Foundations and Trends® in
Computer Graphics and Vision
Vol. 7, Nos. 2–3 (2011) 81–227

DOI: 10.1561/0600000035

now
the essence of knowledge

Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning

By Antonio Criminisi, Jamie Shotton, and Ender Konukoglu

Contents

Foundations and Trends® in
Computer Graphics and Vision
Vol. 7, Nos. 2–3 (2011) 81–227

DOI: 10.1561/0600000035

now
the essence of knowledge

Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning

Antonio Criminisi[1], Jamie Shotton[2], and Ender Konukoglu[3]

[1] *Microsoft Research Ltd., 7 J J Thomson Ave, Cambridge, CB3 0FB, UK, antcrim@microsoft.com*
[2] *Microsoft Research Ltd., 7 J J Thomson Ave, Cambridge, CB3 0FB, UK, jamiesho@microsoft.com*
[3] *Microsoft Research Ltd., 7 J J Thomson Ave, Cambridge, CB3 0FB, UK, enderk@microsoft.com*

Abstract

This review 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 gives us the opportunity to write and optimize the core implementation only once, with application to many diverse tasks.

The proposed model may be used both in a discriminative or generative way and may be applied to discrete or continuous, labeled or unlabeled data.

The main contributions of this review are: (1) Proposing a unified, probabilistic and efficient model for a variety of learning tasks; (2) Demonstrating margin-maximizing properties of classification forests; (3) Discussing probabilistic regression forests in comparison with other nonlinear regression algorithms; (4) Introducing density forests for estimating probability density functions; (5) Proposing an efficient algorithm for sampling from a density forest; (6) Introducing manifold forests for nonlinear dimensionality reduction; (7) Proposing new algorithms for transductive learning and active learning. Finally, we discuss how alternatives such as random ferns and extremely randomized trees stem from our more general forest model.

This document is directed at both students who wish to learn the basics of decision forests, as well as researchers interested in the 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.

1

Overview and Scope

This review 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 review, we formulate diverse learning tasks such as regression, classification and semi-supervised learning as instances of the same general decision forest model. The unified framework further extends to novel uses of forests in tasks such as density estimation and manifold learning. The underlying unified framework gives us the opportunity to implement and optimize the general algorithm for all these tasks only once, and then adapt it to individual applications with relatively small changes.

This review 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 sections.

1.1 A Chronological Literature Review

One of the earlier works on decision trees is the seminal "Classification and Regression Trees (CART)" book by Breiman et al. [12], where the authors describe the basics of decision trees and their use for both classification and regression problems. Following that publication researchers then focused on algorithms for constructing (learning) optimal decision trees for different tasks using available training data. For this purpose, one of the most popular algorithms is "C4.5" of Quinlan [81]. Although, decision trees were proven to be useful, their application remained limited to relatively low dimensional data.

In the nineties, researchers discovered how using ensembles of learners (e.g., generic "weak" classifiers) yields greater accuracy and generalization. This seems particulary true for high dimensional data, as often encountered in real life applications. One of the earliest references to ensemble methods is the boosting algorithm of Schapire [87], where the author discusses how iterative re-weighting of training data can be used to build "strong" classifiers as linear combination of many weak ones.

Combining the ideas of decision trees and ensemble methods gave rise to decision forests, that is, ensembles of randomly trained decision trees. The idea of constructing and using ensembles of trees with randomly generated node tests was introduced for the first time in the work of Amit and Geman [1, 2] for handwritten digit recognition. In that work the authors also propose using the mean of the tree probabilities as output of the tree ensemble.

In the subsequent work of Ho [47] tree training via randomized partitioning of the feature space is discussed further, and in [48] 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.

Breiman's later work in [10, 11] further consolidated the role of random forests and popularized their use. There, the author introduces a different way of injecting randomness in the forest by randomly sampling the labeled training data (namely "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 [107, 108]. Random decision forests where used in [63] for image classification and in [60] for keypoint tracking in videos. Recent years have seen an explosion of forest-based techniques in the machine learning, vision and medical imaging literature [9, 15, 25, 31, 35, 37, 58, 59, 65, 67, 68, 69, 74, 79, 89, 92, 97, 110]. Decision forests compare favorably 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 [39, 91, 66].

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 a *classification* task, where the desired 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 thought of as a *regression* problem. In this case the output is a continuous variable.
- Detecting abnormalities in a medical scan can be achieved by evaluating the image under a learned probability *density* function for scans of healthy individuals.
- Capturing the intrinsic variability of size and shape of different structures in the human brain from magnetic resonance images may be cast as *manifold learning*.

- Interactive image segmentation may be thought of as a *semi-supervised* problem, where the user's brush strokes define labeled data and the rest of image pixels provide *already available* unlabeled data.
- Learning a general rule for detecting tumors in images using minimal amount of manual annotations is an *active learning* problem, where expensive expert annotations can be optimally acquired in the most economical fashion.

The popularity of decision forests is mostly due to their recent success in classification tasks. However, here we show that forests are a more general tool which can be applied to many additional problems. This section 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.

The unification we present, yields both theoretical and practical advantages. In fact, we show how multiple prototypical machine learning problems can all be mapped onto the same general model by means of different parametrizations. As a result, properties of the general framework are inherited by the specific instantiations. The major practical advantage of such unification is that one can implement and optimize the associated inference algorithms only once and then use them, with relatively small modifications, in many applications.

This section presents the model definitions and components in an abstract manner. Some brief concrete examples for different tasks are presented here, with further details in the relevant sections. Before delving into the model description we first provide an intuitive explanation for the basic principles of decision trees. Then we introduce the general mathematical notation that will be used throughout the manuscript. Finally, we extend the tree formalism to the decision forest model.

2.1 Decision Tree Basics

Decision trees have been around for a number of years [12, 81]. Their recent revival is mostly due to the discovery that ensembles of slightly different trees tend to produce higher accuracy on previously unseen

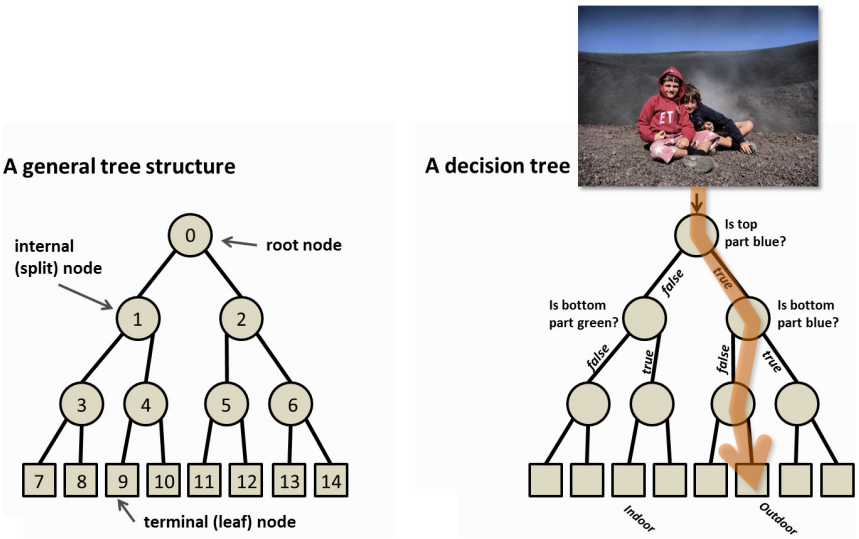


Fig. 2.1 Decision tree. (a) A tree is a set of nodes and edges organized in a hierarchical fashion. A tree is a graph with no loops. Internal nodes are denoted with circles and terminal nodes with squares. (b) A decision tree is a tree where each internal node stores a split (or test) function to be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

data, a property known as *generalization* [2, 11, 47]. Ensembles of trees will be discussed extensively throughout this review. But let us first focus on individual trees.

Tree data structure. A tree is a special type of graph. It is a data structure made of a collection of nodes and edges organized in a hierarchical fashion (Figure 2.1(a)). 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. In contrast to general graphs a tree cannot contain loops. In this review we focus only on binary trees where each internal node has exactly two outgoing edges.

Decision tree. A *decision* tree is a set of questions organized in a hierarchical manner and represented graphically as a tree. For a

given input object, a decision tree estimates an unknown property of the object by asking successive questions about its known properties. Which question to ask next depends on the answer of the previous question and this relationship is represented graphically as a path through the tree which the object follows. The decision is then made based on the terminal node on the path.

For example, imagine we have a photograph and we need to construct an algorithm for figuring out automatically whether it displays an indoor or an outdoor scene. We have no other information but the image pixels. We can start by looking at the top part of the image and ask whether it is blue or not. If it is then that might be the sky. Based on this, we ask another question, for instance whether the bottom part is also blue. If it is not then our belief that this photograph displays an outdoor scene increases. However, if the bottom part of the photo is also blue then perhaps it is an indoor scene and we are looking at a blue wall.

All these questions/tests help our decision making move toward the correct region of the decision space. Also, the more questions the higher the confidence in the response. The tests can be represented hierarchically via a decision tree structure. In a decision tree, each internal node is associated with one such question. In our example, we can think of the image as being injected at the root node, and a test being applied to it (see Figure 2.1(b)). Based on the result of this first test the whole 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 most probable answer based on the questions asked throughout (e.g., "outdoor"). Therefore, key to the good functioning of a decision tree is to establish: (i) the tests associated to each internal node and (ii) the decision-making predictors associated with each leaf.

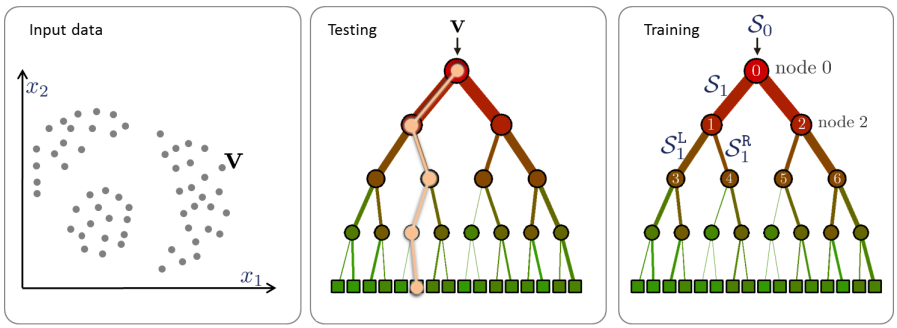
A decision tree can also be thought of as a technique for splitting complex problems into a set of simpler ones. It is a hierarchical piecewise model. Its parameters (i.e., all node tests, the leaves predictors etc.) could be selected by hand for simple problems. In more complex problems (such as vision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

2.2 Mathematical Notation and Basic Definitions

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., $\mathcal{S}$).

Data point and features. A generic object, called *data point*, is denoted by a vector $\mathbf{v} = (x_1, x_2, \ldots, x_d) \in \mathcal{F}$, where the components x_i represent some attributes of the data point, called *features*, see Figure 2.2(a) for an illustration. These features may vary from application to application. For instance, in a computer vision application $\mathbf{v}$ may correspond to a pixel in an image and the x_is represent the responses of a chosen filter bank at that particular location.

The number of features naturally depends on the type of the data point as well as the application. In theory, the dimensionality of the feature space $\mathcal{F}$, d, can be very large, even infinite. In practice, it is often not possible, and further not necessary, to extract all d dimensions of $\mathbf{v}$ ahead of time. Instead we extract only a small portion of d on an as-needed basis. Based on this let us formulate the features of interest that are computed at any single time to be a subset selected from the set



(a) (b) (c)

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 $\mathcal{S}_0$ into the tree and optimizing the parameters of the split nodes so as to optimize a previously energy function. See text for details.

of all possible features as $\boldsymbol{\phi}(\mathbf{v}) = (x_{\phi_1}, x_{\phi_2}, \ldots, x_{\phi_{d'}}) \in \mathcal{F}^{d'} \subset \mathcal{F}$, where d' denotes the dimensionality of the subspace and $\phi_i \in [1, d]$ denote the selected dimensions. In most applications, d can be very large but the dimension of the subspace $\mathcal{F}^{d'}$ is much smaller $d' \ll d$.

Test functions, split functions and weak learners. As explained above a decision tree is a set of tests that are hierarchically organized. In this review we use the terms "split function," "test function," and "weak learner" interchangeably. Each node has associated a different test function. We formulate a test function at a split node j as a function with binary outputs

$$h(\mathbf{v}, \boldsymbol{\theta}_j) : \mathcal{F} \times \mathcal{T} \rightarrow \{0, 1\}, \tag{2.1}$$

where 0 and 1 can be interpreted as "false" and "true" respectively, $\boldsymbol{\theta}_j \in \mathcal{T}$ denote the parameters of the test function at the jth split node. The data point $\mathbf{v}$ arriving at the split node is sent to its left or right child node according to the result of the test function (see Figure 2.3(a)).

Training points and training sets. The last definitions we introduce are the *training point* and the *training set*. A training point is a data point for which the attributes that we are seeking for are actually known. In the example of the previous section a training set would be a set of photos with associated "indoor" or "outdoor" labels. Based on

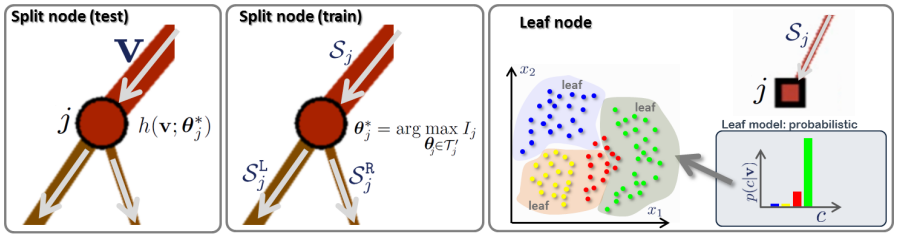


Fig. 2.3 Split and leaf nodes. (a) Split node (testing). A split node is associated with a weak learner (or split function, or test function). (b) Split node (training). Training the parameters $\boldsymbol{\theta}_j$ of node j involves optimizing a chosen objective function (maximizing the information 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.

this definition, a training set, denoted by $\mathcal{S}_0$, is a collection of different training points (Figure 2.2(c)).

In a supervised task a training point is a pair $(\mathbf{v},\mathbf{y})$ where $\mathbf{v}$ is the input feature vector and $\mathbf{y}$ here represents a generic, known label. In an unsupervised task the training points are represented only by their feature response and there is no associated label.

When discussing trees it is convenient to think of subsets of training points as being 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 Figure 2.2(c)), 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.

2.3 Randomly Trained Decision Trees

As mentioned already the functioning of decision trees can be separated into an off-line phase (training) and an on-line one (testing). Here we describe these two phases as well as the other components of the random decision trees that are used in these phases. We take a general approach and keep definitions and explanations at an abstract level.

2.3.1 Tree Testing (on-line)

The basic principle of tree testing is simple. Given a previously unseen data point $\mathbf{v}$ a decision tree hierarchically applies a number of predefined tests (see Figure 2.2(b)). Starting at the root, each split node applies its associated test function $h(\cdot,\cdot)$ to $\mathbf{v}$. Depending on the result

of this *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. The leaf nodes contain a predictor/estimator (e.g., a classifier or a regressor) which associates an output (e.g., a class label or a continuous value) with the input $\mathbf{v}$.

2.3.2 Tree Training (off-line)

The split functions stored at the internal nodes are key for the functioning of the tree. One may think of designing these functions manually. However, this approach would only be possible for very simple problems. For more realistic problems the test functions need to be learned automatically, from exemplar data. Thus, the training phase takes care of selecting the type and parameters of the test function $h(\mathbf{v},\boldsymbol{\theta}_j)$ associated with each split node (indexed by j) by optimizing a chosen objective function defined on an available training set.

The optimization of the split functions proceeds in a greedy manner. At each node j, depending on the subset of the incoming training set $\mathcal{S}_j$ we *learn* the function that "best" splits $\mathcal{S}_j$ into $\mathcal{S}_j^{\mathtt{R}}$ and $\mathcal{S}_j^{\mathtt{L}}$. This problem is formulated as the maximization of an objective function at that node

$$\boldsymbol{\theta}_j^* = \arg\max_{\boldsymbol{\theta}_j \in \mathcal{T}} I_j \tag{2.2}$$

with

$$\begin{aligned} I_j &= I(\mathcal{S}_j, \mathcal{S}_j^{\mathtt{L}}, \mathcal{S}_j^{\mathtt{R}}, \boldsymbol{\theta}_j) \\ \mathcal{S}_j^{\mathtt{L}} &= \{(\mathbf{v},\mathbf{y}) \in \mathcal{S}_j | h(\mathbf{v},\boldsymbol{\theta}_j) = 0\} \\ \mathcal{S}_j^{\mathtt{R}} &= \{(\mathbf{v},\mathbf{y}) \in \mathcal{S}_j | h(\mathbf{v},\boldsymbol{\theta}_j) = 1\} \end{aligned} \tag{2.3}$$

As before, the symbols $\mathcal{S}_j, \mathcal{S}_j^{\mathtt{L}}, \mathcal{S}_j^{\mathtt{R}}$ denote the sets of training points before and after the split (see Figures 2.2(b) and 2.3(b)). The objective function (2.3) is of an abstract form here. Its precise definition and the meaning of "best" depends on the task at hand (e.g., supervised or not,

[1] In this review 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.

continuous or discrete output). For instance, for binary classification, the term "best" can be defined as splitting the training subset $\mathcal{S}_j$ such that the resulting child nodes are as pure as possible, that is, containing only training points of a single class. In this case the objective function can, for instance, be defined as the information gain. Precise definitions and more task-specific details will be given in later sections.

During training we also need to optimize the tree structure (shape). Training starts at the root node, $j = 0$, where the optimum split parameters are found as described earlier. Thus, we construct two child nodes, each receiving a different disjoint subset of the training set. This procedure is then applied to all the newly constructed nodes and the training phase continues. The structure of the tree depends on how and when we decide to stop growing various branches of the tree. Diverse stopping criteria can be applied. For example it is common to stop the tree when a maximum number of levels D has been reached. Alternatively, one can impose a minimum value of the maximum $\max_{\boldsymbol{\theta}_j} I_j$, in other words we stop when the seeked-for attributes of the training points within the leaf node are similar to one another. Tree growing may also be stopped when a node contains too few training points. Avoiding growing full trees[2] has been demonstrated to have positive effects in terms of generalization. In this survey we avoid further post-hoc operations such as tree pruning [44] to keep the training process as simple as possible.

At the end of the training phase we obtain: (i) the (greedily) optimum weak learners associated with each node, (ii) a learned tree structure, and (iii) a different set of training points at each leaf.

2.3.3 Weak Learner Models

The split functions play a crucial role both in training and testing. Up-to-now we have refrained from defining a specific form for these models. In this section, we provide a simple geometric parametrization and a few derived formulations, which will be used throughout this review. We formulate the parametrization of the weak learner model as $\boldsymbol{\theta} = (\boldsymbol{\phi}, \boldsymbol{\psi}, \boldsymbol{\tau})$, where $\boldsymbol{\psi}$ defines the geometric primitive used to separate the data (e.g., an axis-aligned hyperplane, an oblique hyperplane [45, 65],

[2] The term full tree here means a tree where each leaf contains only one training point.

a general surface etc.). The parameter vector $\boldsymbol{\tau}$ 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 $\mathbf{v}$. The optimization given in (2.2) is then defined over all these three parameters. Figure 2.4 illustrates a few possible weak learner models, for example:

Linear data separation. The first parametrization we define is the linear model

$$h(\mathbf{v}, \boldsymbol{\theta}_j) = [\tau_1 > \boldsymbol{\phi}(\mathbf{v}) \cdot \boldsymbol{\psi} > \tau_2], \tag{2.4}$$

where $[\cdot]$ is the indicator function.[3] For instance, in the 2D example in Figure 2.4(b) $\boldsymbol{\phi}(\mathbf{v}) = (x_1\ x_2\ 1)^\top$, and $\boldsymbol{\psi} \in \mathbb{R}^3$ denotes a generic line in homogeneous coordinates. In (2.4) setting $\tau_1 = \infty$ or $\tau_2 = -\infty$ corresponds to using a single-inequality test 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 Figure 2.4(a)). Such axis-aligned weak learners are often used in the boosting literature and they are referred to as *stumps* [107].

Please note that the axis aligned case is over-parametrized in (2.4). Here we choose this parametrization because it highlights the role of the geometric model $\boldsymbol{\psi}$ and it generalizes to more complex cases.

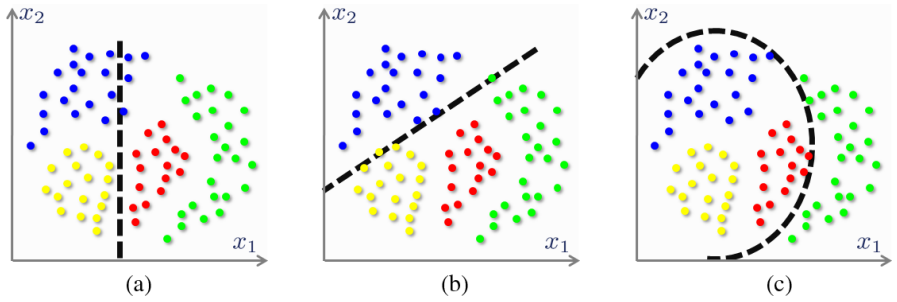


Fig. 2.4 Example weak learners. (a) Axis-aligned hyperplane. (b) General oriented hyperplane. (c) Quadratic (conic in 2D). For ease of visualization here we have $\mathbf{v} = (x_1\ x_2) \in \mathbb{R}^2$ and $\boldsymbol{\phi}(\mathbf{v}) = (x_1\ x_2\ 1)$ in homogeneous coordinates. In general data points $\mathbf{v}$ may have a much higher dimensionality and $\boldsymbol{\phi}$ still a dimensionality of ≤ 2.

[3] 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

$$h(\mathbf{v}, \boldsymbol{\theta}_j) = [\tau_1 > \boldsymbol{\phi}^\top(\mathbf{v})\ \boldsymbol{\psi}\ \boldsymbol{\phi}(\mathbf{v}) > \tau_2] \tag{2.5}$$

with $\boldsymbol{\psi} \in \mathbb{R}^{3\times3}$ 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.

Here we discuss simple weak learner models. But one may use more complex node test functions such as SVM, boosting etc. [104, 110]. However, care must be taken in selecting the complexity of the node test function. In fact, as shown later, the number of degrees of freedom of the weak learner influences heavily the forest generalization properties.

2.3.4 Energy Models

The objective function used during training is essential in constructing decision trees that will perform the desired task. In fact, the result of the optimization problem in (2.2) determines the parameters of the weak learners, which, in turn, determines the path followed by a data and thus its prediction. In summary, through its influence on the choice of weak learners the energy model determines the prediction and estimation behavior of a decision tree.

Developing task specific energy models is an active research area. In this section we discuss the components which are in common to the most widely used objective functions. Later sections will show how small modifications of the training objective lead to different tasks.

The tree training phase is driven by the statistics of the training set. The basic building blocks of the training objective function are the concepts of *entropy* and *information gain*. These concepts are usually discussed in information theory or probability courses. Here we briefly explain them from the point of view of the decision trees and illustrate them with toy examples in Figures 2.5 and 2.6.

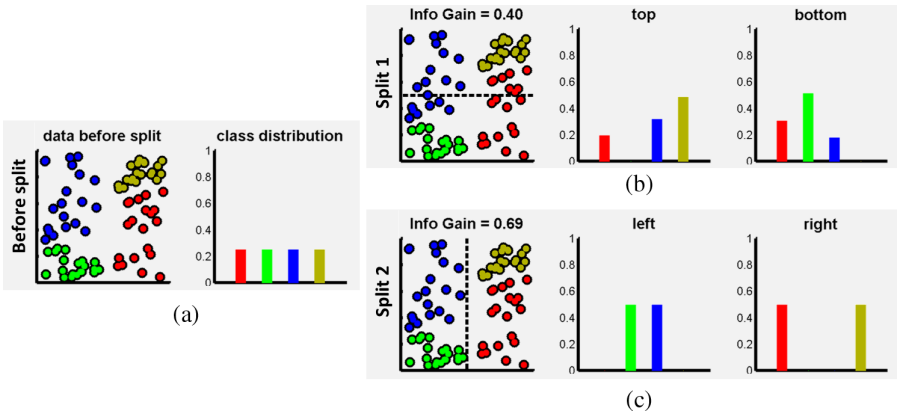


Fig. 2.5 Information gain for discrete, non-parametric distributions. (a) Dataset $\mathcal{S}$ before a split. (b) After a horizontal split. (c) After a vertical split. In this example the vertical split produces purer class distributions in the child nodes. Classes are colour coded.

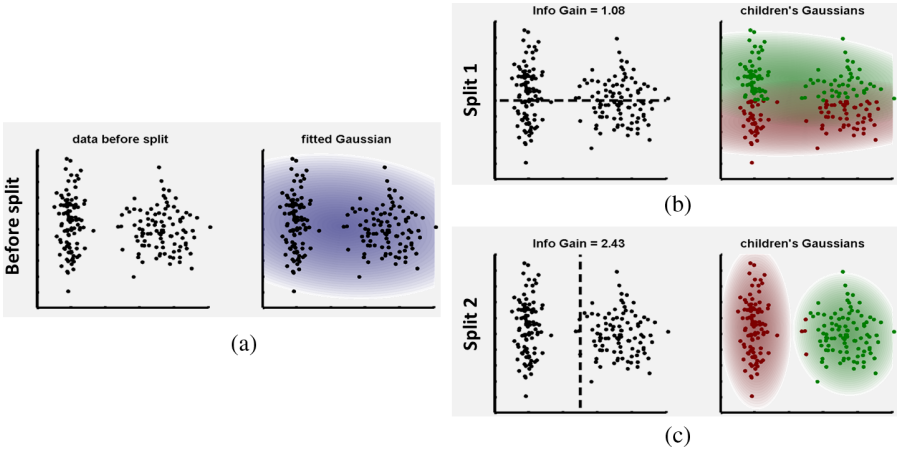


Fig. 2.6 Information gain for continuous, parametric densities. (a) Dataset $\mathcal{S}$ before a split. (b) After a horizontal split. (c) After a vertical split. A vertical split produces better separation and a correspondingly higher information gain.

Figure 2.5 illustrates a toy classification example. The graph in Figure 2.5(a) shows a number of training points on a 2D space, where each coordinate denotes a feature value and the colors indicate the known classes. As mentioned previously, during training our aim is to learn the parameters that best split the training data. In this example our objective is to separate different classes as much as possible. For instance,

if we split the training data horizontally (as shown in Figure 2.5(b)) this produces two sets of data. Each set now contains points from three classes while before the split the dataset had all four classes. Thus, if we use the children (rather than the parent node) we would have more chances of correct prediction; we have reduced the *uncertainty* of prediction. This intuitive explanation can be formulated using quantitative measures for entropy and information gain.

The empirical distribution over classes in Figure 2.5(a) is uniform since we have exactly the same number of points for each color/class. This also means that the entropy of the training set is rather high. For discrete probability distributions we use the Shannon entropy, defined as

$$H(\mathcal{S}) = -\sum_{c \in \mathcal{C}} p(c) \log(p(c)), \tag{2.6}$$

where $\mathcal{S}$ is the set of training points and the letter c indicates the class label. The set of all classes is denoted $\mathcal{C}$ and $p(c)$ indicates the empirical distribution extracted from the training points within the set S. When applying an horizontal split such as the one in Figure 2.5(b) we see that the empirical distributions of the resulting two sets are no longer uniform. The children distributions are more pure, their entropy has decreased and their information content increased. This improvement can be quantified by measuring the information gain

$$I = H(\mathcal{S}) - \sum_{i \in \{\mathtt{L},\mathtt{R}\}} \frac{|\mathcal{S}^i|}{|\mathcal{S}|} H(\mathcal{S}^i). \tag{2.7}$$

In this example, the split has produced an information gain $I = 0.4$. However, the vertical split shown in Figure 2.5(c) separates the training set even better, that is, the resulting children each contain only two colors. This corresponds to even lower child entropies and a higher information gain ($I = 0.69$). This simple example shows how we can use the information gain as a training objective function. Maximizing the information gain helps select the split parameters which produce the highest *confidence* in the final distributions. This concept is at the basis of decision tree training.

The previous example has focused on discrete, categorical distributions. But entropy and information gain can also be defined for continuous-valued labels and continuous distributions. In fact, the definition of the information gain remains the same but this time, instead of using the Shannon entropy, the differential entropy is used

$$H(S) = -\int_{y \in \mathcal{Y}} p(y) \log(p(y)) dy. \tag{2.8}$$

Here y is a continuous label of interest and p is the probability density function estimated from the training points in the set S. From a practical point of view, in the discrete case, the distribution $p(c)$ was defined as the empirical distribution computed from the training set. Similarly in the continuous distribution $p(y)$ can be defined either using parametric distributions or non-parametric methods.

One of the most popular choice in various applications is to use Gaussian-based models to approximate the density $p(y)$ due to their simplicity. The differential entropy of a d-variate Gaussian is defined analytically as

$$H(\mathcal{S}) = \frac{1}{2} \log((2\pi e)^d |\Lambda(\mathcal{S})|). \tag{2.9}$$

Figure 2.6 illustrates the role of the continuous information gain in training, with another toy example. This time we wish to cluster similar points according to their features (again, depicted as the coordinates of a 2D space). Given an arbitrary input data point we wish the tree to predict its associated cluster. In Figure 2.6(a) we have a set $\mathcal{S}$ of training data points represented in a continuous 2D space. Fitting a Gaussian to the entire initial set $\mathcal{S}$ produces the density shown in blue, which has a high differential entropy. Splitting the data horizontally (Figure 2.6(b)) produces two largely overlapping and slightly smaller 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 (Figure 2.6(c)) yields better separation, with peakier Gaussians and a correspondingly higher value of information gain ($I = 2.43$). The fact that the information gain measure can be defined flexibly, for discrete or continuous distributions is a useful property which is at the basis of our unified forest model.

2.3.5 Leaf Prediction Models

During training, besides the tree structure and the weak learners, we also need to learn how to make predictions.

After training, each leaf node remains associated with a subset of (labeled) training data. During testing, a previously unseen point traverses the tree until it reaches a leaf. Since the split nodes act on features, the input test point is likely to end up in a leaf associated with training points which are all similar to itself. Thus, it is reasonable to assume that the associated label must also be similar to that of the training points in that leaf. This justifies using the label statistics gathered in that leaf to predict the label associated with the input test point.

In the most general sense the leaf statistics can be captured using the posterior distributions

$$p(c|\mathbf{v}) \quad \text{and} \quad p(y|\mathbf{v}), \tag{2.10}$$

where c and y represent the discrete or continuous labels, respectively. $\mathbf{v}$ is the data point that is being tested in the tree and the conditioning denotes the fact that the distributions depend on the specific leaf node reached by the test point (see Figure 2.3(c)). Different leaf predictors can be used. For instance, a Maximum A-Posteriori (MAP) estimate may be obtained as $c^* = \arg\max_c p(c|\mathbf{v})$, in the discrete case, as used in [12]. In general, we prefer to keep the entire distribution around so as to be able to reason about uncertainties.

2.3.6 The Randomness Model

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

- random training set sampling [11] (e.g., bagging), and
- randomized node optimization [48].

These two techniques are not mutually exclusive and could be used together. However, in this survey we focus on the second alternative which: (i) enables us to train trees on the entire training data, and

(ii) yields margin-maximization properties for the ensemble models (details in Section 3). On the other hand, bagging yields greater training efficiency.

Randomized node optimization. In (2.2) we show that at each node the optimization is done with respect to the entire parameter space $\mathcal{T}$. However, this has a major drawback associated with it: efficiency. For large dimensional problems, the size of $\mathcal{T}$ can be extremely large considering that the feature/attribute dimension of each data point can be large. Optimizing over $\mathcal{T}$ is therefore not feasible, nor desirable (for reasons that will become clearer later). Instead, when training at the jth node we only make available a small random subset $\mathcal{T}_j \subset \mathcal{T}$ of parameter values. Thus under the randomness model training a tree is achieved by optimizing each split node j by

$$\boldsymbol{\theta}_j^* = \arg\max_{\boldsymbol{\theta}_j \in \mathcal{T}_j} I_j. \tag{2.11}$$

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 tree and (usually) its value is fixed for all nodes. For $\rho = |\mathcal{T}|$ all the split nodes use all the information available and therefore there is no randomness in the system. Vice-versa, when $\rho = 1$ each split node take only a single randomly chosen set of values for its parameter $\boldsymbol{\theta}_j$. Thus, there is no real optimization and we get maximum randomness.

2.4 Ensembles of Trees (Decision Forest)

A random decision forest is an ensemble of randomly trained decision trees. The key aspect of the forest model is the fact that its component trees are all randomly different from one another. This leads to decorrelation between the individual tree predictions and, in turn, results in improved generalization and robustness. The forest model is characterized by the same components as the decision trees. The family of weak learners (test functions), energy model, the leaf predictors and the type of randomness influence the prediction/estimation properties

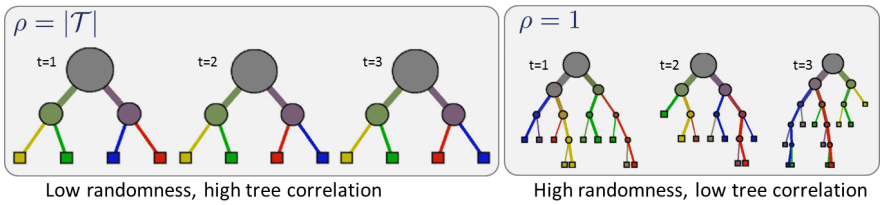


Fig. 2.7 Controlling the amount of randomness and tree correlation. (a) Large values of ρ correspond to little randomness and thus large tree correlation. In this case the forest behaves very much as if it was made of a single tree. (b) Small values of ρ correspond to large randomness in the training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

of the forests. Furthermore, the randomness parameter $\rho = |\mathcal{T}_j|$ controls not only the amount of randomness within each tree but also the amount of correlation between different trees in the forest. In fact, as illustrated in Figure 2.7, when $\rho = |\mathcal{T}|$ all the trees will be identical and as ρ decreases the trees become more decorrelated (different from one another).

In a forest with T trees we use the variable $t \in \{1, \ldots, T\}$ to index each component tree. 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 [89] 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

$$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} p_t(c|\mathbf{v}), \tag{2.12}$$

where $p_t(c|\mathbf{v})$ denotes the posterior distribution obtained by the tth tree. Alternatively one could also multiply the tree outputs together (though the trees are not statistically independent)

$$p(c|\mathbf{v}) = \frac{1}{Z}\prod_{t=1}^{T} p_t(c|\mathbf{v}) \tag{2.13}$$

with the partition function Z ensuring probabilistic normalization.

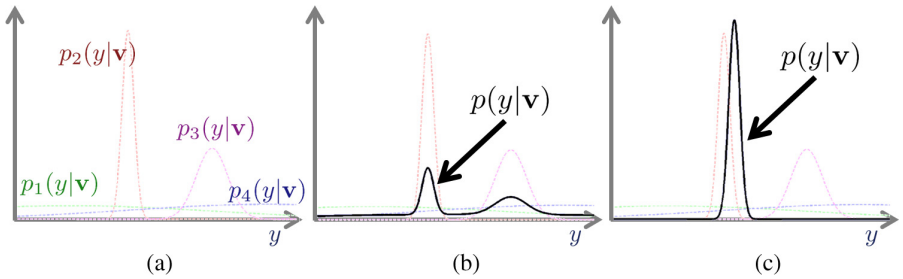


Fig. 2.8 Ensemble model. (a) The posteriors of four different regression trees (shown with different colors). Some correspond to higher confidence than others. (b) An ensemble posterior $p(y|\mathbf{v})$ obtained by averaging all tree posteriors. (c) The ensemble posterior $p(y|\mathbf{v})$ obtained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

Figure 2.8 illustrates tree output fusion for a simple example where the attribute we want to predict is the continuous variable y. Imagine that we have trained a forest with $T = 4$ trees. For a test data point $\mathbf{v}$ we get the corresponding tree posteriors $p_t(y|\mathbf{v})$, with $t = \{1, \ldots, 4\}$. 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 at 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.

2.4.1 Key Model Parameters

The decision forests, their construction and prediction abilities depend on the model parameters. The parameters that most influence the behavior of a decision forest are:

- The maximum allowed tree depth D;
- The amount of randomness (controlled by ρ) and its type;

- The forest size (number of trees) T;
- 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 [25, 92, 110], how learning very deep trees can lead to overfitting, as well as the importance of using very large amounts of training data [91]. In his seminal work Breiman [11] has also shown the importance of randomness and its effect on tree correlation. Additionally, Section 3 will show how the choice of randomness model directly influences a classification forest's generalization.

The choice of stopping criteria has a direct influence on the shape of the trees, e.g., whether they are well balanced or not. In general, very unbalanced trees should be avoided. At the limit they may become just chains of weak learners, with little feature sharing and thus little generalization. A less studied issue is how the weak learners influence the forest's accuracy and its estimated uncertainty. To this end, the next sections will show the effect of ρ on the forest behavior with some simple toy examples and compare the results with existing alternatives. When training a forest it is important to visualize its trees as well as other intermediate variables (e.g., the features and parameters chosen at each node), to make sure the forest has the expected behavior.

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 and helping understand the forests' general properties. Real-world applications will also be discussed briefly to confirm the power of forests in practice.

3

Classification Forests

This section discusses the most common use of decision forests, that is, 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 section also shows that:

- under certain conditions classification forests exhibit margin-maximizing behavior, 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 section for the classification task.

3.1 Classification Algorithms in the Literature

Classification (of pixels or images) is at the heart of modern computer vision and image understanding, as demonstrated by the large interest in the PASCAL Visual Object Class (VOC) recognition challenge [30].

One of the most widely used classification algorithms is the support vector machine (SVM) [106] 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 [34] 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 [107] 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 [21, 103]. In principle, classification trees and forests work, unmodified with any number of classes. For instance, they have been tested on ~ 20 classes in [92] and ~ 30 classes in [91].

Abundant literature has shown the advantage of fusing together multiple simple learners of different types [62, 96, 104, 110, 113]. 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 [4, 19, 24, 60, 83, 92, 66].

3.2 Specializing the Decision Forest Model for Classification

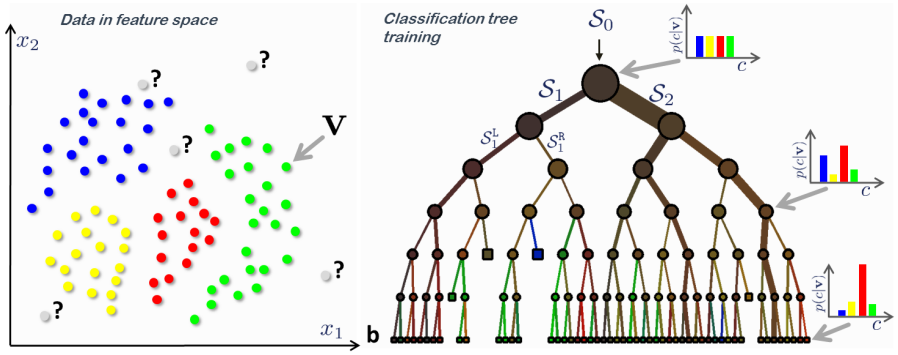
This section specializes the generic model introduced in Section 2 for use in classification.

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

> *Given a labeled 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. Each training point is denoted as a pair $(\mathbf{v}, c)$. In Figure 3.1(a) data points are denoted with circles, with different colors indicating different training labels. Testing points (not available during training) are indicated in gray (their class label is not known in advance).

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 usual the input is represented as a multi-dimensional vector of feature responses



(a) (b)

Fig. 3.1 Classification: training data and tree training. (a) Input data points. The ground-truth label of training points is denoted with different colors. Gray circles indicate unlabeled, previously unseen test data. (b) A binary classification tree. During training a set of labeled 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 toward the leaves.

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

$\mathbf{v} = (x_1, \ldots, x_d) \in \mathbb{R}^d$. 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:

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

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 Figure 3.1(b) 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 toward 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 sections, using an information-gain-like objective function aids unification of diverse tasks under the same forest framework.

Class re-balancing. Note that in some applications one has a very unbalanced distribution of classes in the training set S_0. For instance, when performing semantic image segmentation the number of "background" pixels may dominate all other "object" pixels. This may have a detrimental effect on forest training. This problem may be mitigated simply by resampling the training data so as to have roughly uniform

training distributions. An alternative is to use the known "prior" class distribution to weigh the contribution of each class when computing the information gain at each node.

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 sample $\rho = 1{,}000$ 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 Figure 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.

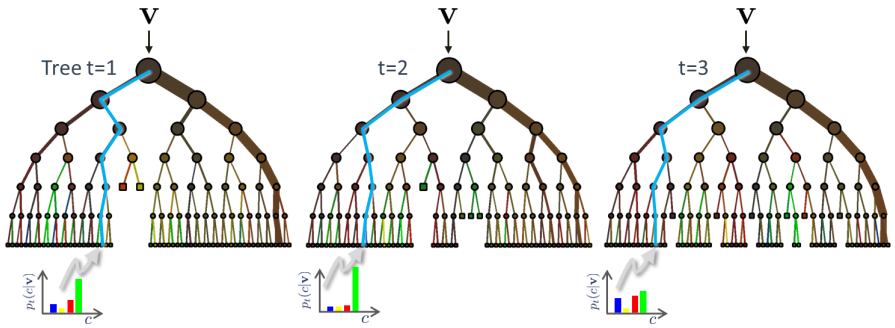


Fig. 3.2 Classification forest testing. During testing the same unlabeled test input data $\mathbf{v}$ is pushed through each component tree. At each internal node a test is applied and the data point sent to the appropriate child. The process is repeated until a leaf is reached. At the leaf the stored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

In later section 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

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 (Figure 3.3(a)). 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 (Figure 3.3(b)). 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 Figure 3.3(a)). In this visualization the color associated with each test point is a linear combination of the colors (red and yellow) corresponding to the two classes; where the mixing weights are proportional to the posterior itself. Thus, intermediate, mixed colors (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 Figure 3.3c_1). This is undesirable. In fact, intuitively one would expect the confidence of classification to be reduced for test data which is "different" than the training data. The

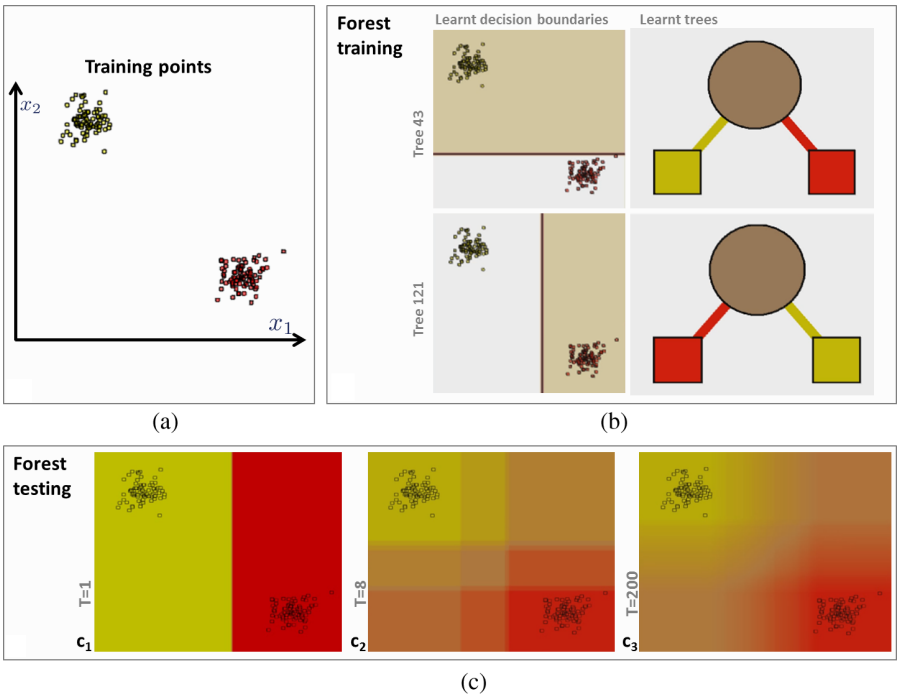


Fig. 3.3 A first classification forest and the effect of forest size T. (a) Training points belonging to two classes. (b) Different training trees produce different partitions and thus different leaf predictors. The color of tree nodes and edges indicates the class probability of training points going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

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 (Figure 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 behavior.

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 survey 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, for example, support vector machines and boosting is that the same classification model can handle both binary and multi-class problems. This is illustrated in Figure 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 Figure 3.4(a) and four classes in Figures 3.4(b) and 3.4(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 Figure 3.4(a′) and grayish ones in Figures 3.4(b′) and 3.4(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 Figure 3.4(b′) 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* Figures 3.4(b) 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 colors in Figure 3.4(c′)). 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 Figure 3.5 illustrates the behavior of classification forests on a four-class training set where there is both mixing of labels

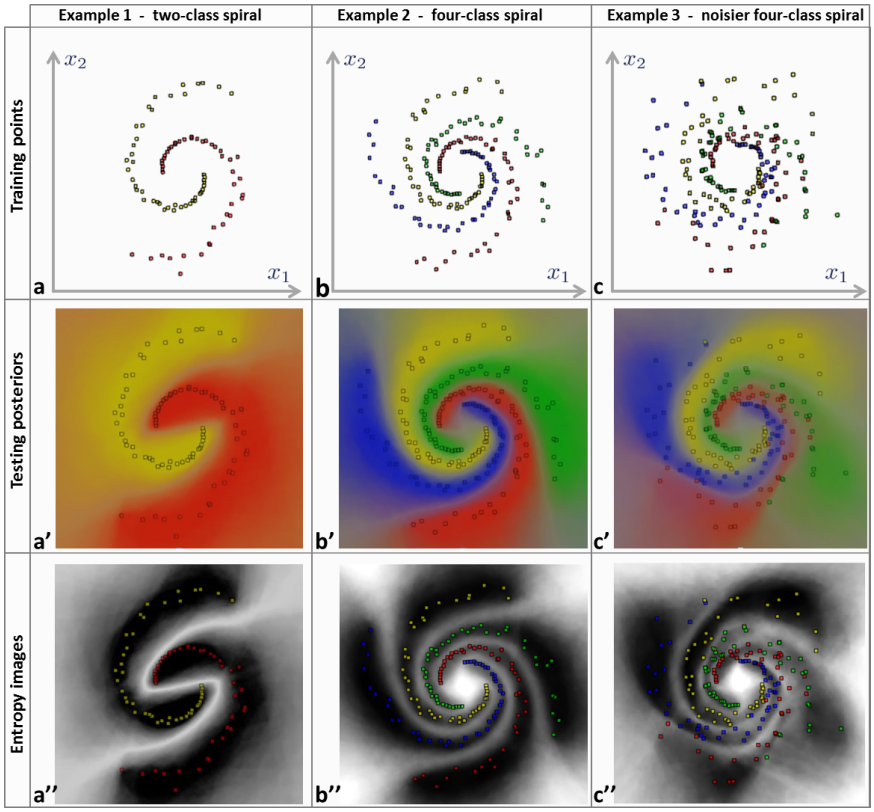


Fig. 3.4 The effect of multiple classes and noise in training data. (a,b,c) Training points for three different experiments: 2-class spiral, 4-class spiral and another 4-class spiral with noisier point positions, respectively. (a′,b′,c′) Corresponding testing posteriors. (a,″b,″c″) Corresponding entropy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

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

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

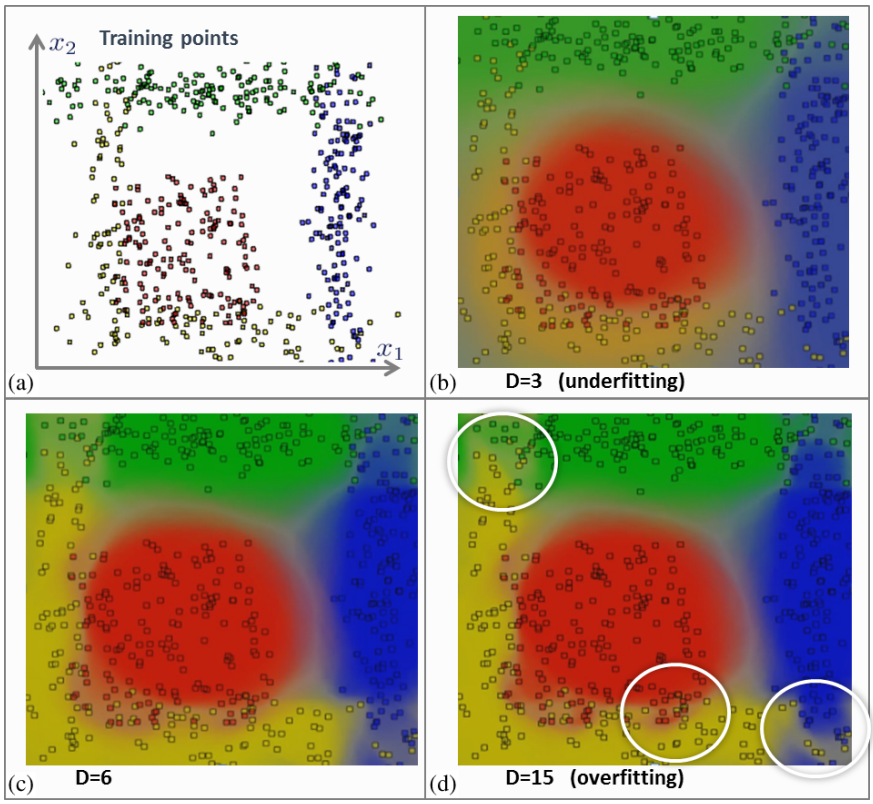


Fig. 3.5 The effect of tree depth. A four-class problem with both mixing of training labels and large gaps. (a) Training points. (b,c,d) Testing posteriors for different tree depths. All experiments were run with $T = 200$ and a conic weak-learner model. The tree depth is a crucial parameter in avoiding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

Finally, we notice that a large value of D ($D = 15$ in the example) tends to produce *overfitting*, that is, 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 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 overlooked in the literature is the effect of a particular choice of weak learner model on the forest behavior.

Figure 3.6 illustrates this point. We are given four sets of well separated point clusters, one cluster per class. We train three forests on those points with different choices of weak learner. The goal is to study the effect of the weak learner on the confidence in regions far from the training data. By comparing Figures 3.6(b), 3.6(c) and 3.6(d) we

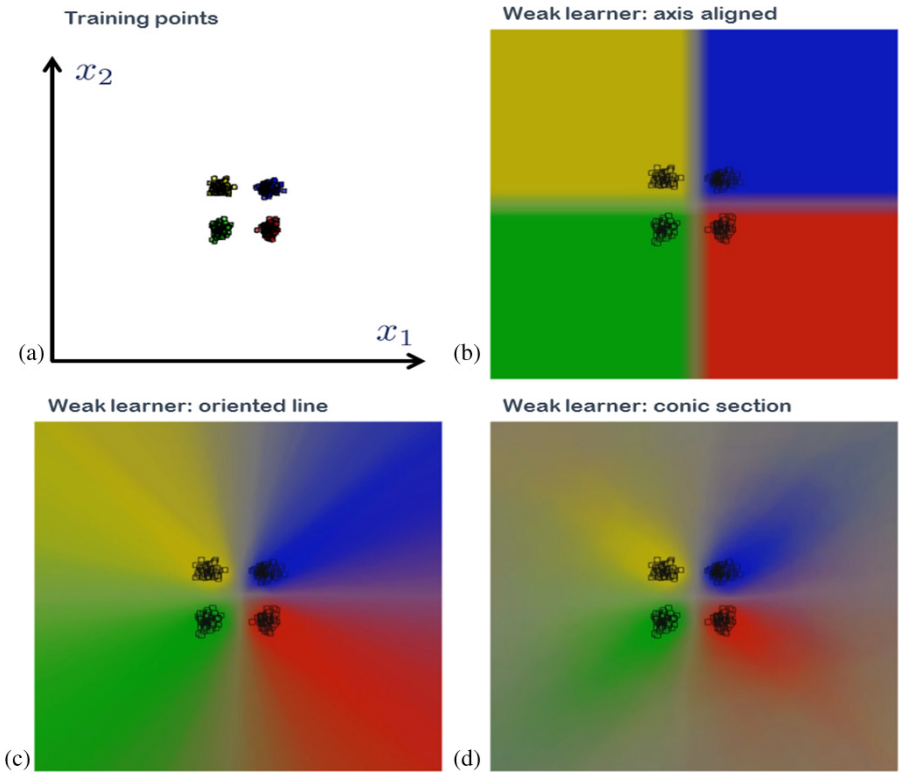


Fig. 3.6 The effect of the weak learner model. (a) A four-class training set. (b) The testing posterior for a forest with axis-aligned weak learners. In regions far from the training points the posterior is overconfident. (c) The testing posterior for a forest with oriented line weak learners. (d) The testing posterior for a forest with conic section weak learners. In (c) and (d) the uncertainty of class prediction increases with distance from the training data points. Here we use $D = 3$ and $T = 200$ for all examples.

observe that an axis-aligned weak-learner produces overconfident predictions in the corners of the space. In this case the confidence value is independent of distance from training points. In contrast, the curved nature of the conic sections yield a more convincing increase of uncertainty with distance from training data.

Another experiment is shown in Figure 3.7 where we are given a set of training points arranged in four spirals, one for each of the four classes. 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

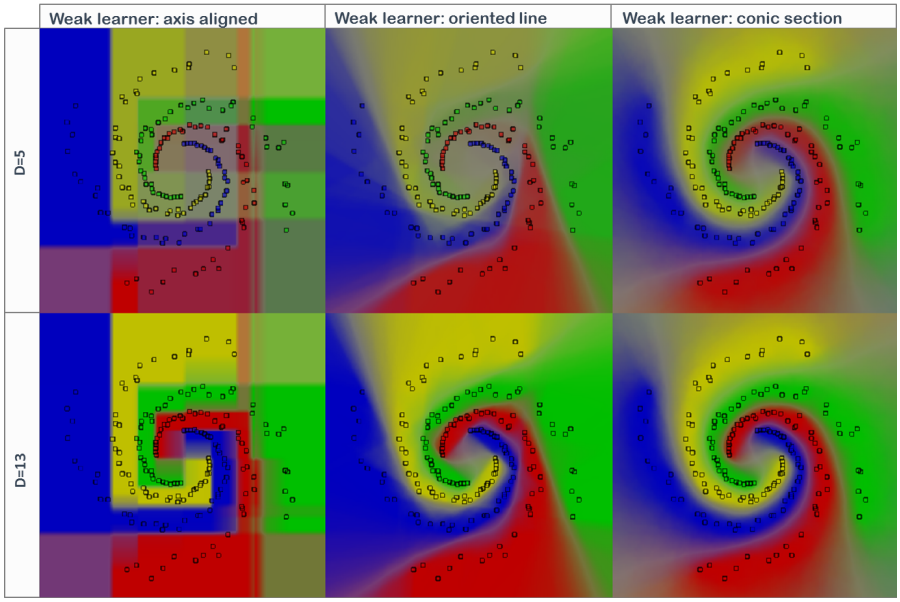


Fig. 3.7 The effect of the weak learner model. The same set of 4-class training data is used to train 6 different forests, for 2 different values of D and 3 different weak learners. For fixed weak learner deeper trees produce larger confidence. For constant D nonlinear weak learners produce the best 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 $T = 400$.

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 colors 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 [45, 65] is a clear improvement, and better still is the nonlinear 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.

3.3.5 The Effect of Randomness

Figure 3.8 shows the same experiment as in Figure 3.7 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 Figure 3.8). Furthermore, larger randomness yields a much lower overall confidence, especially noticeable in shallower trees (washed out colors in the top row).

A disadvantage of the more complex weak learners is that they are associated with 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).

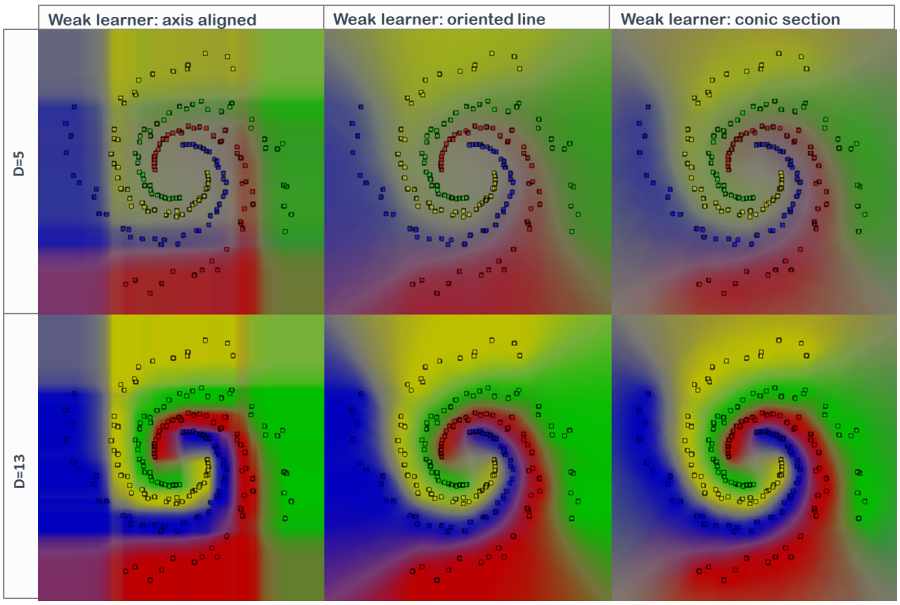


Fig. 3.8 The effect of randomness. The same set of 4-class training data is used to train 6 different forests, for 2 different values of D and 3 different weak learners. This experiment is identical to that in Figure 3.7 except that we have used much more training randomness. In fact $\rho = 5$ for all 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.

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 [49].

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 [58]. 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 Figure 3.9(a). 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 2.3.6).

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

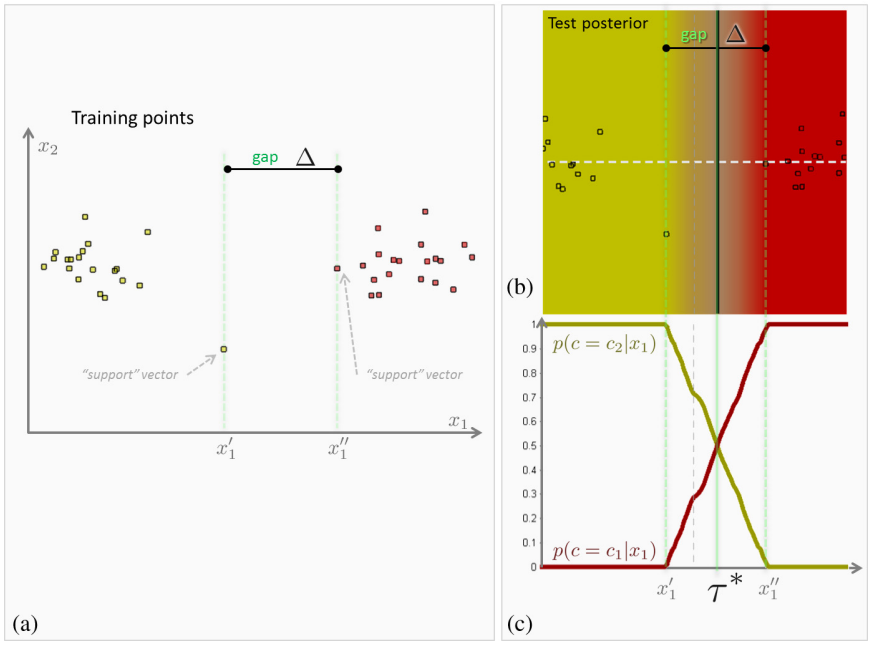


Fig. 3.9 Forest's maximum-margin properties. (a) Input 2-class training points. They are separated by a gap of dimension Δ. (b) Forest posterior. Note that all of the uncertainty band resides within the gap. (c) Cross-sections of class posteriors along the horizontal, white dashed line in (b). Within 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.

separating line tends to be placed somewhere within the gap (see Figure 3.9(a)) 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 Figure 3.3(b)).

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 Figures 3.9(b) and 3.9(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 middle of the gap, that is, the maximum-margin solution. Next, we describe the same concepts more formally.

We are given the two-class training points in Figure 3.9(a). 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, that is,

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

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] that is, 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, that is,

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

[3] Analogous to support vectors in SVM.

(see Figures 3.9(b) and 3.9(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 behavior. 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.

3.4.1 The Effect of Randomness on Optimal Separation

The experiment in Figure 3.9 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 Figure 3.10 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.

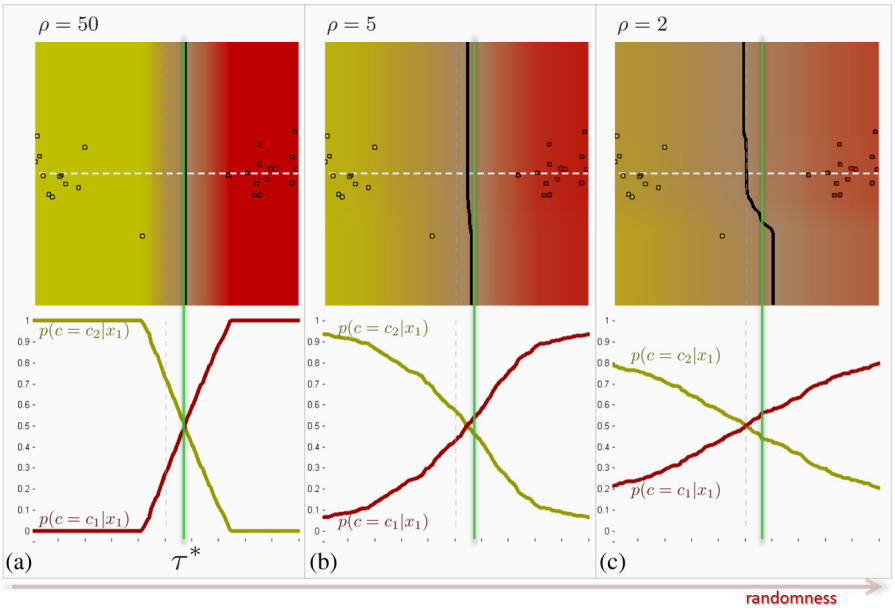


Fig. 3.10 The effect of randomness on the forest margin. (a) Forest posterior for $\rho = 50$ (small randomness). (b) Forest posterior for $\rho = 5$. (c) Forest posterior for $\rho = 2$ (highest randomness). These experiments have used $D = 2, T = 400$ and axis-aligned weak learners. The *bottom row* shows 1D posteriors computed along the white dashed line. Increasing randomness produces less well defined separating surfaces. The optimal separating surface, that is, the loci of points where the class posteriors are equal (shown in *black*) moves toward 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.

The role of the parameter ρ is very similar to that of "slack" variables in SVM [106]. 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 3.11 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

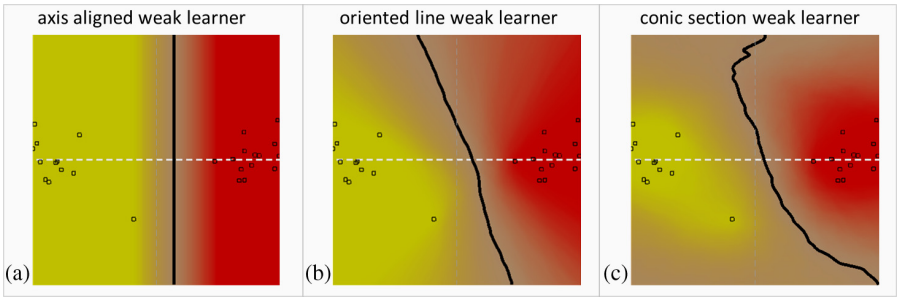


Fig. 3.11 The effect of the weak learner on forest margin. (a) Forest posterior for axis aligned weak learners. (b) Forest posterior for oriented line weak learners. (c) Forest posterior for conic section weak learners. In these experiments we have used $\rho = 50, D = 2, T = 500$. The choice of weak learner affects the optimal, hard separating curve (in black). Individual training points influence the surface differently depending on the amount of randomness in the forest.

training points depending on the value of ρ. Little randomness produces a behavior closer to that of support vector machines.

In classification forests, using linear weak-learners still produces (in general) globally nonlinear classification (see the black curves in Figures 3.10(c) and 3.11(b)). This is due to the fact that multiple simple linear split nodes are organized in a hierarchical fashion.

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 Figure 3.12a we have a linearly separable four-class training set. On it we have trained two forests with $\rho = 50, D = 3, T = 400$. 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 3.12(b) and 3.12(c) show the corresponding testing posteriors. As usual gray 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 neighboring classes. As in the 2-class case the influence of individual training points is dictated by the randomness parameter ρ.

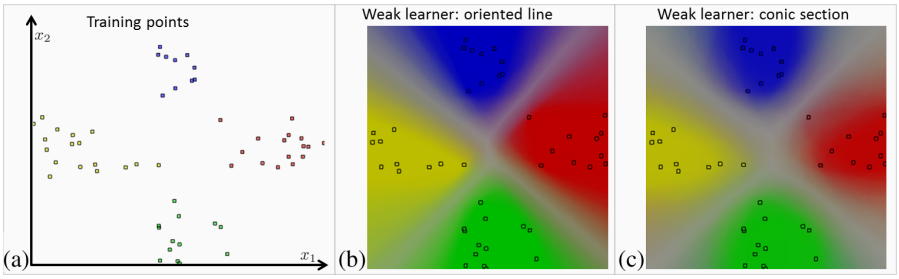


Fig. 3.12 Forest's max-margin properties for multiple classes. (a) Input four-class training points. (b) Forest posterior for oriented line weak learners. (c) Forest posterior for conic section weak learners. Regions of high entropy are shown as gray bands and correspond to loci of optimal separation. In these experiments we have used the following parameter settings $\rho = 50, D = 3, T = 400$.

Finally, when comparing Figures 3.12(b) and 3.12(c) 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 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 Figure 3.13 where we have trained two forests with $\rho = 500, D = 2, T = 400$ and two different randomness models. The forest tested in Figure 3.13(a) uses randomized node optimization (RNO). The one in Figure 3.13(b) 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

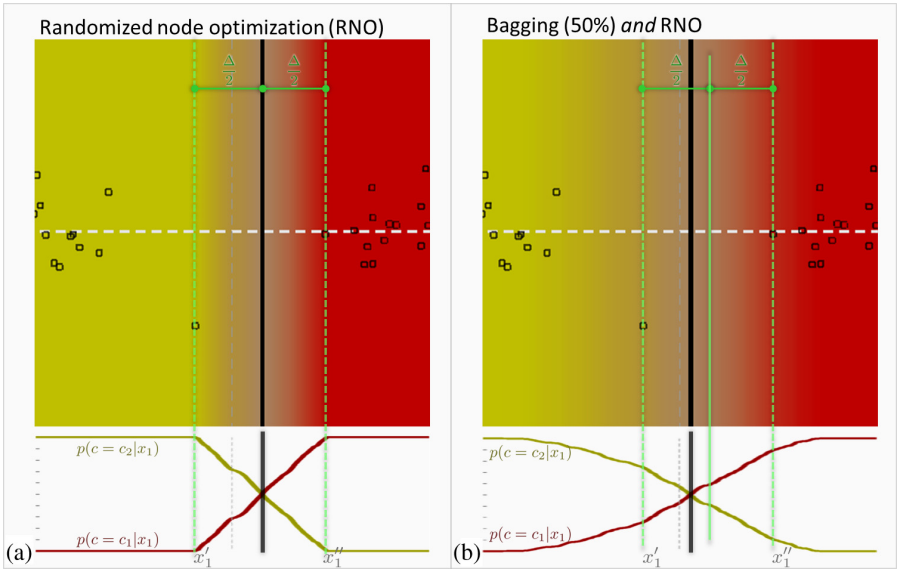


Fig. 3.13 Max-margin: bagging versus randomized node optimization. (a) Posterior for forest trained with randomized node optimization. (b) Posterior for forest trained with bagging. In bagging, for each tree we use 50% random selection of training data with replacement. Loci of optimal separation are shown as black lines. In these experiments we use $\rho = 500$, $D = 2$, $T = 400$ and axis-aligned weak learners. Areas of high entropy are shown in gray to highlight the separating surfaces.

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 Figure 3.13(b). In both cases we have used a large value of ρ to make sure that each tree achieves decent optimality in parameter selection. 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 behavior 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 behavior (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 [49]. In the rest of the survey we use the RNO randomness model because it allows us to use all available

training data and enables us to control the maximum-margin behavior 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 3.14 shows a comparison between classification forests and ModestBoost on two synthetic experiments.[4] Here, for both algorithm

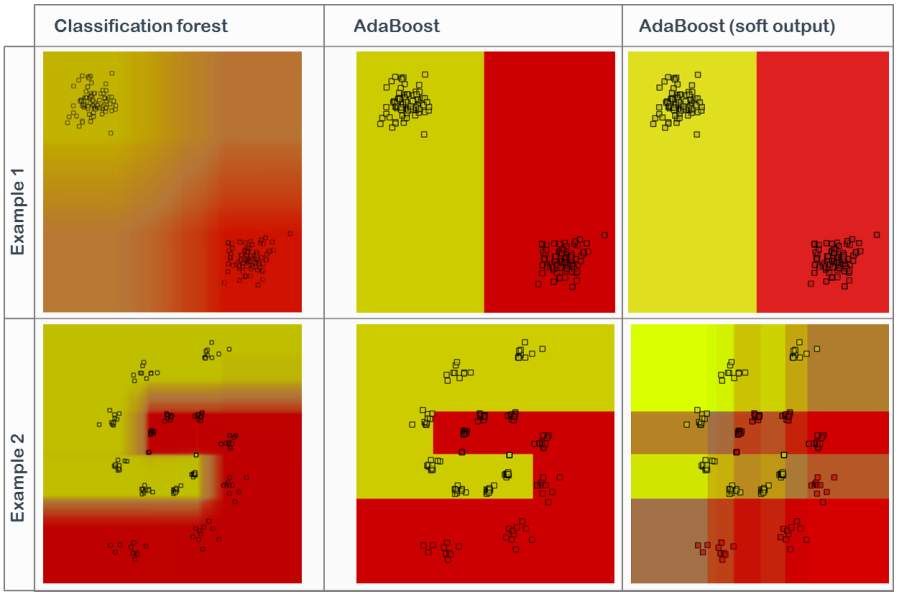


Fig. 3.14 Comparison between classification forests and boosting on two examples. Forests produce a smooth, probabilistic output. High uncertainty is associated with regions between different classes or away from training data. Capitalized produces a hard output. Interpreting the output of a boosted strong classifier as real valued does not seem to produce 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.

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

we use shallow tree stumps ($D = 2$) with axis-aligned split functions as this is what is conventionally used in boosting [108].

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 probabilistic output. 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 variants may produce better behavior.

3.5.2 Comparison with Support Vector Machines

Figure 3.15 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 and both forests and SVMs achieve good separation results. However, forests also produce uncertainty information. Probabilistic SVM counterparts such as the relevance vector machine [102] do produce confidence output but at the expense of further computation.

The role of good confidence estimation is particularly evident in Figure 3.15(b) where we can see how the uncertainty increases as we move away from the training data. The exact shape of the confidence

[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.

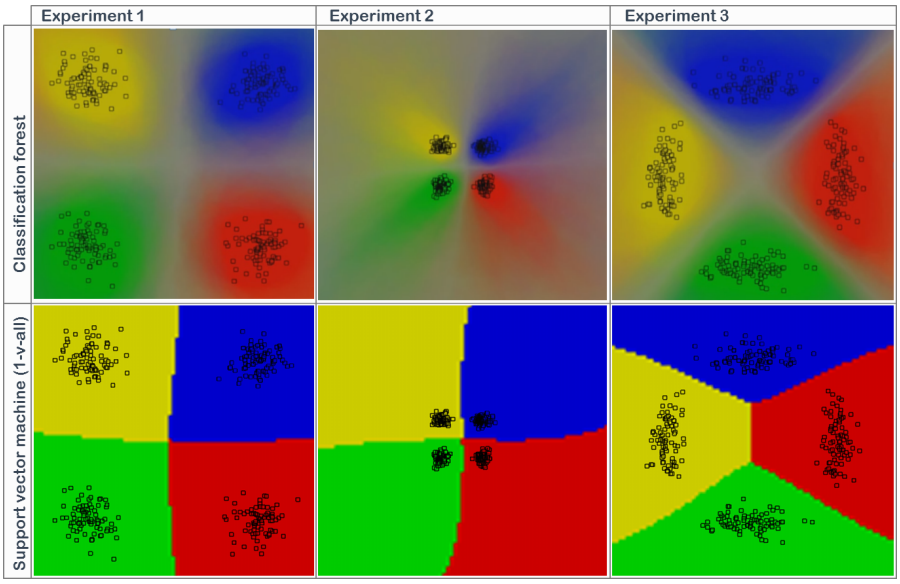


Fig. 3.15 Comparison between classification forests and support vector machines. All forest experiments were run with $D = 3$, $T = 200$ and conic weak learner. The SVM parameters were optimized to achieve best results.

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 recently they have been adapted to work with multiple classes. Figure 3.15(c) 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 [66]. Here we present a summary of the algorithm in [91]

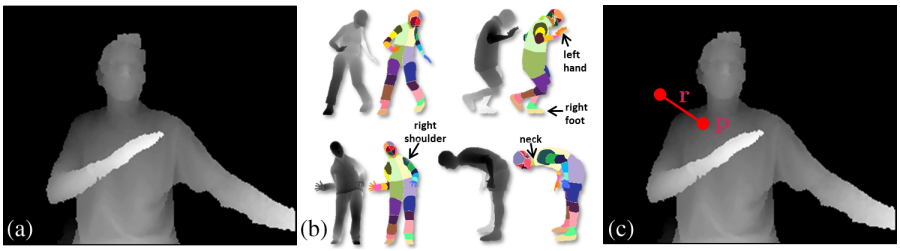


Fig. 3.16 Classification forests in Microsoft Kinect for XBox 360. (a) An input frame as acquired by the Kinect depth camera. (b) Synthetically generated ground-truth labeling of 31 different body parts [91]. (c) One of the many features of a "reference" point $\mathbf{p}$. Given $\mathbf{p}$ computing the feature amounts to looking up the depth at a "probe" position $\mathbf{p}+\mathbf{r}$ and comparing it with the depth of $\mathbf{p}$.

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 Figure 3.16(a) 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 31 different body part classes: $c \in \{\texttt{left hand}, \texttt{right hand}, \texttt{head}, \texttt{l. shoulder}, \texttt{r. shoulder}, \ldots\}$. 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 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 $J(\cdot)$ 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 Figure 3.16(c)). 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 labeled training image pairs as in Figure 3.16(b). Training happens by maximizing the information gain for discrete distributions (3.1). For a split node j its parameters are

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

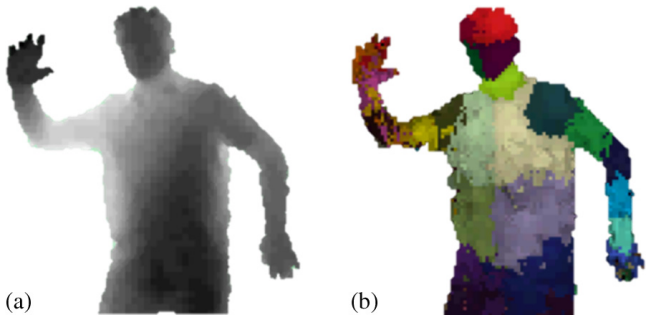


Fig. 3.17 Classification forests in Kinect for XBox 360. (a) An input depth frame with background removed. (b) The body part classification posterior. Different colors corresponding to different body parts, out of 31 different classes.

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 Figure 3.17; with many more shown in the original paper [91]. 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, for example, when changing the weak learner model, or the amount of randomness etc. However, this investigation is beyond the scope of this review.

Moving on from classification, the next section addresses a closely related problem, that of probabilistic, nonlinear regression. Interestingly, regression forests have very recently been used also for skeletal joint prediction in Kinect images [39].

4

Regression Forests

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

Regression forests are used for the nonlinear 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 sections we start with a brief literature survey of linear and nonlinear 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 [52]) 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 nonlinear behavior [88]. 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 [32, 43]. For instance the estimation of multi-view epipolar geometry and image registration transformations can be achieved in this way [43]. 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 [57, 95]). Similar to RANSAC, SVR can deal successfully with large amounts of noise. In Bayesian machine learning Gaussian processes [5, 82] 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 [25, 31, 39, 53, 67]. Next, we discuss how to specialize the generic forest model described in Section 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:

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

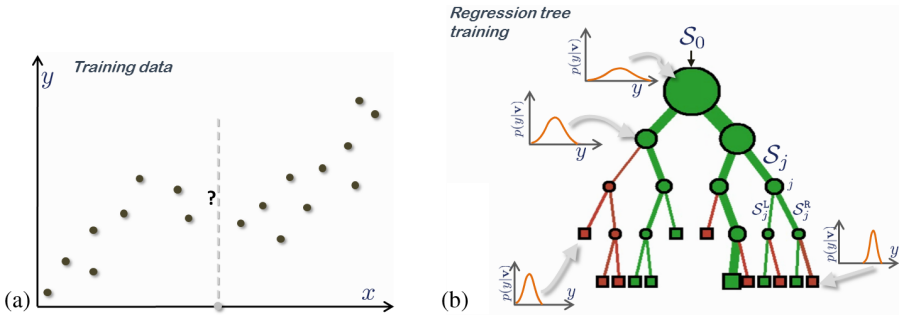


Fig. 4.1 Regression: training data and tree training. (a) Training data points are shown as dark circles. The associated ground truth label is denoted by their position along the y coordinate. The input feature space is one-dimensional in this example ($\mathbf{v} = (x)$). x is the independent input and y is the dependent variable. A previously unseen test input is indicated with a light gray circle on the x axis. (b) A binary regression tree. During training a set of labeled training points $\mathcal{S}_0$ 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 toward the leaves.

Like classification the regression task is inductive, with the main difference being the continuous nature of the output. Figure 4.1(a) provides an illustrative example of training data and associated continuous ground-truth labels. In general, a training point is denoted as a labelled pair $(\mathbf{v}, \mathbf{y})$. A previously unseen test input (unavailable during training) is shown as a light gray 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 input is represented as a multi-dimensional feature response vector $\mathbf{v} = (x_1, \ldots, x_d) \in \mathbb{R}^d$.

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

A regression tree (Figure 4.1(b)) 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 Figure 4.2). Next we specify the precise nature of each model component.

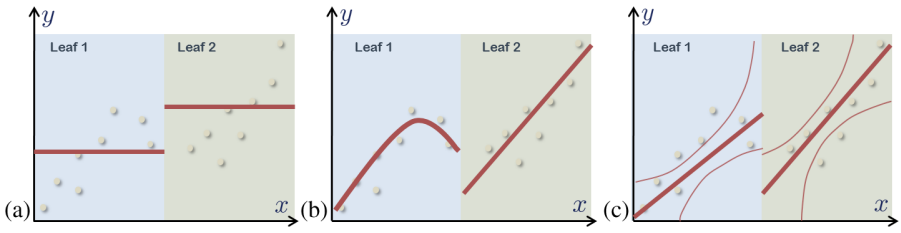


Fig. 4.2 Example predictor models. Different possible predictor models. (a) Constant. (b) Polynomial and linear. (c) Probabilistic-linear. The conditional distribution $p(y|x)$ is returned in the latter.

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 Figure 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 captures both the linear and constant models (see Figures 4.2(a) and 4.2(b)).

In this survey we are interested in output confidence as well as its actual value. Thus for prediction we can use a probability density function over the continuous variable $\mathbf{y}$. So, given the tth 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 Figure 4.2(c) 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 (Figure 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 2.4.

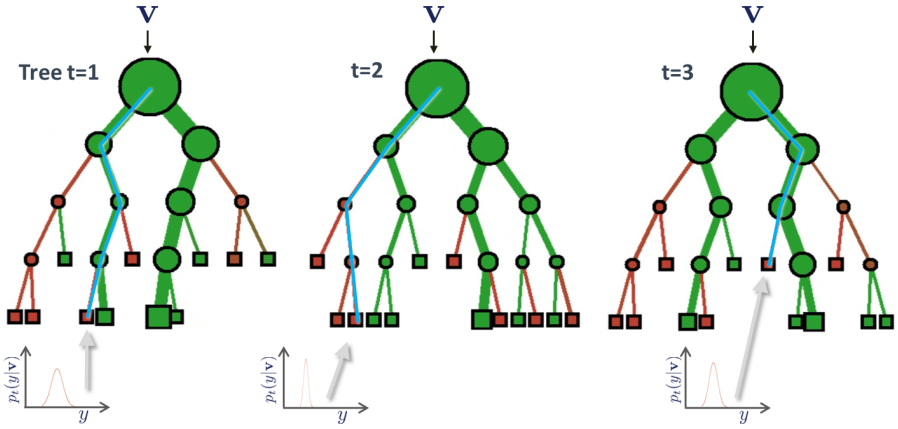


Fig. 4.3 Regression forest: the ensemble model. The regression forest posterior is simply the average of all individual tree posteriors $p(\mathbf{y}|\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} p_t(\mathbf{y}|\mathbf{v})$.

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 jth 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:

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

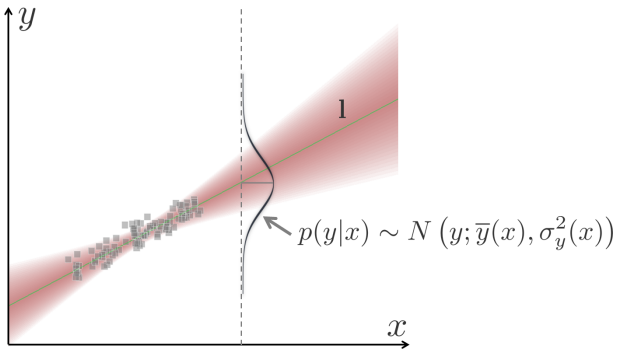


Fig. 4.4 Probabilistic line fitting. Given a set of training points we can fit a line $\mathbf{l}$ to them, for example, by least squares or RANSAC. In this example $\mathbf{l} \in \mathbb{R}^2$. Matrix perturbation theory (see Appendix A) enables us to estimate a probabilistic model of $\mathbf{l}$ from where we can derive $p(y|x)$ (modeled 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.

with $\Lambda_{\mathbf{y}}$ the conditional covariance matrix computed from probabilistic linear fitting (see also Figure 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 (Figure 4.2).

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

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

with $\overline{y}_j$ indicating the mean value of y for all training points reaching the jth 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 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 Figure 4.1b)

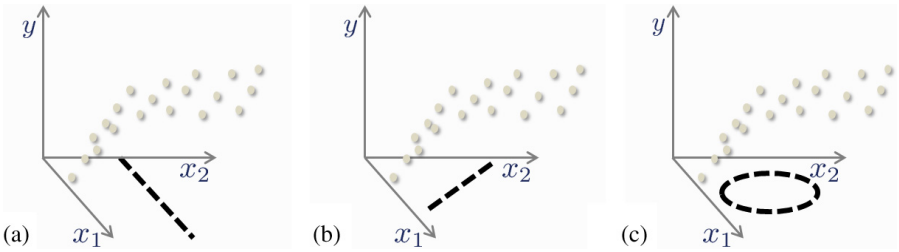


Fig. 4.5 Example weak learners. The (x_1, x_2) plane represents the d-dimensional input domain (independent). The y space represents the n-dimensional continuous output (dependent). The example types of weak learner are like in classification (a) Axis-aligned hyperplane. (b) General oriented hyperplane.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

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 Figure 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 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 behavior.

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 Figure 4.6(a). 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.

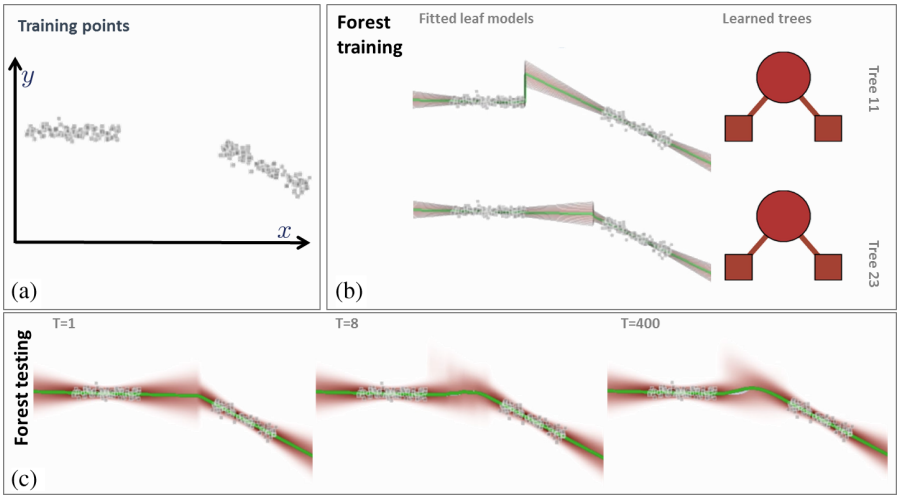


Fig. 4.6 A first regression forest example and the effect of its size T. (a) Training points. (b) Two different shallow trained trees ($D = 2$) split the data into two portions and produce different piece-wise probabilistic-linear predictions. (c) Testing posteriors evaluated for all values of x and 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.

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 (Figure 4.6(b)) are all slightly different from each other as they produce different leaf models (Figure 4.6(b)). During training, as expected each leaf model produces smaller uncertainty near 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 (Figure 4.6(c)) 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 Figure 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

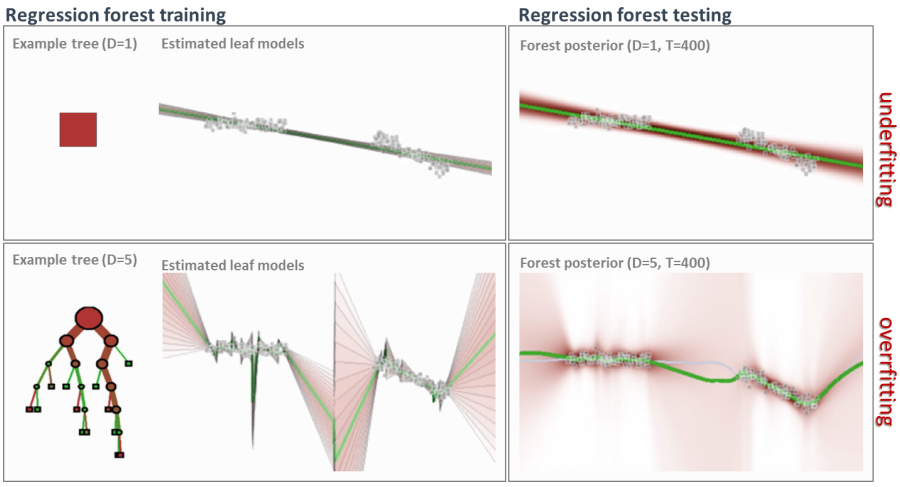


Fig. 4.7 The effect of tree depth. (*Top row*) Regression forest trained with $D = 1$. Trees are degenerate (each tree corresponds only to their root node). This corresponds to conventional linear regression. In this case the data is more complex than a single linear model and thus this forest under-fits. (*Bottom row*) Regression forest trained with $D = 5$. Much deeper trees produce the opposite effect, that is, 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.

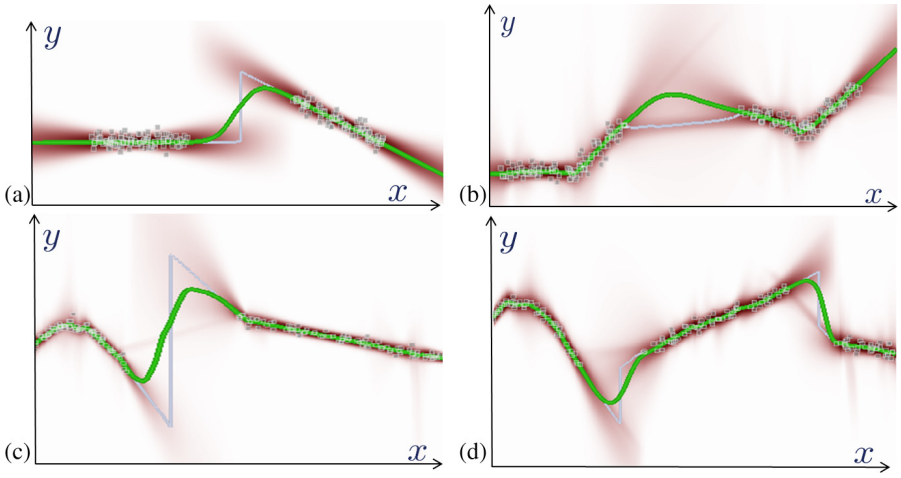


Fig. 4.8 Spatial smoothness, multi-modal posteriors and testing uncertainty. Four more regression experiments. The squares indicate labeled training data. The green curve is the estimated conditional mean $\overline{y}(x) = E[y|x] = \int y \cdot p(y|x)\, dy$ and the gray curve the estimated mode $\hat{y}(x)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

figure) yields over-fitting. 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 gray. These experiments highlight the smooth interpolating behavior 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 [49].

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

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 [82].

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 Gaussian lower case for consistency 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

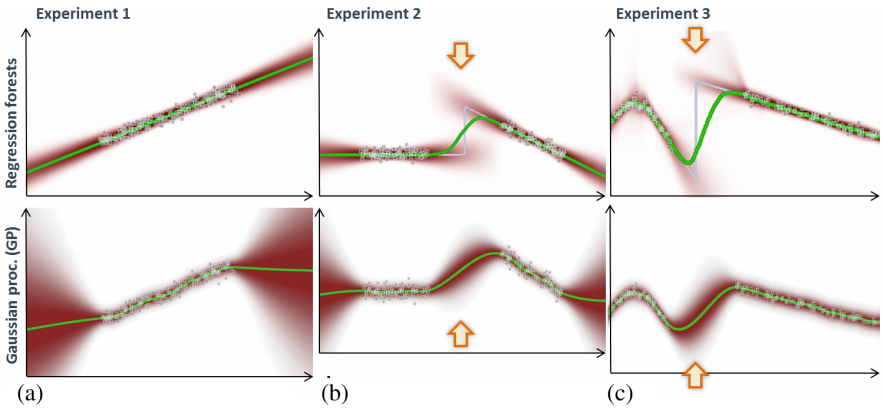


Fig. 4.9 Comparing regression forests with Gaussian processes. (a,b,c) Three training datasets and the corresponding testing posteriors overlaid on top. In both the forest and the GP model uncertainties increase as we move away from training data. However, the actual shape of the posterior is different. (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.

[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`.

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).

Gaussian processes are well known for how they model increasing uncertainty with increasing distance from training points. The bottom row illustrates this point 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 behavior 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 (probabilistic-linear here). One striking difference between the forest and the GP model though is illustrated in Figures 4.9(b) and 4.9(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 behavior 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 [85].

This difference between the two models in the presence of ambiguities is tested further in Figure 4.10. Here the training data itself is arranged in an ambiguous way, as a "non-function" relation (see also [71] 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 Figures 4.10(b) and 4.10(c), respectively. In this case neither technique 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

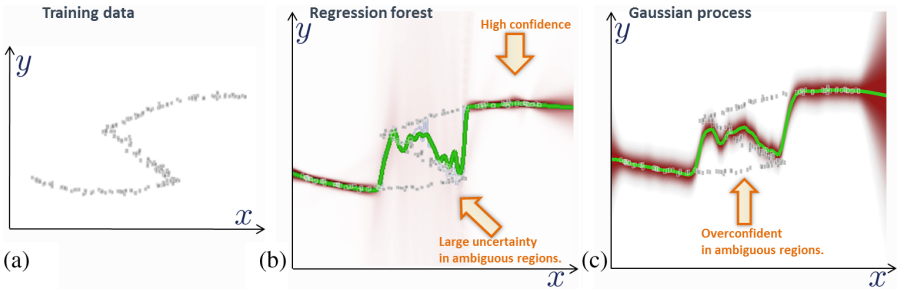


Fig. 4.10 Comparing forests and GP on ambiguous training data. (a) Input labeled training points. The data is ambiguous because a given input x may correspond to multiple values of y. (b) The posterior $p(y|x)$ computed via random regression forest. The middle (ambiguous) region remains associated with high uncertainty (in pink). (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.

think of improving the forest output, for example, by using a mixture of probabilistic-linear predictors at each leaf (as opposed to a single line). Later sections 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 Figure 4.11). This is useful for example, (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 [55]. Details

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

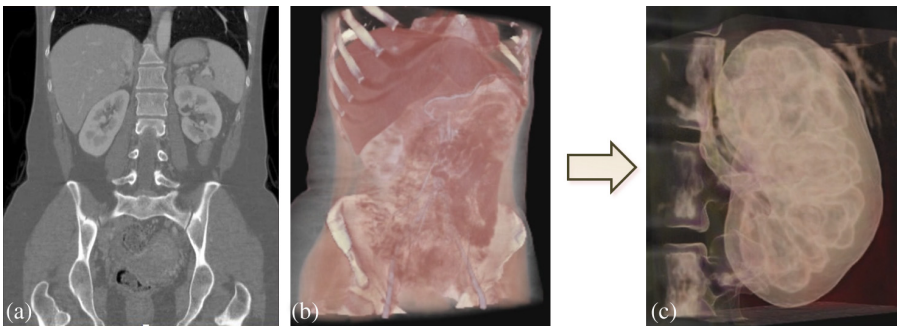


Fig. 4.11 Automatic localization of anatomy in 3D Computed Tomography images. (a) A coronal slice (*frontal view*) from a test 3D CT patient's scan. (b) Volumetric rendering of the scan to aid interpretation. (c) Automatically localized left kidney using regression forest. Simultaneous localization of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

of the algorithm can be found in [25]. 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: $\mathbf{b} = (b^{\mathtt{L}}, b^{\mathtt{R}}, b^{\mathtt{H}}, b^{\mathtt{F}}, b^{\mathtt{A}}, b^{\mathtt{P}}) \in \mathbb{R}^6$ (see Figure 4.12(a)). For simplicity here we 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 [35] 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 for example, the left kidney should be. The votes take the form of relative displacement vectors

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

(see Figure 4.12(b)). 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

[4] A more general parametrization is given in [25].

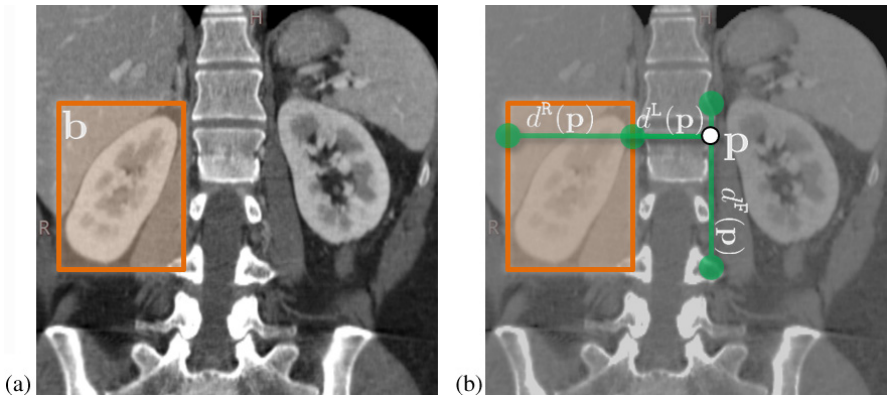


Fig. 4.12 Automatic localization of anatomy in 3D CT images. (a) A coronal view of the abdomen of a patient in a CT scan. The bounding box of the right kidney is shown in orange. (b) Each voxel $\mathbf{p}$ in the volume votes for the position of the six walls of the box via the relative displacements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

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.

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

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

where $J(\mathbf{p})$ denotes the density of the tissue in an element of volume at position $\mathbf{p}$ as measured by the CT scanner (in calibrated Hounsfield Units). The 3D feature box $\mathbf{B}$ (not to be confused with the output organ bounding box) is displaced from the reference point $\mathbf{p}$ (see Figure 4.13(a)). Since for each reference pixel $\mathbf{p}$ 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 labeled 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

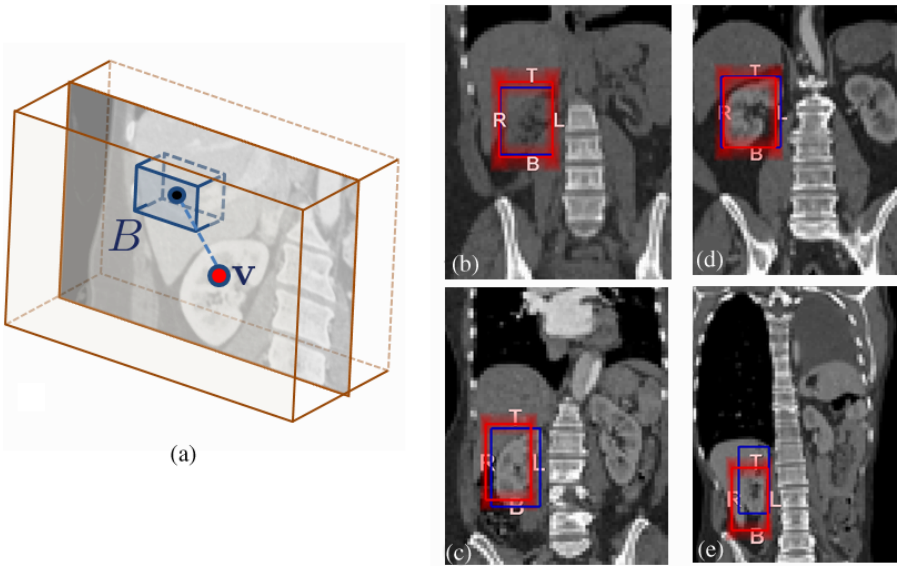


Fig. 4.13 Features and results. (a) Feature responses are defined via integral images in displaced 3D boxes, denoted with **B**. (b,c,d,e) Some results on four different test patients. The right kidney (red box) is correctly localized in all scans. The corresponding ground-truth is shown with a blue 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:

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

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, that is, 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 [25].

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

An important advantage of decision forests (compared to for example, 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' 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

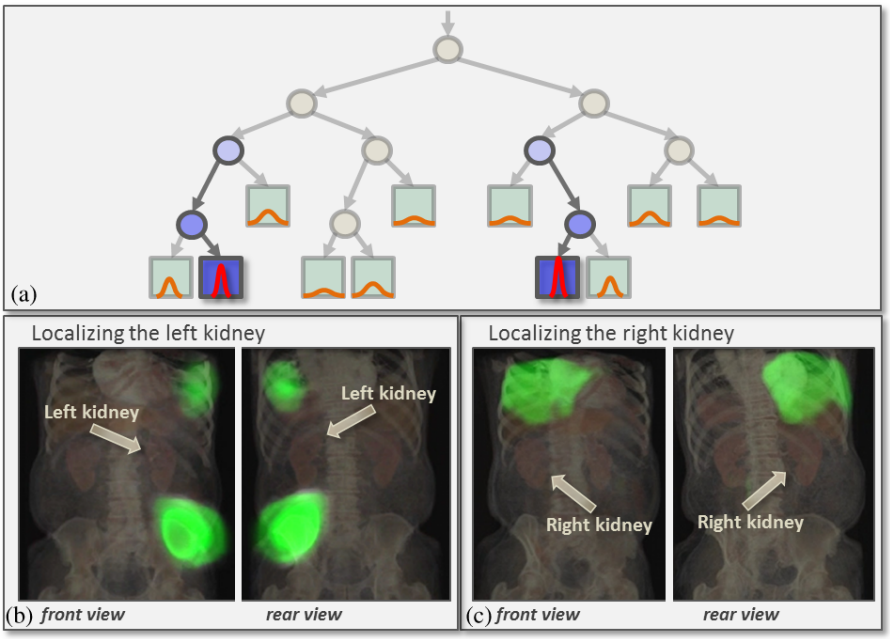


Fig. 4.14 Automatic discovery of salient anatomical landmarks. (a) Leaves associated with the most peaked densities correspond to clusters of points which predict organ locations with high confidence. (b) A 3D rendering of a CT scan and (in green) landmarks automatically selected as salient predictors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

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 [77]. See also [40, 85] 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 [50] for further examples and applications of regression forests to medical image analysis.

5

Density Forests

Sections 3 and 4 have discussed the use of decision forests in supervised tasks, that is, when *labeled* training data is available. In contrast, this section discusses the use of forests in unlabeled scenarios.

For instance, one important task is that of discovering the intrinsic nature and structure of large sets of unlabeled 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 Section 2). Given some observed unlabeled 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 [61]. A very successful probabilistic density model is the Gaussian mixture model (GMM), where complex distributions can be approximated via a linear combination of simple (multivariate) Gaussian components [5, 56]. 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 [76]. The advantage of kernel-based estimation over, for example, 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 neighbor density estimation algorithm [5].

In Breiman's 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 unsupervised information gain-based optimization, which fits well within our unified forest model. Forest-based data clustering has been discussed in [69, 92] for computer vision applications.

For further reading on general density estimation techniques the reader is invited to explore the following material [5, 93].

5.2 Specializing the Forest Model for Density Estimation

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

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

> *Given a set of unlabeled observations we wish to estimate the latent 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 $\mathbf{v} = (x_1, \ldots, x_d) \in \mathbb{R}^d$. 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 Figure 5.1(a). Unlabeled 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 (Figure 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

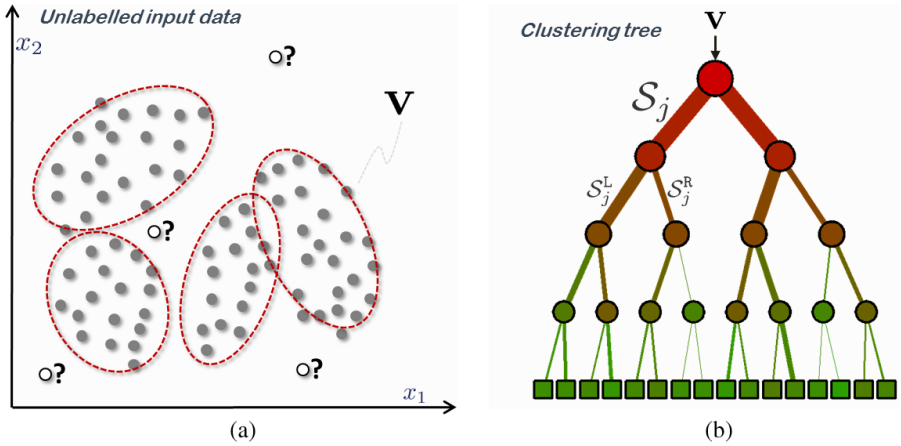


Fig. 5.1 Input data and density forest training. (a) Unlabeled data points using for training a density forest are shown as dark circles. White circles indicate previously unseen test data. (b) Density forests are ensembles of clustering trees.

partitions are created, one by each tree. This is in contrast to the single "soft" clustering generated by the EM algorithm. Furthermore, (ii) the forest-based probability is a combination of tree-based probabilities. 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 points $\mathcal{S}_0 = \{\mathbf{v}\}$ (note the absence of training labels here) we train each individual tree in the forest independently and if possible in parallel. As usual we employ randomized node optimization. Thus, optimizing the jth 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 energy 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, that is, one which applies to unlabeled 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

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

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

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

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 centers 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 tth tree in a forest defines a partition of the data such that

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

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 tth 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 Figure 5.2 for an illustration).

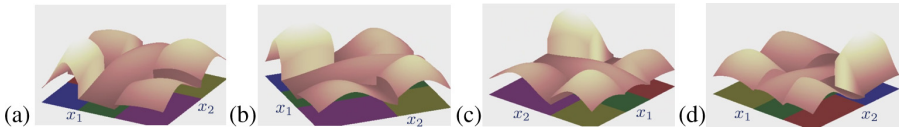


Fig. 5.2 A tree density is piece-wise Gaussian. (a,b,c,d) Different views of a tree density $p_t(\mathbf{v})$ defined over an illustrative 2D feature space. Each individual Gaussian component is defined over a bounded domain. See text for details.

Partition function. Note that in (5.3) each Gaussian is truncated by the boundaries of the partition cell associated with the corresponding leaf (see Figure 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 Figure 5.2). Unfortunately, the cumulative multivariate normal does not have a close form solution. However, approximating its functional form is a well researched problem and a number of good numerical approximations exist [41, 80].

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

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

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 [72, 94]. 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 Figure 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

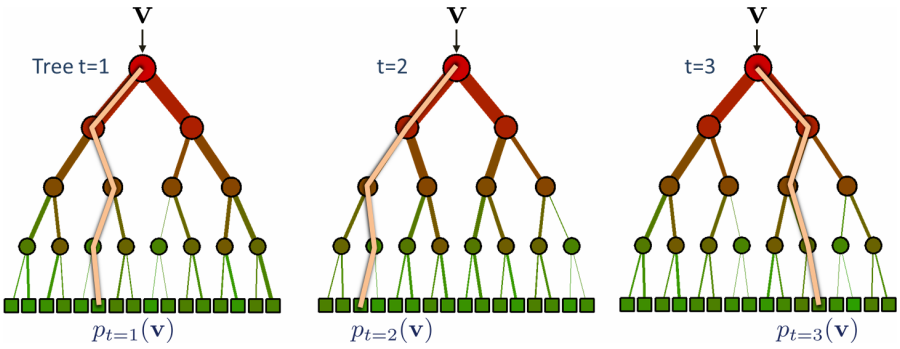


Fig. 5.3 Density forest: the ensemble model. A density forest is a collection of clustering trees trained on unlabeled data. The tree density is the Gaussian associated with the leaf reached by the input test point: $p_t(\mathbf{v}) = \frac{\pi_{l(\mathbf{v})}}{Z_t}\mathcal{N}(\mathbf{v}; \boldsymbol{\mu}_{l(\mathbf{v})},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

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 5.4(a) shows some unlabeled 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. In this simple example $D = 2$ (top row) produces the best results.

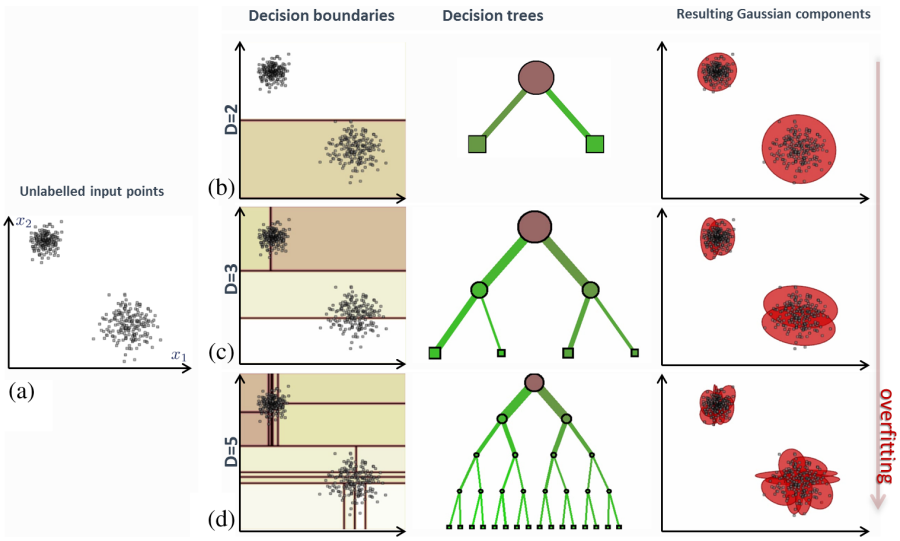


Fig. 5.4 The effect of tree depth on density. (a) Input unlabeled data points in a 2D feature space. (b,c,d) Individual trees out of three density forests trained on the same dataset, for different tree depths D. A forest with unnecessarily deep trees tends to fit to the training noise, thus producing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

5.3.2 The Effect of Forest Size

Figure 5.5 shows the output of six density forests trained on the input data in Figure 5.4(a) 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 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.

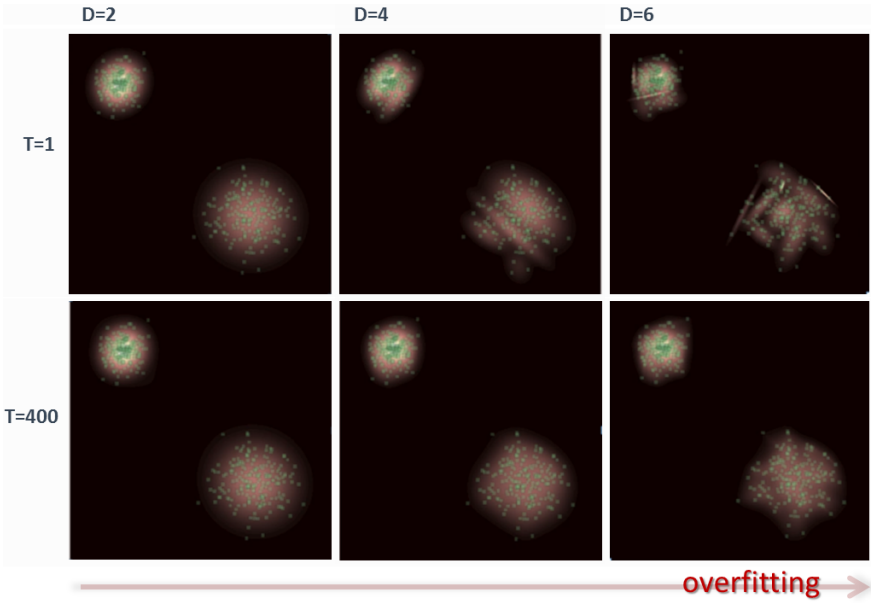


Fig. 5.5 The effect of forest size on density. Densities $p(\mathbf{v})$ for six density forests trained on the same unlabeled dataset for varying T and D. Increasing the forest size T *always* improves the smoothness of the density and the forest generalization, even for deep trees.

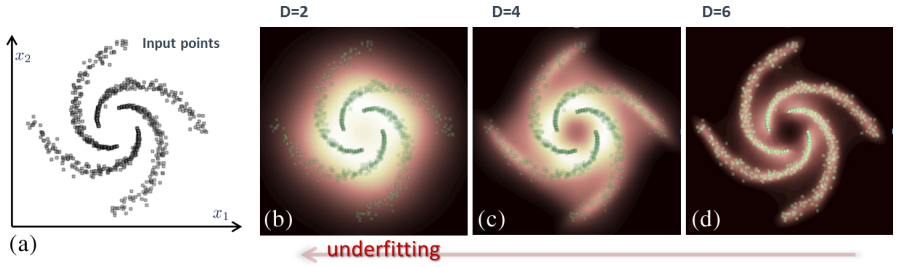


Fig. 5.6 Density forest applied to a spiral data distribution. (a) Input unlabeled data points in their 2D feature space. (b,c,d) Forest densities for different tree depths D. The original training points are overlaid in green. The complex distribution of input data is captured correctly by a deeper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

5.3.3 More Complex Examples

A more complex example is shown in Figure 5.6. The noisy input data is organized in the shape of a four-arm spiral (Figure 5.6(a)). Three density forests are trained on the same dataset with $T = 200$ and varying

depth D. The corresponding densities are shown in Figures 5.6(b), 5.6(c) and 5.6(d). Here, due to the greater complexity of the input data distribution shallower trees yield under-fitting, that is, overly smooth and detail-lacking density estimates. In this example good results are obtained for $D = 6$ as the automatically estimated 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 neighbor 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 Figure 5.7(b) and the corresponding non-parametric densities in Figure 5.7(c) and 5.7(d) clearly shows much smoother results for the forest output. Both the Parzen and the nearest neighbor estimators produce artifacts due to hard choices of, for example, the Parzen window bandwidth or the number k of neighbors. 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 section. Next, we compare (qualitatively) density forests with variants of the Gaussian mixture model.

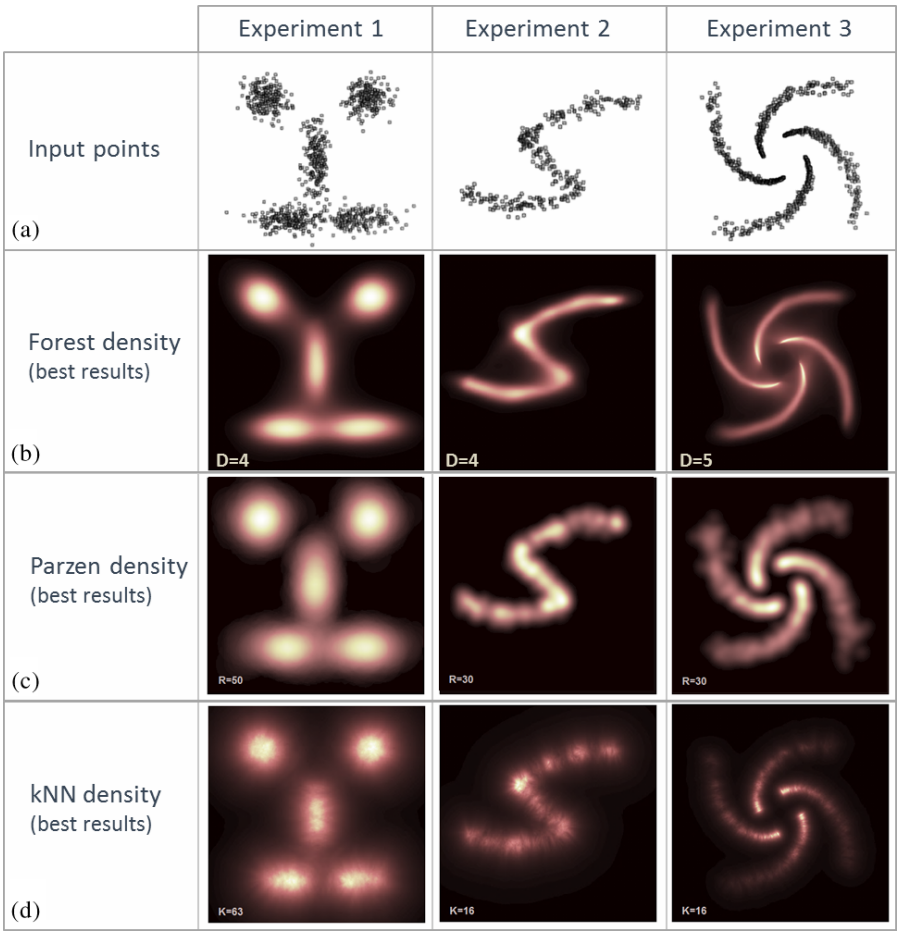


Fig. 5.7 Comparison between density forests and non parametric estimators. (a) Input unlabeled points for three different experiments. (b) Forest-based densities. Forests were computed with $T = 200$ and varying depth D. (c) Parzen window densities (with Gaussian kernel). (d) K-nearest neighbor densities. 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).

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 Figure 5.7(a). Figure 5.7(b) shows the (visually) best results obtained with a GMM, using EM for its parameter estimation [5]. We can observe

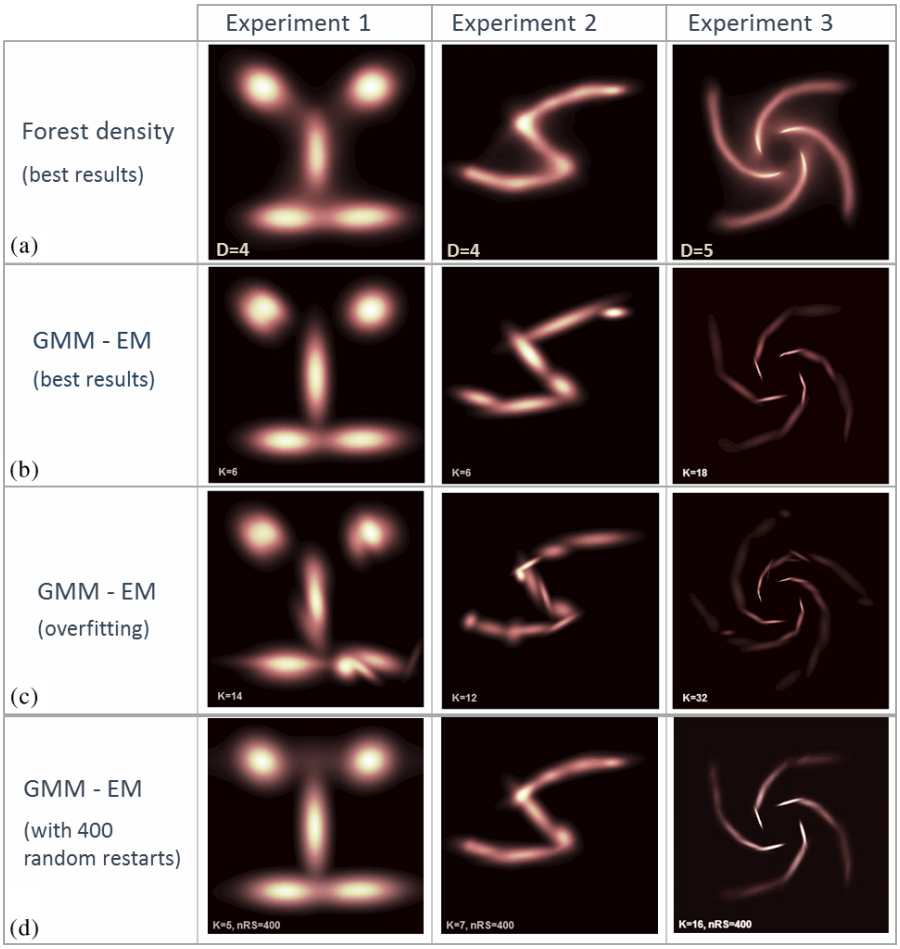


Fig. 5.8 Comparison with GMM EM. (a) Forest-based densities. Forests were computed with $T = 200$ and optimized depth D. (b) GMM density with a relatively small number of Gaussian components. The model parameters are learned via EM. (c) GMM density with a larger number of Gaussian components. Increasing 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 especially for the spiral example.

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 Figure 5.7(c). 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 Figure 5.7(c), 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 randomness (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.

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

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

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

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

$$T \times (D \times B + G) \tag{5.8}$$

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 very efficient to compute.[1] Under these circumstances we may be able to ignore the term $D \times B$ in (5.8) and the cost of testing a density forest becomes comparable to that of a conventional, *single* GMM with T components.

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

Comparing training costs between the two models is a little harder because it involves the number of EM iterations (for 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 unlabeled 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 Figure 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 a Gaussian.

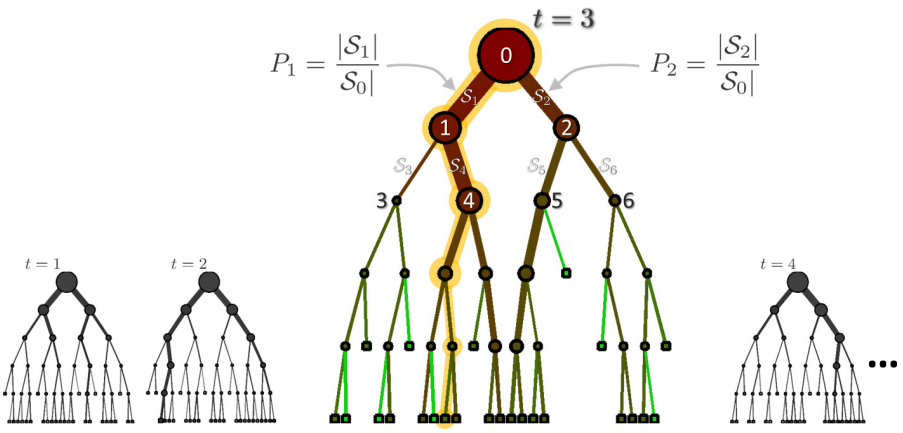


Fig. 5.9 Drawing random samples from the generative density model. Given a trained density forest we can generate random samples by: (i) selecting one of the component trees, (ii) randomly navigating down to a leaf and, (iii) drawing a sample from the associated Gaussian. The precise algorithmic steps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

Given a density forest with T trees:

(1) Draw uniformly a random tree index $t \in \{1, \ldots, T\}$ 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 Figure 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.

Given a set of T GMMs learned with random restarts:

(1) Draw uniformly a GMM index $t \in \{1, \ldots, T\}$ 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.

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 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

$$N \times (1 + 1) \times J + N \times G \tag{5.9}$$

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

For comparison, sampling from a random-restart GMM is illustrated in Algorithm 5.2. It turns out that the cost of drawing N samples under such a model is identical to 5.9. It is interesting to see how although the

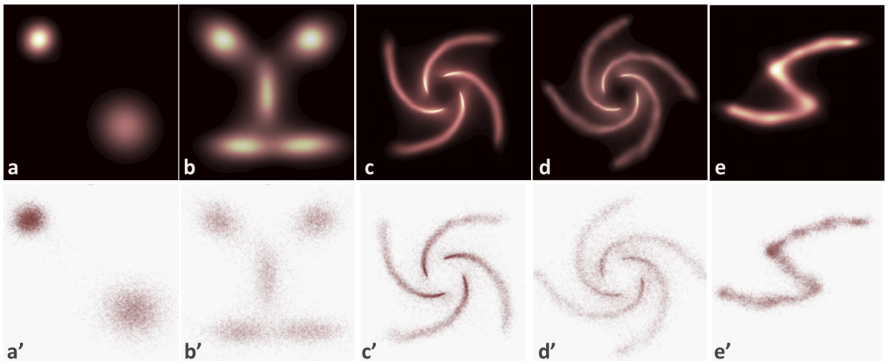


Fig. 5.10 Sampling results (*Top row*). Densities learned from hundreds of training points, via density forests. (*Bottom row*) Random points generated from the learned forests. We draw 10,000 random points per experiment (different experiments in different columns).

two algorithms are built upon different data structures, their steps are very similar. 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 a simple algorithm produces good results both for simpler, Gaussian-mixture distributions (Figures 5.10(a) and 5.10(b)) as well as more complex densities like spirals and other convolved shapes (Figures 5.10(c), 5.10(d) and 5.10(e)).

5.6 Dealing with Non-function Relations

Section 4 concluded by showing shortcomings of regression forests trained on inherently ambiguous training data, that is, 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 Figure 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 modeled properly could increase the prediction confidence (see also [71]).

We repeat a variant of this experiment in Figure 5.11. However, this time a *density* forest is trained on the "S-shaped" training set. In contrast to the regression approach in Section 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 *unlabeled*. 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

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

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

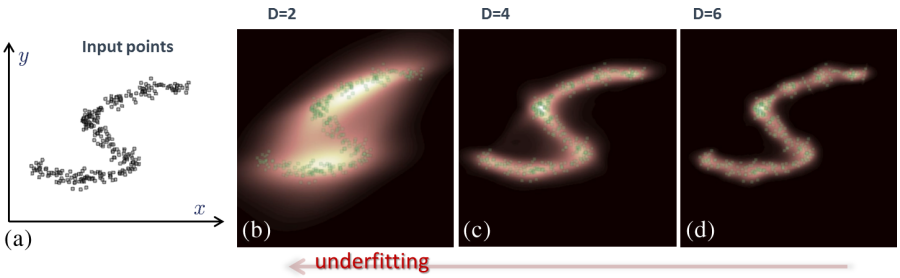


Fig. 5.11 Training density forest on a "non-function" dataset. (a) Input *unlabeled* training points on a 2D feature space. (b,c,d) Three density forests are trained on such data, and the corresponding densities shown in the figures. Dark pixels correspond to small density and vice-versa. The original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

the covariance

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

In Figure 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 5.12(a) shows the training points and an input (test) 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 5.12(b) 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 Figure 5.12(b)). As shown in Figure 5.12(c) 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 (domain bounded) Gaussians are encountered, one per leaf (see Figures 5.12(d) and 5.13). Consequently, the tree conditional is piece-wise Gaussian and defined as follows:

$$p_t(y|x = x^*) = \frac{1}{Z_{t,x^*}} \sum_{l \in \mathcal{L}_{t,x^*}} [y_l^{\mathtt{B}} \leq y < y_l^{\mathtt{T}}] \pi_l \, \mathcal{N}(y; \mu_{y|x,l}, \sigma_{y|x,l}^2) \qquad (5.10)$$

with the leaf conditional mean $\mu_{y|x,l} = \mu_y + \frac{\sigma_{xy}^2}{\sigma_{yy}^2}(x^* - \mu_x)$ and variance $\sigma_{y|x,l}^2 = \sigma_{yy}^2 - \frac{\sigma_{xy}^4}{\sigma_{xx}^2}$. In (5.10) $\mathcal{L}_{t,x^*}$ denotes the subset of all leaves in the

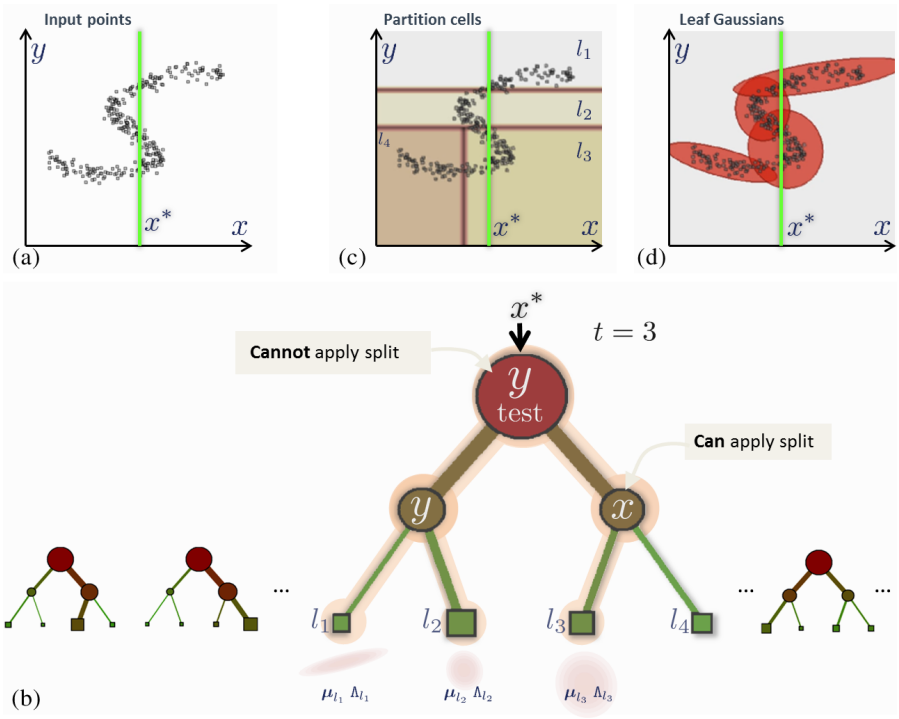


Fig. 5.12 Regression from density forests. (a) 2D training points are shown in black. The green vertical line denotes the value x^* of the independent variable. We wish to estimate $p(y|x = x^*)$. (b) When testing a tree on the input x^* some split nodes cannot apply their associated split function and 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.

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

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

with ϕ denoting the 1D cumulative normal distribution function

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

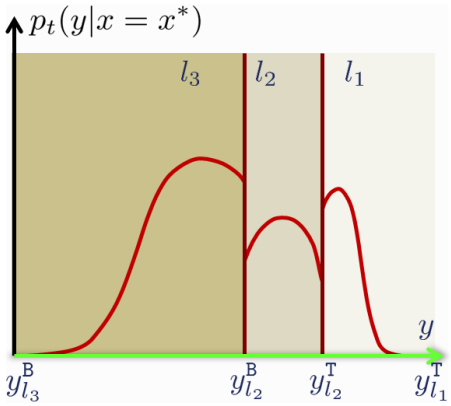


Fig. 5.13 The tree conditional is a piece-wise Gaussian. See text for details.

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, for example, the conditional $p(y|x = x_3)$ in Figure 5.14 with the regression distribution in 4.10b we see that now the conditional shows three very distinct modes rather than a large, 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 section by discussing the issue of efficiently drawing random samples from the conditional model $p(y|x)$.

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 (Figure 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

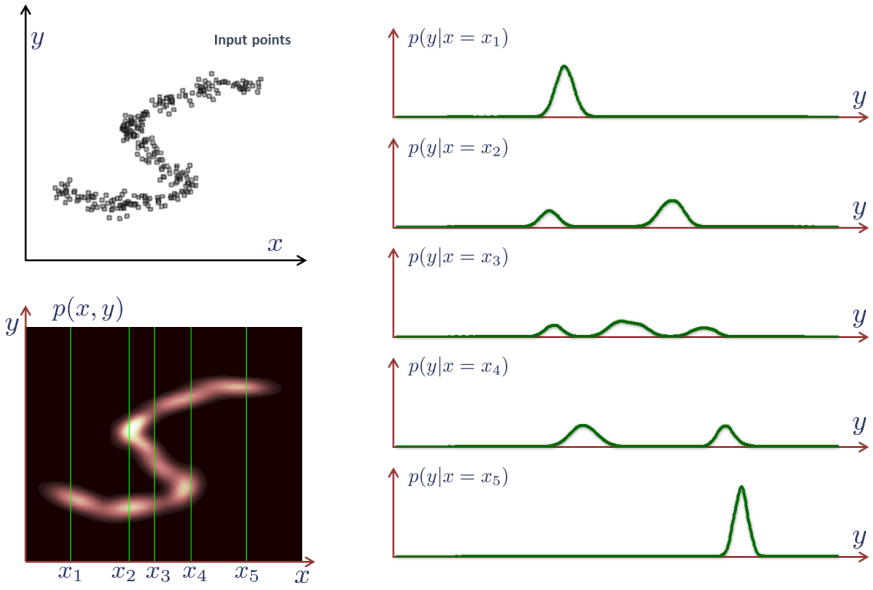


Fig. 5.14 Regression from density forests. The conditionals $p(y|x = x_i)$ show multimodal behavior. This is an improvement compared to regression forests.

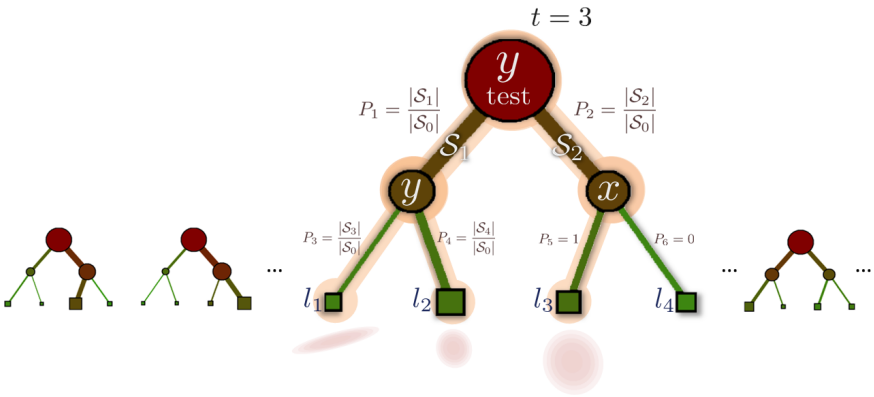


Fig. 5.15 Sampling from conditional model. Since x is known and y unknown y-nodes cannot apply the associated split function. When sampling from such a tree a child of a y-node is chosen randomly. Instead, the child of an x-node is selected deterministically. See text for details.

Figure 5.16, for five fixed values of the independent variable x. In Figure 5.16(b) darker regions indicate overlapping sampled points. Three distinct clusters of points are clearly visible along the $x = x_3$ line, two

Given a density forest with T trees trained with axis-aligned weak learners and an input value $x = x^*$:

(1) Sample uniformly $t \in \{1, \ldots, T\}$ 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.

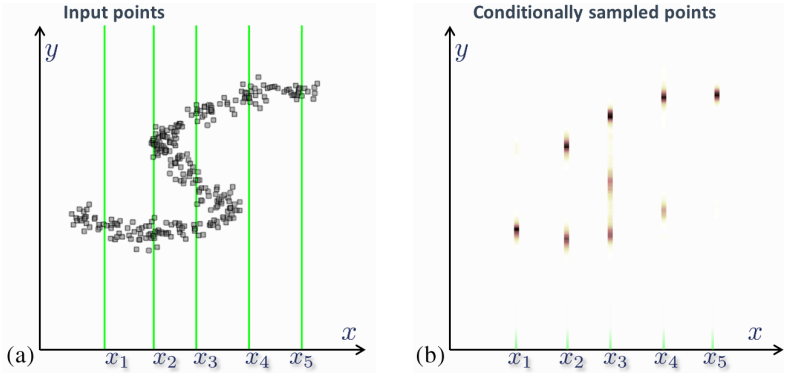


Fig. 5.16 Results on conditional point sampling. Tens of thousands of random samples of y are drawn for five fixed positions in x following Algorithm 5.3. In (b) the multimodal nature of the underlying conditional becomes apparent from the empirical distribution of the samples.

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, 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 (Figure 5.17b), via the multivariate inverse probability integral transform algorithm [27]. 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, Figure 5.17c). Finally, a quantitative comparison between the estimated density ($p(\mathbf{v})$) and the ground-truth one ($p_{\mathtt{gt}}(\mathbf{v})$)

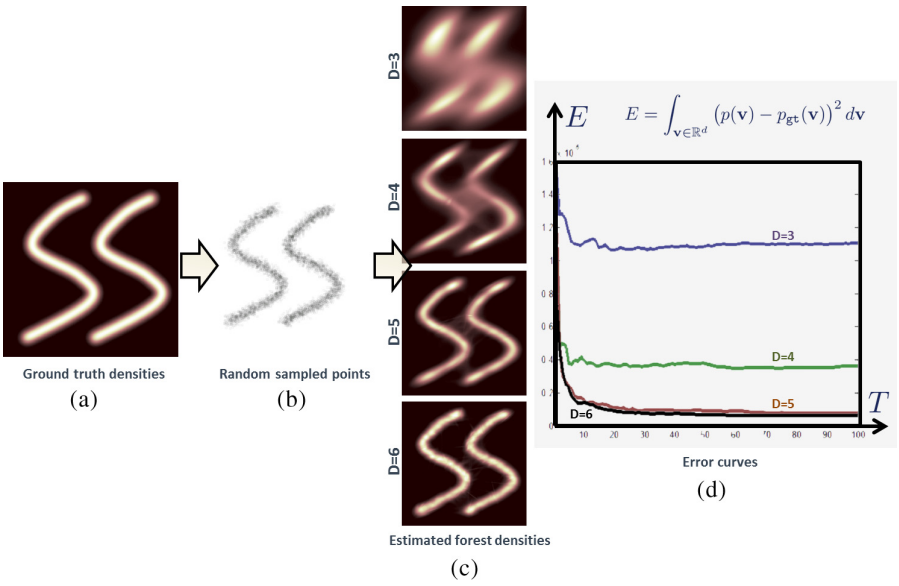


Fig. 5.17 Quantitative evaluation of forest density estimation. (a) An input ground-truth density (non-Gaussian in this experiment). (b) Thousands of random points drawn randomly from the density. The points are used to train four density forests with different depths. (c) During testing the forests 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 $D \in \{3, 4, 5, 6\}$.

can be carried out. The density reconstruction error is computed here as a sum of squared differences:

$$E = \sum_{\mathbf{v}} (p(\mathbf{v}) - p_{\mathtt{gt}}(\mathbf{v}))^2. \tag{5.11}$$

Alternatively one may consider the technique in [99]. Note that due to probabilistic normalization the maximum value of the error in (5.11) is 4. The curves in Figure 5.17d show how the reconstruction error diminishes with increasing forest size and depth. Unsurprisingly, in our 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

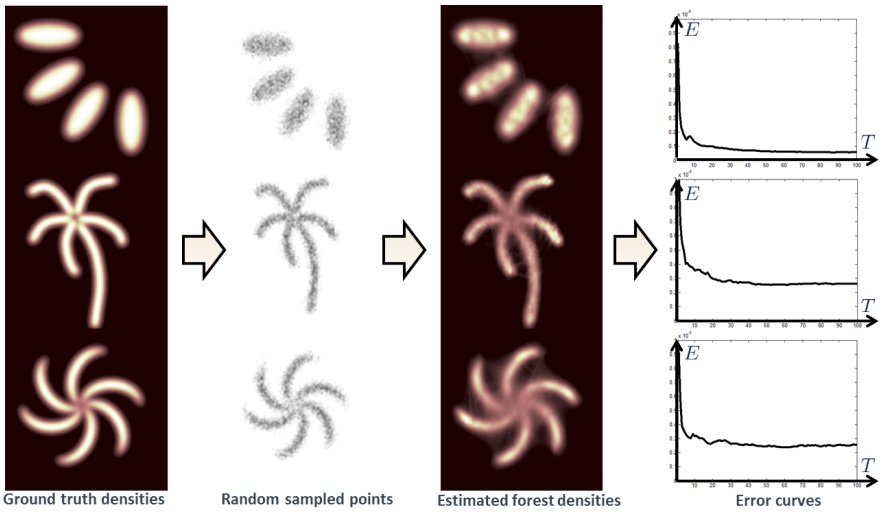


Fig. 5.18 Quantitative evaluation, further results. (a) Input ground-truth densities. (b) Thousands of points sampled randomly from the ground-truth densities. (c) Densities estimated by the forest. Density values are computed for all points in the domain (not just the training points). (d) Error curves 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 center of the spiral arms).

results in those troublesome areas. In Figure 5.18 using larger and larger sets of sampled training points would produce lower and lower reconstruction errors. A more thorough analysis of the consistency of our density estimator is deferred to future work.[2] Density forests are the backbone of manifold learning and semi-supervised learning, described next.

[2] See "consistent estimator" in wikipedia for a definition of consistency.

6

Manifold Forests

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

This section 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 intrinsic distribution (e.g., on a hyperplane, 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) [52]. PCA is based on the computation of directions of maximum data variability. This is obtained simply by eigen-decomposition of the data covariance matrix computed in the original space. Therefore, PCA is a linear model and as such has considerable limitations in real problems. A popular, nonlinear technique is "isometric feature mapping" (or IsoMap) [101] which estimates low dimensional embeddings that tend to preserve geodesic distances between point pairs.

Manifold forests build upon "Laplacian eigenmaps" [3] 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 [90]. Recent probabilistic interpretation of spectral dimensionality reduction may be found in [26, 70]. 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, for example, for cardiac analysis [29, 111], registration [42], and brain image analysis [36]. A more thorough exploration of the vast literature on manifold learning and dimensionality reduction is beyond the scope of this review. 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 [33, 46]. 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 (Section 2) for use in manifold learning.

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

> *Given a set of k unlabeled 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 $\mathbf{f}: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}, \mathbf{f}(\mathbf{v}_i) = \mathbf{v}'_i$ such that $d' \ll d$ and that preserves the observations' relative geodesic distances.*

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

In Figure 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

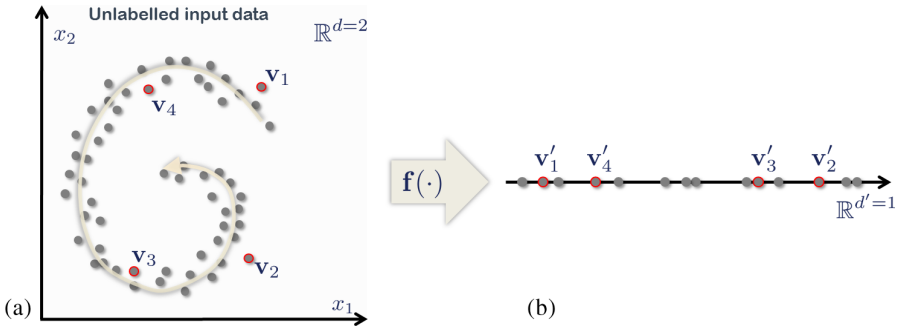


Fig. 6.1 Manifold learning and dimensionality reduction. (a) Input, unlabeled data points are denoted with circles. They live in a high-dimensional space (here $d = 2$ for illustration clarity). A red outline highlights some selected points of interest. (b) The target space is much lower dimensionality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

are as close as possible to the corresponding 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 section builds upon density forests, with much of the mathematical modeling borrowed from Section 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 and unsupervised information 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 section 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 domain bounded 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. 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 section, at its leaves a clustering tree t defines a partition of the input points

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

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

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

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 [3] 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 might 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 [3, 70] which we summarize here for completeness.

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 $\mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_{k-1}$ be the solutions of (6.7) in *increasing* order of eigenvalues

$$\begin{aligned} \mathtt{L}\mathbf{e}_0 &= \lambda_0 \mathbf{e}_0 \\ \mathtt{L}\mathbf{e}_1 &= \lambda_1 \mathbf{e}_1 \\ \vdots \quad &\quad \vdots \\ L\mathbf{e}_{k-1} &= \lambda_{k-1} \mathbf{e}_{k-1} \end{aligned} \tag{6.8}$$

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

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

where $i \in \{1, \ldots, k\}$ 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 neighborhood system 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:

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

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 tth 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 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 Figure 6.2). Each image defines a data point in a high dimensional space. When training a manifold forest we can imagine that, for example, 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

Fig. 6.2 Image similarity via ensemble clustering. Different trees (whose leaves are denoted by different color 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 `W`. In `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.

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 mean operation (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 Figure 6.2 as features we could use averages of pixel colors within rectangles placed within the image frame. Position and size of the rectangles is selected during training. This would allow the system to learn, for example, that brown-colored regions are expected toward the bottom of the image for beach scenes, long vertical edges are expected in cityscapes, etc.

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 $d' \ll k$ 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 [3]. 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 section.

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 Figure 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 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, unlabeled 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, for example, in Figure 6.4(b)-leftmost, by the vertical banding artifacts; and in Figure 6.4(d)-leftmost by the single

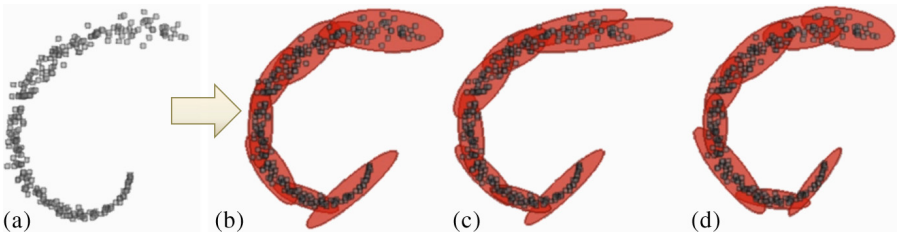


Fig. 6.3 Different clusterings induced by different trees. (a) The input data in 2D. (b,c,d) Different partitions corresponding to different trees in the same manifold forest. A given pair of points will belong to the same cluster (leaf) in some trees and not in others.

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

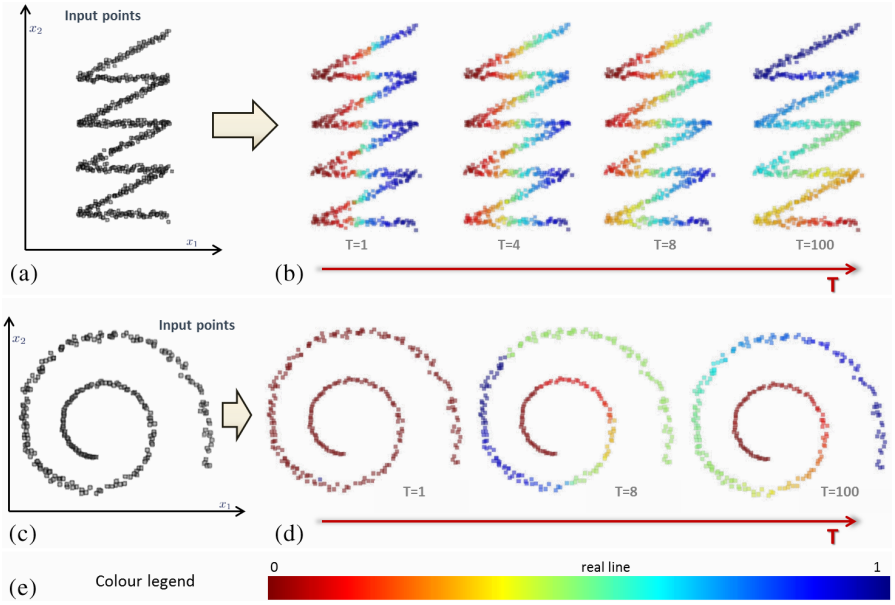


Fig. 6.4 Manifold forest and nonlinear dimensionality reduction. The effect of T. (a,c) Input 2D points for two different synthetic experiments. (b,d) Nonlinear mapping from the original 2D space to the 1D real line is color coded, from dark red to dark blue. In both examples a small forest (small T) 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 color legend. Different colors, from red to blue, denote the position of the mapped points in their target 1D space.

red color for all points. However, as the number of trees increases the affinity matrix `W` better approximates the true pairwise graph affinity. Consequently the color coding (linearly going from dark blue to dark red) starts to follow correctly the intrinsic 1D structure 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.

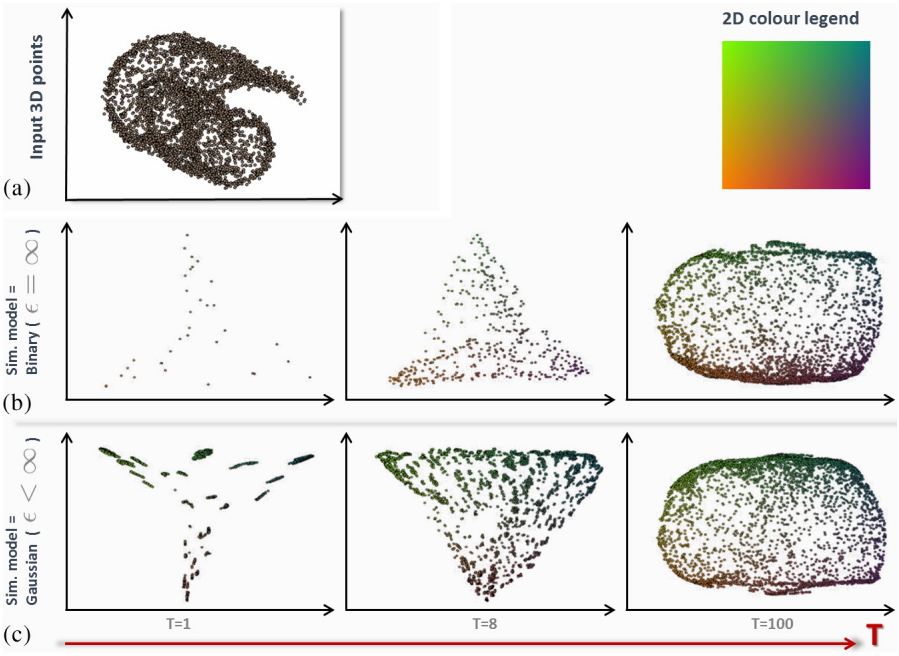


Fig. 6.5 The effect of the similarity model. In this experiment we map 3D points into their intrinsic 2D manifold. (a) The input 3D data is a variant of the well known "Swiss Roll" dataset. The noisy points are organized as a spiral in one plane with a sinusoidal component in the orthogonal direction. (b) Different mappings into the 2D plane for increasing forest size T. Here we use binary affinity. (c) 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).

Figure 6.5 compares the binary and Gaussian affinity models in a 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 Figure 6.5). 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 color-coded by the automatically

Fig. 6.6 Unfolding the "Swiss Roll" manifold. Different 3D views from various viewpoints. Color-coding indicates the mapped 2D manifold. See color-legend in Figure 6.5.

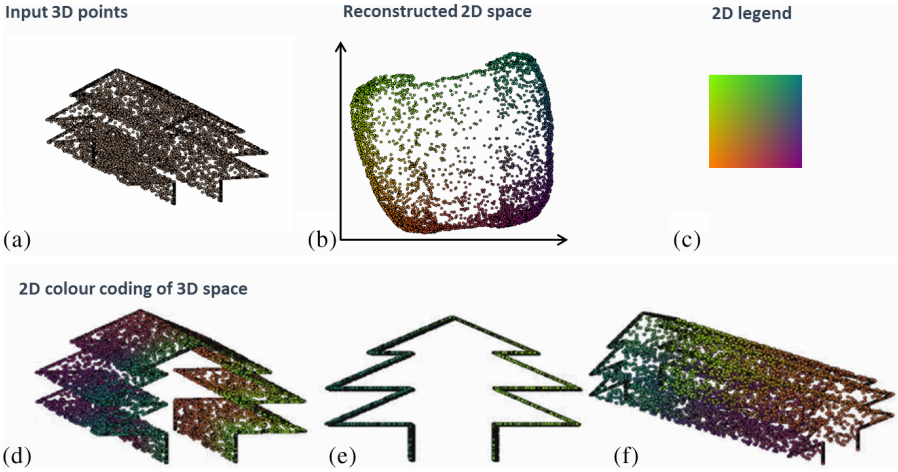


Fig. 6.7 The "Christmas Tree" manifold. (a) The unlabeled input data points in their 3D space. (b) The reconstructed 2D space. (c) The 2D color legend. (d,e,f) Different views of the 3D points with color coding corresponding to the automatically discovered 2D mapping.

discovered underlying 2D manifold. The mapped colors 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 Figure 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 section 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 [3]. 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.

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

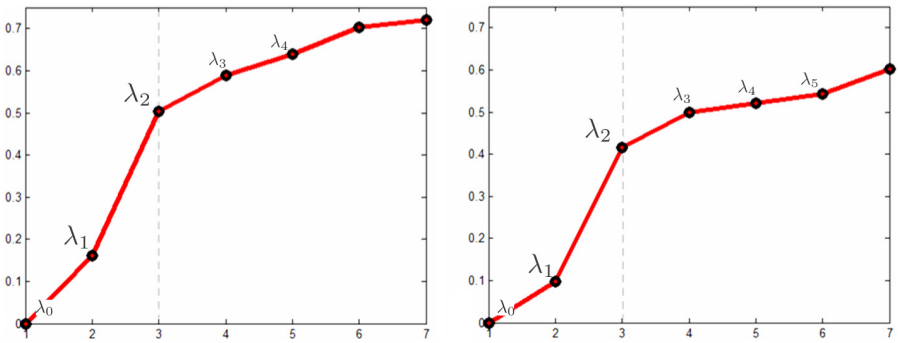


Fig. 6.8 Discovering the manifold intrinsic dimensionality. (a) The sorted eigenvalues of the normalized graph Laplacian for the "Swiss Roll" 3D example, with binary affinity model. (b) As above but with Gaussian affinity. In both curves there is a clear elbow in correspondence of λ_2 thus indicating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

value of d. Similarly, Gaussian affinities produce sharper elbows than binary affinities.

6.4 Learning Manifold of Object Shapes

This section discusses the possible use of manifold forests for shape analysis.

Specifically, we are given a set of manually generated object shapes (represented as binary segmentation masks) and we wish to map them onto a low-dimensional space so that visually similar shapes are mapped to positions close to one another. We can think of this task as learning a shape similarity function. This task is unsupervised in the sense that we ignore associations of input masks with their object class name.

In this experiment the input images and corresponding object segmentations come from the PASCAL VOC database [30]. For simplicity, here we focus on images of three object classes only: bicycles, aeroplanes, and cars. Figure 6.9 shows some of the input images and their corresponding object segmentations (red silhouette).

Each segmentation mask is represented as a d-dimensional array $\mathbf{v} = (x_1, \ldots, x_i, \ldots, x_d) \in \mathbb{R}^d$ of feature responses, with $d = 150$. The feature responses x_i are computed as the value of the signed distance (distance from the known object outline) at a given point position. In Figure 6.9 the distance map is shown as a gray-level image, with darker values indicating points inside the silhouette, and vice-versa. The positions of the randomly generated 150 points used to extract features (shown as green circles) is fixed throughout.

We wish to map this 150-dimensional representation of shapes onto a 2D space. We do so by training a manifold forest with maximum depth $D = 5$ and size $T = 100$. Following the algorithm described earlier we obtain the mapping shown in Figure 6.10. In the figure we observe that despite the lack of supervision, images of aeroplanes are clustered together and the same for the silhouettes of bicycles and cars. The presence of occlusions and/or missing parts does not seem to affect the results much.

Once again, here at each node the features that provide the best split are chosen automatically out of a large pool of features. Notice that in

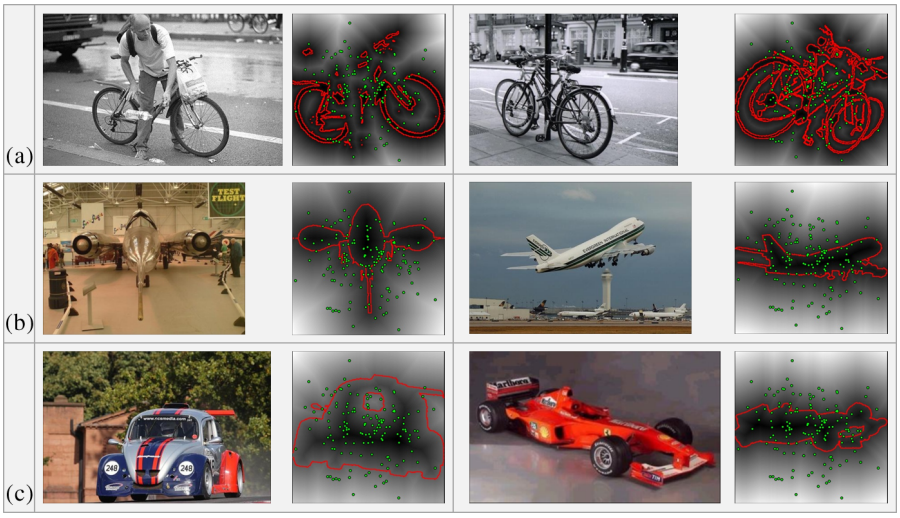


Fig. 6.9 Learning manifolds of object shapes. (a) Some images of bicycles and corresponding manual segmentations (red outline). The right panel also shows the distance map (as a gray-level image) and the set of randomly selected points (in green) used to extract feature responses. (b,c) Same as in (a) but for aeroplanes and cars, respectively. The object masks have been pre-rotated, centered and scaled (via PCA analysis) to remove variations due to similarity transformations and help capture only intrinsic shape differences.

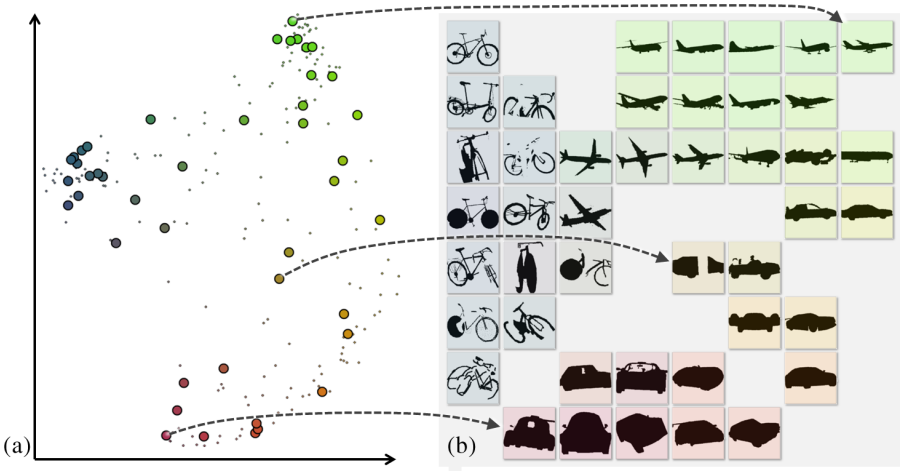


Fig. 6.10 Learning manifolds of object shapes. (a) The target 2D space. Each point corresponds to a learned 2D representation of one of the input shapes. Larger circles denote some selected shapes, for visualization. (b) The selected shapes visualized in a regular grid arrangement. The overlaid colors aid visual clarity as they help match the silhouettes with the corresponding circles in (a).

this case the original dimensionality d must be finite since the entropy has to be computed on the entire multi-variate Gaussian representation.

6.5 Learning Manifold of Text Documents

This section illustrates the possible use of manifold forests in a non-vision application, that is, the automatic organization of news reports and automatic discovery of correlations between text documents.

In this experiment we downloaded some news reports from the web (a snapshot of news taken on the 23 November 2011) using different search engines and search terms. The reports are then saved as unstructured text files with no associated labels. Everything here is completely unsupervised. Each document $\mathbf{v}$ is represented as an array of d features. Each feature x_i is simply the count of occurrences of a given keyword in the selected document $\mathbf{v}$. In this case the vocabulary of keywords is defined manually and it includes items like: "crisis," "Berlusconi," "Europe," "mortgage," etc. Thus in this case $\mathbf{v} \in \mathbb{N}^d$. The size of the vocabulary is $d = 100$.

We train a manifold forest with $T = 200$ trees and depth $D = 4$ and use it to map all documents from their original high-dimensional space to a 2D space. Here we use a binary affinity model and axis-aligned weak learners. The results are illustrated in Figure 6.11 where we observe that documents talking about similar topics have been positioned in nearby locations. The topics themselves have been discovered automatically and without supervision. For instance, there is a cluster of documents in the center referring to the European financial crisis of November 2011 (gray-purple squares). Within this central region there are sub-clusters referring more specifically to the crisis in Italy (gray-blue), Greece (gray), and Spain (purple), respectively. Then, there is a region referring to the role of the German chancellor A. Merkel in the crisis (in orange). Furthermore, the green squares (spatially contiguous to the Italian news cluster) contain reports related to a different topic, that is, the defeat of the Manchester City football team in Naples (in Italy).

Note that here all these associations have been discovered completely automatically and in an unsupervised manner. The documents

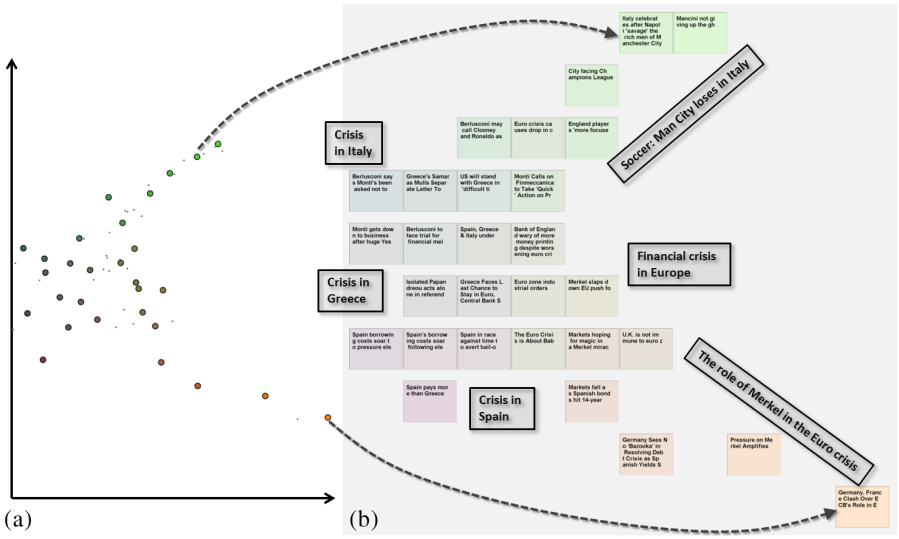


Fig. 6.11 Learning manifolds of news reports. (a) The target 2D space. Each point corresponds to a learned 2D representation of one of the input news documents. Larger circles denote some selected documents, for visualization. (b) The selected documents (just the titles shown here) visualized as boxes in a regular grid. The overlaid colors aid visual clarity as they help matching the document titles with the corresponding circles in (a). The black boxes indicating the general topics have been overlaid manually for clarity.

have been mapped onto a space where Euclidean distances make semantic sense, to some extent. Regarding the choice of features, in this toy example we use simple histograms of words from a predefined vocabulary. However which words to use and with what weight is automatically determined by the training algorithm via the optimization of the objective function. Thus, one may think of extending the size of the chosen vocabulary to be the entire English dictionary. Alternative, more sophisticated features may be used too.

6.6 Discussion

In this section we have discussed some of the advantages of manifold forests and have studied 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 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 the more trees the better the behavior.

Manifold forests solve the problem of finding appropriate data neighborhoods adaptively and efficiently. The actual mapping from the original space to a lower dimensional one has been achieved here via Laplacian eigenmaps, however, alternative dimensionality-reduction algorithms may be used.

A more rigorous validation of manifold forests 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 sections have discussed the use of decision forests in supervised problems (for example, regression and classification) as well as unsupervised ones (for example, density and manifold estimation). This section 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 labeled training data and a large set on unlabeled data. 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 unlabeled data to improve classification.

Semi-supervised machine learning is interested in the problem of transferring existing ground-truth labels to the unlabeled (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 sections (Sections 3 and 4), where the test data is *not* available at training time. This section 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 section 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 discussed.

7.1 Literature on Semi-supervised Learning

A popular technique for semi-supervised learning is transductive support vector machines [51, 109]. Transductive SVM (TSVM) is an extension of the popular support vector machine algorithm [106] which maximizes the separation of both labeled and unlabeled data. The experimental section of this section will present comparisons between forests and TSVM.

Excellent, recent references for semi-supervised learning and active learning are [18, 20, 100, 112] which provide a nice structure to the vast amount of literature on these topics. A thorough literature survey is beyond the scope of this paper and here we focus on forest-based models.

In [58] 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, 28]. 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 Section 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 labeled and unlabeled data we wish to associate a class label to all the unlabeled data.

Unlike inductive classification here all unlabeled "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 $\mathbf{v} = (x_1, \ldots, x_d) \in \mathbb{R}^d$.

Here we consider two types of input data: *labeled* $\mathbf{v}^{\mathtt{l}} \in \mathcal{L}$ and *unlabeled* $\mathbf{v}^{\mathtt{u}} \in \mathcal{U}$. This is illustrated in Figure 7.1(a), where data points are denoted with circles. Colored circles indicate labeled training points, with different colors denoting different labels. Unlabeled data is shown in gray. Figures 7.1(b) and 7.1(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 labeled data. Both labeled and unlabeled data are used to optimize an objective function with two components: a supervised and an unsupervised one, as described next.

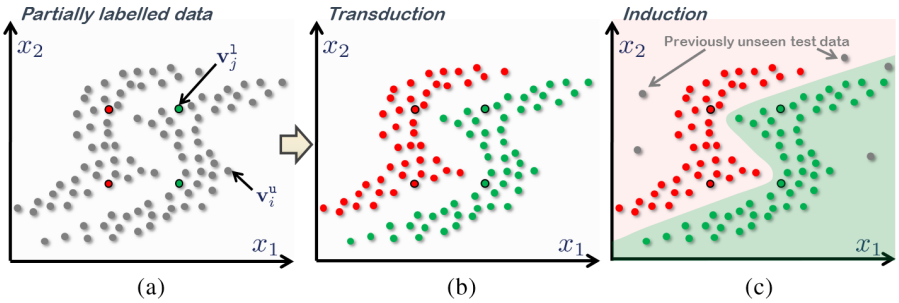


Fig. 7.1 Semi-supervised forest: input data and problem statement. (a) Partially labeled input data points in their two-dimensional feature space. Different colors denote different labels. Unlabeled data is shown in gray. (b) In transductive learning we wish to propagate the existing ground-truth labels to the many, available unlabeled data points. (c) In inductive learning we wish to learn a generic function that can be applied to previously unavailable test points (gray circles). Training a conventional classifier on the labeled data only would produce a sub-optimal classification surface, that is, 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 labeled training data as well as separating different high density regions from one another. This is achieved by maximizing the following mixed information gain:

$$I_j = I_j^{\mathtt{u}} + \alpha I_j^{\mathtt{s}}. \tag{7.1}$$

In the equation above $I_j^{\mathtt{s}}$ is a supervised term and depends only on the labeled training data. In contrast, $I_j^{\mathtt{u}}$ is the unsupervised term and depends on all data, both labeled and unlabeled. 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^{u} is defined via differential entropies defined over continuous parameters (that is, the parameters of the Gaussian associated with each cluster):

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

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

The ensemble model. During testing, a semi-supervised classification tree 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 labeled dataset (as in Figure 7.2(a)) 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 Figure 7.2(b), 7.2(c), and 7.2(d). The different colored 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

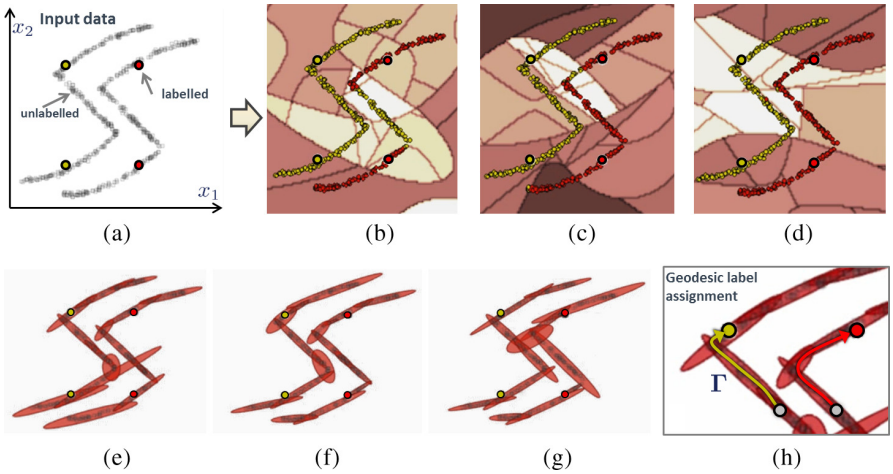


Fig. 7.2 Label transduction in semi-supervised forests. (a) Input points, only four of which are labeled as belonging to two classes (red and yellow). (b,c,d) Different transductive trees produce different partitions of the feature space. Different regions of high data density tend to be separated by cluster boundaries. Geodesic optimization enables assigning labels to the originally unlabeled 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.

within. Label transduction from annotated data to unannotated data can be achieved directly via the following minimization:

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

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 $\mathbf{s}_0 = \mathbf{v}^{\mathrm{u}}, \mathbf{s}_{L(\Gamma)} = \mathbf{v}^{\mathrm{l}}$, respectively. The local distances $d(\cdot,\cdot)$ are defined as symmetric Mahalanobis distances

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

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, for example, Euclidean ones) discourages paths from cutting across regions of low data density, a key requirement for correct transduction.[1] In effect we have defined geodesics on the latent probability density function.

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

7.4 Induction from Transduction

Previously we have described how to propagate class labels from labeled training points to already available unlabeled ones. Here we describe how to infer a general probabilistic classification rule $p(c|\mathbf{v})$ that may be applied to previously unavailable 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

[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 [23].

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 (Figures 7.2(b), 7.2(c), and 7.2(d)). After label propagation we also have attached a class label to *all* available data (with, in general, different trees assigning different classes to the points in $\mathcal{U}$). 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 inductive classification results on the same example as in Figure 7.2. Now the inductive classification posterior is tested

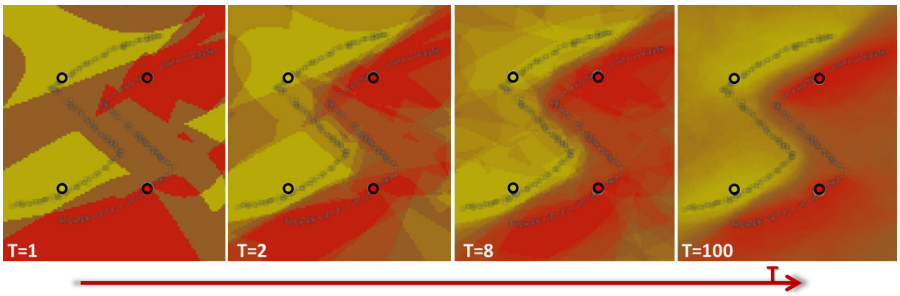


Fig. 7.3 Learning a generic, inductive classification rule. Output classification posteriors, tested on all points in a rectangular section of the feature space. Labeled training points are indicated by colored circles (only four of those per image). Available unlabeled data are shown by small gray squares. 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 unlabeled points, thus demonstrating the validity of the use of unlabeled data densities. From left to right the number of trees in the forest increases from $T = 1$ to $T = 100$. See text for details.

on all points 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 unlabeled points (small gray squares). Finally we observe that classification uncertainty is greater in the middle due to its increased distance from the four ground-truth labeled points (yellow and red circles).

Discussion. In summary, by using our mixed information gain and some geodesic-based label transduction the generic decision forest model can be readily adapted for use in semi-supervised tasks. Semi-supervised forests can be used both for transduction as well as (density-aware) induction with an efficient, single-step 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 for example, self-training techniques [84]. Self-training techniques work by: (i) training a supervised classifier, (ii) classifying the unlabeled 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 unlabeled 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 amount of ground-truth labelled data.

The effect of additional labeled data and active learning. As observed already, the central region in Figure 7.4(a) 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 its confidence. This effect is indeed illustrated in Figure 7.4(b).

As expected, a guided addition of further labeled 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 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

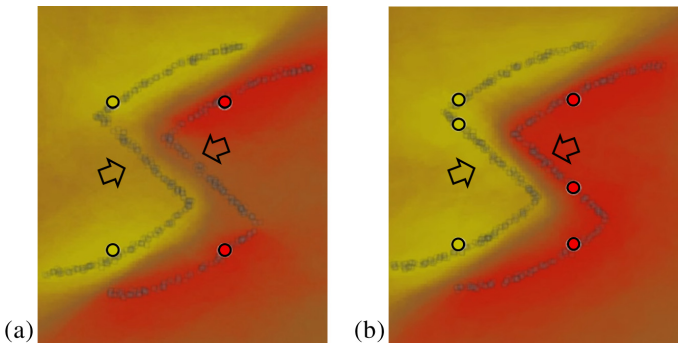


Fig. 7.4 Active learning. (a) Test forest posterior trained with only four labeled points and hundreds of unlabeled ones. The middle region shows lower confidence (pointed at by two arrows). (b) As before, but with two additional labeled points placed in regions of high uncertainty. The overall confidence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

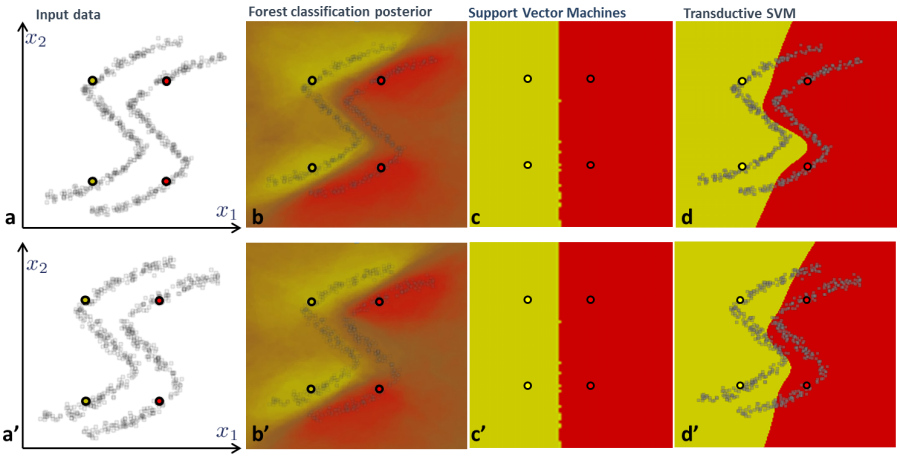


Fig. 7.5 Comparing semi-supervised forests with SVM and transductive SVM. (a) Input partially labeled data points. (b) Semi-supervised forest classification posterior. The probabilistic output captures prediction uncertainty (mixed-color pixels in the central region). (c) Unsurprisingly, conventional SVM produces a vertical separating surface and it is not affected by the unlabeled 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′).

SVM [106] as well as transductive SVM [51, 109], 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 labeled or unlabeled sets, or both) is reflected in lower prediction confidence. Second, while transductive SVM manages to exploit the presence of available unlabeled 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.

Handling multiple classes. Being tree-based models semi-supervised forests can natively handle multiple (>2) classes. This is demonstrated in Figure 7.6 with a four-class synthetic experiment. The input points are randomly drawn from four bi-variate Gaussians. Out

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

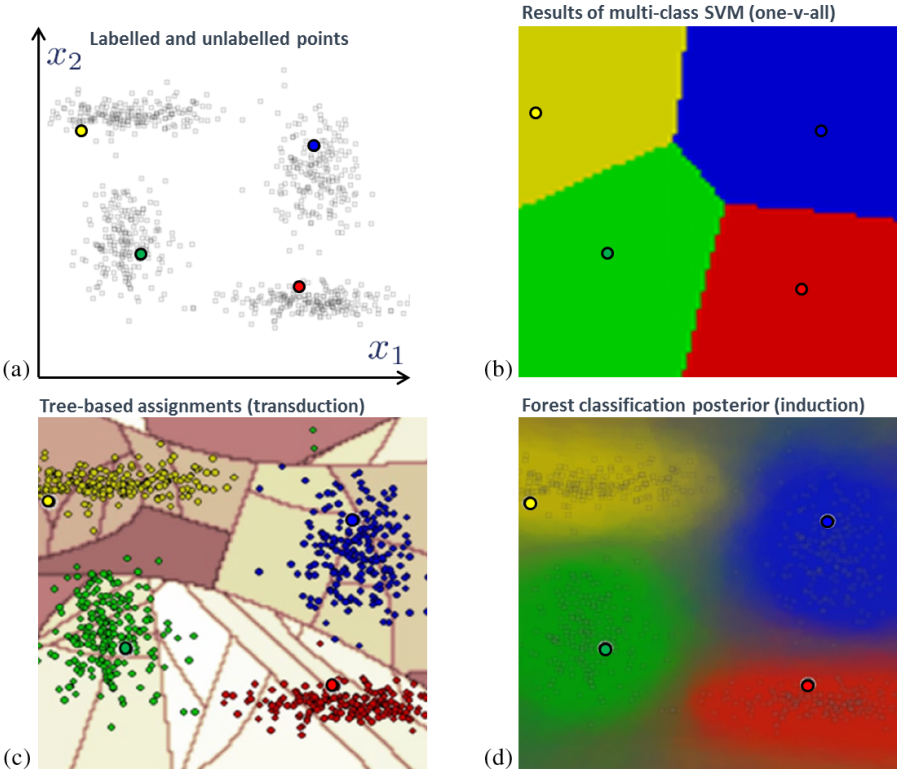


Fig. 7.6 Handling multiple classes. (a) Partially labeled input data. We have 4 labeled points for the 4 classes (different colors for different classes). (b) Classification results for one-v-all support vector machines. (c) Transduction results based on a single decision tree. Originally unlabeled points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

of hundreds of points only four are labeled with their respective classes (shown in different colors). Conventional one-v-all SVM classification results in hard class assignments (Figure 7.6(b)). Tree-based transductive label propagation (for a single tree) is shown in Figure 7.6(c). Note that slightly different assignments are achieved for different trees. The forest-based inductive posterior (computed for $T = 100$) is shown in Figure 7.6(d) where the contribution of previously unlabeled points to the shape of the final posterior is clear. Regions of low confidence in the posterior correspond to regions of low density in the data.

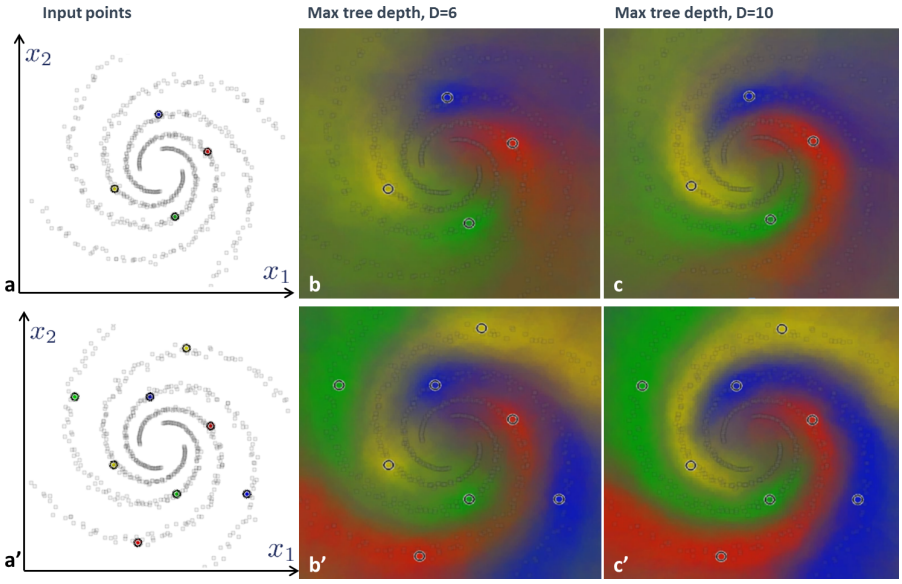


Fig. 7.7 Semi-supervised forest: effect of tree depth. (a) Input labeled and unlabeled points. We have 4 labeled points and 4 classes (color coded). (a′) As in (a) but with double the amount of labeled data. (b,b′) Semi-supervised forest inductive classification posterior for $D = 6$ tree levels. (c,c′)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

The effect of tree depth. We conclude this section by studying the effect of the depth parameter D in Figure 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 ground-truth labeled 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 labeled data. In this relatively complex example, accurate and sharp classification is achieved with just 2×4 labeled data points (for $D = 10$ tree levels) and hundreds of unlabeled 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 section we discuss a few important ones.

8

Random Ferns and Other Forest Variants

This section 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 [38, 69].

In our decision model the amount of randomness in the optimization of split nodes is controlled by the parameter $\rho = |\mathcal{T}_j|$ (Section 2). In our randomized node optimization model when training the jth 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

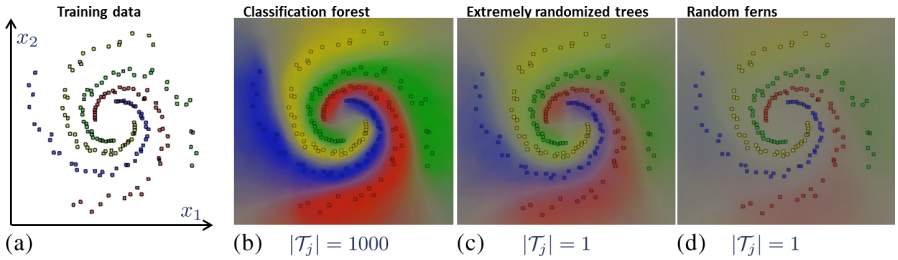


Fig. 8.1 Forests, extremely randomized trees and ferns. (a) Input training points for four classes. (b) Posterior of a classification forest. (c) Posterior of an ensemble of extremely randomized trees. (d) Posterior of a random fern. The randomness parameter is changed as illustrated. All other parameters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

instance of the general decision forest model with the additional constraint that $\rho = 1 \forall j$. In this case no node training is performed.

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 $\rho = 1{,}000$ 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 Figures 8.1(b) and 8.1(c), 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. On the flip side, extremely randomized trees are very efficient to train.

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 [75, 77].

Figure 8.2 illustrates this point. As usual training points are indicated with colored circles, with different colors indicating different classes. In both a decision tree and a decision fern the first node (root)

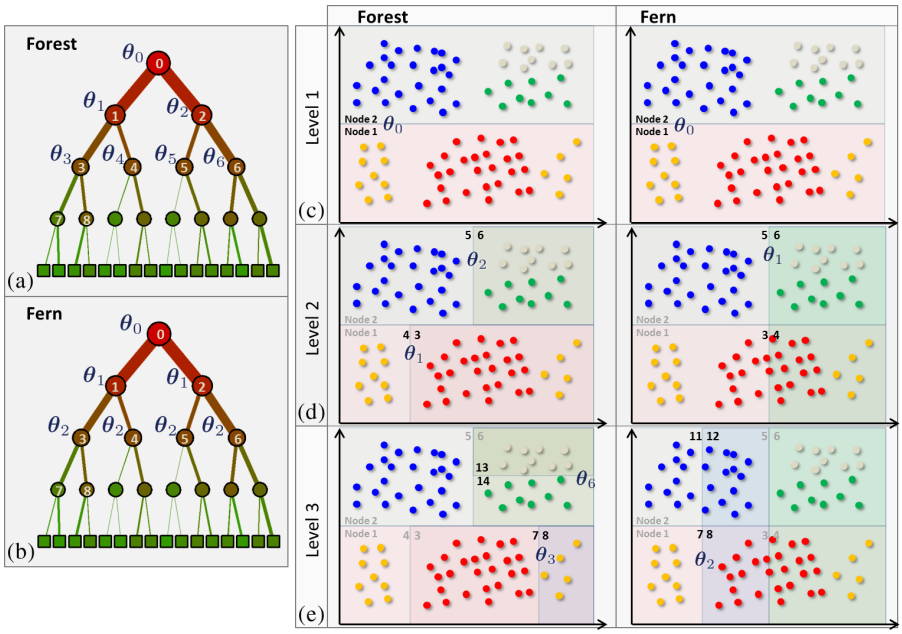


Fig. 8.2 Forests and ferns. A set of labeled training data is used to train a forest and a fern. Here simple axis-aligned weak learners are employed. A fern has fewer parameters than the forest and thus the fern typically requires deeper trees than a forest to split equally well the input training data.

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 (Figure 8.2(d)). 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 Figures 8.2(d) and 8.2(e)). Consequently, in order to split exactly the linearly separable input dataset in this example the fern requires more levels than the forest. This explains why in Figure 8.1(c) we see lower prediction confidence (very washed-out colors) 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, that is, 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 labeled 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 (for example, unnormalized histograms) can be quickly updated (for example, by simply adding the new counts in the appropriate bins). The work in [86] presents further details.

8.4 Structured-output Forests

Often decision forests are used for the semantic segmentation of images. This involves assigning a class to each pixel (voxel) in the image domain (for example, as in Microsoft Kinect for XBox 360). However, such class decisions are often made independently for each pixel. Classic Markov random fields and conditional random fields [8] add generic spatial priors to achieve more homogeneous outputs by smoothing noisy local evidence (possibly conditioned on the image data).

Recent work on structured-output decision forests tries to overcome the limitations of such generic models and learn a class-aware model of spatial context [73]. Several different techniques have been proposed, some of which are summarized below.

8.4.1 Entangled Forests

Entangled forests [68] 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 class probabilities output of previous split nodes in the same tree.

The basic idea stems from the work on Probabilistic Boosting Trees [104] and autocontext [105]. 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 [68]. In [68] 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 5 cm below the right lung and 5 cm above the right kidney is likely to be in the liver.

Biased randomization. The work in [68] 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 Structured Semantic Image Labeling

The work in [54] presents a different approach, where the authors train a classification forest to predict "label patches," as opposed to individual labels for each pixels. Thus each leaf predicts the $n \times n$ structured labels of an entire image patch. In this models difficulties arise when defining an efficient energy for training, and the authors propose some effective approximations. During testing a "patch reconciliation" step

ensures the assignment of a good label for each pixel out of many label patch proposals.

8.4.3 Decision Tree Fields

Nowozin et al. [74] proposed another technique for learning models of context which go beyond conventional conditional random fields. In fact, they combine random decision forests and random fields together in a decision tree field model. The authors build on ideas from [79] and propose a model where both the per-pixel likelihoods as well as the graph pairwise potentials are conditioned on the underlying image content. Different types of pair-wise weights are learned from images using random forests. 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.

Exploring new forest-based models for structured-output prediction is a recent research trend that promises to produce interesting algorithms for the semantic segmentation of photographs as well as medical images.

8.5 Further Forest Variants

The "STAR" model in [78] 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" [64]. Node harvest has the advantage of high interpretability, but seems to work best in low signal-to-noise conditions.

This section has presented only a small subset of the most interesting variants on tree-based machine learning techniques.

9

Conclusions

This review has proposed a unified 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 also 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 popular 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 for example, to figure out optimal ways of incorporating priors (for example, 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 within the same forest [39]. 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 survey 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 [49].

Appendix A — Deriving the Regression Information Gain

This section 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 Figure 1) 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 $\mathtt{A}$

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

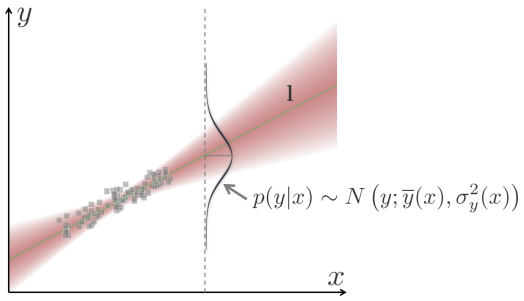


Fig. 1 Probabilistic line fitting. Given a set of training points we can fit a line model to them. For instance, in this example $\mathbf{l} \in \mathbb{R}^2$. Matrix perturbation theory enables us to compute the entire conditional density $p(\mathbf{l}|x)$ from where we can derive $p(y|x)$. Training a regression tree involves minimizing the uncertainty of the prediction $p(y|x)$. Therefore, the training objective is a function of σ_y^2.

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:

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

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 [22, 98] we can estimate a Gaussian density of the line parameters: $\mathbf{l} \sim \mathcal{N}(\bar{\mathbf{l}}, \Lambda_l)$, as follows.

The generic ith 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

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

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

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

Finally the 3×3 line covariance matrix is

$$\Lambda_{\mathbf{l}} = \mathtt{J}\ \mathtt{S}\ \mathtt{J} \tag{A.1}$$

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 kth eigenvalues of the matrix $\mathtt{M}$ and $\mathbf{u}_k$ its corresponding eigenvector. The 3×3 matrix $\mathtt{S}$ in (A.1) is

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

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 $\mathcal{N}(\bar{\mathbf{l}}, \Lambda_l)$. However, what we want is the conditional distribution $p(y|x)$ (see Figure 1) 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 section we will see how this is used in the regression information gain.

In its explicit form a line equation is $y = ax + b$ 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

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

See also Figure 1.

Regression information gain. In a regression forest the objective function of the jth split node is

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

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

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

by substituting (A.2) in (A.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 (A.3) yields the information gain

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

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).

Acknowledgments

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] Y. Amit and D. Geman, "Randomized inquiries about shape; an application to handwritten digit recognition," Technical Report 401, Department of Statistics, University of Chicago, IL, 1994.

[2] Y. Amit and D. Geman., "Shape quantization and recognition with randomized trees," *Neural Computation*, vol. 9, pp. 1545–1588, 1997.

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

[4] B. Benfold and I. Reid, "Unsupervised learning of a scene-specific coarse gaze estimator," in *International Conference on Computer Vision*, Barcelona, Spain, 2011.

[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 *Institute of Electrical and Electronics Engineers International Conference on Computer Vision*, 2007.

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

[11] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 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 *Computer Vision and Pattern Recoginition*, 2011.

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

[15] R. Caruana, N. Karampatziakis, and A. Yessenalina, "An empirical evaluation of supervised learning in high dimensions," in *International Conference on Machine Learning*, pp. 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, eds., *Semi-Supervised Learning*. Cambridge, MA: MIT Press, 2006.

[19] Y. Chen, T.-K. Kim, and R. Cipolla, "Silhouette-based object phenotype recognition using 3D shape priors," in *International Conference on Computer Vision*, Barcelona, Spain, 2011.

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

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

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

[23] A. Criminisi, T. Sharp, and A. Blake, "Geos: Geodesic image segmentation," in *Proceedings of European Conference on Computer Vision*, 2008.

[24] A. Criminisi, J. Shotton, and S. Bucciarelli, "Decision forests with long-range spatial context for organ localization in CT volumes," in *Medical Image Computing and Computer Assisted Intervention Workshop on Probabilistic Models for Medical Image Analysis*, 2009.

[25] A. Criminisi, J. Shotton, D. Robertson, and E. Konukoglu, "Regression forests for efficient anatomy detection and localization in CT studies," in *Medical Image Computing and Computer Assisted Intervention Workshop on Medical Computer Vision: Recognition Techniques and Applications in Medical Imaging*, Beijing, 2010.

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

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

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

[29] N. Duchateau, M. De Craene, G. Piella, and A. F. Frangi, "Characterizing pathological deviations from normality using constrained manifold learning," in *Medical Image Computing and Computer Assisted Intervention*, 2011.

[30] M. Everingham, L. van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal on Computer Vision*, vol. 88, pp. 303–338, June 2010.

[31] G. Fanelli and J. Gall, "Real time head pose estimation with random regression forests," in *Institute of Electrical and Electronics Engineers Computer Vision and Pattern Recognition*, 2011.

[32] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the Association for Computing Machinery*, vol. 24, pp. 381–395, 1981.

[33] Y. Freund, S. Dasgupta, M. Kabra, and N. Verma, "Learning the structure of manifolds using random projections," in *Neural Information Processing Systems*, 2007.

[34] 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*, vol. 55, no. 1, 1997.

[35] J. Gall and V. Lempitsky, "Class-specific Hough forest for object detection," in *Institute of Electrical and Electronics Engineers Computer Vision and Pattern Recognition*, Miami, 2009.

[36] S. Gerber, T. Tasdizen, S. Joshi, and R. Whitaker, "On the manifold structure of the space of brain images," in *Medical Image Computing and Computer Assisted Intervention*, 2009.

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

[38] P. Geurts, "Extremely randomized trees," in *Machine Learning*, 2003.

[39] R. Girshick, J. Shotton, P. Kohli, A. Criminisi, and A. Fitzgibbon, "Efficient regression of general-activity human poses from depth images," in *Institute of Electrical and Electronics Engineers International Conference on Computer Vision*, 2011.

[40] R. Guerrero, R. Wolz, and D. Rueckert, "Laplacian eigenmaps manifold learning for landmark localization in brain MR images," in *Medical Image Computing and Computer Assisted Intervention*, 2011.

[41] S. S. Gupta, "Probability integrals of multivariate normal and multivariate t," *Annals of Mathematical Statistics*, vol. 34, no. 3, 1963.

[42] J. Hamm, D. H. Ye, R. Verma, and C. Davatzikos, "GRAM: A framework for geodesic registration on anatomical manifolds," *Medical Image Analysis*, vol. 14, no. 5, 2010.

[43] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd Edition, 2003.

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

[45] D. Heath, S. Kasif, and S. Salzberg, "Induction of oblique decision trees," *Journal of Artificial Intelligence Research*, vol. 2, no. 2, pp. 1–32, 1993.

[46] C. Hegde, M. B. Wakin, and R. G. Baraniuk, "Random projections for manifold learning — proofs and analysis," in *Neural Information Processing Systems*, 2007.

[47] T. K. Ho, "Random decision forests," in *International Conference on Document Analysis and Recognition*, pp. 278–282, 1995.
[48] T. K. Ho, "The random subspace method for constructing decision forests," *Institute of Electrical and Electronics Engineers Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.
[49] http://research.microsoft.com/groups/vision/decisionforests.aspx.
[50] http://research.microsoft.com/projects/medicalimageanalysis/.
[51] T. Joachims, "Advances in kernel methods — support vector learning," chapter *Making Large-Scale SVM Learning Practical*, The MIT Press, 1999.
[52] I. T. Jolliffe, *Principal Component Analysis*. Springer-Verlag, 1986.
[53] 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 *Medical Image Computing and Computer Assisted Intervention*, 2011.
[54] P. Kontschieder, S. Rota Buló, H. Bischof, and M. Pelillo, "Structured class-labels in random forests for semantic image labelling," in *International Conference on Computer Vision*, Barcelona, Spain, 2011.
[55] 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 *Society of Photographic Instrumentation Engineers PIE Medical Imaging*, 2011.
[56] M. Kristan, D. Skocaj, and A. Leonardis, "Incremental learning with gaussian mixture models," in *Computer Vision Winter Workshop (CVWW)*, Moravske Toplice, Slovenia, 2008.
[57] C. H. Lampert, "Kernel methods in computer vision," *Foundations and Trends in Computer Graphics and Vision*, vol. 4, no. 3, 2008.
[58] C. Leistner, A. Saffari, J. Santner, and H. Bischoff, "Semi-supervised random forests," in *International Conference on Computer Vision*, 2009.
[59] 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.
[60] V. Lepetit and P. Fua, "Keypoint recognition using randomized trees," *Institute of Electrical and Electronics Engineers Transactions on Pattern Analysis and Machine Intelligence*, 2006.
[61] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of Berkeley Symposium on Mathematical Statistics and Probability*, 1967.
[62] T. Malisievicz, A. Gupta, and A. A. Efros, "Ensemble of exemplar-svms for object detection and beyond," in *International Conference on Computer Vision*, Barcelona, Spain, 2011.
[63] R. Maree, P. Geurts, J. Piater, and L. Wehenkel, "Random subwindows for robust image classification," in *Proceedings of Computer Vision and Pattern Recognition*, 2005.
[64] N. Meinshausen, "Node harvest," *The Annals of Applied Statistics*, vol. 4, no. 4, 2010.

[65] B. Menze, B. M. Kelm, D. N. Splitthoff, U. Koethe, and F. A. Hamprecht, "On oblique random forests," in *Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Database*, 2011.
[66] Microsoft Corp. Redmond WA, Kinect for XBox 360.
[67] A. Montillo and H. Ling, "Age regression from faces using random forests," in *International Conference on Image Processing*, 2009.
[68] A. Montillo, J. Shotton, J. E. Winn, D. Metaxas, E. Iglesias, and A. Criminisi, "Entangled decision forests and their application for semantic segmentation of CT images," in *Information Processing in Medical Imaging (IPMI)*, 2011.
[69] F. Moosman, B. Triggs, and F. Jurie, "Fast discriminative visual codebooks using randomized clustering forests," in *Proceedings of Neural Information Processing Systems*, 2006.
[70] B. Nadler, S. Lafon, R. R. Coifman, and I. G. Kevrekidis, "Diffusion maps, spectral clustering and eigenfunctions of Fokker-Plank operators," in *Proceedings of Neural Information Processing Systems*, 2005.
[71] R. Navaratnam, A. W. Fitzgibbon, and R. Cipolla, "The joint manifold model for semi-supervised multi-valued regression," in *International Conference on Computer Vision*, 2007.
[72] R. M. Neal, "Annealed importance sampling," *Statistics and Computing*, vol. 11, pp. 125–139, 2001.
[73] S. Nowozin and C. H. Lampert, "Structured learning and prediction in computer vision," *Foundations and Trends in Computer Graphics and Vision*, vol. 6, no. 3, 4, 2010.
[74] S. Nowozin, C. Rother, S. Bagon, T. Sharp, B. Yao, and P. Kohli, "Decision tree fields," in *International Conference on Computer Vision*, 2011.
[75] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua, "Fast keypoint recognition using random ferns," *Institute of Electrical and Electronics Engineers Transactions on Pattern Aanalysi and Machine Intelligence*, vol. 32, no. 3, 2010.
[76] E. Parzen, "On estimation of a probability density function and mode," *Annals of Mathematical Statistics*, vol. 33, pp. 1065–1076, 1962.
[77] 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 *Medical Image Computing and Computer Assisted Intervention*, Toronto, 2011.
[78] O. Pauly, D. Mateus, and N. Navab, "STARS: A new ensemble partitioning approach," in *International Conference on Computer Vision Workshop on Information Theory In Computer Vision and Pattern Recognition*, 2011.
[79] N. Payet and S. Todorovic, "$(rf)^2$ random forest random field," in *Neural Information Processing Systems*, 2010.
[80] R. L. Plackett, "A reduction formlula for normal multivariate integrals," *Biometrika*, vol. 41, 1954.
[81] J. R. Quinlan, *C4.5: Programs for Machine Learning*. 1993.
[82] C. E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[83] G. Rogez, J. Rihan, S. Ramalingam, P. Orrite, and C. Torr, "Randomized trees for human pose detection," in *Computer Vision and Pattern Recognition*, 2008.
[84] C. Rosenberg, M. Hebert, and H. Schneiderman, "Semi-supervised self-training of object detection models," in *Institute of Electrical and Electronics Engineers Workshop on Applications of Computer Vision*, 2005.
[85] M. R. Sabuncu and K. V. Leemput, "The relevance voxel machine (RVoxM): A bayesian method for image-based prediction," in *Medical Image Computing and Computer Assisted Intervention*, 2011.
[86] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischoff, "On-line random forests," in *International Conference on Computer Vision workshop on On-Line learning for Computer Vision*, 2009.
[87] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990.
[88] G. A. F. Seber and C. J. Wild, *Non Linear Regression*. New York: John Wiley and Sons, 1989.
[89] T. Sharp, "Implementing decision trees and forests on a GPU," in *European Conference on Computer Vision*, 2008.
[90] J. Shi and J. Malik, "Normalized cuts and image segmentation," in *Proceedings of Computer Vision and Pattern Recognition (Computer Vision and Pattern Recognition)*, Washington, DC, USA, 1997.
[91] 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 *Institute of Electrical and Electronics Engineers Computer Vision and Pattern Recognition*, 2011.
[92] J. Shotton, M. Johnson, and R. Cipolla, "Semantic texton forests for image categorization and segmentation," in *Institute of Electrical and Electronics Engineers Computer Vision and Pattern Recognition*, 2008.
[93] B. W. Silverman, *Density Estimation*. London: Chapman and Hall, 1986.
[94] J. Skilling, "Bayesian inference and maximum entropy methods in science and engineering," in *American Institute of Physics*, 2004.
[95] A. J. Smola and B. Scholkopf, "A tutorial on support vector regression," Technical Report, 1998.
[96] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf, "Large scale multiple kernel learning," *Journal of Machine Learning Research*, vol. 7, July 2006.
[97] 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.
[98] G. Stewart and J. Sun, *Matrix Perturbation Theory*. Elsevier, 1990.
[99] G. J. Szekely and M. L. Rizzo, "Testing for equal distributions in high dimensions," *Interstat*, 2004.
[100] M. Szummer and T. Jaakkola, "Partially labelled classification with markov random walks," in *Neural Information Processing Systems*, 2001.
[101] J. B. Tenenbaum, V. deSilva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, 2000.

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

[103] A. Torralba, K. P. Murphy, and W. T. Freeman, "Sharing visual features for multiclass and multiview object detection," *Institute of Electrical and Electronics Engineers Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[104] Z. Tu, "Probabilistic boosting-tree: Learning discriminative models for classification, recognition, and clustering," in *Proceedings of the Institute of Electrical and Electronics Engineers International Conference on Computer Vision (International Conference on Computer Vision)*, pp. 1589–1596, 2005.

[105] Z. Tu, "Auto-context and its application to high-level vision tasks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

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

[107] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Conference Vision*, 2004.

[108] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," in *International Conference on Computer Vision*, 2003.

[109] J. Wang, "On transductive support vector machines," in *Prediction and Discovery*, American Mathematical Society, 2007.

[110] P. Yin, A. Criminisi, I. Essa, and J. Winn, "Tree-based classifiers for bilayer video segmentation," in *Computer Vision and Pattern Recognition*, 2007.

[111] Q. Zhang, R. Souvenir, and R. Pless, "On manifold structure of cardiac MRI data: Application to segmentation," in *Institute of Electrical and Electronics Engineers Computer Vision and Pattern Recognition*, Los Alamitos, CA, USA, 2006.

[112] X. Zhu and A. Goldberg, "Introduction to semi-supervised learning," Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers, 2009.

[113] A. Zien and C. S. Ong, "Multiclass multiple kernel learning," in *International Conference on Machine Learning*, 2007.