Randomised Decision Forests for Classification

Tae-Kyun (T-K) Kim

Senior Lecturer

https://labicvl.github.io/

Further reading:

Breiman L, Random Forests. Machine Learning, 45 (1), pp 5-32, 2001.

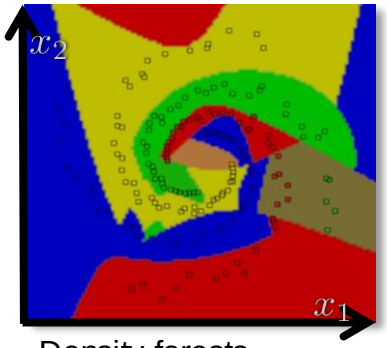
A brief history

- Decision Forest (DF), also known as Random Forest (RF) or randomised trees, was originally proposed by [Breiman, 2001] and extensively studied in the field of Computer Vision.
- Conceptually, a DF is based on the concept of decision tree [Quinlan, 1986], whereby at test time each internal tree node routes data to its left or right child-node by applying a threshold to a projection of the input features.
- Each input ultimately ends up at a leaf node , where a prediction function is stored during training and applied during testing.
- On top of that, [Breiman, 2001] combined the idea of randomised feature selection and bagging [Breiman, 1996] with an ensemble of decision trees and termed it as random forest (decision forest).

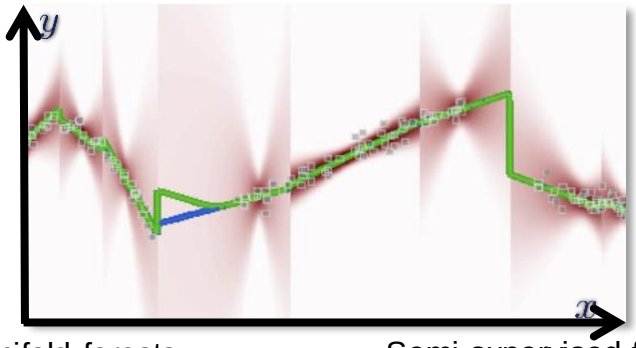
A brief history

- DF for classification and regression were first introduced in [Breiman, 2001].
- Over the years, many variants have been proposed, including clustering forest [Schölkopf et al. 2006], manifold forest [Bonde et al. 2010], semi-supervised forest [Leistner et al. 2009], etc.

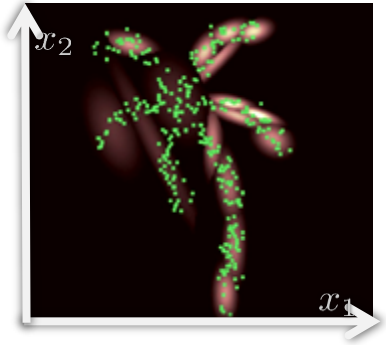
Classification forests



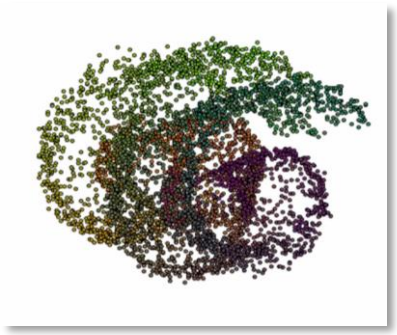
Regression forests



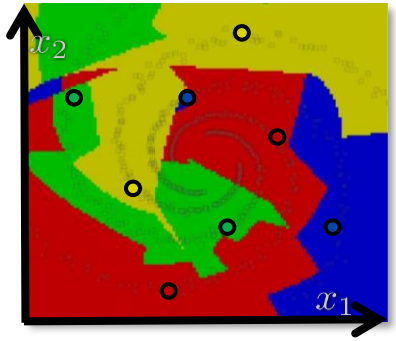
Density forests



Manifold forests



Semi-supervised forests

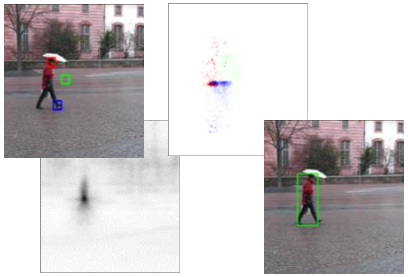


Animations from Criminisi et al.

A brief history: applications in computer vision

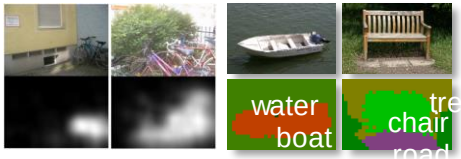
- Recent years have seen an explosion of forest-based techniques in the field.
- Hough forest for joint classification and regression was proposed [Gall and Lempitsky, 2009].
- Random decision forests were used for visual vocabulary and categorisation in [Mossmann et al. 2006, Shotton et al. 2008], and for keypoint tracking in videos in [V. Lepetit et al. 2006].
- Decision forests compare favourably with respect to other techniques [R. Caruana et al. 2008] and have led to a great success in body pose estimation with a depth camera [Shotton et al. 2011].
- DF has been extended to hand pose estimation [Tang et al. 2013].

Hough Forest



[Gall *et al.*, 09]

Visual Words



[Moosmann *et al.*, 06]
[Shotton *et al.*, 08]

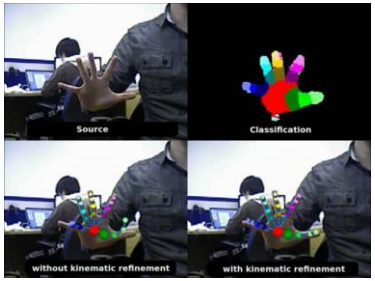
Keypoint Recognition

[Lepetit *et al.*, 06]

Body Pose

[Shotton *et al.*, 11]

Hand Pose



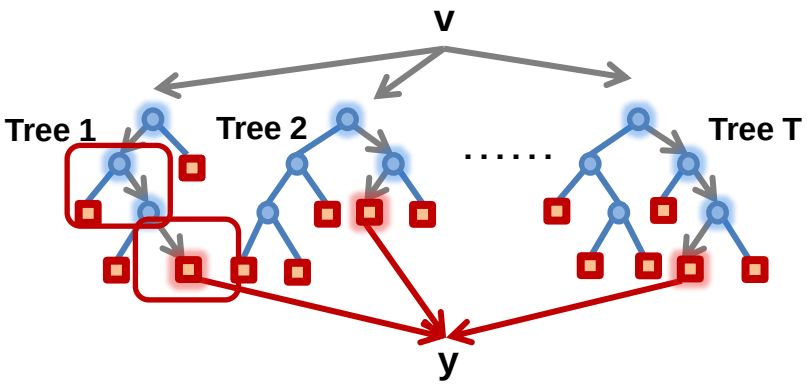
[Tang *et al.*, 13]

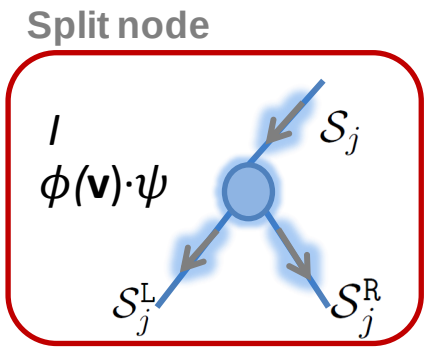
Randomised decision forests

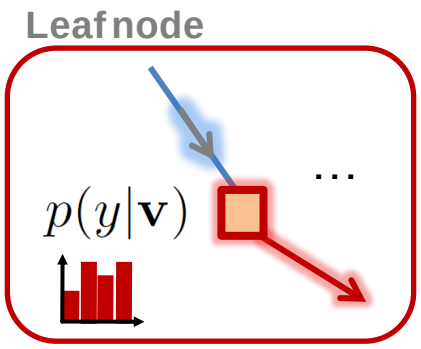
Randomised Forests is an ensemble of bagged tree learners with randomized feature selection.

Variants usually modify one or several of the following components of RF:

- Node objective function *I*, a criterion for selecting the best node weak learner (split feature, threshold)
- Split feature function, $\phi(\mathbf{v})\cdot\psi$, for splitting data **v**.
- Predictor, a function that is used for predicting output **y**.
- Tree structure, which inherently has a hierarchical architecture.







Why decision forest

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 additional problems e.g. regression (and codebook).

DF has quite a few desired properties as a classifier over other classifiers.

- Top-tier performance of DF has been empirically proved among modern classifiers.
  - In the comparison study by [Caruana et al. 2008, Delgado et al., 2014], ensemble classifiers outperform all other families, in both 2-class and multiclass cases.
  - A DF-based method achieves the best accuracy among all 179 classifiers.
- Good generalisation performance on new unseen samples is conveniently offered by the injected randomisation without parameter regularisation, preventing overfitting.

Why decision forest

- Multi-class/multi-variate problems are inherently supported by the tree structure of DF.
  - Compared to binary classifiers like SVM or adaboost, DF does not require a one-vs.-one or one-vs.-rest scheme for multi-class problems, hence more efficient.
- High dimensional data can be efficiently handled without extra computation. The node-splitting scheme can efficiently handle high dimensional input data, without a dimension reduction preprocessing step.
- DF can nicely scale up to a large training set. A large dataset is mandatory for a high dimensional parameter space coverage. Its recursive node partitioning, or node-wise optimisation (a small data set at a node) is by far more time-efficient than a global optimiser.
- A real-time solution is achieved. At the testing time, a input $\boldsymbol{v}$ is given by a sequential decision-making on the traversal of a binary tree. Each node test is very simple, often axis-aligned or two-pixel tests.

Notations

Input data vector: $\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$

Output/label: y

Subset of training points reaching node j: $\mathcal{S}_j$

Subset of points going to the left child node: $\mathcal{S}_j^{\mathrm{L}}$

Subset of points going to the right child node: $\mathcal{S}_j^{\mathrm{R}}$, s.t. $\mathcal{S}_j = \mathcal{S}_j^{\mathrm{L}} \cup \mathcal{S}_j^{\mathrm{R}}$

Node test parameters: $\theta = (\phi, \psi, \tau)$, $\boldsymbol{\theta} \in \mathcal{T}$

Split feature function: $\phi \cdot \psi$

Threshold: τ

Node weak learner: $h(\mathbf{v}, \boldsymbol{\theta}_j) \in \{\mathtt{true}, \mathtt{false}\}$

Node objective function: $I = I(\mathcal{S}_j, \boldsymbol{\theta})$

Stopping criteria: e.g. max tree depth = *D*

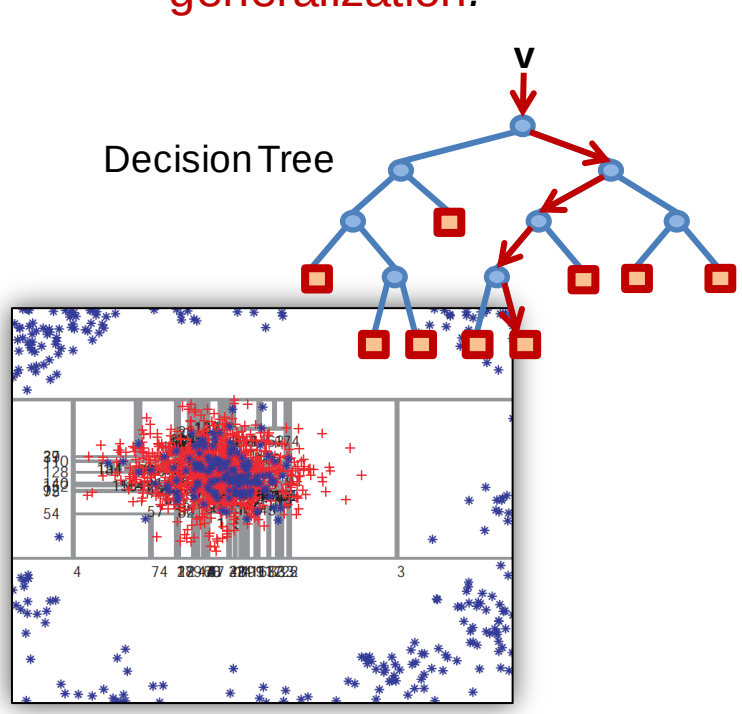
Leaf predictor model: $p(y|\mathbf{v})$

Forest size i.e. number of trees: T

Ensemble model: $p(y|\mathbf{v}) = \frac{1}{T} \sum_t^T p_t(y|\mathbf{v})$

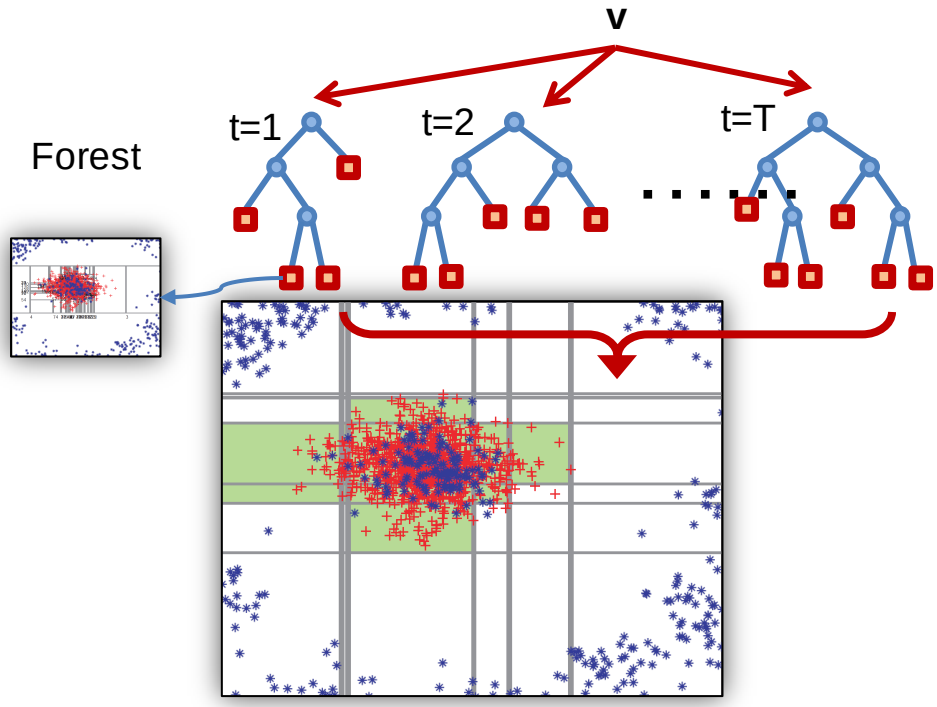
Decision tree

- Decision trees is a classic technique. It lacks a principled way of tree pruning, it tends to highly overfit to training data.
- Their recent revival is mostly thanks to the success of Decision forests: ensembles of slightly different trees with feature randomisation tend to produce higher accuracy on previously unseen data, i.e. good generalization.



Overfit (axis-aligned weaklearners, 2 class problem)

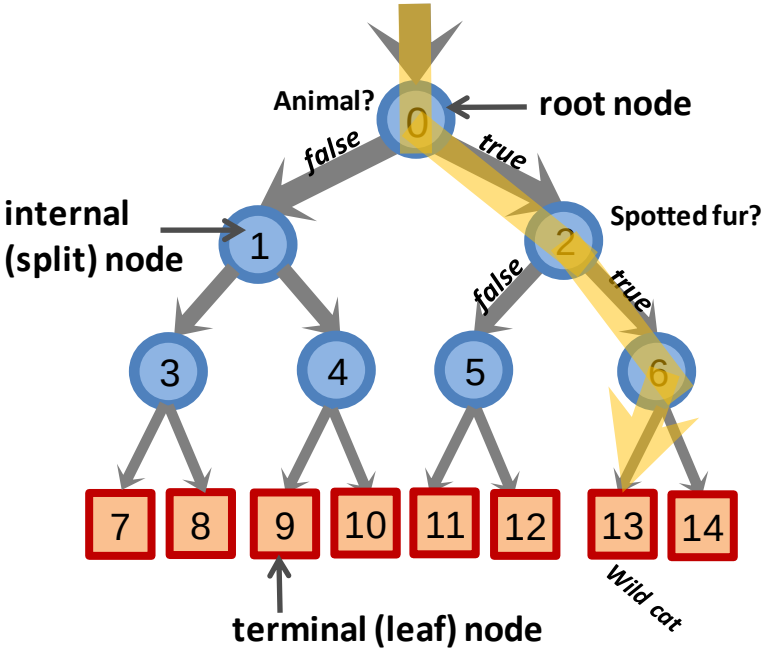
VS



Generalised, smooth decision regions (axis-aligned weaklearners, 2 class problem)

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.
- 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.



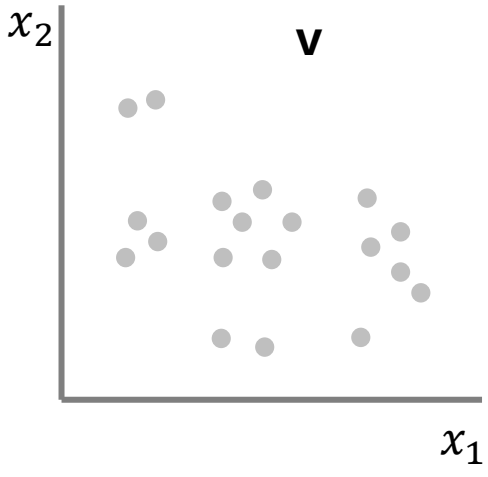
Decision tree - data point and features

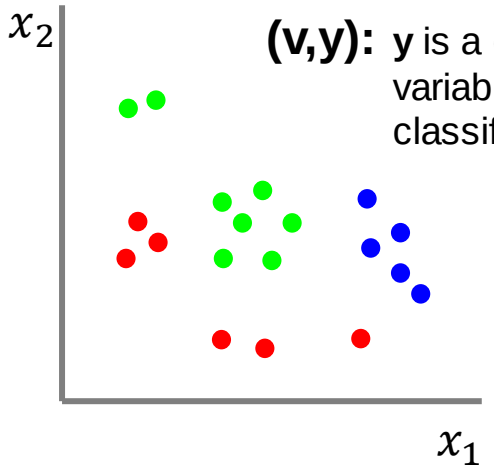
– A generic object, called data point, is denoted by a vector $\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$ where the components x_i represent some attributes of the data point, called *features*.

– The number of features naturally depends on the type of the data point as well as the application.

– The dimensionality of the feature space *d* can be very large. In practice, it is often not possible, and further not necessary, to extract all *d* dimensions of **v** ahead of time.

– In a supervised task (classification, regression), a training point is a pair (**v**,**y**) where **y** represents a generic, known label.

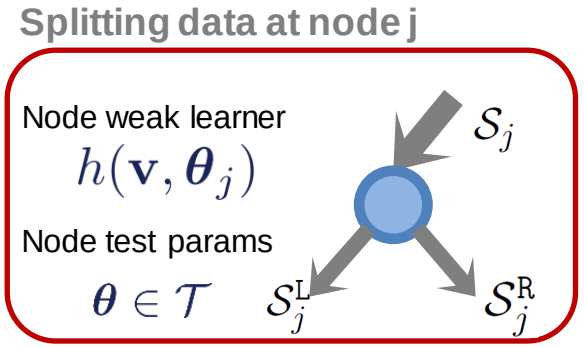




(v,y): **y** is a categorical variable c in classification problems.

Decision tree - weak learners

- A decision tree is a set of tests that are hierarchically organized.
- We use the terms "split function," "test function," and "weak learner" interchangeably.
- At each internal node of a decision tree, a binary split function *h* is required to route the data to its left or right child node.



- Each node has associated a different test function.
- We formulate a test function at a split node *j* as a function with binary outputs

$$h(\mathbf{v},\theta_j) = \{0 \text{ or} 1\}$$

where 0 and 1 can be interpreted as "false" and "true" respectively,

- $\theta_j \in \mathcal{T}$ denote the parameters of the test function at the *j*-th split node.
- The data point **v** arriving at the split node is sent to its left or right child node according to the result of the test function.

Decision tree - weak learner models

- Now the question left is the choice of *h*.
- The split function plays a crucial role both in training and testing.
- Different types of split functions have been extensively discussed: linear, non-linear, axis-aligned, and two-pixel test.
- We provide a simple geometric parametrization and a few derived formulations.
- We formulate the parametrization of the weak learner model as $\theta = (\phi, \psi, \tau)$.
- The filter function ϕ selects some features of choice out of the entire vector **v**.
- ψ defines the geometric primitive used to separate the data (e.g., an axis-aligned hyperplane, an oblique hyperplane, a general surface etc.)
- The parameter τ is a threshold for the inequalities used in the binary test.

Node weak learner: linear

– Linear function is formulated as

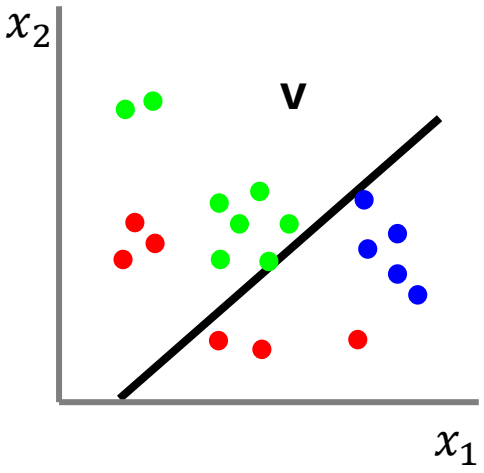
$$h(\mathbf{v}, \boldsymbol{\theta}) = [\tau_1 > \boldsymbol{\phi}(\mathbf{v}) \cdot \boldsymbol{\psi} > \tau_2]$$

where [ ] is the indicator function.

– In the 2D example,

$$\boldsymbol{\phi}(\mathbf{v}) = (x_1 \; x_2 \; 1)^\top$$

with $\boldsymbol{\psi} \in \mathbb{R}^3$ a generic line in homogeneous coordinates.



– Setting $\tau_1 = \infty$ or $\tau_2 = -\infty$ corresponds to using a single-inequality test function.

– Parameters of ψ, τ are decided during training.

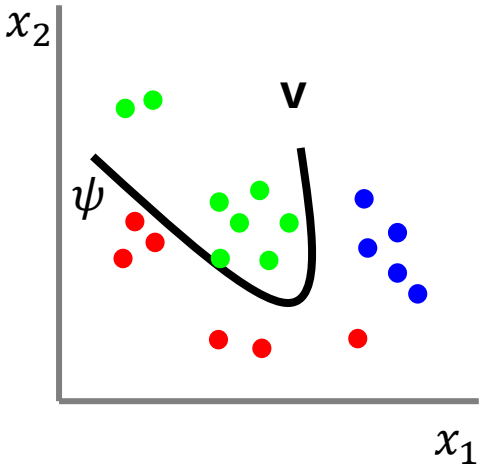
– By comparing with τ linearly separates the data into left and right child nodes.

Node weak learner: non-linear

- If not linearly separable, a non-linear feature function defined below can be used,
- In 2D case,

$$h(\mathbf{v}, \boldsymbol{\theta}) = \left[\tau_1 > \boldsymbol{\phi}^\top(\mathbf{v})\, \boldsymbol{\psi}\, \boldsymbol{\phi}(\mathbf{v}) > \tau_2\right]$$
$$\boldsymbol{\phi}(\mathbf{v}) = (x_1\ x_2\ 1)^\top$$

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



- The conic section acts like a non-linear hyper-surface separating the input data.
- Nonlinear weak learner is computationally expensive.
- In practice, since **v** often needs to pass multiple tree nodes (feature functions), it is empirically proved that a weak feature (axis-aligned, two-pixel tests) has already work well.

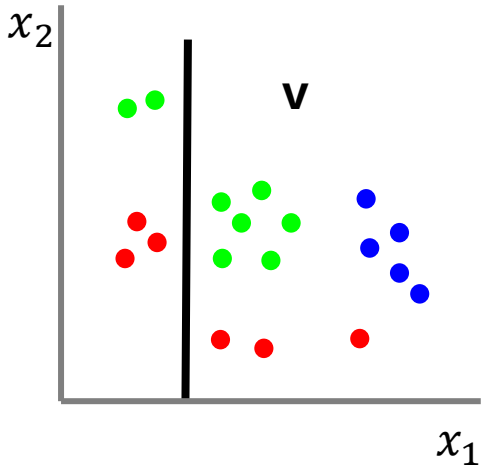
Node weak learner: axis-aligned

- A special case of linear weak learner model is one where the line ψ is aligned with one of the axes of the feature space.
- Axis-aligned feature only selects one dimension of the input $\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$
- Geometrically it looks like a hyperplane parallel to all axes except for the chosen one, hence the name.
- More formally, we define

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

- In the 2D example, $\boldsymbol{\phi}(\mathbf{v}) = (x_1 \; x_2 \; 1)^\top$

$$\boldsymbol{\psi} = (1 \; 0 \; \psi_3) \text{ or } \boldsymbol{\psi} = (0 \; 1 \; \psi_3)$$



- Or simply,

$h(\mathbf{v},\theta) = [x_i > \tau]$, for any $i \in \{1, \ldots, d\}$

- Such axis-aligned weak learners are most often used in the vision literature, since their computational cost are negligible like a look-up table.
- Using axis-aligned features highly accelerates classification time, often leading to a real-time solution.

Node weak learner: two-pixel test

- Another special case of linear weak learner model widely adopted in vision literature is two-pixel test.
- It is a linear feature that is slightly more complex than the axis-aligned one.
- Given an input feature vector $\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$

$$h(\mathbf{v},\theta) = [x_i - x_j > \tau] \qquad \text{for any } i, j \in \{1, \ldots, d\} \; and \; i \neq j$$

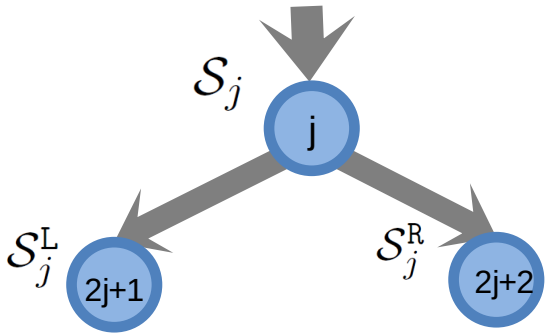
- Not only does this feature use one more dimension than the axis-aligned one, the different signs of the two terms is actually a discrete differentiation operator, which is an approximation of the gradient.

Training sets

- In a supervised task (classification, regression) a training point is a pair (**v**,**y**), where **v** is the input feature vector and **y** here represents a generic, known label.
- In an unsupervised task the training points are represented only by their features 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 S_1 denotes the subset of training points reaching node 1 (nodes are numbered in breadth-first order, starting from 0 for the root), and S_1^{L}, S_1^{R} denote the subsets going to the left and to the right children of node 1, respectively.
- In binary trees the following properties apply:

$$\mathcal{S}_j = \mathcal{S}_j^{\mathrm{L}} \cup \mathcal{S}_j^{\mathrm{R}} \qquad \mathcal{S}_j^{\mathrm{L}} \cap \mathcal{S}_j^{\mathrm{R}} = \emptyset$$

$$\mathcal{S}_j^{\mathrm{L}} = \mathcal{S}_{2j+1} \qquad \mathcal{S}_j^{\mathrm{R}} = \mathcal{S}_{2j+2}$$



Tree training

- The split functions stored at the internal nodes are key for the functioning of the tree.
- The split functions are learned automatically, from exemplar data.
- Thus, the training phase decides the parameters of the test function $h(\mathbf{v}, \theta_j)$ associated with each split node (indexed by j) by optimizing a chosen objective function defined on an available training set.
- At each node j, depending on the subset of the incoming training set S_j, we learn the function that "best" splits S_j into S_j^{L}, S_j^{R}.
- This problem is formulated as the maximization of an objective function at that node

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

where

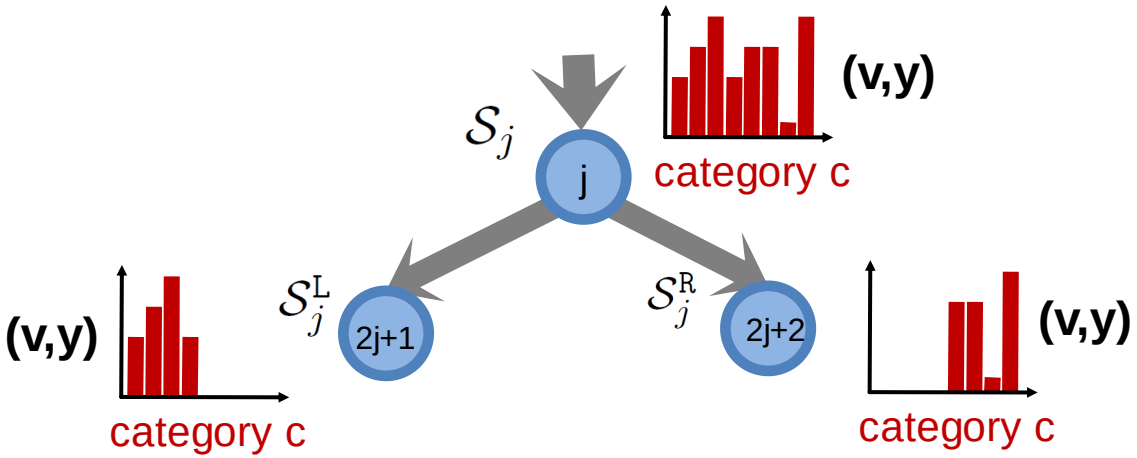
$$I_j = I(\mathcal{S}_j, \mathcal{S}_j^{\mathrm{L}}, \mathcal{S}_j^{\mathrm{R}}, \boldsymbol{\theta}_j)$$

$$\mathcal{S}_j^{\mathrm{L}} = \{(\mathbf{v}, \mathbf{y}) \in \mathcal{S}_j | h(\mathbf{v}, \boldsymbol{\theta}_j) = 0\}$$

$$\mathcal{S}_j^{\mathrm{R}} = \{(\mathbf{v}, \mathbf{y}) \in \mathcal{S}_j | h(\mathbf{v}, \boldsymbol{\theta}_j) = 1\}$$

Tree training

- The full set of all possible node test parameters is denoted by $\mathcal{T}$
- Node test parameters $\boldsymbol{\theta} \in \mathcal{T}$
- Try all node test parameters $\boldsymbol{\theta} \in \mathcal{T}$ and choose the best parameter that maximises the objective.



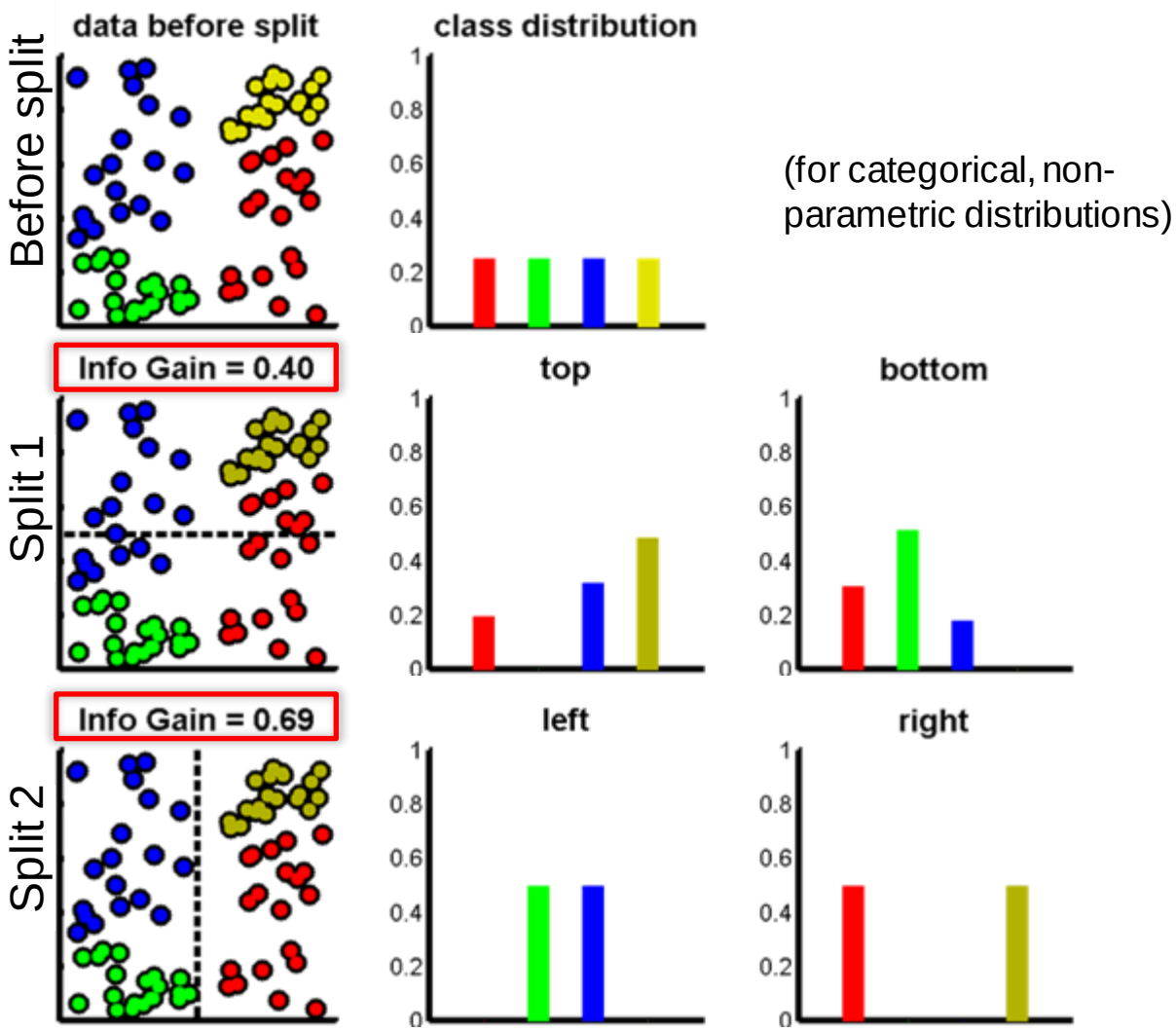
y is a categorical variable c in classification problems.

Energy models, objective function

- The energy model determines the prediction and estimation behaviour of a decision tree, through its influence on the choice of weak learners.
- The objective function used during training constructs decision trees that will perform the desired task.
- The result of the optimization problem determines the parameters of the weak learners, which in turn determines the path followed by a data and its prediction.

- A toy classification example is illustrated in the next slide. The graph shows a number of training points in a 2D space, where each coordinate denotes a feature value and the colors indicate the known classes.
- During training our aim is to learn the parameters that best split the training data.
- Our objective is to separate different classes as much as possible.

A toy classification example, information gain



(for categorical, non-parametric distributions)

Objective function, information gain

- For instance if we split the training data horizontally, 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 before split 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.

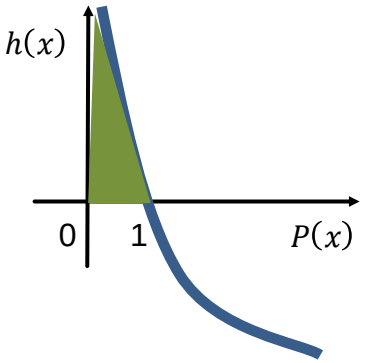
Entropy

- If two random variables x and y are unrelated, we define

$$h(x,y) = h(x) + h(y)$$

- As the joint probability $P(x,y) = P(x)P(y)$, we define

$$h(x) = -logP(x)$$

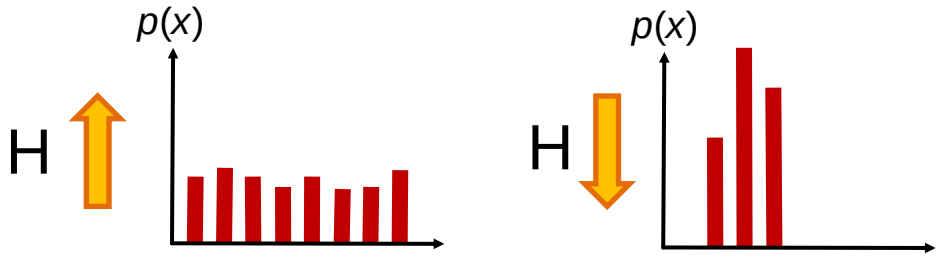


- The average amount of h(x), called entropy, on a discrete variable x is

$$H(x) = \sum_x P(x)h(x)$$

$$= -\sum_x P(x)\log P(x)$$

- The entropy is higher when the distribution is more uniform.

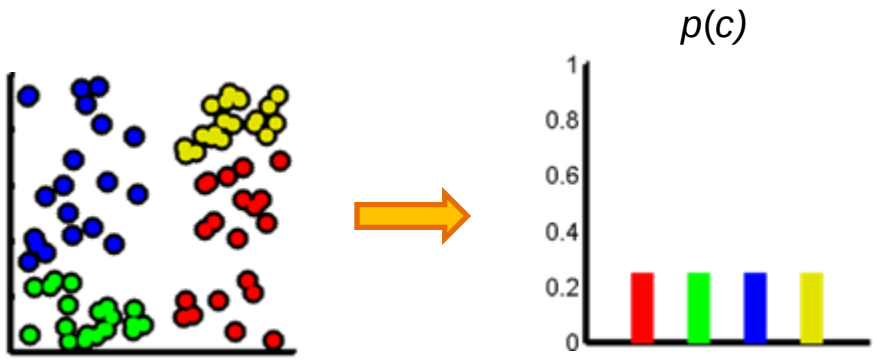


Objective function, information gain

– For discrete probability distributions, here *y* is a categorical variable c, we use the Shannon entropy, defined as

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

where *S* is the set of training points and the letter *c* indicates the class label. The set of all classes is denoted *C* and *p*(*c*) indicates the empirical distribution extracted from the training points within the set *S*.

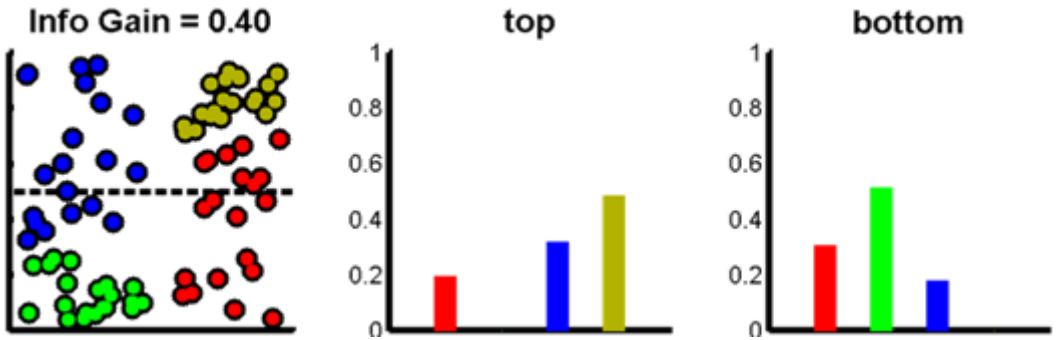


Objective function, information gain

- When applying an horizontal split in the toy example, 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

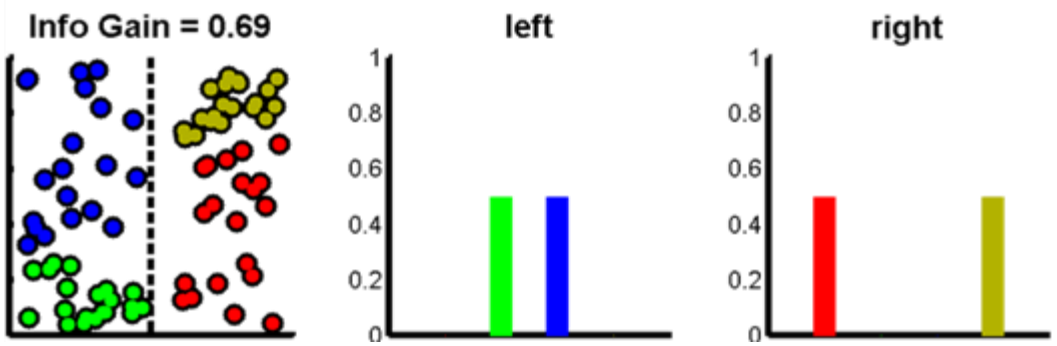
$$I(\mathcal{S}, \boldsymbol{\theta}) = H(\mathcal{S}) - \sum_{i \in \{\mathrm{L},\mathrm{R}\}} \frac{|\mathcal{S}^i|}{|\mathcal{S}|} H(\mathcal{S}^i)$$

- Note that in practice, since S is the same for all node test parameters, we can skip computing the entropy H(S).



A toy classification example, information gain

- The vertical split separates the training set even better, that is, the resulting children each contain only two colors/classes.
- This corresponds to even lower child entropies, and a higher information gain (I = 0.69).
- 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.



Stopping criteria

- Training starts at the root node, $j = 0$, where the optimum split parameters are found as described earlier.
- 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 tree structure (shape) 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 (or *depth*) D has been reached.
  - Alternatively, when the *information gain* is smaller than a predefined minimum value.
    - In other words we stop when the seeked-for attributes of the training points within the leaf node are similar to one another.
  - when a node contains *too few training* points.
- Avoiding growing full trees has been demonstrated to have positive effects in terms of generalization.

Leaf prediction models

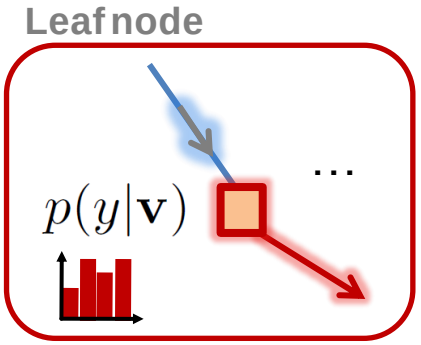
- During training, besides the tree structure and the weak learners, we also need to learn how to make predictions.
- After training, each leaf node remains associated with a subset of (labeled) training data.
- During testing, a previously unseen point traverses the tree until it reaches a leaf.
- Since 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.

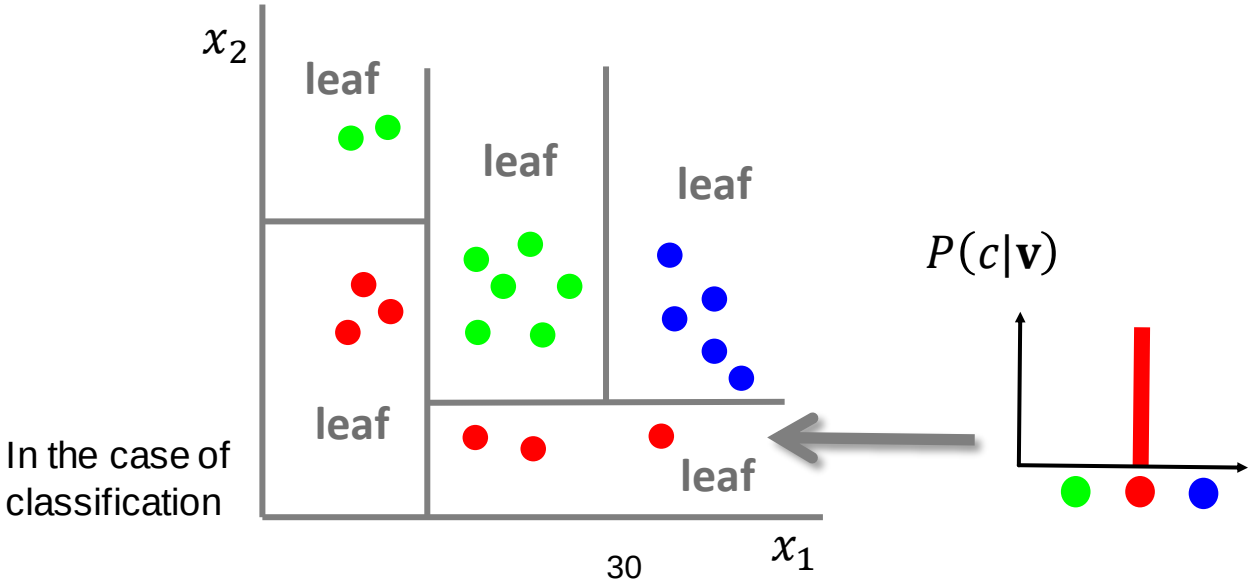
Leaf prediction models

– The leaf statistics can be captured using the posterior distributions

$$p(c|\mathbf{v}) \quad \text{or} \quad p(y|\mathbf{v})$$

where *c* and *y* represent the discrete or continuous labels, respectively. **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.



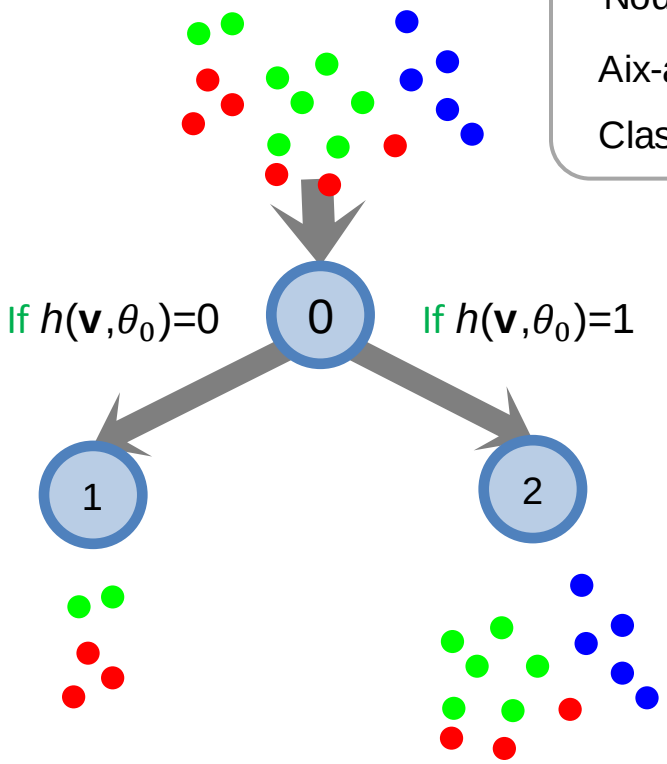


Learning a binary decision tree - recursive partitioning

– Given data vectors and their categorical labels,

$$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d \qquad c \in \mathcal{C} \qquad \text{with} \qquad \mathcal{C} = \{c_k\}$$

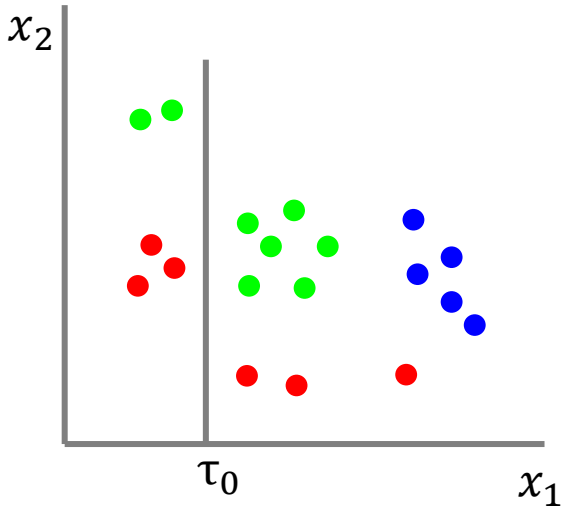
tree grows by successively partitioning the input data or space.



Node j Node test parameter θ_j

Aix-aligned weak learner $h(\mathbf{v},\theta_j) = [x_i > \tau_j]$, for any $i \in \{1, \ldots, d\}$

Class prob dist $p(c|\mathbf{v})$ Split nodes Leaf nodes



Learning a binary decision tree - recursive partitioning

– Given data vectors and their categorical labels,

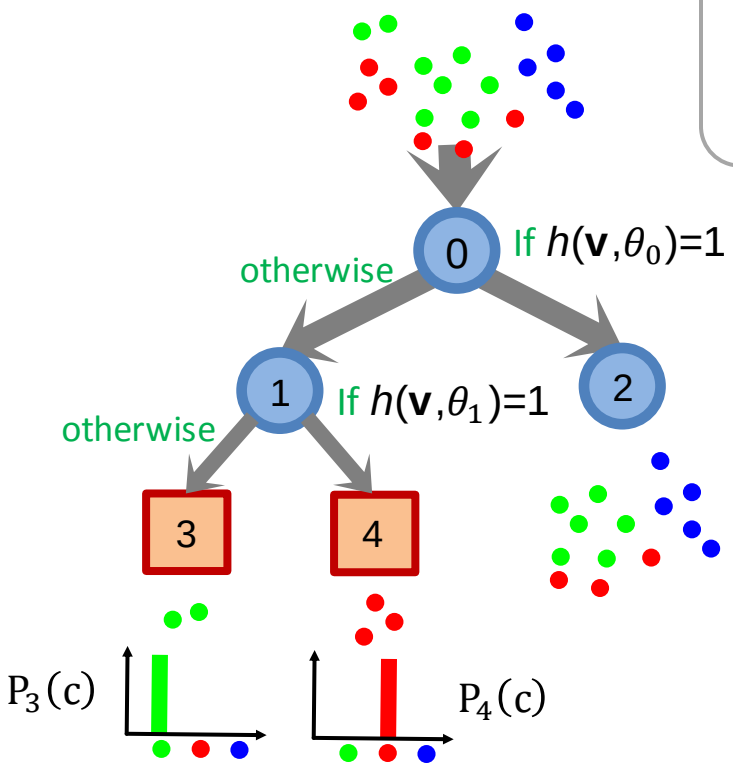
$$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d \qquad c \in \mathcal{C} \qquad \text{with} \qquad \mathcal{C} = \{c_k\}$$

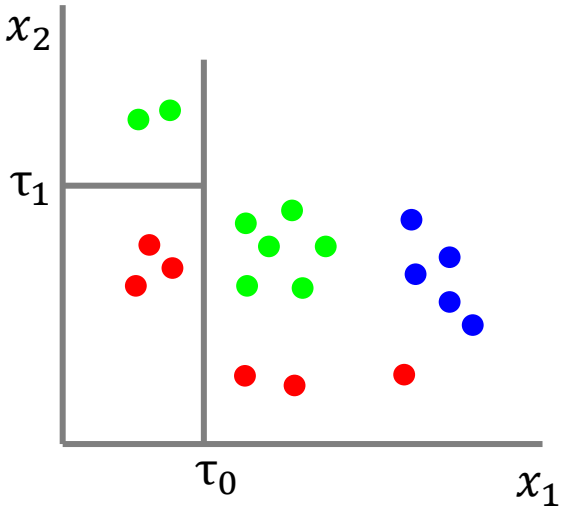
tree grows by successively partitioning the input data or space.

Node j Node test parameter θ_j

Aix-aligned weak learner $h(\mathbf{v},\theta_j) = [x_i > \tau_j]$, for any $i \in \{1, \ldots, d\}$

Class prob dist $p(c|\mathbf{v})$ Split nodes Leaf nodes





Learning a binary decision tree - recursive partitioning

– Given data vectors and their categorical labels,

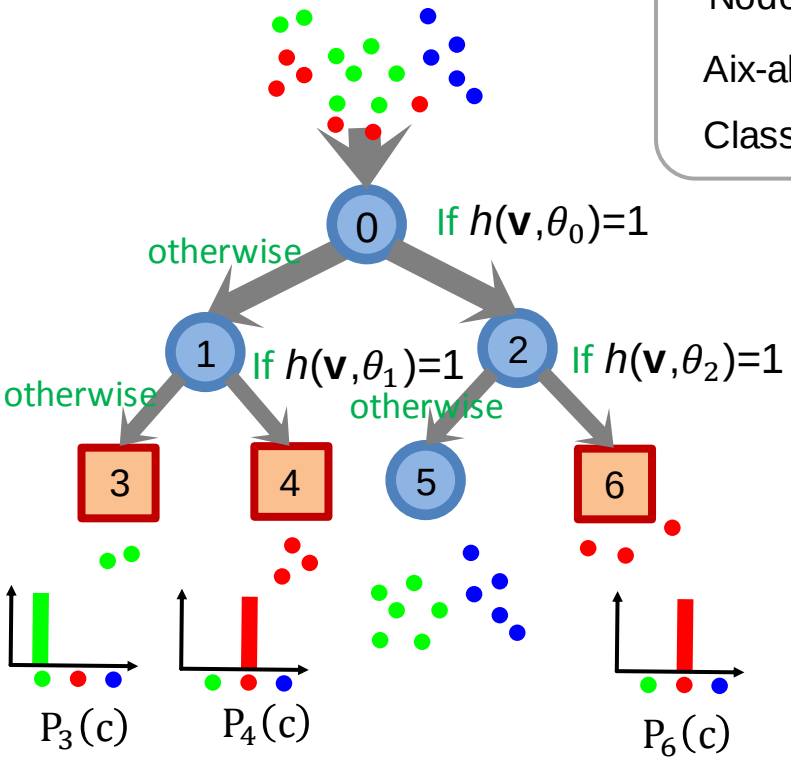
$$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d \qquad c \in \mathcal{C} \qquad \text{with} \qquad \mathcal{C} = \{c_k\}$$

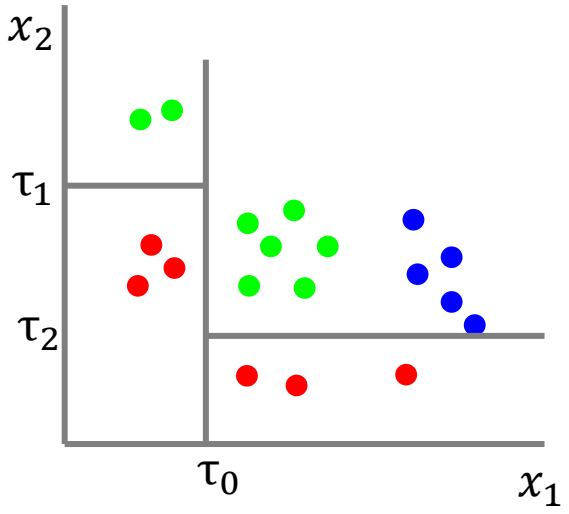
tree grows by successively partitioning the input data or space.

Node j Node test parameter θ_j

Aix-aligned weak learner $h(\mathbf{v},\theta_j) = [x_i > \tau_j]$, for any $i \in \{1, \ldots, d\}$

Class prob dist $p(c|\mathbf{v})$ Split nodes Leaf nodes





Learning a binary decision tree - recursive partitioning

– Given data vectors and their categorical labels,

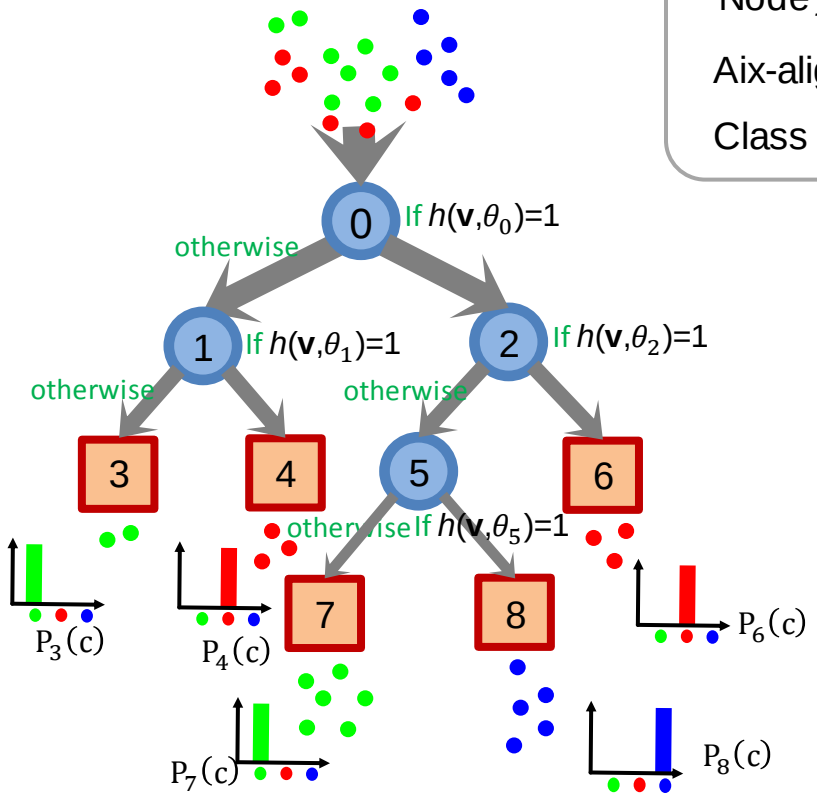
$$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d \qquad c \in \mathcal{C} \qquad \text{with} \qquad \mathcal{C} = \{c_k\}$$

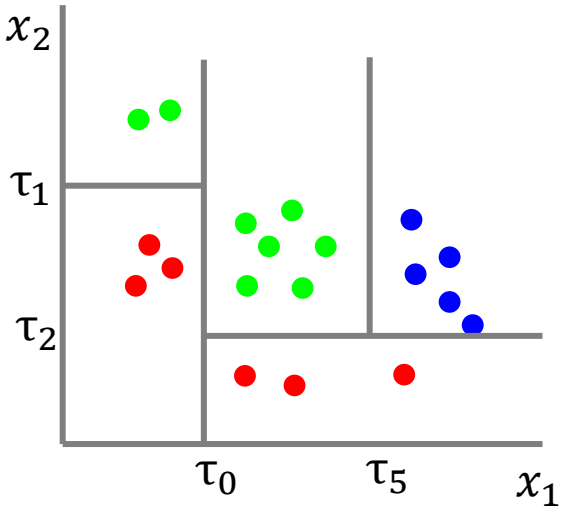
tree grows by successively partitioning the input data or space.

Node j Node test parameter θ_j

Aix-aligned weak learner $h(\mathbf{v},\theta_j) = [x_i > \tau_j]$, for any $i \in \{1, \ldots, d\}$

Class prob dist $p(c|\mathbf{v})$ Split nodes Leaf nodes



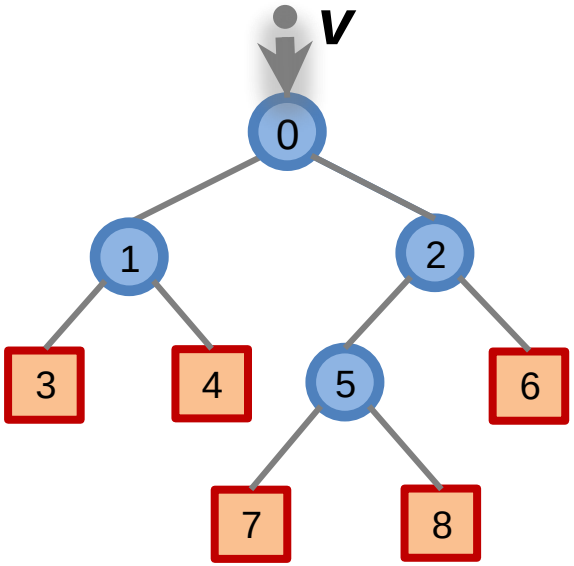


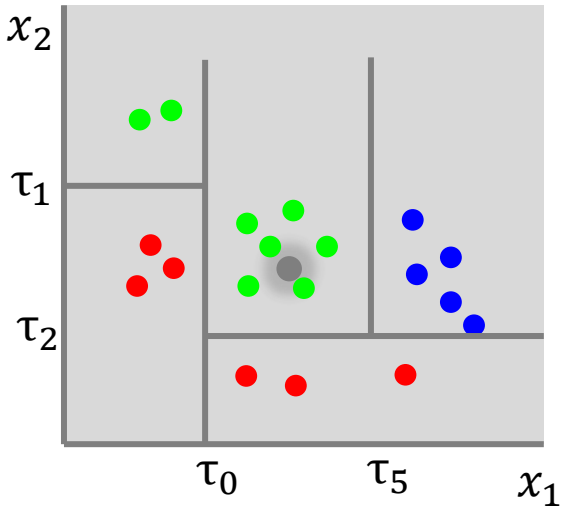
Tree testing

- Given a previously unseen data point **v,** a decision tree applies hierarchically a number of predefined tests.
- Starting at the root, each split node applies its associated test function $h(\mathbf{v},\theta_j)$.
- Depending on the result of this *binary* test the data is sent to the right or left child.
- 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 **v**.

Fast evaluation by a decision tree

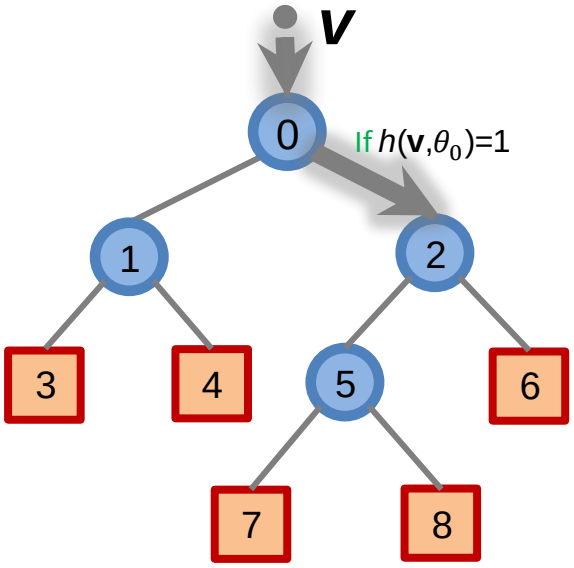
– A test input ***v*** is given by a sequential decision-making on the traversal of a binary tree.

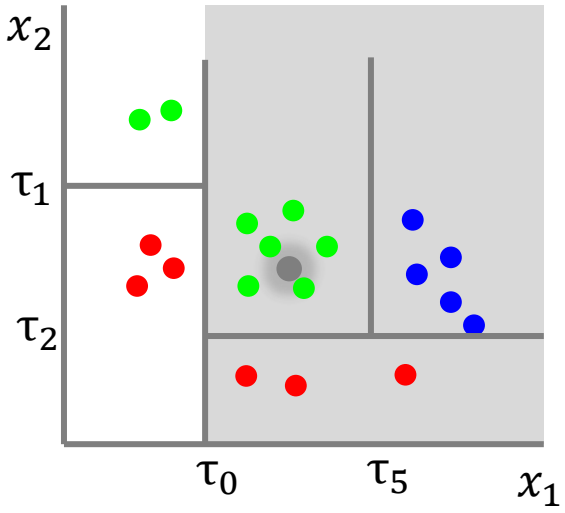




Fast evaluation by a decision tree

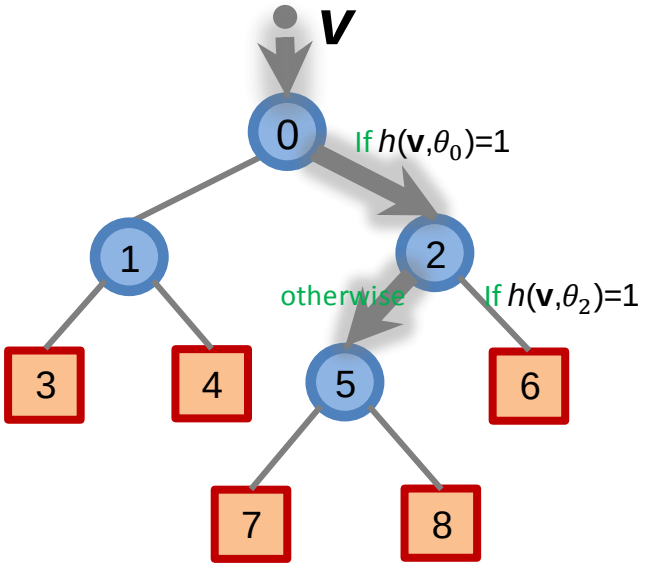
– A test input ***v*** is given by a sequential decision-making on the traversal of a binary tree.

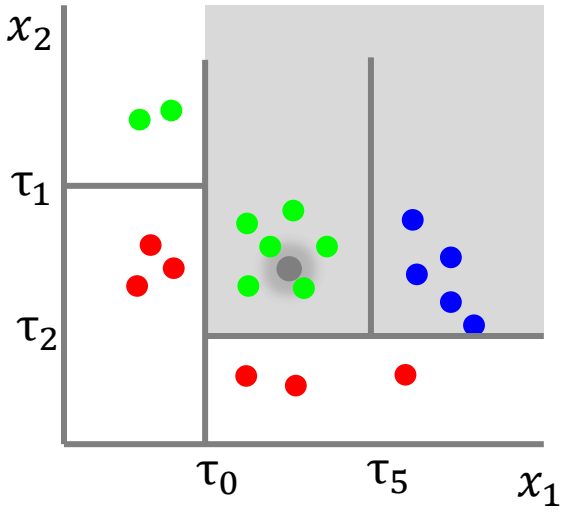




Fast evaluation by a decision tree

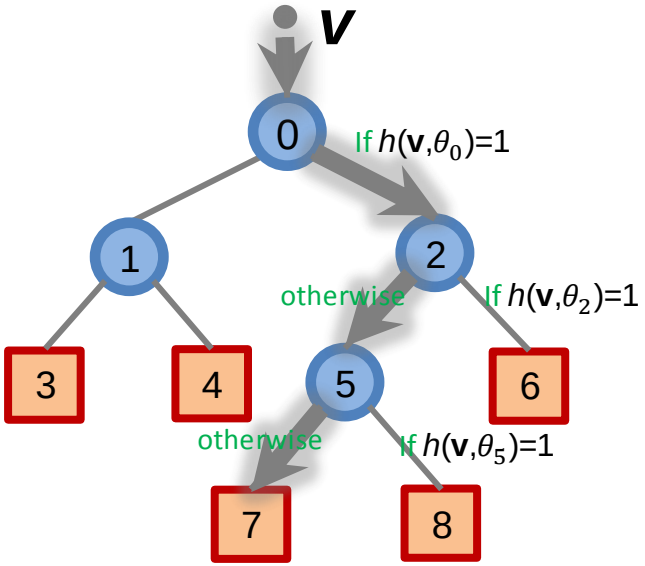
– A test input ***v*** is given by a sequential decision-making on the traversal of a binary tree.

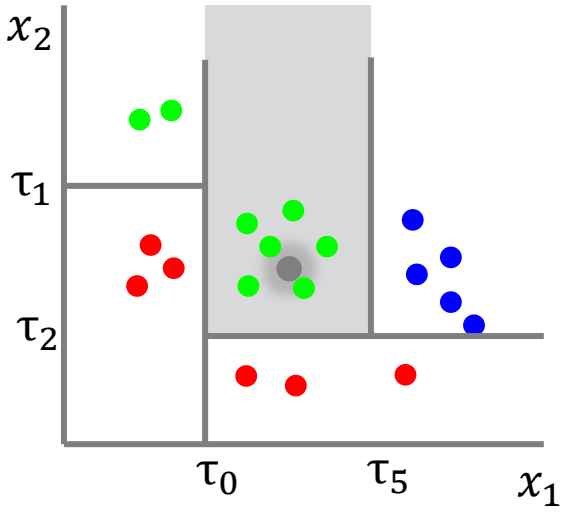




Fast evaluation by a decision tree

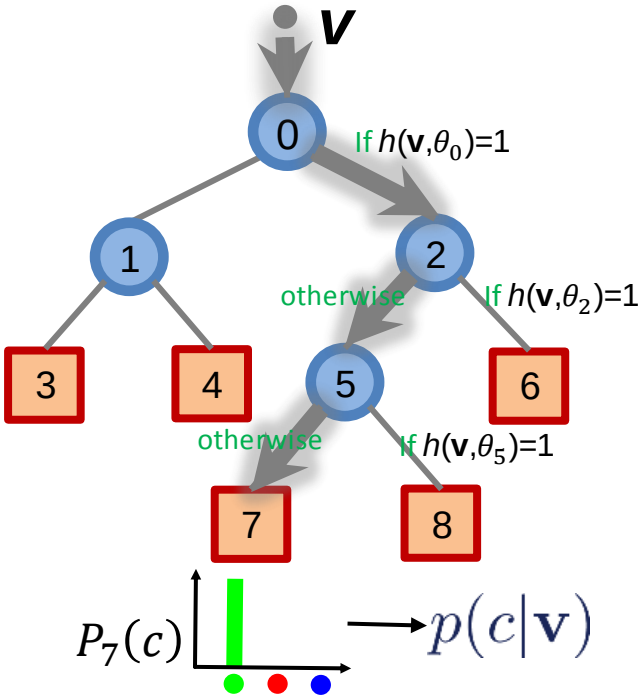
– A test input ***v*** is given by a sequential decision-making on the traversal of a binary tree.

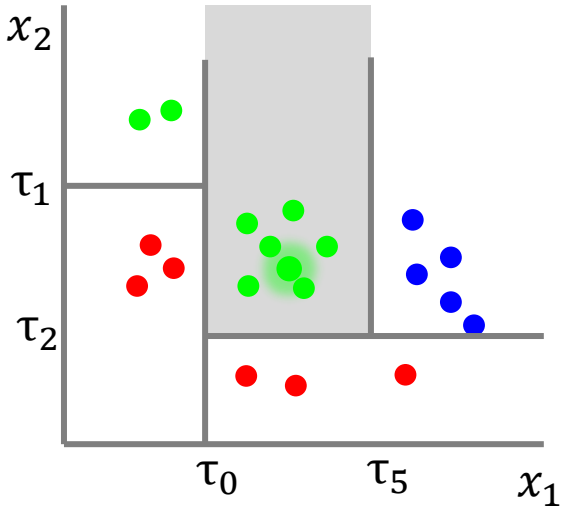




Fast evaluation by a decision tree

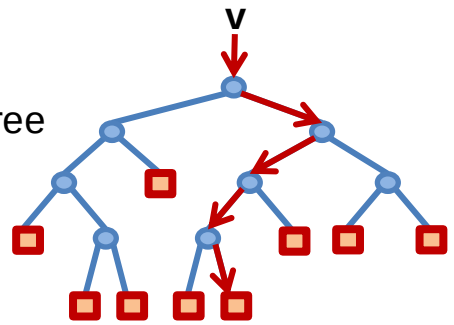
– A test input ***v*** is given by a sequential decision-making on the traversal of a binary tree.



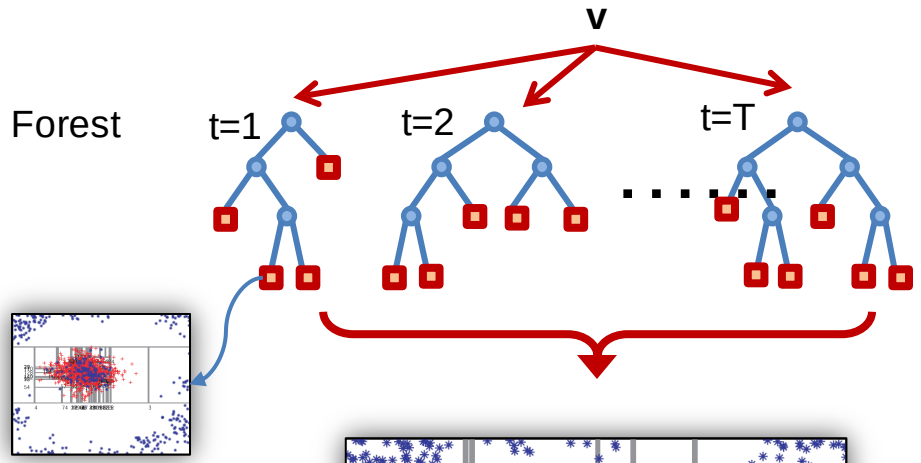


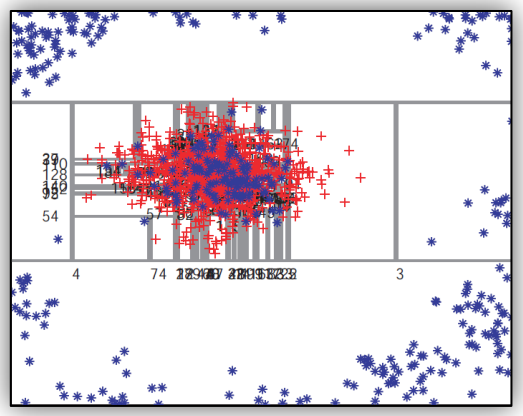
Overfitting

Decision Tree



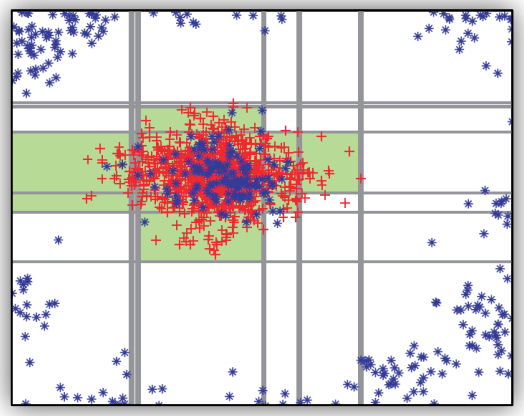
Forest





Overfit (axis-aligned weaklearners, 2 class problem)

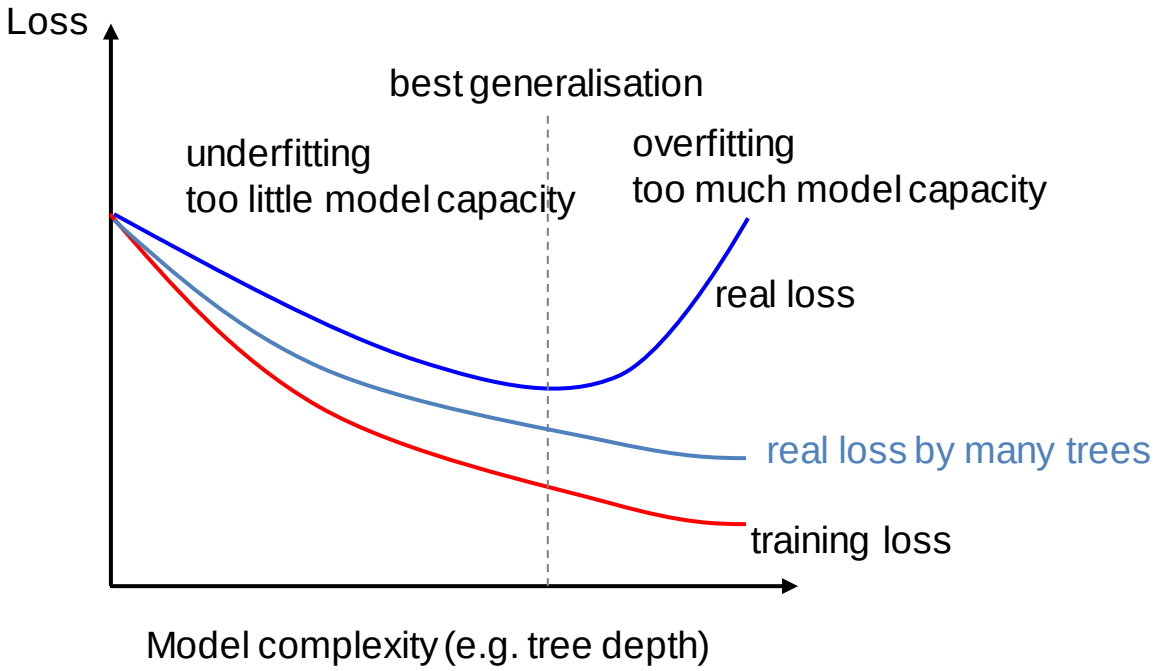
VS



Generalised, smooth decision regions (axis-aligned weaklearners, 2 class problem)

Overfitting and underfitting

- As complexity increases, the model overfits the data:
  - Training loss (classification error of training data) decreases
  - Real loss (classification error of testing data) increases
- We need to penalize model complexity = to regularize



Summary on decision tree

Pros

- It provides very fast evaluation.
- It is a highly scalable algorithm for training.

Cons

- It severely overfits to training data.
- There is no principled way of tee pruning.

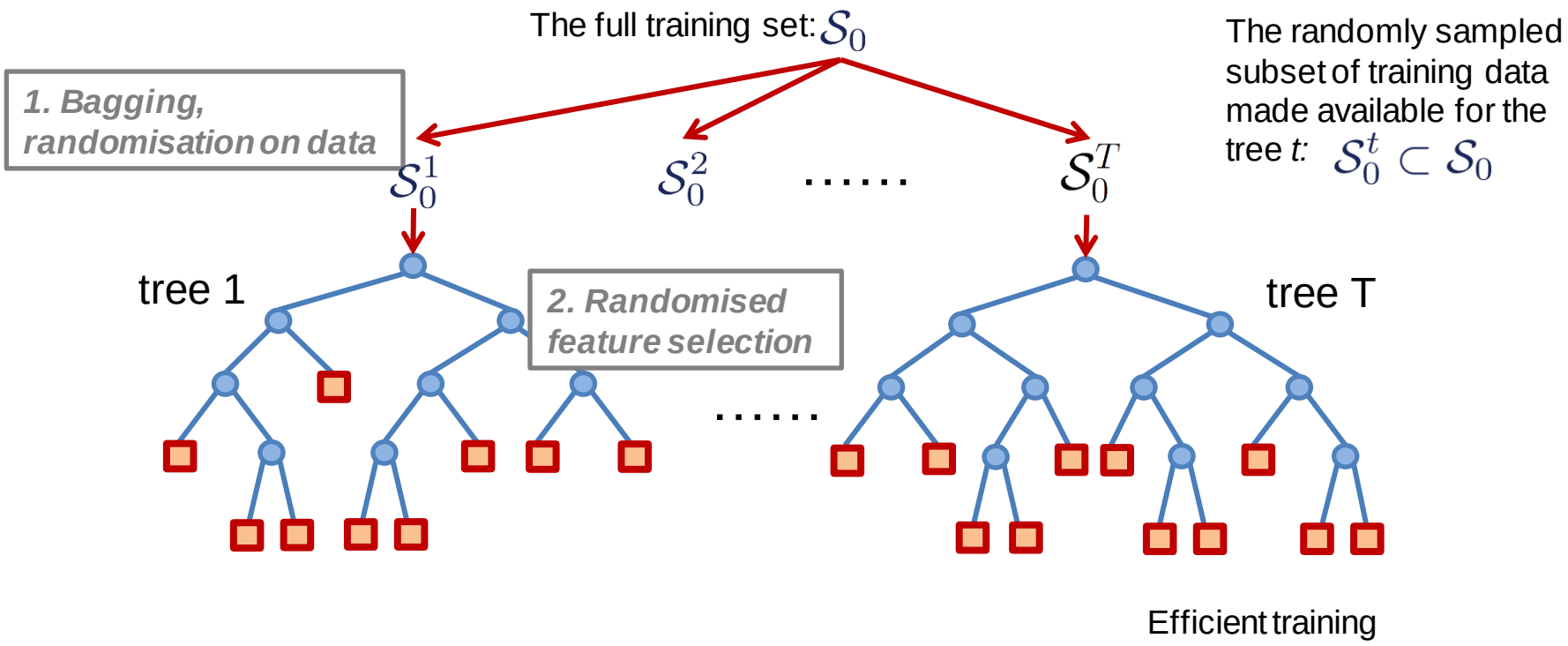
Ensembles of trees (decision forest)

- 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 additionally the type of randomness influence the prediction/estimation properties of the forests.

Randomness model

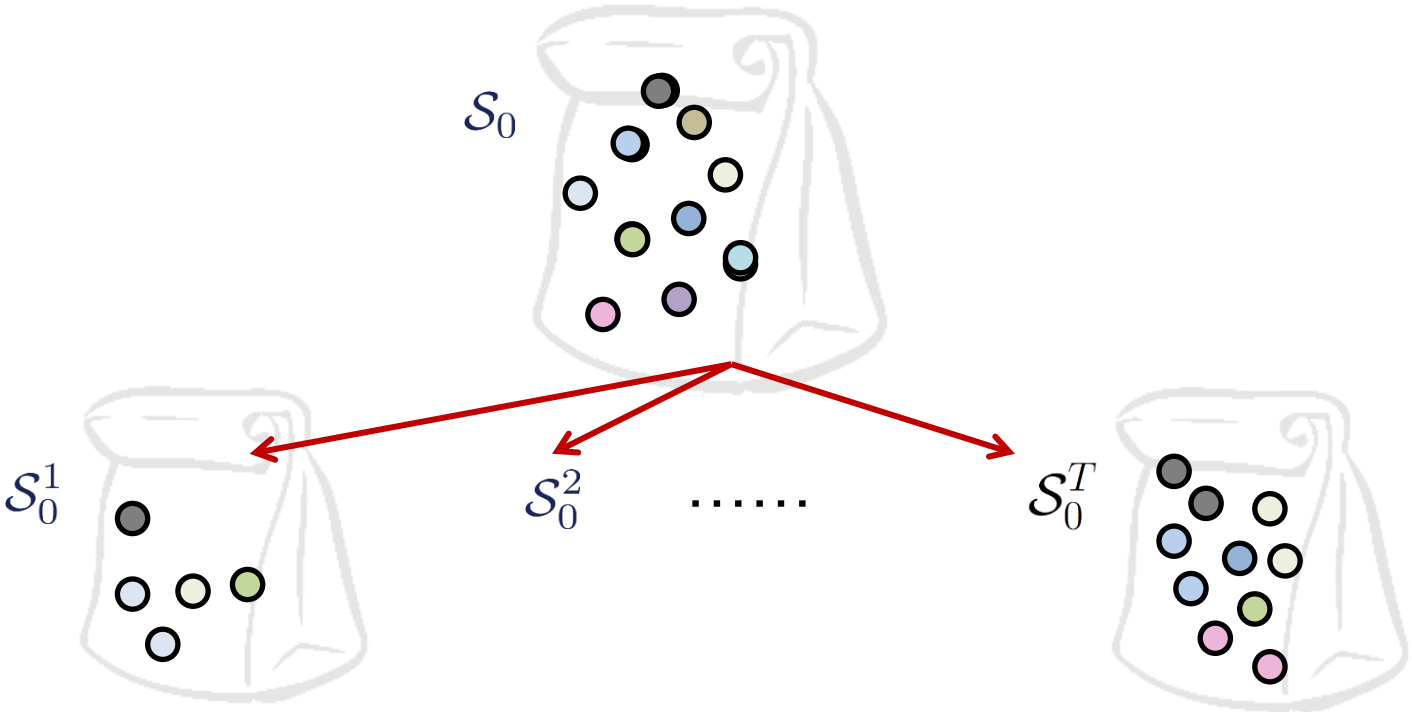
Randomness is injected into the trees during the training phase. Two techniques used together are:

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



Bagging (Bootstrap AGGregatING) - randomizing the training set

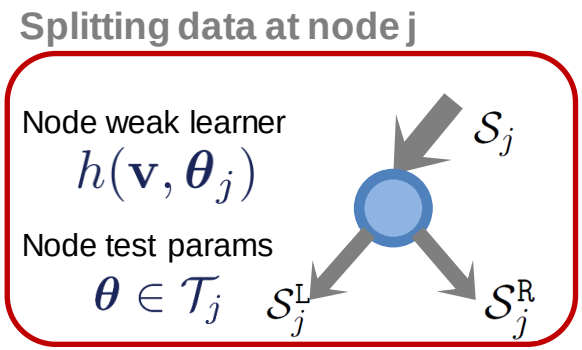
- Given a data set $\mathcal{S}_0$ of size n, it generates T data subsets $\mathcal{S}_0^t$, t=1,…,T.
- Each subset has e.g. n_t=n, by sampling data from $\mathcal{S}_0$ uniformly and with replacement.
- Some data are repeated in $\mathcal{S}_0^t$. If n_t=n and n is large, $\mathcal{S}_0^t$ is likely to have 63.2% of unique data.



Randomized node optimization

- In decision tree, we show that at each node the optimization is done with respect to the entire parameter space $\mathcal{T}$ i.e. $\boldsymbol{\theta} \in \mathcal{T}$.
- 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 of tree diversity).
- Instead, when training at the *j*-th 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

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



Randomized node optimization

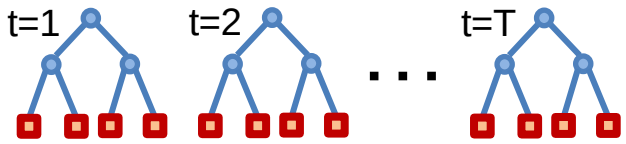
- The amount of randomness is controlled by the ratio $|\mathcal{T}_j|/|\mathcal{T}|$.
- Note that in some cases we may have $|\mathcal{T}| = \infty$.
- At this point it is convenient to introduce a parameter $\rho = |\mathcal{T}_j|$. The parameter $\rho = 1, \ldots, |\mathcal{T}|$ controls the degree 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 θ_j. Thus, there is no real optimization and we get maximum randomness.

Randomized node optimization

- 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.
- As illustrated, when $\rho = |\mathcal{T}|$ all the trees will be identical and as ρ decreases the trees become more decorrelated (different from one another).

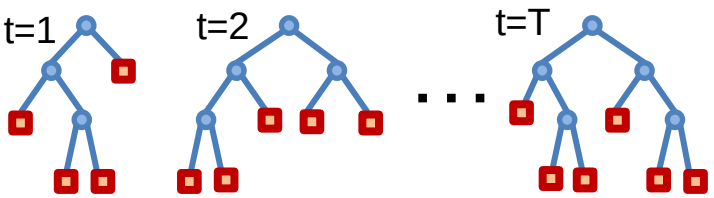
The effect of ρ

$\rho = |\mathcal{T}|$



Low randomness, high tree correlation

$\rho = 1$



High randomness, low tree correlation

Tree correlation vs strength

- Randomisation on data and feature increases diversity among trees.
- For the fixed depth, the randomised feature decreases strength of each tree.
- This compromising issue is further explained in the perspective of a generic committee machine.

Committee machine

– We consider multiple models or experts, $\mathrm{y}_t(x),\ t = 1, \ldots, T$.

– Output of each model is

$$\mathrm{y}_t(x) = h(x) + \epsilon_t(x)$$

where $h(x), \epsilon_t(x)$ are the true value and error of each model.

– The average sum-of-squares error is

$$\mathrm{E}[\{y_t(x) - h(x)\}^2] = \mathrm{E}[\epsilon_t(x)^2]$$

– The average error by acting individually is

$$\mathrm{E}_{av} = \frac{1}{T}\sum_{t=1}^{T} \mathrm{E}[\epsilon_t(x)^2]$$

Committee machine

– The committee machine is

$$y_{com}(x) = \frac{1}{T}\sum_{t=1}^{T} y_t(x)$$

– The expected error of the committee machine is

$$E_{com} = E\left[\left\{\frac{1}{T}\sum_{t=1}^{T} y_t(x) - h(x)\right\}^2\right]$$

$$= E\left[\left\{\frac{1}{T}\sum_{t=1}^{T} \epsilon_t(x)\right\}^2\right] = E[\frac{1}{T^2}(\epsilon_1^2 + \epsilon_1\epsilon_2 + \epsilon_2^2 + \cdots)]$$

Committee machine

– If we assume

$$E\left[\epsilon_i(x)\epsilon_j(x)\right] = 0,$$

$$\text{for any } i, j \in \{1, \ldots, T\} \; and \; i \neq j$$

then we obtain

$$E_{com} = \frac{1}{T} E_{av}$$

– In practice, the errors are typically highly correlated, but we can still expect that

$$E_{com} \leq E_{av}$$

Leaf prediction models and testing

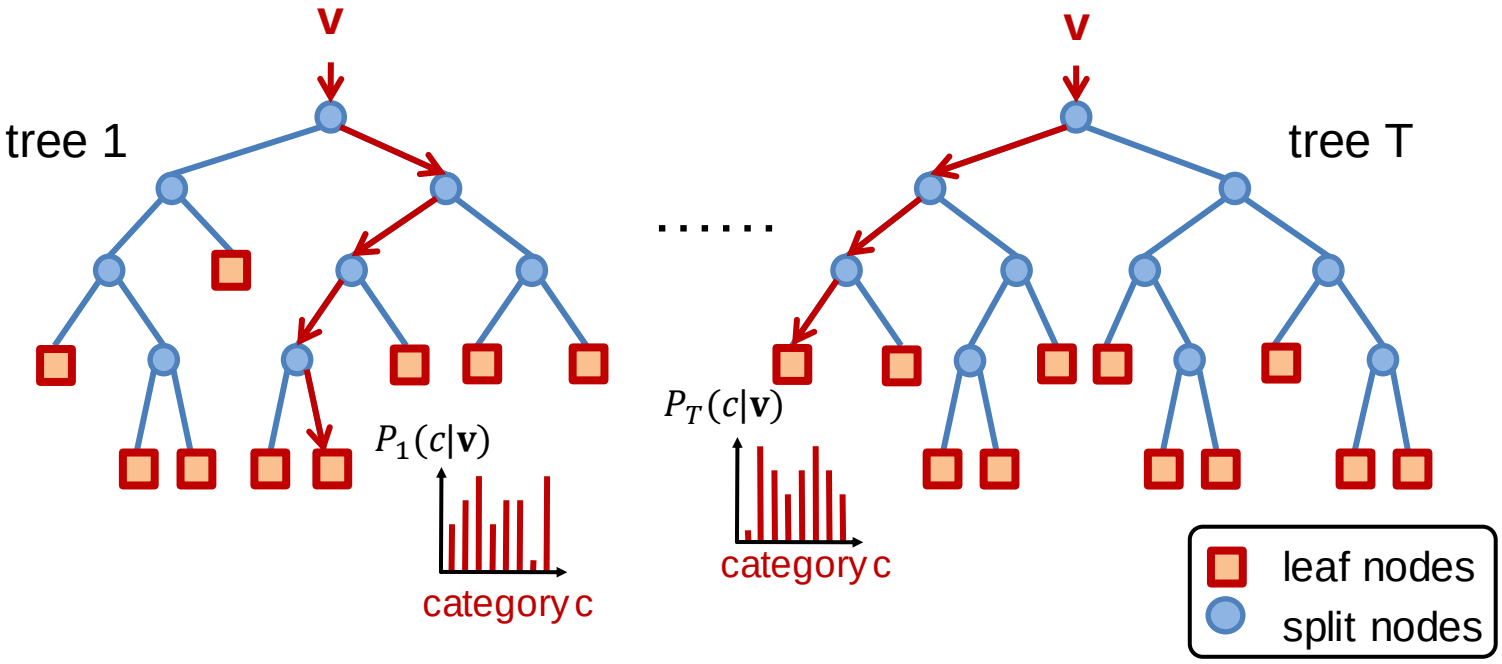
- In a forest with T trees we use the variable $t \in \{1, \ldots, T\}$ to index each component tree.
- All trees are trained independently (and possibly in parallel).
- During testing, each test point $\mathbf{v}$ is simultaneously pushed through all trees (starting at the root) until it reaches the corresponding leaves.
- Tree testing can also often be done in parallel, thus achieving high computational efficiency on modern parallel CPU or GPU hardware.
- Combining all tree predictions into a single forest prediction is done by a simple averaging operation. In classification

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

where $p_t(c|\mathbf{v})$ denotes the posterior distribution obtained by the t-th tree.

Forest of trees: evaluation

– A data point is passed down all trees, and the leaf nodes that the data point reaches are collected.



– Classification is done by $P(c|\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|\mathbf{v})$

Leaf prediction models and testing

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

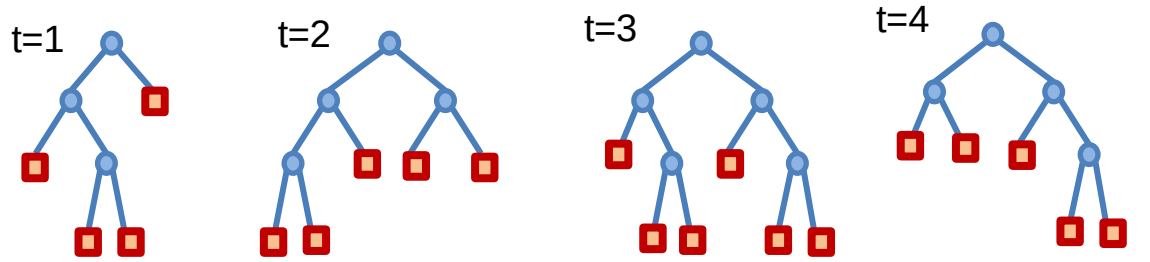
$$p(c|\mathbf{v}) = \frac{1}{Z}\prod_{t=1}^{T} p_t(c|\mathbf{v})$$

with *Z* ensuring probabilistic normalization.

Leaf prediction models and testing

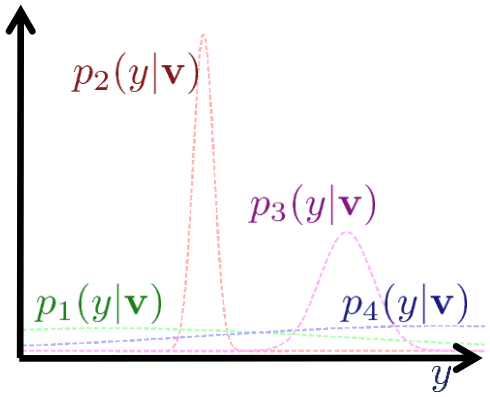
- Tree output fusion is illustrated in the next slide, for a simple example where the attribute we want to predict is the continuous variable y.
- Imagine that we have trained a forest with $T = 4$ trees.
- For a test data point **v,** we get the corresponding tree posteriors $p_t(y|\mathbf{v})$, with $t = \{1, \ldots, 4\}$.
- 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, or may do majority voting.

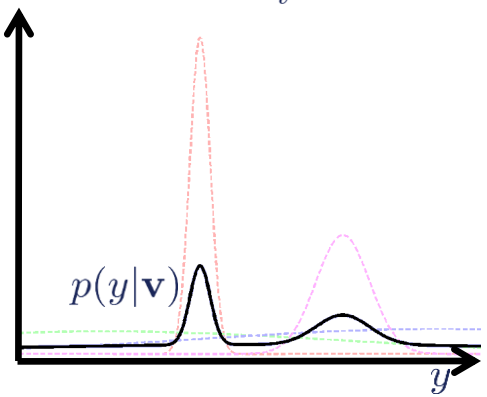
An example forest to predict continuous variables

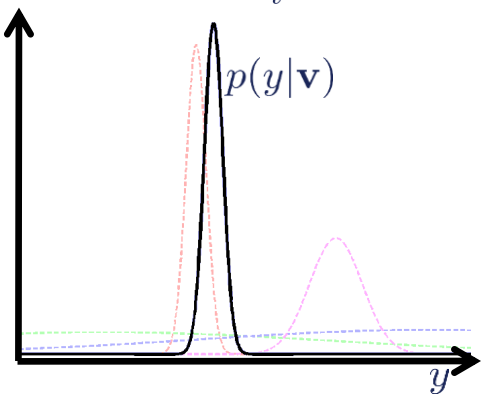


$$p(y|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(y|\mathbf{v})$$

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







Key model parameters

- The decision forests, their construction and prediction abilities depend on the model parameters. The parameters that most influence the behaviour 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

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

Key model parameters

- Several studies have pointed out how the testing accuracy increases monotonically with the forest size T, how learning very deep trees can lead to overfitting, as well as the importance of using very large amounts of training data.

- The choice of randomness model directly influences a classification forest's generalization. In the seminal work of Breiman, the importance of randomness and its effect on tree correlation has been shown.

- 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.

Classification forest summary

Classification forest enjoys a number of useful properties:

- It naturally handles problems with more than two classes
- It provides a probabilistic output
- It generalizes well to previously unseen data (by randomising training data and randomised node optimisation)
- It is efficient thanks to their parallelism and reduced set of tests per data point
- It achieves a highly scalable training, and often a real-time testing solution
- With a little modification, Forest can be applied to a series of different tasks like classification, regression or clustering

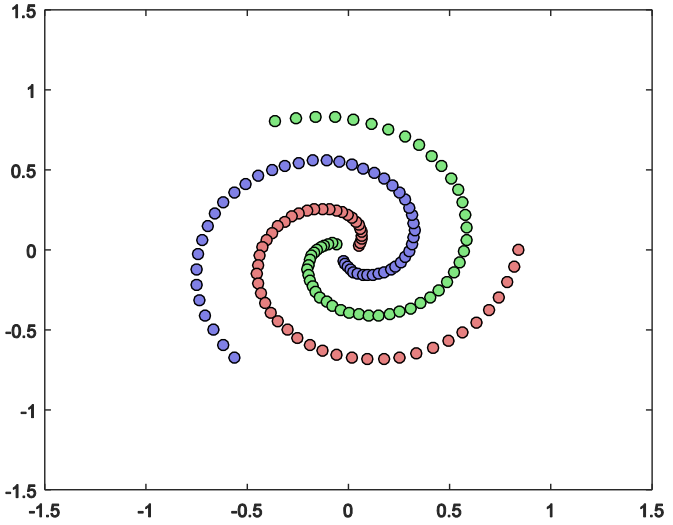
Disadvantages of RF include:

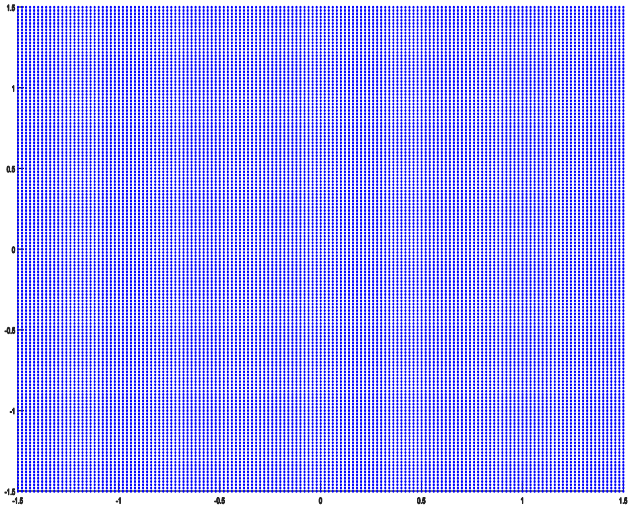
- It lacks of theoretical support, which is due to the difficulties in formulating the whole algorithm in an analytical form
- As a result, sometimes one can not avoid choosing the parameters (number of trees, maximum depth, and the degree of randomness in node optimisation) empirically

Classifying toy spiral data

Toy Spiral data

- Training
  - data_train(:,1:2) : [num_data x dim] Training 2D vectors
  - data_train(:,3) : [num_data x 1] Label of training data, {1,2,3}
- Testing
  - data_test(:,1:2) : [num_data x dim] Testing 2D vectors, 2D points in the uniform dense grid within the range of [-1.5, 1.5]
  - data_test(:,3) : N/A





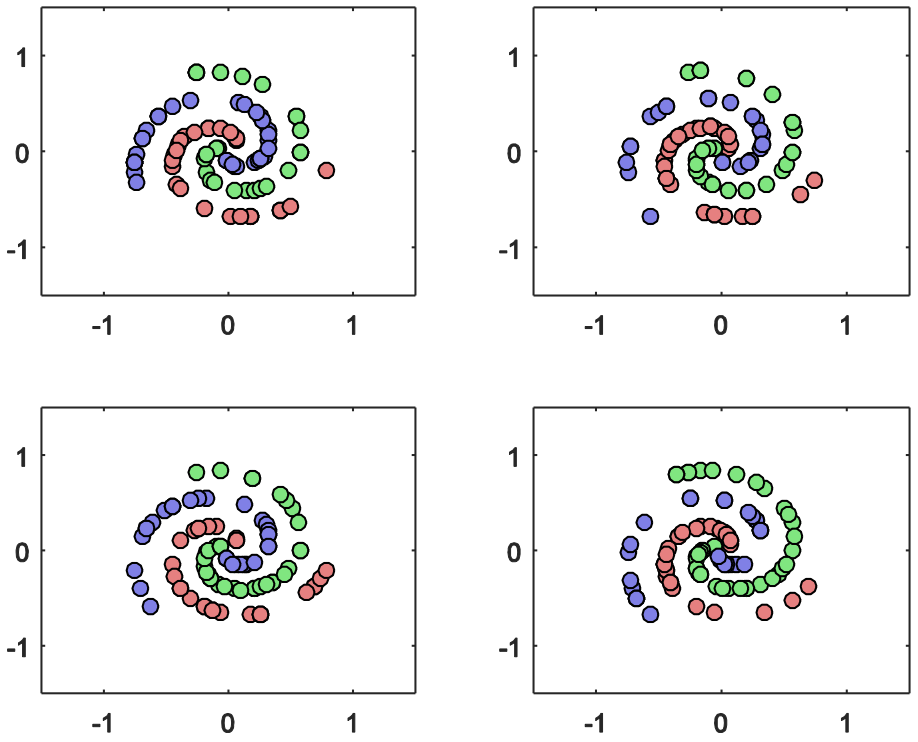
Set random forest parameters

```
param.num = 10;          % Number of trees
param.depth = 5;         % Maximum depth of trees
param.splitNum = 3;      % Number of set of split parameters θ_j i.e. ρ = 3
param.split = 'IG';      % Objective function 'information gain'
```

Train random forest

Do bagging to create subsets of training data.

- A new training set for each trees is generated by random sampling from dataset with replacement.
- Sampling fraction is set as 1 - 1/e (63.2%).
- Plot first 4 data subsets.

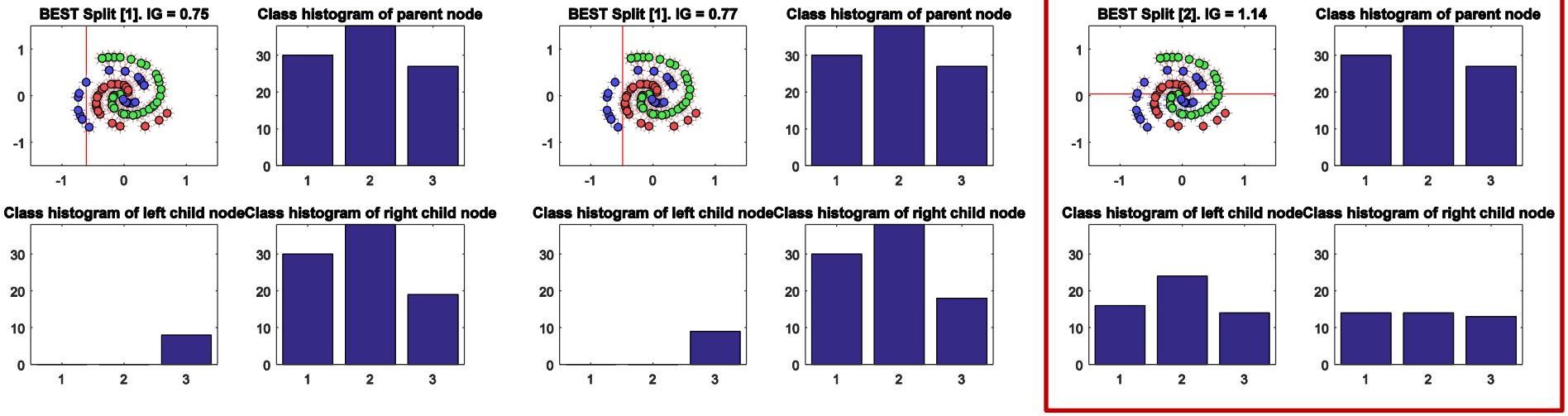


Grow a tree by splitting nodes

We grow the first tree out of (param.num=10) trees.

Split the first (root) node.

- Use axis-aligned weak learner. Pick a feature randomly.
- Find the data range of the randomly chosen feature. Pick a random value within the range as a threshold.
- Split data with this feature and threshold.
- Repeat the above 3 times (param.splitNum = 3) and choose the best split parameter.

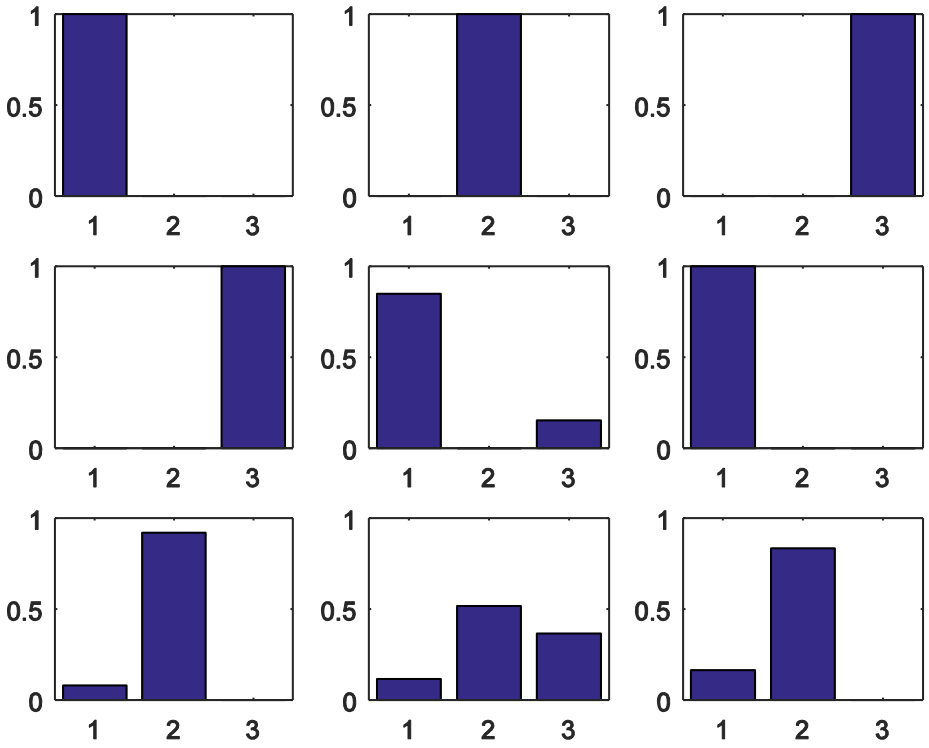


Split the nodes recursively

Grow a tree for j (the node index) = 1:2^(param.depth-1)-1

- We use a fixed depth (param.depth=5) and emtiness as stop criteria.
- Visualise the class distributions of the first 9 leaf nodes.

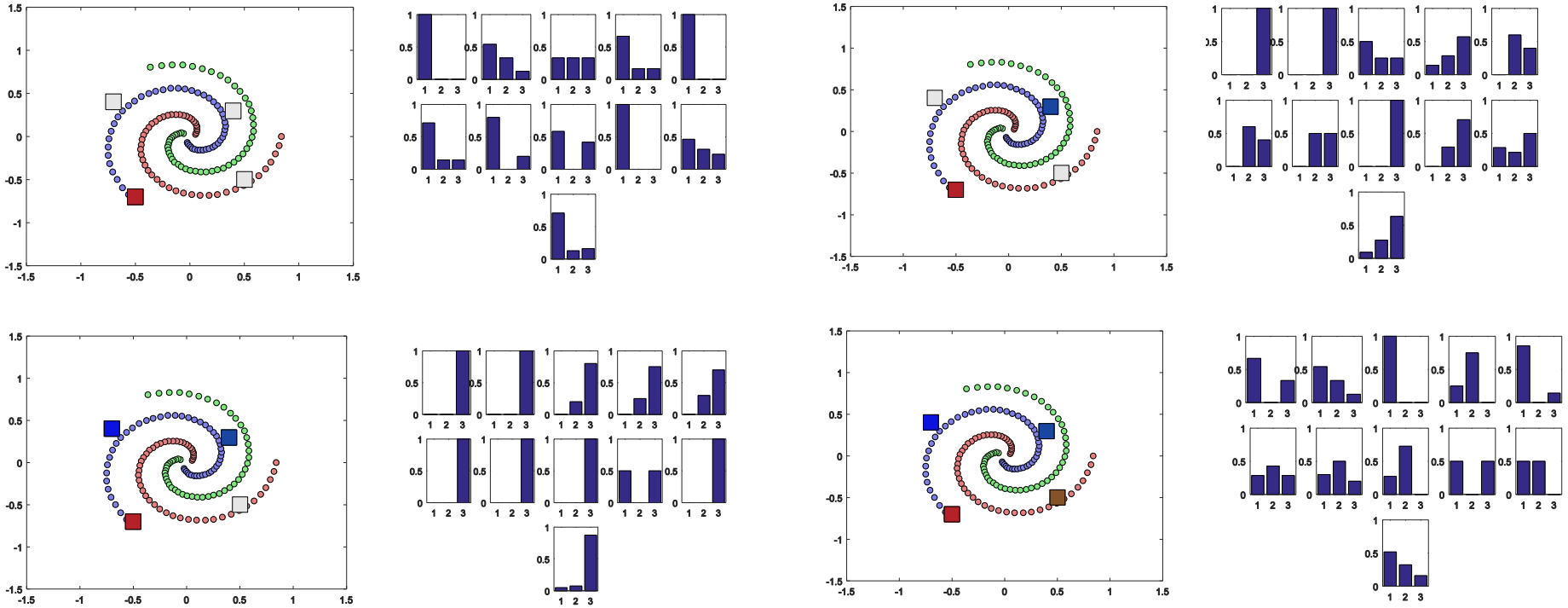
Grow all other trees



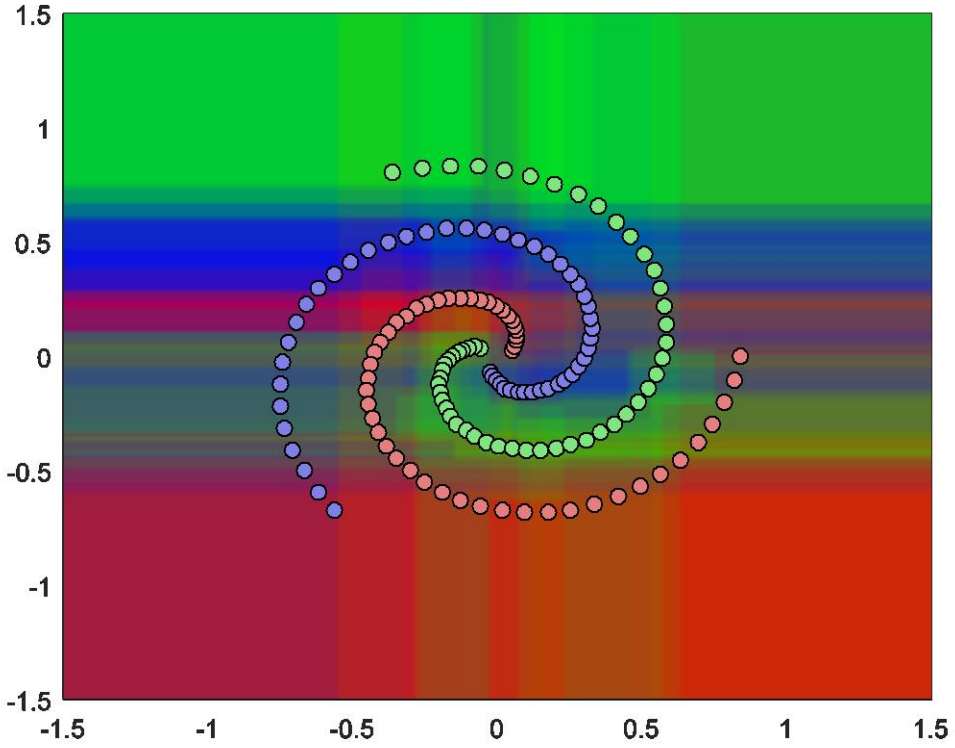
Testing

Grab the few data points and evaluate them one by one by the leant RF.

Visualise the class distributions of the leaf noes which the data point arrives at and the averaged class distribution.



Test on the dense data by RF



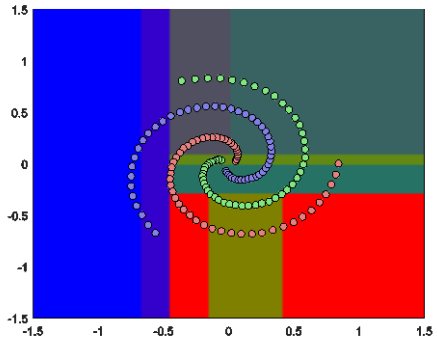
Effect of model parameters

Change the number of trees for T ∈ [1,3,5,10,20].

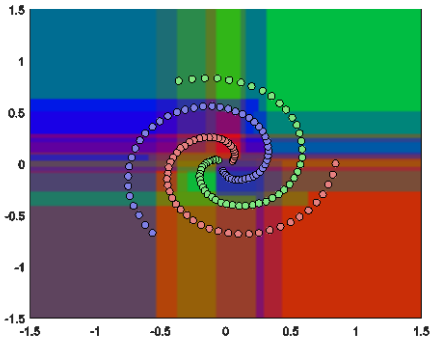
Fix other parameters.

- param.depth = 5; % Maximum depth of trees
- param.splitNum = 3; % Number of set of split parameters θ_j i.e. $\rho = 3$
- param.split = 'IG'; % Objective function 'information gain'

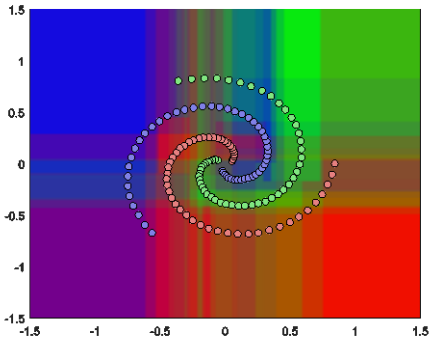
T=1



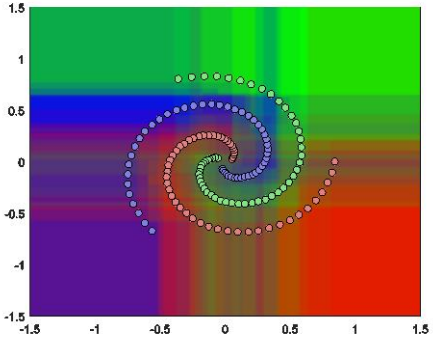
T=3



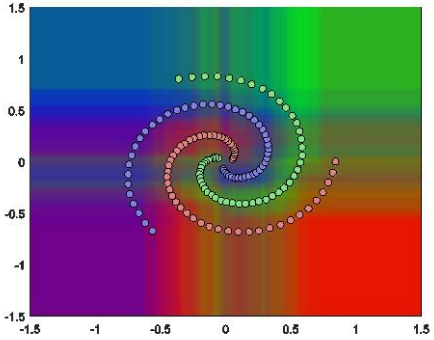
T=5



T=10



T=20



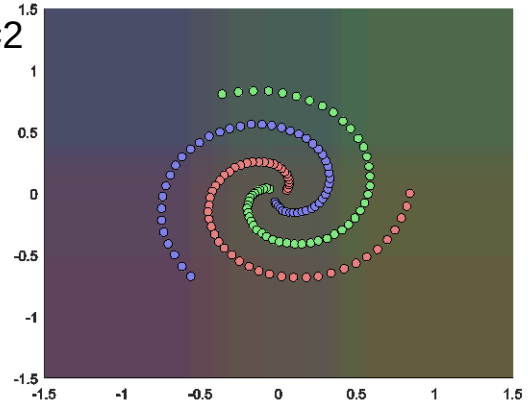
Effect of model parameters

Change the tree depth for D ∈ [2,5,7,11].

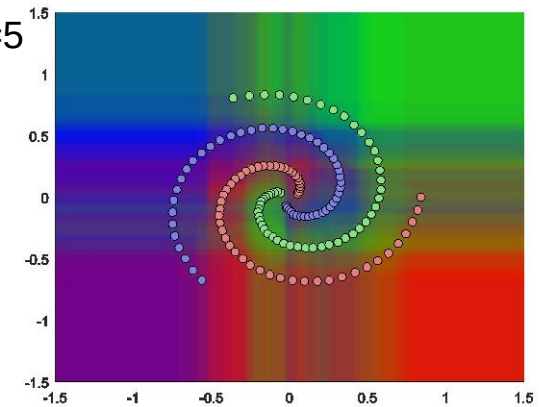
Fix other parameters.

- param.num = 20; % Number of trees
- param.splitNum = 3; % Number of set of split parameters θ_j i.e. $\rho = 3$
- param.split = 'IG'; % Objective function 'information gain'

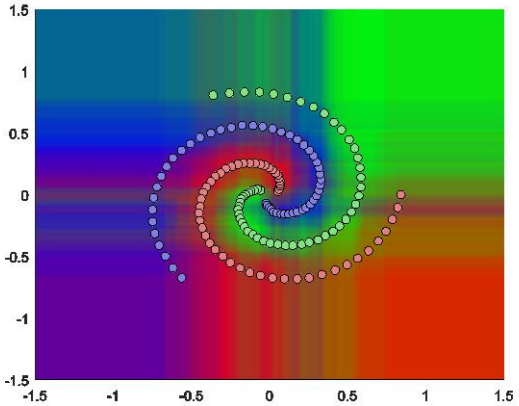
D=2



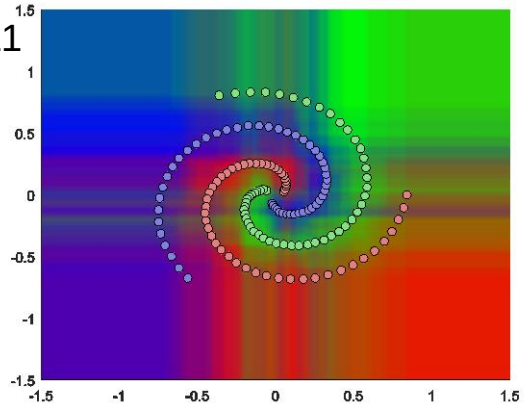
D=5



D=7



D=11



Visual object categorisation

Caltech101 dataset:

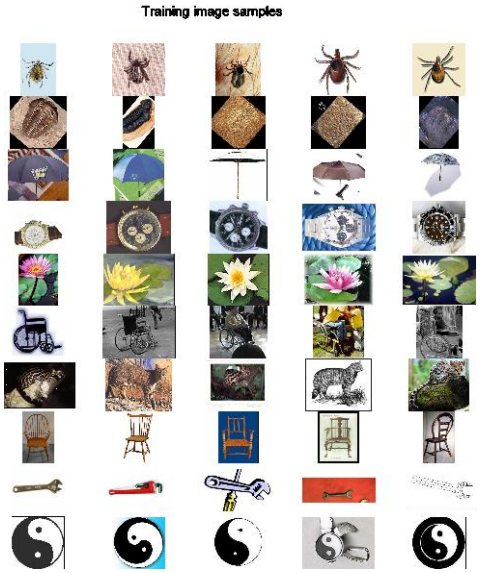
- folderName = './Caltech_101/101_ObjectCategories'
- Number of classes: 10
- Number of training images: 15 per class, randomly selected
- Number of testing images: 15 per class, randomly selected

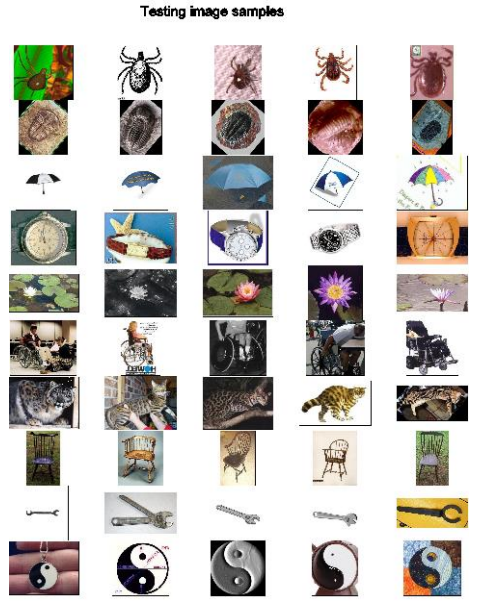
Feature descriptor:

- It works on gray scale images
- It extracts multi-scaled dense SIFT features (for details of image description, see http://www.vlfeat.org/matlab/vl_phow.html)
- Feature dimension is 128.

Building visual vocabulary (codebook):

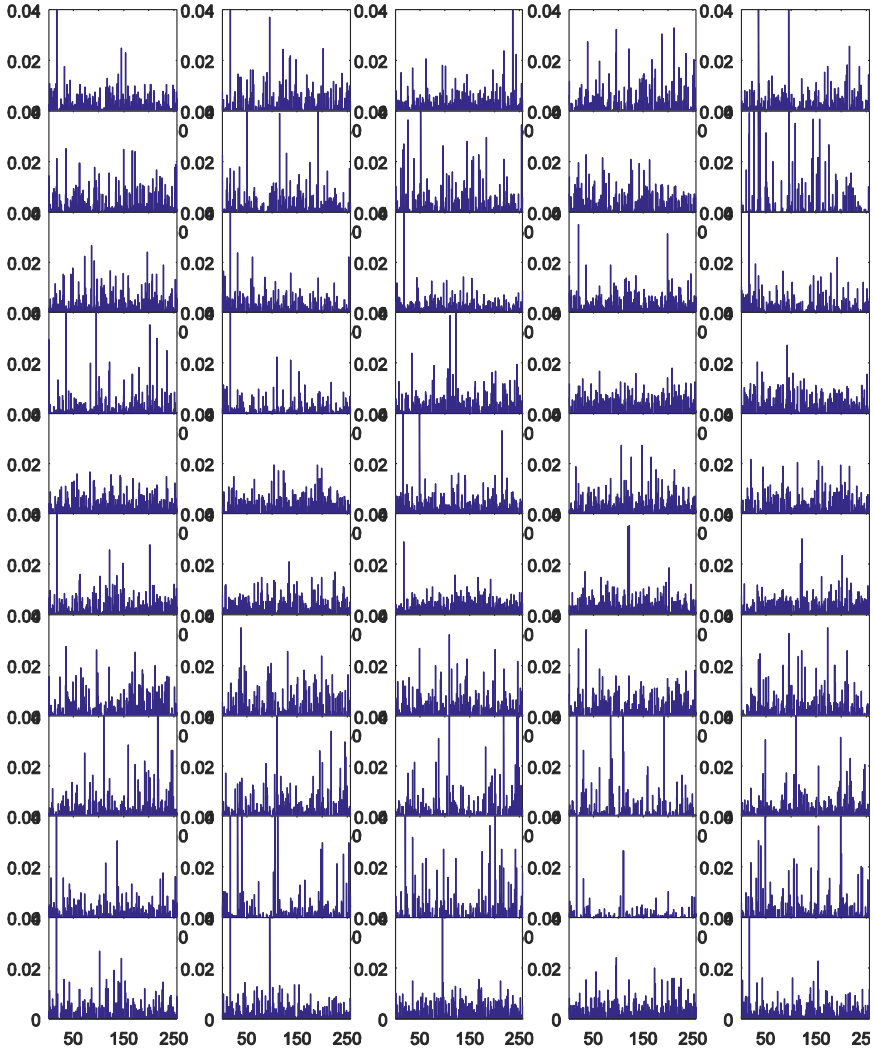
- We randomly select 100k descriptors for k-means clustering



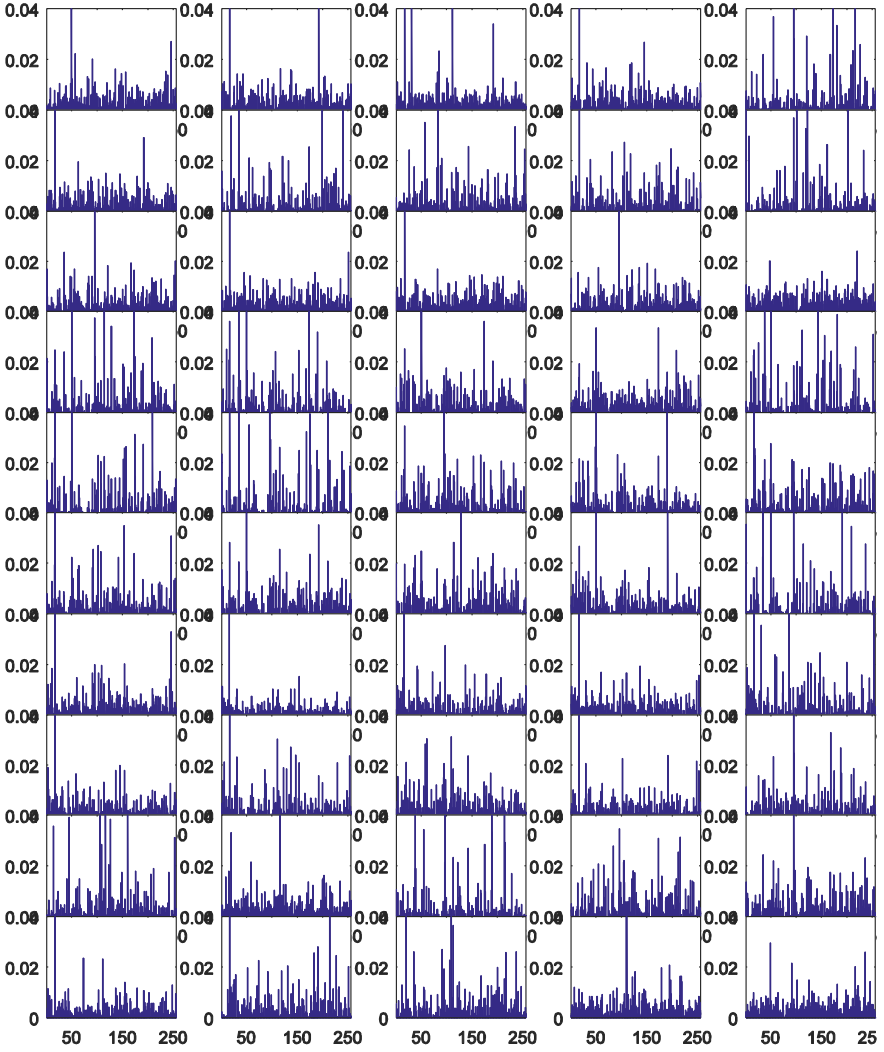


Bag of words histogram

Image representations: 256-D histograms



Testing image representations: 256-D histograms



Visual categorisation result

```
param.num = ??;         % Number of trees
param.depth = ??;       % Maximum depth of trees
param.splitNum = ??;    % Number of set of split parameters θ_j i.e. ρ = 3
param.split = 'IG';     % Objective function 'information gain'
```