaussian affinity models in a

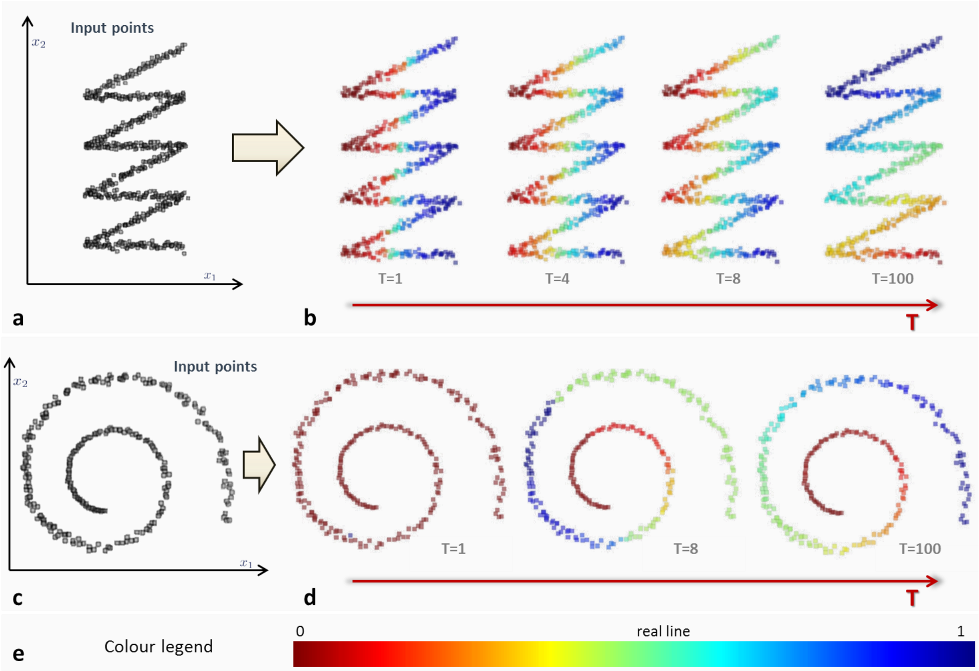


Fig. 6.4: **Manifold forest and non-linear dimensionality reduction. The effect of T. (a,c)** Input 2D points for two different synthetic experiments. **(b,d)** Non-linear mapping from the original 2D space to the 1D real line is colour coded, from dark red to dark blue. In both examples a small forest (small T) does not capture correctly the intrinsic 1D manifold. For larger values of T (*e.g.* $T = 100$) the accuracy of such a mapping increases. **(e)** The colour legend. Different colours, from red to blue, denote the position of the mapped points in their target 1D space.

synthetic example. The input points are embedded within a 3D space with their intrinsic manifold being a 2D rectangle. A small number of trees in both cases produces a roughly triangular manifold, but as T increases the output manifold becomes more rectangular shaped. Notice that our model preserves local distances only. This is not sufficient to reproduce sharp 90-degree angles (see rightmost column in fig. 6.5).

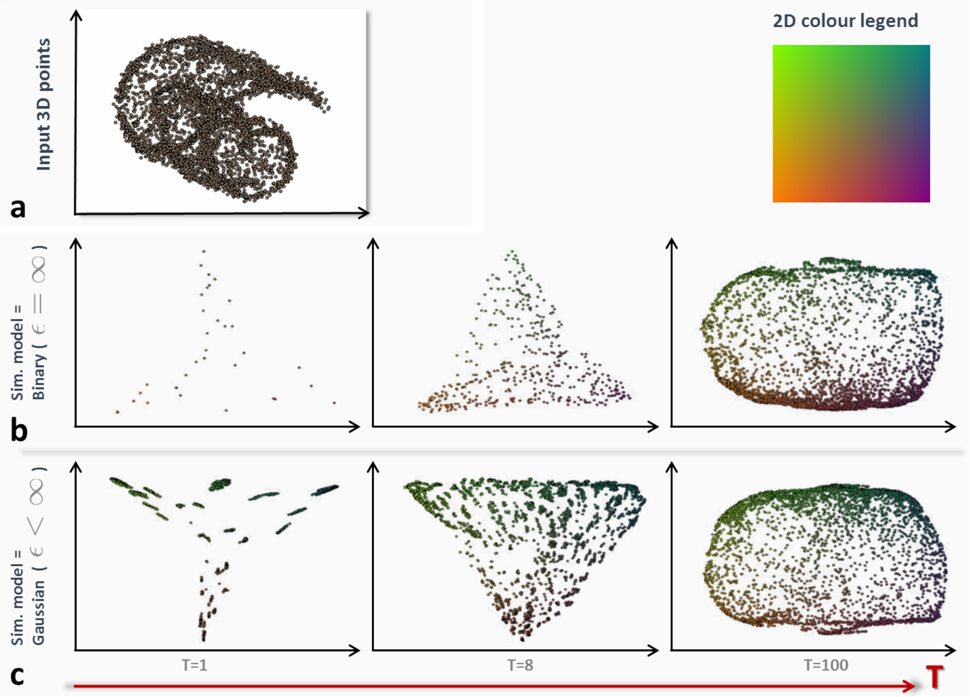


Fig. 6.5: **The effect of the similarity model.** In this experiment we map 3D points into their intrinsic 2D manifold. **(a)** The input 3D data is a variant of the well known "Swiss Roll" dataset. The noisy points are organized as a spiral in one plane with a sinusoidal component in the orthogonal direction. **(c)** Different mappings into the 2D plane for increasing forest size T. Here we use binary affinity. **(d)** As above but with a Gaussian affinity model. A sufficiently large forest manages to capture the roughly rectangular shape of the embedded manifold. For this experiment we used max forest size $T = 100$, $D = 4$ and weak learner = oriented hyperplane (linear).

For a sufficiently large forest both models do a reasonable job at reorganizing the data points on the target flat surface.

Figure 6.6 shows three views of the "Swiss Roll" example from different viewpoints. Its 3D points are colour-coded by the discovered un-

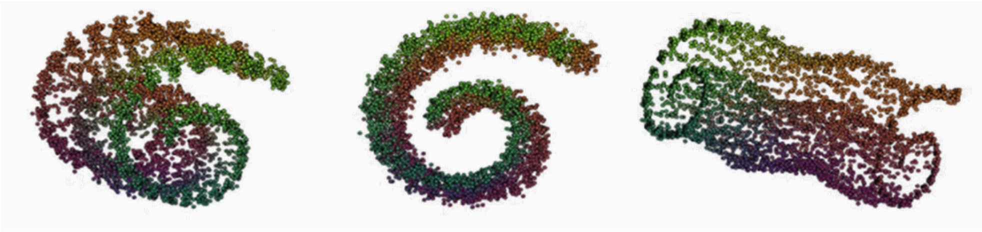


Fig. 6.6: **The "Swiss Roll" manifold.** Different 3D views from various viewpoints. Colour-coding indicates the mapped 2D manifold. See colour-legend in fig. 6.5.

derlying 2D manifold. The mapped colours confirm the roughly correct 2D mapping of the original points. In our experiments we have observed that the binary model converges (with T) more slowly than the Gaussian model, but with clear advantages in terms of speed. Furthermore, the length parameter ϵ in (6.4) may be difficult to set appropriately (because it has no immediate interpretation) for complex data such as images (see fig. 6.2). Therefore, a model which avoids this step is advantageous.

Figure 6.7 shows a final example of a 3D space being mapped onto the underlying planar manifold. Once again binary affinities were used here.

6.3.3 Discovering the manifold intrinsic dimensionality

We conclude this chapter by investigating how we can choose the optimal dimensionality of the target space. In terms of accuracy it is easy to see that a value of d' identical to the dimensionality of the original space would produce the best results because there would be no loss of information. But one criterion for choosing d' is to drastically reduce the complexity of the target space. Thus we definitely wish to use small values of d'. By plotting the (ordered) eigenvalues it is also clear that there are specific dimensionalities at which the spectrum presents sharp changes [4]. This indicates that there are values of d' such that if we used $d' + 1$ we would not gain very much. These special loci can be used to define "good" values for the target dimensionality.

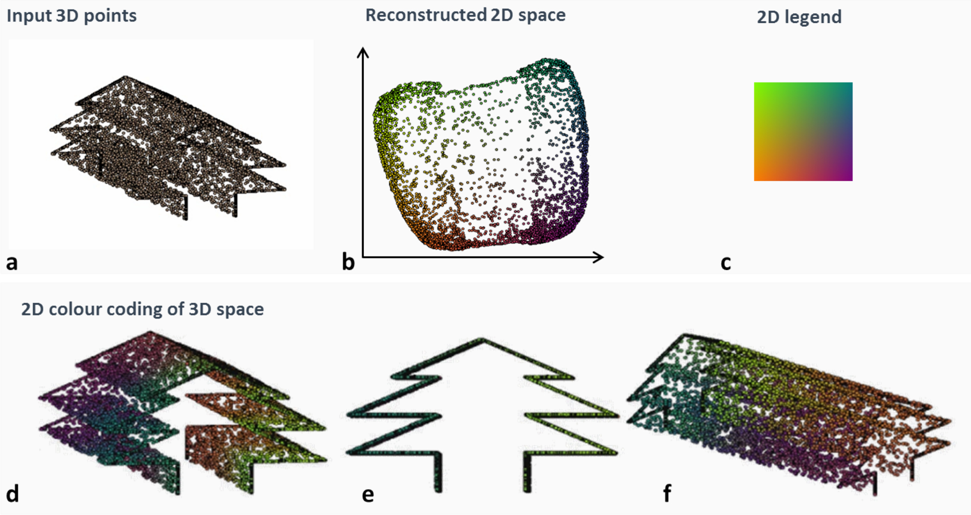


Fig. 6.7: **The "Christmas Tree" manifold. (a)** The unlabelled input data points in their 3D space. **(b)** The reconstructed 2D space. **(c)** The 2D colour legend. **(d,e,f)** Different views of the 3D points with colour coding corresponding to the automatically discovered 2D mapping.

Figure 6.8 plots the eigenvalue spectra for the "Swiss Roll" dataset and the binary and Gaussian affinity models, respectively. As expected from theory $\lambda_0 = 0$ (corresponding to a translation component that we ignore). The sharp elbow in the curves, corresponding to λ_2 indicates an intrinsic dimensionality $d = 2$ (correct) for this example. In our experiments we have observed that higher values of T produce a more prominent elbow in the spectrum and thus a clearer choice for the value of d. Similarly, Gaussian affinities produce sharper elbows than binary affinities.

6.3.4 Discussion

Above we have discussed some of the advantages of manifold forests and have studies the effects of its parameters. For example we have seen that manifold forests can be efficient, avoid the need to predefine the features to be used, and can provide guidance with respect to

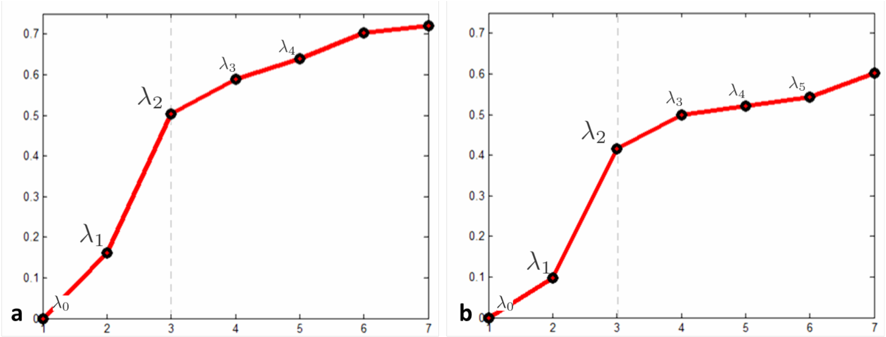


Fig. 6.8: **Discovering the manifold intrinsic dimensionality. (a)** The sorted eigenvalues of the normalized graph Laplacian for the "Swiss Roll" 3D example, with binary affinity model. **(b)** As above but with Gaussian affinity. In both curves there is a clear elbow in correspondence of λ_2 thus indicating an intrinsic dimensionality $d' = 2$. Here we used forest size $T = 100$, $D = 4$ and weak learner = linear.

the optimal dimensionality of the target space. On the flip side it is important to choose the forest depth D carefully, as this parameter influences the number of clusters in which the data is partitioned and, in turn, the smoothness of the recovered mapping. In contrast to existing techniques here we also need to choose a weak-learner model to guide the way in which different clusters are separated. The forest size T is not a crucial parameter since, as usual, the more trees the better the behaviour.

The fact that the same decision forest model can be used for manifold learning and nonlinear dimensionality reduction is an interesting discovery. This chapter has only presented the manifold forest model and some basic intuitions. However, a more thorough experimental validation with real data is necessary to fully assess the validity of such model. Next, we discuss a natural continuation of the supervised and unsupervised models discussed so far, and their use in semi-supervised learning.

7

Semi-supervised forests

Previous chapters have discussed the use of decision forests in supervised problems (*e.g.* regression and classification) as well as unsupervised ones (*e.g.* density and manifold estimation). This chapter puts the two things together to achieve semi-supervised learning. We focus here on semi-supervised *classification* but the results can be extended to regression too.

In semi-supervised classification we have available a small set of labelled training points and a large set on unlabelled ones. This is a typical situation in many scenarios. For instance, in medical image analysis getting hold of numerous anonymized patients scans is relatively easy and cheap. However, labeling them with ground-truth annotations requires experts time and effort and thus is very expensive. A key question then is if we can exploit the existence of unlabelled data to improve classification.

Semi-supervised machine learning is interested in the problem of transferring existing ground-truth labels to the unlabelled (and already available) data. When in order to solve this problem we make use of the underlying data distribution then we talk of *transductive* learning. This is in contrast with the *inductive* learning already encountered in previous chapters (chapters 3 and 4), where the test data is *not* available at training time. This chapter focuses on transductive classification and also revisits the inductive process in the light of its transductive counterpart. Decision forests can address both tasks accurately and efficiently.

Intuitively, in transductive classification we wish to separate the data so as to: (i) keep different known class labels in different regions and, (ii) make sure that classification boundaries go through areas of low data density. Thus, it is necessary to borrow concepts from both supervised classification and density estimation.

After a brief literature survey, we show how to adapt the generic forest model to do transductive semi-supervised classification. This chapter also shows how, given a transductive forest we can easily upgrade it to a general inductive classification function for previously unseen test data. Numerous examples and comparative experiments illustrate advantages and disadvantages of semi-supervised forests over alternative popular algorithms. The use of decision forests for the related *active learning* task is also briefly mentioned.

7.1 Literature on semi-supervised learning

A popular technique for semi-supervised learning is transductive support vector machines [47, 101]. Transductive SVM (TSVM) is an extension of the popular support vector machine algorithm [97] which maximizes the separation of both labelled and unlabelled data. The experimental section of this chapter will present comparisons between forests and TSVM.

Excellent, recent references for semi-supervised learning and active learning are [18, 19, 91, 104] which provide a nice structure to the vast amount of literature on these topics. A thorough literature survey is well beyond the scope of this paper and here we focus on forest-based models.

In [52] the authors discuss the use of decision forests for semi-supervised learning. They achieve this via an iterative, deterministic annealing optimization. Tree-based semi-supervised techniques for vision and medical applications are presented in [13, 17, 27]. Here we introduce a new, simple and efficient, one-shot semi-supervised forest algorithm.

7.2 Specializing the decision forest model for semi-supervised classification

This section specializes the generic forest model introduced in chapter 2 for use in semi-supervised classification. This model can also be extended to semi-supervised regression though this is not discussed here.

Problem statement. The transductive classification task is summarized here as follows:

Given a set of both labelled and unlabelled data we wish to associate a class label to all the unlabelled data.

Unlike inductive classification here all unlabelled "test" data is already available during training.

The desired output (and consequently the training labels) are of discrete, categorical type (unordered). More formally, given an input point $\mathbf{v}$ we wish to associate a class label c such that $c \in \{c_k\}$. As usual the input is represented as a multi-dimensional feature response vector $\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$.

Here we consider two types of input data: *labelled* $\mathbf{v}^{\mathrm{l}} \in \mathcal{L}$ and *unlabelled* $\mathbf{v}^{\mathrm{u}} \in \mathcal{U}$. This is illustrated in fig. 7.1a, where data points are denoted with circles. Coloured circles indicate labelled training points, with different colours denoting different labels. Unlabelled data is shown in grey. Figures 7.1b,c further illustrate the difference between transductive and inductive classification.

What are semi-supervised forests? A transductive forest is a collection of trees that have been trained on partially labelled data. Both labelled and unlabelled data are used to optimize an objective function with two components: a supervised and an unsupervised one, as described next.

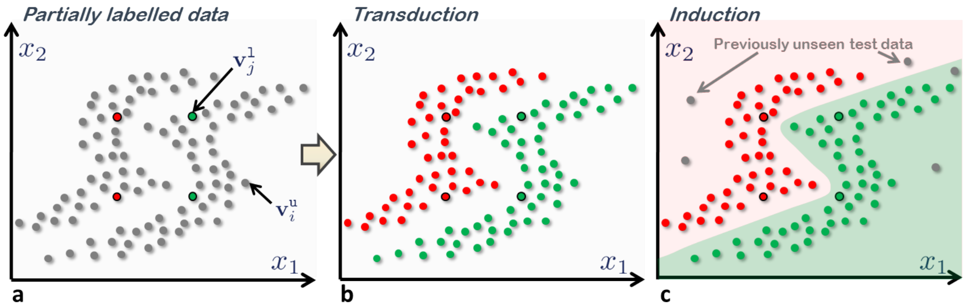


Fig. 7.1: **Semi-supervised forest: input data and problem statement.** **(a)** Partially labelled input data points in their two-dimensional feature space. Different colours denote different labels. Unlabelled data is shown in grey. **(b)** In transductive learning we wish to propagate the existing ground-truth labels to the many, available unlabelled data points. **(c)** In inductive learning we wish to learn a generic function that can be applied to previously unavailable test points (grey circles). Training a conventional classifier on the labelled data only would produce a sub-optimal classification surface, *i.e.* a vertical line in this case. Decision forests can effectively address both transduction and induction. See text for detail.

The training objective function. As usual, forest training happens by optimizing the parameters of each internal node j via

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

Different trees are trained separately and independently. The main difference with respect to other forests is that here the objective function I_j must encourage both separation of the labelled training data as well as separating different high density regions from one another. This is achieved via the following mixed information gain:

$$I_j = I_j^{\mathrm{u}} + \alpha I_j^{\mathrm{s}}. \qquad (7.1)$$

In the equation above I_j^{s} is a supervised term and depends only on the labelled training data. In contrast, I_j^{u} is the unsupervised term and depends on all data, both labelled and unlabelled. The scalar parameter α is user defined and it specifies the relative weight between the two terms.

As in conventional classification, the term $I_j^{\mathtt{s}}$ is an information gain defined over discrete class distributions:

$$I_j^{\mathtt{s}} = H(\mathcal{S}_j) - \sum_{i \in \{\mathtt{L},\mathtt{R}\}} \frac{|\mathcal{S}_j^i|}{|\mathcal{S}_j|} H(\mathcal{S}_j^i) \tag{7.2}$$

with the entropy for a subset $\mathcal{S} \subseteq \mathcal{L}$ of training points $H(\mathcal{S}) = -\sum_c p(c) \log p(c)$ with c the ground truth class labels of the points in $\mathcal{L}$.

Similarly, as in density estimation, the unsupervised gain term $I_j^{\mathtt{u}}$ is defined via differential entropies defined over continuous parameters (*i.e.* the parameters of the Gaussian associated with each cluster):

$$I_j^{\mathtt{u}} = \log |\Lambda(\mathcal{S}_j)| - \sum_{i \in \{\mathtt{L},\mathtt{R}\}} \frac{|\mathcal{S}_j^i|}{|\mathcal{S}_j|} \log |\Lambda(\mathcal{S}_j^i)| \tag{7.3}$$

for all points in $\mathcal{S}_j \subseteq (\mathcal{U} \cup \mathcal{L})$. Like in chapter 5 we have made the working assumption of Gaussian node densities.

The ensemble model. During testing, a semi-supervised classification trees t yields as output the posterior $p_t(c|\mathbf{v})$. Here we think of the input point $\mathbf{v}$ as already available during training ($\mathbf{v} \in \mathcal{U}$, for transduction) or previously unseen (for induction). The forest output is the usual posterior mean:

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

Having described the basic model components next we discuss details of the corresponding label propagation algorithm.

7.3 Label propagation in transduction forest

This section explains tree-based transductive label propagation. Figure 7.2 shows an illustrative example. We are given a partially labelled

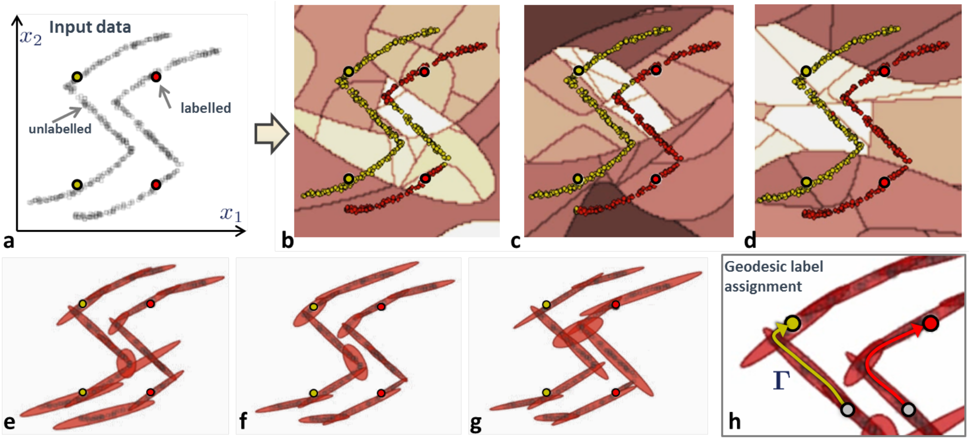


Fig. 7.2: **Label transduction in semi-supervised forests. (a)** Input points, only four of which are labelled as belonging to two classes (red and yellow). **(b,c,d)** Different transductive trees produce different partitions of the feature space. Different regions of high data density tend to be separated by cluster boundaries. Geodesic optimization enables assigning labels to the originally unlabelled points. Points in the central region (away from original ground-truth labels) tend to have less stable assignments. In the context of the entire forest this captures uncertainty of transductive assignments. **(e,f,g)** Different tree-induced partitions correspond to different Gaussian Mixture models. **(h)** Label propagation via geodesic path assignment.

dataset (as in fig. 7.2a) which we use to train a transductive forest of size T and maximum depth D by maximizing the mixed information gain (7.1).

Different trees produce randomly different partitions of the feature space as shown in fig. 7.2b,c,d. The different coloured regions represent different clusters (leaves) in each of the three partitions. If we use Gaussian models then each leaf stores a different Gaussian distribution learned by maximum likelihood for the points within. Label transduction from annotated data to unannotated data can be achieved directly via the following minimization:

$$c(\mathbf{v}^{\mathrm{u}}) \Leftarrow c\left(\arg\min_{\mathbf{v}^{\mathrm{l}}\in\mathcal{L}} D(\mathbf{v}^{\mathrm{u}},\mathbf{v}^{\mathrm{l}})\right) \qquad \forall\ \mathbf{v}^{\mathrm{u}} \in \mathcal{U}. \tag{7.4}$$

The function $c(\cdot)$ indicates the class index associated with a point (known in advance only for points in $\mathcal{L}$). The generic geodesic distance $D(\cdot,\cdot)$ is defined as

$$D(\mathbf{v}^{\mathrm{u}},\mathbf{v}^{\mathrm{l}}) = \min_{\mathbf{\Gamma}\in\{\mathbf{\Gamma}\}} \sum_{i=0}^{L(\mathbf{\Gamma})-1} d(\mathbf{s}_i,\mathbf{s}_{i+1}),$$

with $\mathbf{\Gamma}$ a geodesic path (here represented as a discrete collection of points), $L(\mathbf{\Gamma})$ its length, $\{\mathbf{\Gamma}\}$ the set of all possible geodesic paths and the initial and end points $\mathbf{s}_0 = \mathbf{v}^{\mathrm{u}}, \mathbf{s}_n = \mathbf{v}^{\mathrm{l}}$, respectively. The local distances $d(\cdot,\cdot)$ are defined as symmetric Mahalanobis distances

$$d(\mathbf{s}_i,\mathbf{s}_j) = \frac{1}{2}\left(\mathbf{d}_{ij}{}^{\top}\Lambda_{l(\mathbf{v}_i)}^{-1}\mathbf{d}_{ij} + \mathbf{d}_{ij}{}^{\top}\Lambda_{l(\mathbf{v}_j)}^{-1}\mathbf{d}_{ij}\right)$$

with $\mathbf{d}_{ij} = \mathbf{s}_i - \mathbf{s}_j$ and $\Lambda_{l(\mathbf{v}_i)}$ the covariance associated with the leaf reached by the point $\mathbf{v}_i$. Figure 7.2h shows an illustration. Using Mahalanobis local distances (as opposed to *e.g.* Euclidean ones) discourages paths from cutting across regions of low data density, a key requirement for semi-supervised learning.[1] In effect we have defined geodesics on the space defined by the automatically inferred probability density function.

Some example results of label propagation are shown in fig. 7.2b,c,d. Figures 7.2e,f,g illustrate the corresponding Gaussian clusters associated with the leaves. Following label transduction (7.4) all unlabelled points remain associated with one of the two labels (fig. 7.2b,c,d). Note that such transducted labels are different for each tree, and they are more stable for points closer to the original labelled data. When looking at the entire forest this yields uncertainty in the newly obtained labels. Thus, in contrast to some other transductive learning algorithms

[1] Since all leaves are associated with the same Gaussian the label propagation algorithm can be implemented very efficiently by acting on each leaf cluster rather than on individual points. Very efficient geodesic distance transform algorithms exist [22].

a semi-supervised forest produces a probabilistic transductive output $p(c|\mathbf{v}^{\mathrm{u}})$.

Usually, once transductive label propagation has been achieved one may think of using the newly labelled data as ground-truth and train a conventional classifier to come up with a general, inductive classification function. Next we show how we can avoid this second step and go directly from transduction to induction without further training.

7.4 Induction from transduction

Previously we have described how to propagate class labels from labelled training points to already available unlabelled ones. Here we describe how to infer a general probabilistic classification rule $p(c|\mathbf{v})$ that may be applied to previously unseen test input ($\mathbf{v} \notin \mathcal{U} \cup \mathcal{L}$).

We have two alternatives. First, we could apply the geodesic-based algorithm in (7.4) to every test input. But this involves T shortest-path searches for each $\mathbf{v}$. A simpler alternative involves constructing an inductive posterior from the existing trees, as shown next.

After transduction forest training we are left with T trained trees and their corresponding partitions (fig. 7.2b,c,d). After label propagation we also have attached a class label to *all* available data (with different trees possibly assigning different classes to the points in U). Now, just like in classification, counting the examples of each class arriving at each leaf defines the tree posteriors $p_t(c|\mathbf{v})$. These act upon the entire feature space in which a point $\mathbf{v}$ lives and not just the already available training points. Therefore, the inductive forest class posterior is the familiar

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

Here we stress again that the tree posteriors are learned from all (existing and transducted) class labels ignoring possible instabilities in class assignments. We also highlight that building the inductive posterior is extremely efficient (it involves counting) and does not require training a whole new classifier.

Figure 7.3 shows classification results on the same example as in fig. 7.2. Now the inductive classification posterior is tested on all points

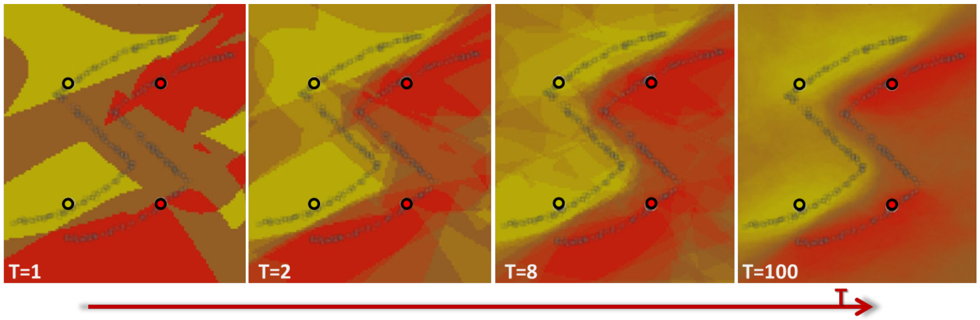


Fig. 7.3: **Learning a generic, inductive classification rule.** Output classification posteriors, tested on all points in a rectangular section of the feature space. Labelled training points are indicated by coloured circles (only four of those per image). Available unlabelled data are shown by small grey squares. Note that a purely inductive classification function would separate the left and right sides of the feature space with a vertical line. In contrast here the separating surface is "S"-shaped because affected by the density of the unlabelled points, thus demonstrating the validity of the use of unlabelled data densities. From left to right the number of trees in the forest increases from $T = 1$ to $T = 100$. See text for details.

within a rectangular section of the feature space. As expected a larger T produces smoother posteriors. Note also how the inferred separating surface is "S"-shaped because it takes into account the unlabelled points (small grey squares). Finally we observe that classification uncertainty is greater in the middle due to its increased distance from the four ground-truth labelled points (yellow and red circles).

Discussion. In summary, by using our mixed information gain and some geodesic manipulation the generic decision forest model can be readily adapted for use in semi-supervised tasks. Semi-supervised forests can be used both for transduction and (refined) induction without the need for a two-stage training procedure. Further efficiency is due to the parallel nature of forests. Both for transduction and induction the output is fully probabilistic. We should also highlight that semi-supervised forests are very different from *e.g.* self-training techniques [75]. Self-training techniques work by: (i) training a supervised classifier, (ii) classifying the unlabelled data, (iii) using the newly classified data (or perhaps only the most confident subset) to train a new classifier, and so on. In contrast, semi-supervised forests are not iterative. Additionally, they are driven by a clear objective function, the maximization of which encourages the separating surface to go through regions of low data density, while respecting existing ground-truth annotations.

Next we present further properties of semi-supervised forests (such as their ability to deal with any number of classes) with toy examples and comparisons with alternative algorithms.

7.5 Examples, comparisons and effect of model parameters

This section studies the effect of the forest model parameters on its accuracy and generalization. The presented illustrative examples are designed to bring to life different properties. Comparisons between semi-supervised forests with alternatives such as transductive support vector machines are also presented.

Figure 7.3 has already illustrated the effect of the presence of unlabelled data as well as the effect of increasing the forest size T on the shape and smoothness of the posterior. Next we discuss the effect of increasing the labelled points.

The effect of additional labelled data and active learning. As observed already, the central region in fig. 7.4a shows higher classification uncertainty (dimmer, more orange pixels). Thus, as typical of active learning [14] we might decide to collect and label additional data precisely in those low-confidence regions. This should have the effect of refining the classification posterior and increasing it confidence. This effect is indeed illustrated in fig. 7.4b.

As expected, a guided addition of further labelled data in regions of high uncertainty increases the overall predictor confidence. The importance of having a probabilistic output is clear here as it is the confidence of the prediction (and not the class prediction itself) which guides, in

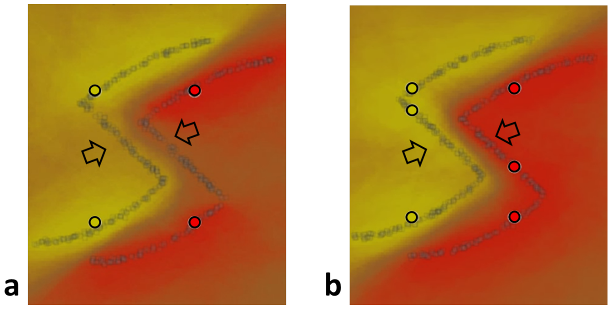


Fig. 7.4: **Active learning. (a)** Test forest posterior trained with only four labelled points and hundreds of unlabelled ones. The middle region shows lower confidence (pointed at by two arrows). **(b)** As before, but with two additional labelled points placed in regions of high uncertainty. The overall confidence of the classifier increases considerably and the overall posterior is sharper. Figure best seen on screen.

an economical way, the collection of additional training data. Next we compare semi-supervised forests with alternative algorithms.

Comparison with support vector machines. Figure 7.5 shows a comparison between semi-supervised forests and conventional SVM [97] as well as transductive SVM [47, 101], on the same two input datasets. [2]

In the figure we observe a number of effects. First, unlike SVM the forest captures uncertainty. As expected, more noise in the input data (either in the labelled or unlabelled sets, or both) is reflected in lower prediction confidence. Second, while transductive SVM manages to exploit the presence of available unlabelled data it still produces a hard, binary classification. For instance, larger amounts of noise in the training data is not reflected in the TSVM separating surface.

[2] In this example the SVM and transductive SVM results were generated using the "SVM-light" Matlab toolbox in `http://svmlight.joachims.org/`.

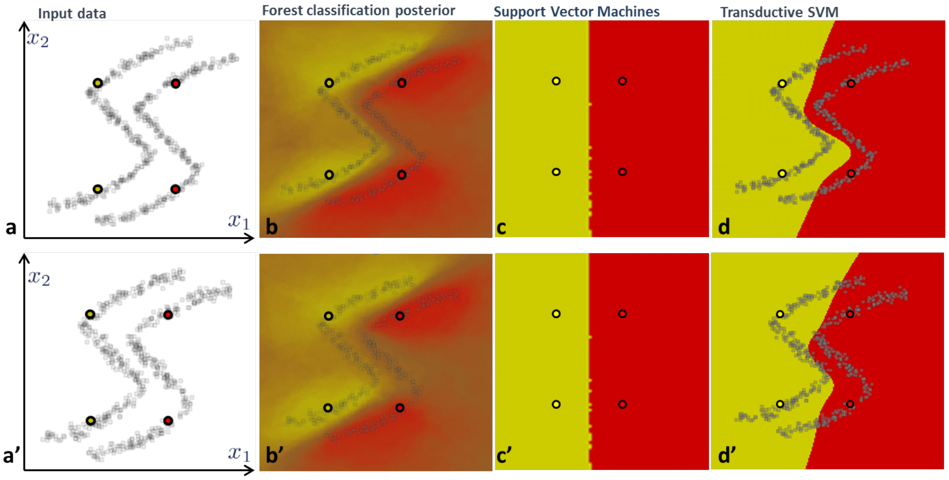


Fig. 7.5: **Comparing semi-supervised forests with SVM and transductive SVM. (a)** Input partially labelled data points. **(b)** Semi-supervised forest classification posterior. The probabilistic output captures prediction uncertainty (mixed-colour pixels in the central region). **(c)** Unsurprisingly, conventional SVM produces a vertical separating surface and it is not affected by the unlabelled set. **(d)** Transductive SVM follows regions of low density, but still does not capture uncertainty. **(a')** As in (a) but with larger noise in the point positions. **(b')** The increased input noise is reflected in lower overall confidence in the forest prediction. **(c',d')** as (c) and (d), respectively, but run on the noisier training set (a').

Handling multiple classes. Being tree-based models semi-supervised forests can natively handle multiple (> 2) classes. This is demonstrated in fig. 7.6 with a four-class synthetic experiment. The input points are randomly drawn from four bi-variate Gaussians. Out of hundreds of points only four are labelled with their respective classes (shown in different colours). Conventional one-v-all SVM classification results in hard class assignments (fig. 7.6b). Tree-based transductive label propagation (for a single tree) is shown in fig. 7.6c. Note that slightly different assignments are achieved for different trees. The forest-based

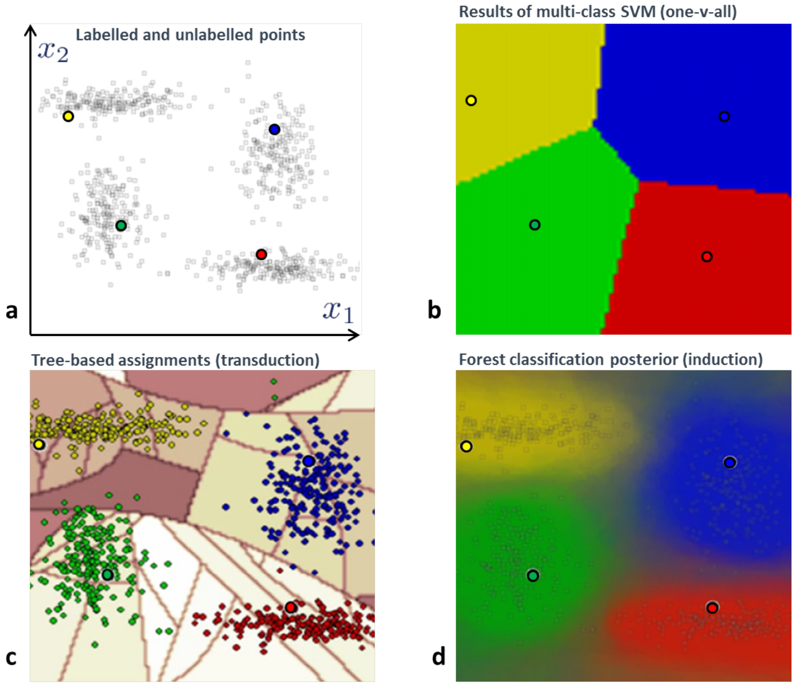


Fig. 7.6: **Handling multiple classes.** **(a)** Partially labelled input data. We have 4 labelled points for the 4 classes (different colours for different classes). **(b)** Classification results for one-v-all support vector machines. **(c)** Transduction results based on a single decision tree. Originally unlabelled points are assigned a label based on tree-induced geodesic distances. **(d)** Final semi-supervised classification posterior. Unlabelled points nicely contribute to the shape of the posterior (*e.g.* look at the elongated yellow blob). Furthermore, regions of low confidence nicely overlap regions of low data density.

inductive posterior (computed for $T = 100$) is shown in fig. 7.6d where the contribution of previously unlabelled points to the shape of the final posterior is clear. Regions of low confidence in the posterior correspond to regions of low density.

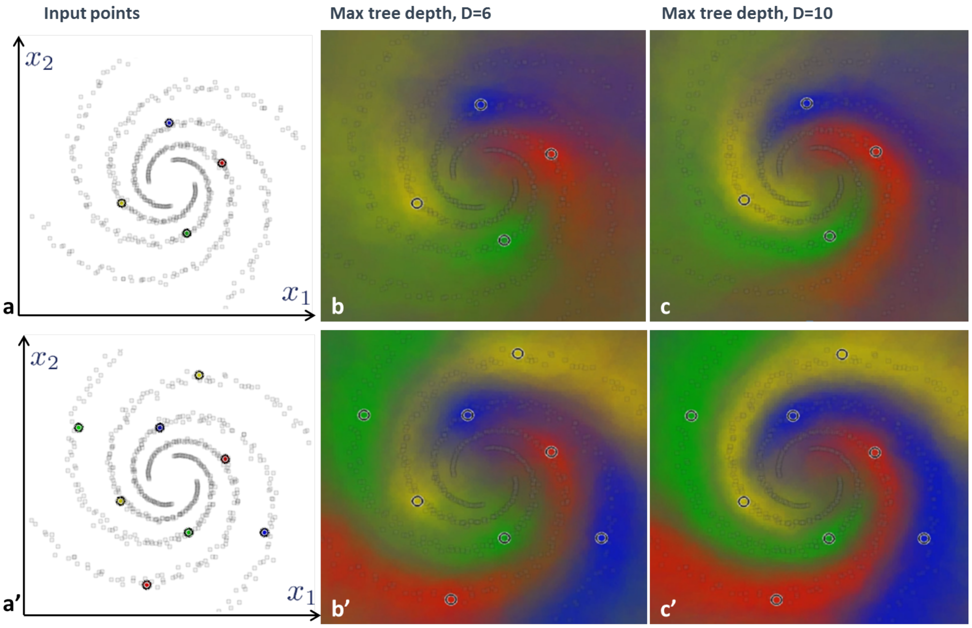


Fig. 7.7: **Semi-supervised forest: effect of depth. (a)** Input labelled and unlabelled points. We have 4 labelled points and 4 classes (colour coded). **(a')** As in (a) but with double the labelled data. **(b,b')** Semi-supervised forest classification posterior for $D = 6$ tree levels. **(c,c')** Semi-supervised forest classification posterior for $D = 10$ tree levels. The best results are obtained in (c'), with largest amount of labelled data and deepest trees.

The effect of tree depth. We conclude this chapter by studying the effect of the depth parameter D in fig. 7.7. The figure shows two four-class examples. The input data is distributed according to four-arm spirals. In the top row we have only four labelled points. In the bottom row we have eight. Similar to classification forests, increasing the depth D from 6 to 10 produces more accurate and confident results. And so does increasing the amount of labelled data. In this relatively complex example, accurate and sharp classification is achieved with just 2×4 labelled data points (for $D = 10$ tree levels) and hundreds of unlabelled ones.

The recent popularity of decision forests has meant an explosion of different variants in the literature. Although collecting and categorizing all of them is a nearly impossible task, in the next chapter we discuss a few important ones.

8

Random ferns and other forest variants

This chapter describes some of the many variants on decision forests that have emerged in the last few years. Many such variations can be seen as special instances of the same general forest model. Specifically here we focus on: random ferns, extremely randomized trees, entangled forests, online training and the use of forests on random fields.

8.1 Extremely randomized trees

Extremely randomized trees (ERT) are ensembles of randomly trained trees where the optimization of each node parameters has been greatly reduced or even removed altogether [36, 61].

In our decision model the amount of randomness in the selection/optimization of split nodes is controlled by the parameter $\rho = |\mathcal{T}_j|$ (section 2.2). In our randomized node optimization model when training the j^{th} internal node the set $\mathcal{T}_j$ is selected at random from the entire set of possible parameters $\mathcal{T}$. Then optimal parameters are chosen only within the $\mathcal{T}_j$ subset. Consequently, extremely randomized trees are a specific instance of the general decision forest model with the additional constraint that $\rho = 1 \quad \forall j$. In this case no node training is performed.

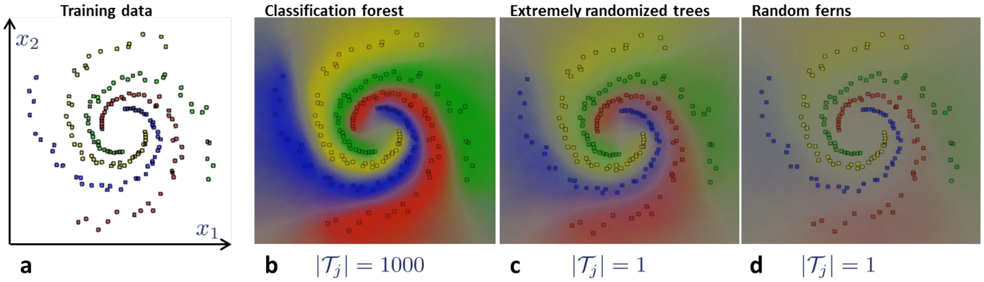


Fig. 8.1: **Forests, extremely randomized trees and ferns. (a)** Input training points for four classes. **(b)** Posterior of a classification forest. **(c)** Posterior of an ensemble of extremely randomized trees. **(d)** Posterior of a random fern. The randomness parameter is changed as illustrated. All other parameters are kept fixed. Extremely randomized trees are faster to train than forests but produce a lower-confidence posterior (in this example). The additional constraints of random ferns yield further loss of posterior confidence.

Figure 8.1 shows a comparison between classification forests and extremely randomized trees for a toy example. Some training points belonging to four different classes are randomly distributed along four spiral arms. Two decision forests were trained on the data. One of them with $\rho = 1000$ and another with $\rho = 1$ (extremely randomized). All other parameters are kept identical ($T = 200, D = 13$, weak learner = conic section, predictor = probabilistic). The corresponding testing posteriors are shown in fig. 8.1b, and fig. 8.1c, respectively. It can be observed that the increased randomness produces lower overall prediction confidence. Algorithmically higher randomness yields slower convergence of test error as a function of the forest size T.

8.2 Random ferns

Random ferns can also be thought of as a specific case of decision forests. In this case the additional constraint is that the same test parameters are used in all nodes of the same tree level [66, 68].

Figure 8.2 illustrates this point. As usual training points are indicated with coloured circles with different colours indicating different classes. In both a decision tree and a decision fern the first node (root) does an equally good job at splitting the training data into two subset. Here we consider only axis-aligned weak learners. In this example going to the next level starts to show the difference between the two models (fig. 8.2d). The fact that all parameters $\boldsymbol{\theta}$ of all nodes in the same level are identical induces partitions of the feature space with complete hyper-surfaces (as opposed to the "half-surfaces" used by the forest, see fig. 8.2d,e). Consequently, in order to split exactly the linearly separable input dataset the fern requires deeper levels than the forest. This explains why in fig. 8.1c we see lower prediction confidence (very washed-out colours) as compared to extremely randomized trees or full forests.

The fact that extremely randomized trees and random ferns are lower-parametric versions of decision forests can be an advantage in some situations. For instance, in the presence of limited training data ERT and ferns run less risk of overfitting than forests. Thus, as usual the best optimal model to use depends on the application at hand.

8.3 Online forest training

One of the advantages of decision forests is that thanks to their parallelism they are efficient both during training and testing. Most of the time they are used in an off-line way, *i.e.* they are trained on a training data and then tested on previously unseen test data. The entirety of the training data is assumed given in advance. However, there are many situations where the labelled training data may be arriving at different points in time. In such cases it is convenient to be able to update the learned forest quickly, without having to start training from scratch. This second mode of training is often referred to as *on-line* training.

Given a forest trained on a starting training set, the simplest form of on-line training is that of keeping the learned parameters and forest structure fixed and only update the leaf distributions. As new training data is available it can be simply "pushed through" all trees until it reaches the corresponding leaves. Then, the corresponding distributions (*e.g.* unnormalized histograms) can be quickly updated (*e.g.* by sim-

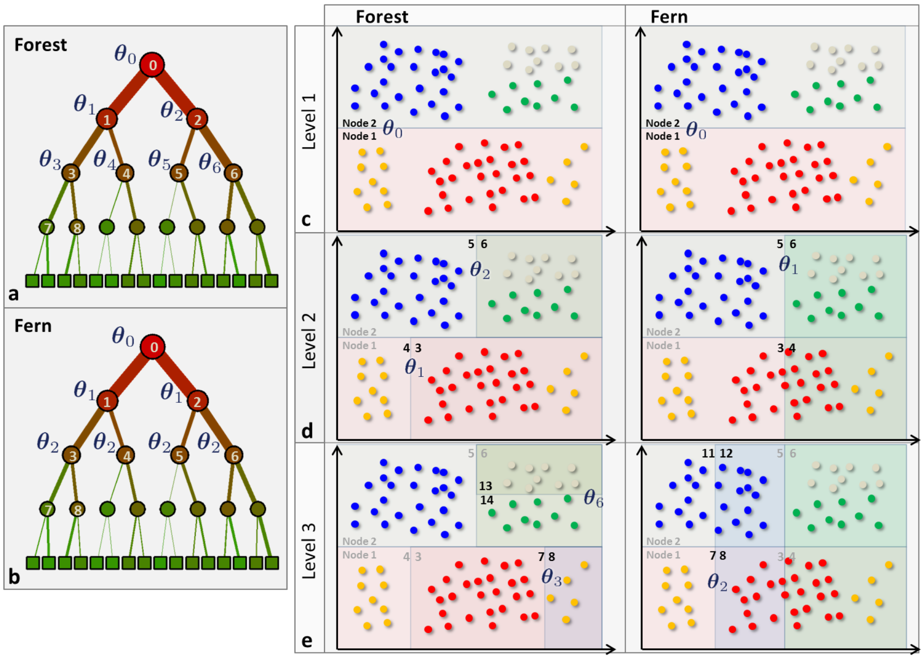


Fig. 8.2: **Forests and ferns.** A set of labelled training data is used to train a forest and a fern. Here simple axis-aligned weak learners are employed. A fern has fewer parameters than the forest and thus the fern typically requires deeper trees than a forest to split equally well the input training data.

ply adding the new counts in the appropriate bins). The work in [77] presents further details.

8.4 Structured-output Forests

Often decision forests are used for the semantic segmentation of images. This involves assigning a class posterior to each pixel (voxel) in the image domain (*e.g.* in Microsoft Kinect). However, such class decisions are often made independently for each pixel. Classic Markov random fields [8] add generic spatial priors to achieve more homogeneous outputs by smoothing noisy local evidence. In this section we mention two techniques which try to improve on this generic smoothness model and learn a class-specific model of spatial context.

8.4.1 Entangled forests

Entangled forests [60] are decision forests where the feature vector used as input to a split node is a function of: (i) the image data *and* (ii) the output of previous split nodes in the same tree.

The basic idea stems from the work on Probabilistic Boosting Trees [95] and autocontext [96]. In the latter the author shows how a sequence of trees, where each uses the output of the previous tree as input, yields better results than using a single tree. In fact, each stage moves us one step closer from the original image data to its "semantic" meaning.

However, due to their hierarchical structure each tree is composed of multiple subtrees. So, the idea of taking the output of a tree as input for the next can also be applied *within* the same decision tree/forest, as shown in [60]. In [60] the authors extend the feature pool by using both image intensities and various combinations of class posteriors extracted at different internal nodes in a classification forest. They show much improved generalization with shallower (and thus more efficient) forests. One of the reasons why entangled forests work well is because of learned, class-specific context. For example, the system learns that a voxel which is 5cm below the right lung and 5cm above the right kidney is likely to be in the liver.

Biased randomization. The work in [60] also introduces a variant on randomized node optimization where the available test parameters $\mathcal{T}_j$ are no longer drawn uniformly from $\mathcal{T}$, but according to a learned proposal distribution. This increases both training efficiency and testing accuracy as it reduces the enormous search space (possibly infinite) to a more manageable subset which is still highly discriminative.

8.4.2 Decision tree fields

Recently, Nowozin et al. [65] proposed another technique for learning class-specific models of context. They combined random decision forests and random fields together in a decision tree field model. In this model, both the per-pixel likelihoods as well as the spatial smoothing priors are dependent on the underlying image content. Different types of pair-wise weights are learned from images using randomized decision trees. By using approximate likelihood functions the training of the decision tree field model remains efficient, however, the test-time inference requires the minimization of a random field energy and therefore may prohibit its use in real-time applications, at present.

8.5 Further forest variants

The “STAR” model in [69] can also be interpreted as a forest of T, randomly trained non-binary trees of depth $D = 1$. The corresponding training and testing algorithms are computational efficient. A related model, made of multiple single nodes is “node harvest” [57]. Node harvest has the advantage of high interpretability, but seems to work best in low signal-to-noise conditions.

This chapter has presented only a small subset of all possible variants on tree-based machine learning techniques. Further interesting ones exist, but collecting all of them in the same document is a near impossible task.

Conclusions

This paper has presented a general model of decision forest and shown its applicability to various different tasks including: classification, regression, density estimation, manifold learning, semi-supervised learning and active learning.

We have presented both a tutorial on known forest-related concepts as well as a series of novel contributions such as demonstrating margin-maximizing properties, introducing forest-based density estimation and manifold forests, and discussing a new algorithm for transductive learning. Finally, we have studied for the first time the effects of important forest parameters such as the amount of randomness and the weak learner model on accuracy.

A key advantage of decision forests is that the associated inference algorithms can be implemented and optimized once. Yet relatively small changes to the model enable the user to solve many diverse tasks, depending on the application at hand. Decision forests can be applied to supervised, unsupervised and semi-supervised tasks.

The feasibility of the decision forest model has been demonstrated both theoretically and in practice, with synthetic experiments and in some commercial applications. Whenever possible, forest results have been compared directly with well known alternatives such as support vector machines, boosting and Gaussian processes. Amongst other advantages, the forest's intrinsic parallelism and consequent efficiency are very attractive for data-heavy practical applications.

Further research is necessary *e.g.* to figure out optimal ways of incorporating priors (*e.g.* of shape) within the forest and to increase their generalization further. An interesting avenue that some researchers have started to pursue is the idea of combining classification and regression [37]. This can be interesting as the two models can enrich one another. The more exploratory concepts of density forest, semi-supervised forest and manifold forests presented here need more testing in real applications to demonstrate their feasibility. We hope that this work can serve as a springboard for future exciting research to advance the state of the art in automatic image understanding for medical image analysis as well as general computer vision.

For further details, animations and demo videos, the interested reader is encouraged to view the additional material available at [1].

Appendix A – Deriving the regression information gain

This chapter shows the mathematical derivation leading to the continuous regression information gain measure in (4.2). We start by describing probabilistic linear regression.

Least squares line regression. For simplicity the following description focuses on fitting a line to a set of 2D points but it can be easily generalized to hyperplanes in a higher dimensional space. We are given a set of points (as shown in fig. 3) and we wish to estimate a probabilistic model of the line through those points. A 2D point $\mathbf{x}$ is represented in homogeneous coordinates as $\mathbf{x} = (x \;\; y \;\; 1)^\top$. A line in homogeneous coordinates is written as the 3-vector $\mathbf{l} = (l_x \;\; l_y \;\; l_w)^\top$. If a point is on the line then $\mathbf{l} \cdot \mathbf{x} = 0$. Thus, for n points we can setup the linear system

$$\mathtt{A}\,\mathbf{l} = \mathbf{0}$$

with the $n \times 3$ matrix $\mathtt{A}$

$$\mathtt{A} = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \cdot & \cdot & \cdot \\ x_n & y_n & 1 \end{pmatrix}.$$

The input points are in general noisy and thus it is not possible to find the line exactly. As usual in these cases we use the well known least squares technique where we define a cost function $C = \mathbf{l}^\top \mathtt{A}^\top \mathtt{A}\mathbf{l}$ to be minimized while satisfying the constraint $||\mathbf{l}|| = 1$. The corresponding Lagrangian is

$$\mathcal{L} = \mathbf{l}^\top \mathtt{A}^\top \mathtt{A}\mathbf{l} - \lambda(\mathbf{l}^\top \mathbf{l} - 1).$$

Taking the derivative of $\mathcal{L}$ and setting it to 0 as follows

$$\frac{\partial \mathcal{L}}{\partial \mathbf{l}} = 2\mathtt{A}^\top \mathbf{A}\mathbf{l} - 2\lambda\mathbf{l} = 0$$

leads to the following eigen-system:

$$\mathtt{A}^\top \mathtt{A}\mathbf{l} = \lambda\mathbf{l}.$$

Therefore, the optimal line solution $\bar{\mathbf{l}}$ is the eigenvector of the 3×3 matrix $\mathtt{M} = \mathtt{A}^\top \mathtt{A}$ corresponding to its minimum eigenvalue.

Estimating the distribution of line parameters. By assuming noisy training points and employing matrix perturbation theory [21, 89] we can estimate a Gaussian density of the line parameters: $\mathbf{l} \sim \mathcal{N}\left(\bar{\mathbf{l}}, \Lambda_l\right)$, as follows.

The generic i^{th} row in the "design" matrix $\mathtt{A}$ is $\mathbf{a}_i = (x_i \;\; y_i \;\; 1) = \mathbf{x}_i^\top$. Thus the corresponding covariance is

$$\mathcal{E}\left[\mathbf{a}_i^\top \mathbf{a}_i\right] = \Lambda_i$$

with $\mathcal{E}$ denoting expectation and where the point covariance Λ_i takes the form

$$\Lambda_i = \begin{pmatrix} \sigma_{x_i}^2 & \sigma_{x_i y_i} & 0 \\ \sigma_{x_i y_i} & \sigma_{y_i}^2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

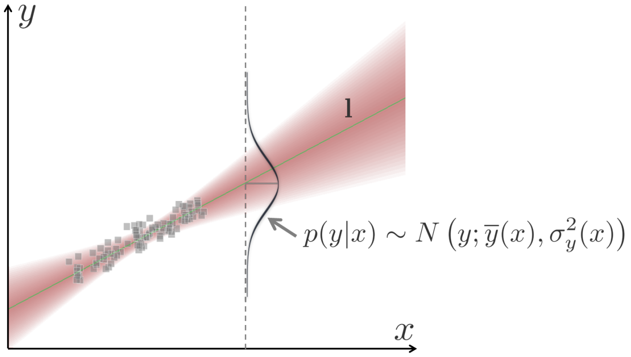


Fig. 3: **Probabilistic line fitting.** Given a set of training points we can fit a line model to them. For instance, in this example $\mathbf{l} \in \mathbb{R}^2$. Matrix perturbation theory enables us to compute the entire conditional density $p(\mathbf{l}|x)$ from where we can derive $p(y|x)$. Training a regression tree involves minimizing the uncertainty of the prediction $p(y|x)$. Therefore, the training objective is a function of σ_y^2.

Finally the 3×3 line covariance matrix is

$$\mathtt{\Lambda}_{\mathbf{l}} = \mathtt{J}\ \mathtt{S}\ \mathtt{J} \tag{1}$$

with the 3×3 Jacobian matrix

$$\mathtt{J} = -\sum_{k=2}^{3} \frac{\mathbf{u}_k \mathbf{u}_k^\top}{\lambda_k}$$

where λ_k denotes the k^{th} eigenvalues of the matrix $\mathtt{M}$ and $\mathbf{u}_k$ its corresponding eigenvector. The 3×3 matrix $\mathtt{S}$ in (1) is

$$\mathtt{S} = \sum_{i=1}^{n} \left(\mathbf{a}_i^\top \mathbf{a}_i \mathbf{l}^\top \mathtt{\Lambda}_i \mathbf{l} \right).$$

Therefore the distribution over $\mathbf{l}$ remains completely defined. Now, given a set of (x, y) pairs we have found the maximum-likelihood line model $\mathcal{N}\left(\bar{\mathbf{l}}, \mathtt{\Lambda}_l\right)$. However, what we want is the conditional distribution $p(y|x)$ (see fig. 3) this is discussed next.

Estimating the conditional $p(y|x)$**.** In regression forests we are given an input point x and the mean and covariance of the line parameters $\mathbf{l}$ for the leaf reached by the input point. The task now is to estimate of the conditional probability $p(y|x)$. At the end of this chapter we will see how this is used in the regression information gain.

In its explicit form a line equation is $y = a\ x + b$ with $a = -l_x/l_y$ and $b = -l_w/l_y$. Thus we can define $\mathbf{l}' = (a\ b)^\top$ with

$$\mathbf{l}' = f(\mathbf{l}) = \begin{pmatrix} -l_x/l_y \\ -l_w/l_y \end{pmatrix}.$$

Its 2×2 covariance is then $\Lambda_{\mathbf{l}'} = \boldsymbol{\nabla} f\ \Lambda_{\mathbf{l}}\ \boldsymbol{\nabla} f^\top$ with

$$\boldsymbol{\nabla} f = \begin{pmatrix} -\frac{1}{l_y} & \frac{l_x}{l_y^2} & 0 \\ 0 & \frac{l_w}{l_y^2} & -\frac{1}{l_y} \end{pmatrix}$$

Now we can rewrite the line equation as $y = g(\overline{\mathbf{x}}) = \mathbf{l}' \cdot \overline{\mathbf{x}}$ with $\overline{\mathbf{x}} = (x\ 1)^\top$ and the variance of y becomes

$$\sigma_y^2(x) = \boldsymbol{\nabla} g\ \Lambda_{\mathbf{l}'}\ \boldsymbol{\nabla} g^\top$$

with $\boldsymbol{\nabla} g = \overline{\mathbf{x}}^\top$. So, finally the conditional density $p(y|x)$ remains defined as

$$p(y|x) = N\left(y; \overline{y}, \sigma_y^2(x)\right). \tag{2}$$

See also fig. 3.

Regression information gain. In a regression forest the objective function of the j^{th} split node is

$$I_j = H(\mathcal{S}_j) - \sum_{\mathtt{i} \in \{\mathtt{L},\mathtt{R}\}} \frac{|\mathcal{S}_j^{\mathtt{i}}|}{|\mathcal{S}_j|} H(\mathcal{S}_j^{\mathtt{i}}) \tag{3}$$

with the entropy for a generic training subset $\mathcal{S}$ defined as

$$H(\mathcal{S}) = -\frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \int_y p(y|x) \log p(y|x)\ dy \tag{4}$$

by substituting (2) in (4) we obtain

$$H(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \frac{1}{2} \log\left((2\pi e)^2 \sigma_y^2(x)\right)$$

which when plugged into (3) yields the information gain

$$I_j \propto \sum_{x_j \in \mathcal{S}_j} \log\left(\sigma_y(x_j)\right) - \sum_{\mathtt{i} \in \{\mathtt{L},\mathtt{R}\}} \left(\sum_{x_j \in \mathcal{S}_j^{\mathtt{i}}} \log\left(\sigma_y(x_j)\right) \right)$$

up to a constant scale factor which has no influence over the node optimization procedure and thus can be ignored.

In this appendix we have derived the regression information gain for the simple case of 1D input x and 1D output y. It is easy to upgrade the derivation to multivariate variables, yielding the more general regression information gain in (4.2).

Acknowledgements

The authors would like to thank: J. Winn, M. Szummer, D. Robertson, T. Sharp, C. Rother, S. Nowozin, P. Kohli, B. Glocker, A. Fitzgibbon, A. Zisserman, K. Simonyan, A. Vedaldi, N. Ayache and A. Blake for the many, inspiring and often animated conversations on decision forests, their theory and their practical issues.

References

[1] http://research.microsoft.com/groups/vision/decisionforests.aspx.

[2] http://research.microsoft.com/projects/medicalimageanalysis/.

[3] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588, 1997.

[4] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 2003.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[6] C. M. Bishop, M. Svensen, and C. K. I. Williams. GTM: the generative topographic mapping. *Neural Computation*, 1998.

[7] A. Bjoerck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[8] A. Blake, P. Kohli, and C. Rother. *Markov Random Fields for Vision and Image Processing*. The MIT Press, 2011.

[9] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *IEEE ICCV*, 2007.

[10] L. Breiman. Random forests. Technical Report TR567, UC Berkeley, 1999.

[11] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[12] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[13] I. Budvytis, V. Badrinarayanan, and R. Cipolla. Semi-supervised video segmentation using tree structured graphical models. In *CVPR*, 2011.

[14] A. Burr. Active learning literature survey. Technical Report 2010-09-14, Univ. Wisconsin Madison, 2010. Computer Sciences Technical Report.

[15] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *International Conference on Machine Learning*, pages 96–103, 2008.

[16] L. Cayton. Algorithms for manifold learning. Technical Report CS2008-0923, UCSD, 2005.

[17] P. Chandna, S. Deswal, and M. Pal. Semi-supervised learning based prediction of musculoskeletal disorder risk. *Journal of Industrial and Systems Engineering*, 2010.

[18] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning.* MIT Press, Cambridge, MA, 2006.

[19] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

[20] K. Crammer and Y. Singer. On the algorithmic implementation of multi-class SVMs. *Journal of Machine Learning Research*, 2001.

[21] A. Criminisi. *Accurate Visual metrology from Single and Multiple Uncalibrated Images.* Distinguished dissertation series. Springer, 2001.

[22] A. Criminisi, T. Sharp, and A. Blake. Geos: Geodesic image segmentation. In *Proc. ECCV*, 2008.

[23] A. Criminisi, J. Shotton, and S. Bucciarelli. Decision forests with long-range spatial context for organ localization in CT volumes. In *MICCAI workshop on Probabilistic Models for Medical Image Analysis*, 2009.

[24] A. Criminisi, J. Shotton, D. Robertson, and E. Konukoglu. Regression forests for efficient anatomy detection and localization in CT studies. In *MICCAI workshop on Medical Computer Vision: Recognition Techniques and Applications in Medical Imaging*, Beijing, 2010. Springer.

[25] J. De Porte, B. M. Herbst, W. Hereman, and S. J. van Der Walt. An introduction to diffusion maps. *Techniques*, 2008.

[26] L. Devroye. *Non-Uniform Random Variate Generation.* Springer-Verlag, New York, 1986.

[27] K. Driessens, P. Reutemann, B. Pfahringer, and C. Leschi. Using weighted nearest neighbour to benefit from unlabelled data. In *Advances in Knowledge Discovery and Data Mining. 10th Pacific-Asia Conference*, 2010.

[28] N. Duchateau, M. De Craene, G. Piella, and A. F. Frangi. Characterizing pathological deviations from normality using constrained manifold learning. In *MICCAI*, 2011.

[29] G. Fanelli and J. Gall. Real time head pose estimation with random regression forests. In *IEEE CVPR*, 2011.

[30] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. of the ACM*, 24:381–395, 1981.

[31] Y. Freund, S. Dasgupta, M. Kabra, and N. Verma. Learning the structure of manifolds using random projections. In *NIPS*, 2007.

[32] Y. Freund and R. E. Schapire. A decision theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 1997.

[33] J. Gall and V. Lempitsky. Class-specific Hough forest for object detection. In *IEEE CVPR*, Miami, 2009.

[34] S. Gerber, T. Tasdizen, S. Joshi, and R. Whitaker. On the manifold structure of the space of brain images. In *MICCAI*, 2009.

[35] E. Geremia, O. Clatz, B. H. Menze, E. Konukoglu, A. Criminisi, and N. Ayache. Spatial decision forests for MS lesion segmentation in multi-channel magnetic resonance. *Neuroimage*, 2011.

[36] Pierre Geurts. Extremely randomized trees. In *Machine Learning*, 2003.

[37] R. Girshick, J. Shotton, P. Kohli, A. Criminisi, and A. Fitzgibbon. Efficient regression of general-activity human poses from depth images. In *IEEE ICCV*, 2011.

[38] R. Guerrero, R. Wolz, and D. Rueckert. Laplacian eigenmaps manifold learning for landmark localization in brain MR images. In *MICCAI*, 2011.

[39] S. S. Gupta. Probability integrals of multivariate normal and multivariate t. *Annals of Mathematical Statistics*, 34(3), 1963.

[40] J. Hamm, D. H. Ye, R. Verma, and C. Davatzikos. GRAM: a framework for geodesic registration on anatomical manifolds. *Medical Image Analysis*, 14(5), 2010.

[41] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision (2nd ed.).* Cambridge University Press, 2003.

[42] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* 2001.

[43] D. Heath, S. Kasif, and S. Salzberg. Induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2(2):1–32, 1993.

[44] C. Hegde, M. B. Wakin, and R. G. Baraniuk. Random projections for manifold learning - proofs and analysis. In *NIPS*, 2007.

[45] T. K. Ho. Random decision forests. In *Intl Conf. on Document Analysis and Recognition*, pages 278–282, 1995.

[46] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Trans. PAMI*, 20(8):832–844, 1998.

[47] T. Joachims. *Advances in Kernel Methods - Support Vector Learning*, chapter Making Large-Scale SVM Learning Practical. The MIT Press, 1999.

[48] I. T. Jolliffe. *Principal Component Analysis.* Springer-Verlag, 1986.

[49] B. M. Kelm, S. Mittal, Y. Zheng, A. Tsymbal, D. Bernhardt, F. Vega-Higuera, K. S. Zhou, P. Meer, and D. Comaniciu. Detection, grading and classification of coronary stenoses in computed tomography angiography. In *MICCAI*, 2011.

[50] E. Konukoglu, A. Criminisi, S. Pathak, D. Robertson, S. White, D. Haynor, and K. Siddiqui. Robust linear registration of CT images using random regression forests. In *SPIE Medical Imaging*, 2011.

[51] C. H. Lampert. Kernel methods in computer vision. *Foundations and Trends in Computer Graphics and Vision*, 4(3), 2008.

[52] C. Leistner, A. Saffari, J. Santner, and H. Bischoff. Semi-supervised random forests. In *ICCV*, 2009.

[53] V. Lempitsky, M. Verhoek, A. Noble, and A. Blake. Random forest classification for automatic delineation of myocardium in real-time 3D echocardiography. In *Functional Imaging and Modelling of the Heart (FIMH)*, 2009.

[54] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Trans. PAMI*, 2006.

[55] J.B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of 5th berkeley Symposium on Mathematical Statistics and Pobability*. University of California Press, 1967.

[56] R. Maree, P. Geurts, J. Piater, and L. Wehenkel. Random subwindows for robust image classification. In *Proc. CVPR*, 2005.

[57] N. Meinshausen. Node harvest. *The Annals of Applied Statistics*, 4(4), 2010.

[58] B. Menze, B.M. Kelm, D.N. Splitthoff, U. Koethe, and F.A. Hamprecht. On oblique random forests. In *Proc. ECML/PKDD*, 2011.

[59] A. Montillo and H. Ling. Age regression from faces using random forests. In *ICIP*, 2009.

[60] A. Montillo, J. Shotton, J. E. Winn, Metaxas D. Iglesias, E., and A. Criminisi. Entangled decision forests and their application for semantic segmentation of CT images. In *Information Processing in Medical Imaging (IPMI)*, 2011.

[61] F. Moosman, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *Proc. NIPS*, 2006.

[62] B. Nadler, S. Lafon, R. R. Coifman, and I. G. Kevrekidis. Diffusion maps, spectral clustering and eigenfunctions of Fokker-Plank operators. In *Proc. of NIPS*, 2005.

[63] R. Navaratnam, A. W. Fitzgibbon, and R. Cipolla. The joint manifold model for semi-supervised multi-valued regression. In *ICCV*, 2007.

[64] R. M. Neal. Annealed importance sampling. *Statistics and Computing*, 11:125–139, 2001.

[65] S. Nowozin, C. Rother, S. Bagon, T. Sharp, B. Yao, and P. Kohli. Decision tree fields. In *ICCV*, 2011.

[66] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Trans. PAMI*, 32(3), 201.

[67] E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962.

[68] O. Pauly, B. Glocker, A. Criminisi, D. Mateus, A. Martinez Moller, S. Nekolla, and N. Navab. Fast multiple organs detection and localization in whole-body MR Dixon sequences. In *MICCAI*, Toronto, 2011.

[69] O. Pauly, D. Mateus, and N. Navab. STARS: A new ensemble partitioning approach. In *ICCV Workshop on Information Theory In Computer Vision and Pattern Recognition*, 2011.

[70] N. Payet and S. Todorovic. $(rf)^2$ random forest random field. In *NIPS*, 2010.

[71] R. L. Plackett. A reduction formlula for normal multivariate integrals. *Biometrika*, 41, 1954.

[72] J. R. Quinlan. *C4.5: Programs for Machine Learning.* 1993.

[73] C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning.* The MIT Press, 2006.

[74] G. Rogez, J. Rihan, S. Ramalingam, and P. Orrite, C. Torr. Randomized trees for human pose detection. In *CVPR*, 2008.

[75] C. Rosenberg, M. Hebert, and H. Schneiderman. Semi-supervised self-training of object detection models. In *Seventeenth IEEE Workshop on Applications of Computer Vision*, 2005.

[76] M. R. Sabuncu and K. V. Leemput. The relevance voxel machine (RVoxM): A bayesian method for image-based prediction. In *MICCAI*, 2011.

[77] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischoff. On-line random forests. In *ICCV workshop on On-Line learning for Computer Vision*, 2009.

[78] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[79] G.A.F. Seber and C.J. Wild. *Non Linear Regression.* New York: John Wiley and Sons, 1989.

[80] T. Sharp. Implementing decision trees and forests on a GPU. In *ECCV*, 2008.

[81] J. Shi and J. Malik. Normalized cuts and image segmentation. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, Washington, DC, USA, 1997.

[82] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from a single depth image. In *IEEE CVPR*, 2011.

[83] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *IEEE CVPR*, 2008.

[84] B. W. Silverman. *Density Estimation.* Chapman and Hall, London, 1986.

[85] J. Skilling. Bayesian inference and maximum entropy methods in science and engineering. In *AIP*, 2004.

[86] A. J. Smola and B. Scholkopf. A tutorial on support vector regression. Technical report, 1998.

[87] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large Scale Multiple Kernel Learning. *Journal of Machine Learning Research*, 7, July 2006.

[88] A. Statnikov, L. Wang, and C. A. Aliferis. A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. *BMC Bioinformatics*, 2008.

[89] G. Stewart and J.G. Sun. *Matrix Perturbation Theory.* Elsevier, 1990.

[90] G. J. Szekely and M. L. Rizzo. Testing for equal distributions in high dimensions. *Interstat*, Nov 2004.

[91] M. Szummer and T. Jaakkola. Partially labelled classification with markov random walks. In *NIPS*, 2001.

[92] J. B. Tenenbaum, V. deSilva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, (290), 2000.

[93] M. E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

[94] A. Torralba, K. P. Murphy, and W. T. Freeman. Sharing visual features for multiclass and multiview object detection. *IEEE Trans. PAMI*, 2007.

[95] Z. Tu. Probabilistic boosting-tree: Learning discriminative models for classification, recognition, and clustering. In *Proc. of the 10-th IEEE Intl Conference on Computer Vision (ICCV)*, pages 1589–1596, 2005.

[96] Z. Tu, K. L. Narr, P. Dollar, I. Dinov, P. M. Thompson, and A. W. Toga. Brain anatomical structure segmentation by hybrid discriminative/generative models. *IEEE Trans. on Medical Imaging*, 27(4), 2008.

[97] V. Vapnik. *The nature of statistical learning theory.* Springer Verlag, 2000.

[98] P. Viola and M. J. Jones. Robust real-time face detection. *IJCV*, 2004.

[99] P. Viola, M. J. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. In *ICCV*, 2003.

[100] Microsoft Corp. Redmond WA. Kinect for XBox 360.

[101] J. Wang. On transductive support vector machines. In *Prediction and Discovery*. American Mathematical Society, 2007.

[102] P. Yin, A. Criminisi, I. Essa, and J. Winn. Tree-based classifiers for bilayer video segmentation. In *CVPR*, 2007.

[103] Q. Zhang, R. Souvenir, and R. Pless. On manifold structure of cardiac MRI data: Application to segmentation. In *IEEE Computer Vision and Pattern Recognition*, Los Alamitos, CA, USA, 2006.

[104] X. Zhu and A. Goldberg. *Introduction to Semi-Supervised Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2009.

[105] A. Zien and C. S. Ong. Multiclass multiple kernel learning. In *ICML*, 2007.