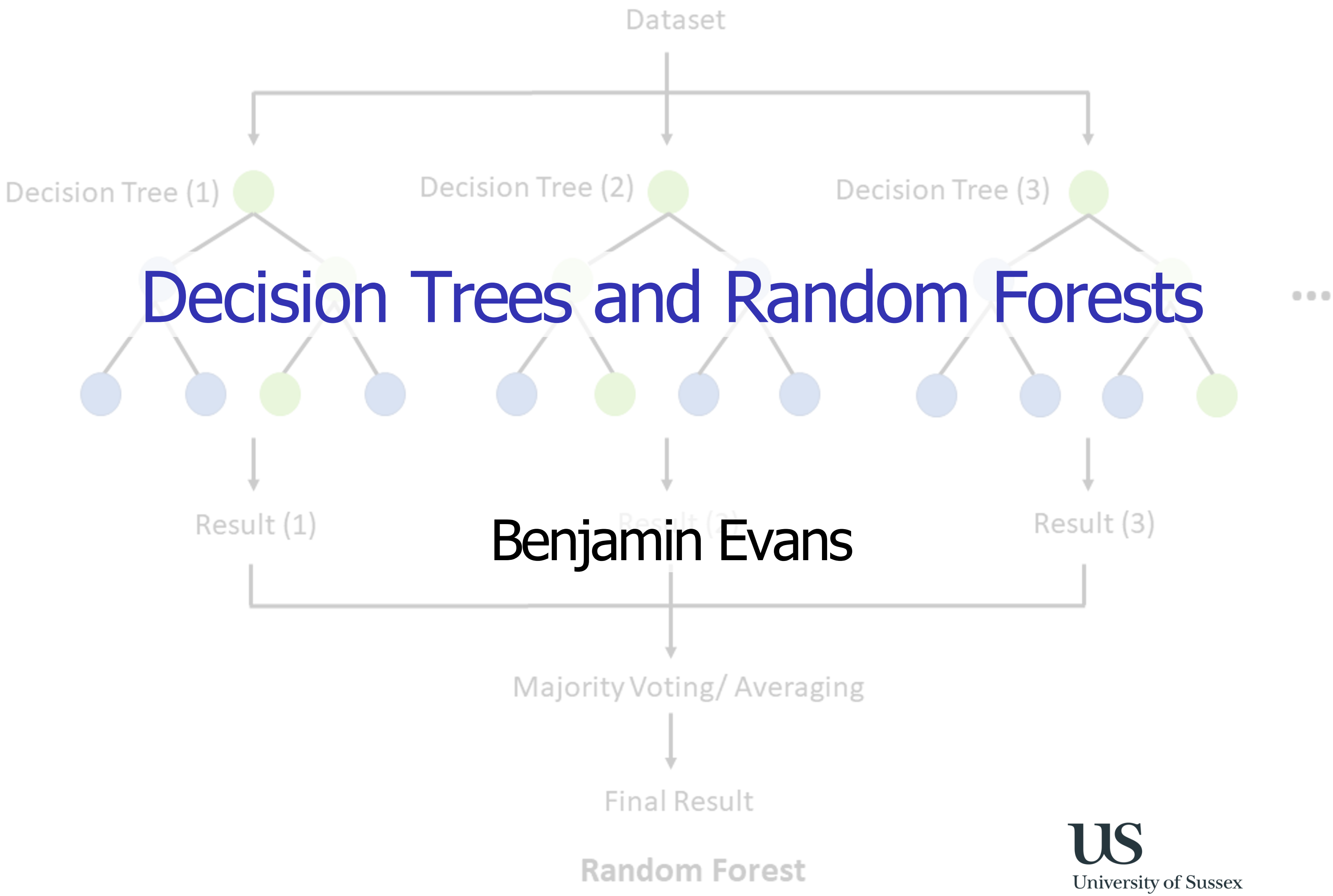
Decision Trees and Random Forests



Benjamin Evans

University of Sussex

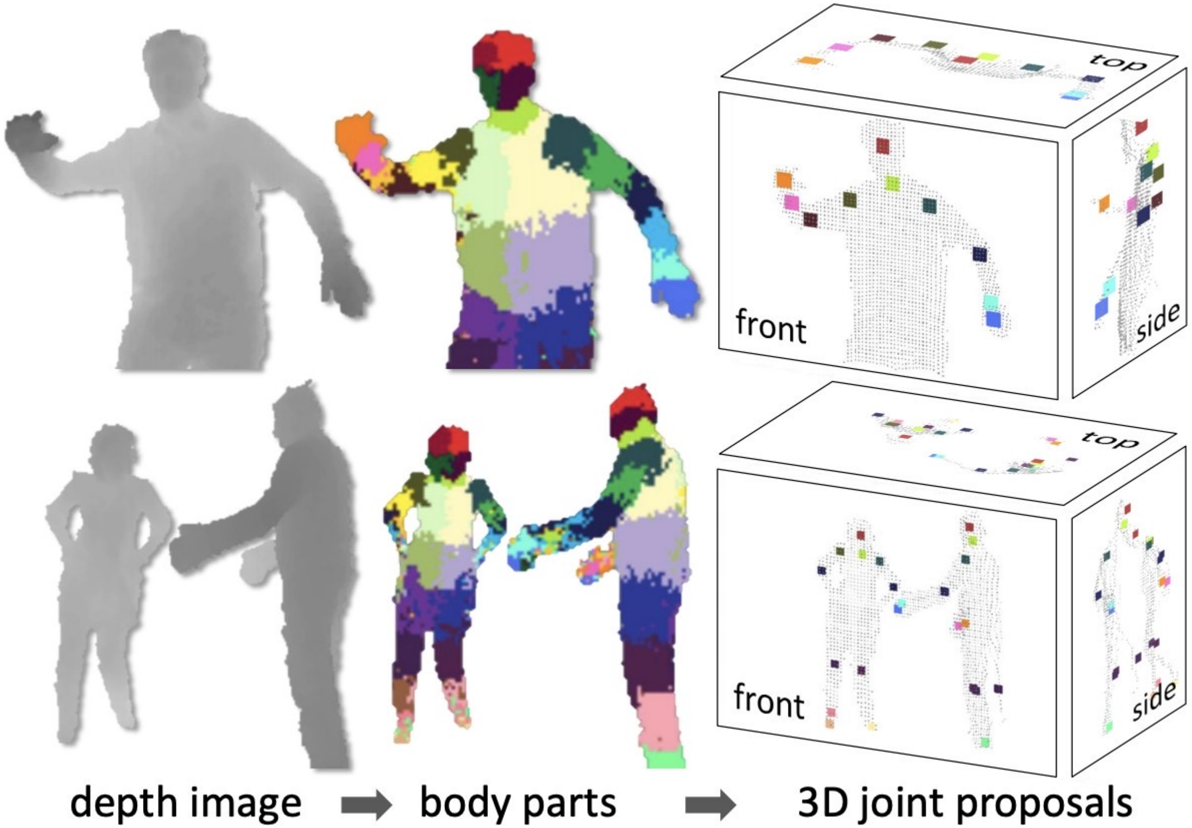
Today's Outline

- Decision trees as a non-parametric prediction model
  - **Classifiers** or Regressors
  - What they can express, limitations and strengths
  - A generic algorithm for learning them
- Ensemble of decision trees: a random forest

Learning Outcomes

1. Understand how decision tree classifiers work and how they are trained.
2. Understand their advantages and disadvantages.
3. Explain the concept of ensembles, how they are trained, and why random forests are effective models.

Applications: XBox-Kinect



Real-Time Human Pose Recognition in Parts from Single Depth Images, Shotton et al. CVPR 2011

US University of Sussex

Properties of Decision Trees?

- Decision trees are powerful predictive models that partition the input space to allow predictions that are <u>non-linear</u> in the input.
- That have been applied to problems such as *classification*, *regression* and *probability density estimation*.
- Decision trees are *readily interpreted*, their decision making process can be understood by humans and used in knowledge-based systems.
- Decision trees are highly expressive:
  - Good for separating complex data.
  - …but need to worry about over-fitting.

An Example

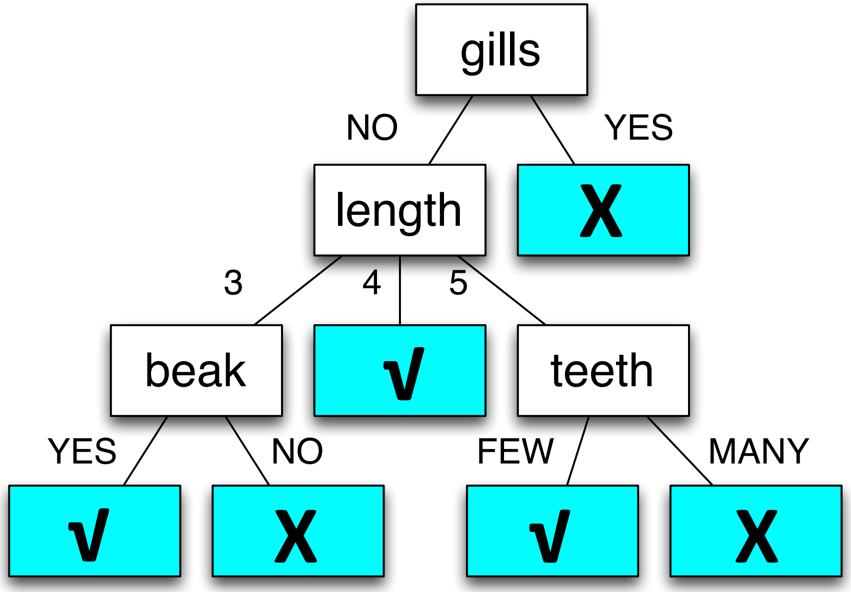
- Suppose we are building a binary classifier of sea creatures.
- We collect some properties that are assumed significant for identification:
  - Whether or not they have gills
  - The length of the creatures in metres
  - Whether they have a beak or not
  - Whether they have few or many teeth
- We compile a collection of examples describing sea-creatures that are manually labelled.

Sample Data

	gills	length	beak	teeth	class
example1	yes	3	no	few	×
example2	no	5	yes	few	✓
example3	no	3	yes	many	✓
example4	yes	5	no	many	×
example5	no	5	yes	many	×
example6	no	4	yes	many	✓
example7	no	5	no	few	✓
example. . .	no	3	no	few	×

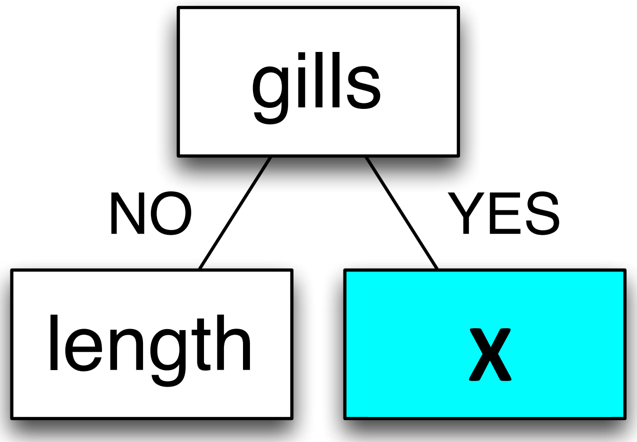
Decision Tree as a Classifier

- Recast observed properties into a tree of questions (nodes):
  - *nodes* labelled by features
  - *edges* labelled by values of those features
  - *leaf nodes* labelled by class labels



Tests and Splits

- Node is a test on a single feature:
  - e.g. are there gills or not?
- The set of edges/values at a node is called a split
  - e.g. {yes, no}
- A path from the root to a leaf is a logical conjunction of tests



University of Sussex

A Little Bit of Logic

- A test is a simple proposition:
  - i.e. a statement that can be **true** or **false**

Test	true	false
gills=yes	1,4,...	2,3,5,6,7,...
length=3	1,3,...	2,4,5,6,7,...
teeth=many	3,4,5,6,...	1,2,7,...

- A test can also be understood as denoting a set of instances
  - i.e. just those instances for which the test is **true**

	gills	length	beak	teeth	class
example1	yes	3	no	few	×
example2	no	5	yes	few	✓
example3	no	3	yes	many	✓
example4	yes	5	no	many	×
example5	no	5	yes	many	×
example6	no	4	yes	many	✓
example7	no	5	no	few	✓
example...	no	3	no	few	×

A Little Bit More Logic

Operator	English	Symbol
negation	not	$\neg$
conjunction	and	$\wedge$
disjunction	or	$\vee$
implication	if. . .then	$\rightarrow$

- Complex tests can be built using logical operators:

  e.g. a conjunction of tests:

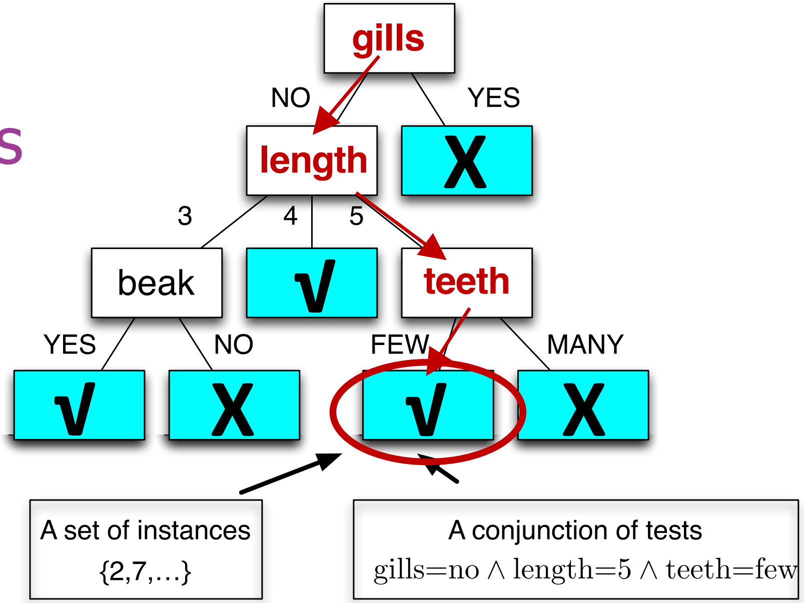
$$\text{gills=no} \wedge \text{length=5} \wedge \text{teeth=few}$$

true just in the case that each of the simple tests is **true**.

- Can also be understood as denoting a set of instances.

Path, Logic and Instances

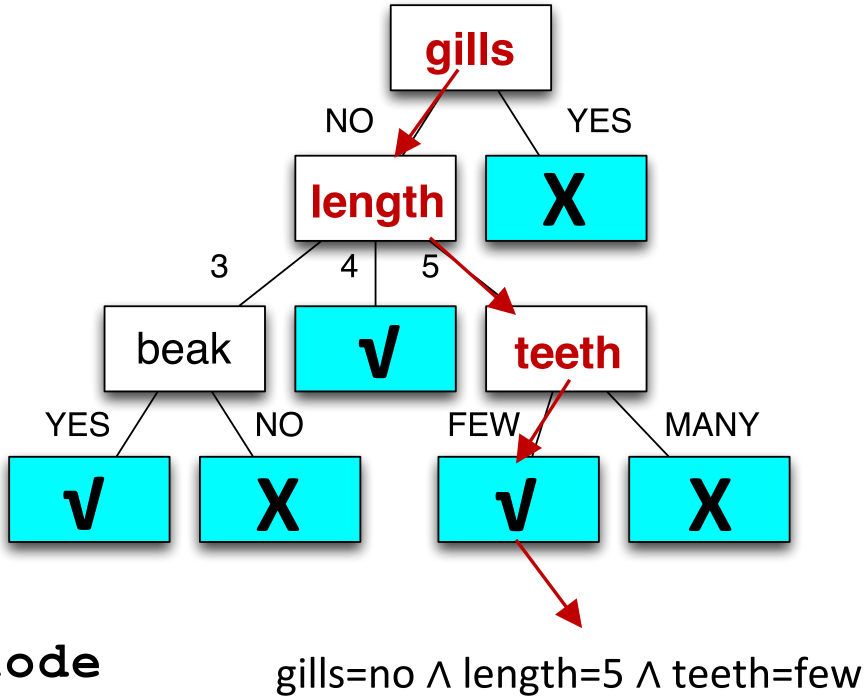
- A path from the root to a leaf encodes a logical conjunction of tests
- Leaf node represents:
  - a logical expression
  - a set of instances



	gills	length	beak	teeth	class
example1	yes	3	no	few	×
example2	no	5	yes	few	✓
example3	no	3	yes	many	✓
example4	yes	5	no	many	×
example5	no	5	yes	many	×
example6	no	4	yes	many	✓
example7	no	5	no	few	✓
example...	no	3	no	few	×

Decision Tree Classification

- Classify instances by following a path.
  - Start at the root and finish at a leaf.




```
set node=root
until isleaf(node) do{
    follow true edge to next node
}
return label(node)
```

University of Sussex

Learning a Decision Tree

- Start with examples *D* and features *F*
  - take a divide and conquer approach

For each new node you might add:

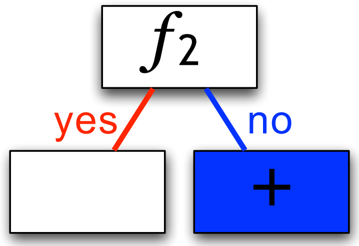
1. IF : all examples *D* are (pretty much) of the same class, then just label them as that class and terminate;
2. ELSE : choose a feature from *F* to split on and build a decision tree from each corresponding subset of *D*

- There are some issues to resolve:
  - pretty much of the same class?
  - choose a feature to split on?

Learning a Decision Tree: Example

	f_1	f_2	class
ex1	a	yes	–
ex2	a	no	+
ex3	b	yes	–
ex4	c	yes	+
ex5	c	no	+

- Examples are not (pretty much) of the same class.
- Choose a feature to split on.
  - suppose we choose f_2



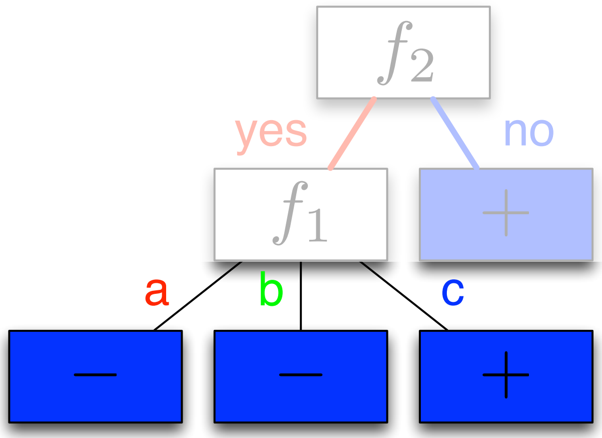
  - right node can be labelled as a leaf with value +
  - build a decision tree for the remaining instances at left node

University of Sussex

Learning a Decision Tree: Example

	f_1	f_2	class
ex1	a	yes	$-$
ex3	b	yes	$-$
ex4	c	yes	$+$

- Examples still not (pretty much) of same class, so split on f_1:



- Tree is finished: all leaves are labelled consistently

University of Sussex

Learning a Decision Tree: Algorithm

if root homogeneous then
just label it and return

GrowTree($\mathcal{D}$, $\mathcal{F}$):

if Homogeneous($\mathcal{D}$) OR $\mathcal{F}$ == { } **then**
  return new tree with single leaf with Label($\mathcal{D}$);
S := BestSplit($\mathcal{D}$, $\mathcal{F}$);
partition $\mathcal{D}$ into subsets $\mathcal{D}_i$ according to S
for each i **do**
  if $\mathcal{D}_i$!= { } then
    $\mathcal{T}_i$:= GrowTree($\mathcal{D}_i$,$\mathcal{F}$ / {S});
  else
    $\mathcal{T}_i$:= a single leaf with Label($\mathcal{D}$)
  endif
endfor
return new tree with root test S and subtrees $\mathcal{T}_i$

Learning a Decision Tree: Algorithm

if root homogeneous then
just label it and return

GrowTree($\mathcal{D}$, $\mathcal{F}$):

if Homogeneous($\mathcal{D}$) OR == { } **then**
  return new tree with single leaf with Label($\mathcal{D}$);
S := BestSplit($\mathcal{D}$, $\mathcal{F}$);
partition $\mathcal{D}$ into subsets $\mathcal{D}_i$ according to S
for each i **do**
  if $\mathcal{D}_i$!= { } then
    $\mathcal{T}_i$:= GrowTree($\mathcal{D}_i$, $\mathcal{F}$/{S});
  else
    $\mathcal{T}_i$:= a single leaf with Label($\mathcal{D}$)
  endif
endfor
return new tree with root test S and subtrees $\mathcal{T}_i$

Otherwise
DIVIDE:
find a good split ...

University of Sussex

Learning a Decision Tree: Algorithm

if root homogeneous then
just label it and return

```
GrowTree(D, F):

if Homogeneous(D) OR    ==   { } then
   return new tree with single leaf with Label(D);
S := BestSplit(D, F);
partition D into subsets D_i according to S
for each i do
   if D_i != { } then
      T_i := GrowTree(D_i, F/{S});
   else
      T_i := a single leaf with Label(D)
   endif
endfor
return new tree with root test S and subtrees T_i
```

Otherwise
DIVIDE:
find a good split ...

… and CONQUER:
build a children tree for each subset

University of Sussex

Homogeneity, Labels, and Splitting

- Algorithm assumes the following functions are defined:
  - Homogeneous(*D*) : if *D* is homogeneous (enough) to be assigned a single label return **true** else return **false**.
  - Label(*D*) : return most appropriate label for *D*.
  - BestSplit(*D*,*F*) : return best split of *D* (i.e. identify best feature to split on).
- There are different possible instantiations of these functions.

Homogeneity, Labels, and Splitting

- What does Homogeneous(*D*) mean?
  - Could simply say a set of instances is homogeneous if all instances have the same class label.
  - Clearly, this label should be returned by Label(*D*).
  - Homogeneity can be relaxed, so more generally, Label(*D*) should return the **majority** label.
- What does BestSplit(*D*,*F*) mean?
  - Assume for the moment binary (+,-) classification and just simple Boolean features.
  - Ideally split *D* into two: one set of just + instances and one set of just - instances (both sets are pure).
- Why would pure subsets be the ideal case?

University of Sussex

Impurity of a Set of Instances

Binary classification:

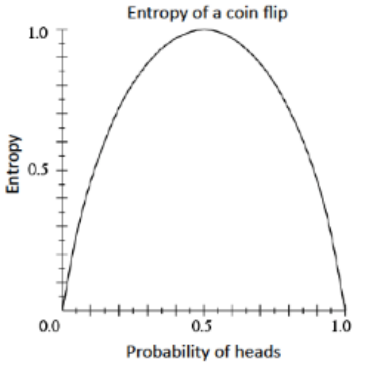
- In general when training we may have splits at a node that have mixes of + and - instances (at least one set is impure).
- We can write $[n_+, m_-]$ to denote a set with a mix of n positive instances and m negative instances.
- Assumption: (im)purity of a set of instances defined in terms of proportion $p = n_+/(n_+ + m_-)$.

University of Sussex

Measuring Impurity

Measures adopted in practice include:

1. Entropy:
   - For $C > 2$ classes, it is: $\sum_{i=1}^{C} p_i \times \log_2 \frac{1}{p_i}$
   - For $C = 2$ classes (binary), it is: $-p \log_2 p - (1-p) \log_2 (1-p)$



2. Gini index:
   - The expected error rate:
     - p_i is the probability that a random instance in the leaf node belongs to class i,
     - $1 - p_i$ is the probability that it is misclassified.
   - For $C > 2$ classes, it is: $\sum_{i=1}^{C} p_i (1 - p_i)$
   - For $C = 2$ classes (binary), it is just: $2p(1-p)$

Impurity of a Split

- Suppose that a possible split of D results in subsets $D_1, D_2, \ldots, D_k$
- How do we judge the "goodness" of this split?
- Have impurity measure $\text{Imp}(D)$ for instances D
  - e.g. Entropy or Gini index
- Define $\text{Imp}(D_1, D_2, \ldots, D_k)$ as weighted average:

$$w_1 \times \text{Imp}(D_1) + w_2 \times \text{Imp}(D_2) + \ldots + w_k \times \text{Imp}(D_k)$$

where $$w_i = |D_i| \Big/ \textstyle\sum_{1,\ldots,k} |D_i|$$

BestSplit Algorithm

run through each feature in turn

BestSplit($\mathcal{D}$, $\mathcal{F}$):

min_impurity := 1;
for each f in $\mathcal{F}$ **do**
  split $\mathcal{D}$ into $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ given k values of f;
  if Imp($\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$) < min_impurity **then**
    min_impurity := Imp($\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$);
    best_feature := f;
  endif
endfor
return best_feature;

keep track of split with lowest impurity

return best feature to split on

University of Sussex

Why Favour Purity?

- Earlier we asked why pure subsets are the ideal case.
  - answer has to do with **generalisation**.

    - "Purer splits yield smaller trees, with shorter paths from root to leaves."
- Shorter paths mean fewer tests when classifying instances.
- Fewer tests mean fewer features to be examined before assigning a class label.
- Occam's razor again! Fewer features examined and tested on means greater generalisation.

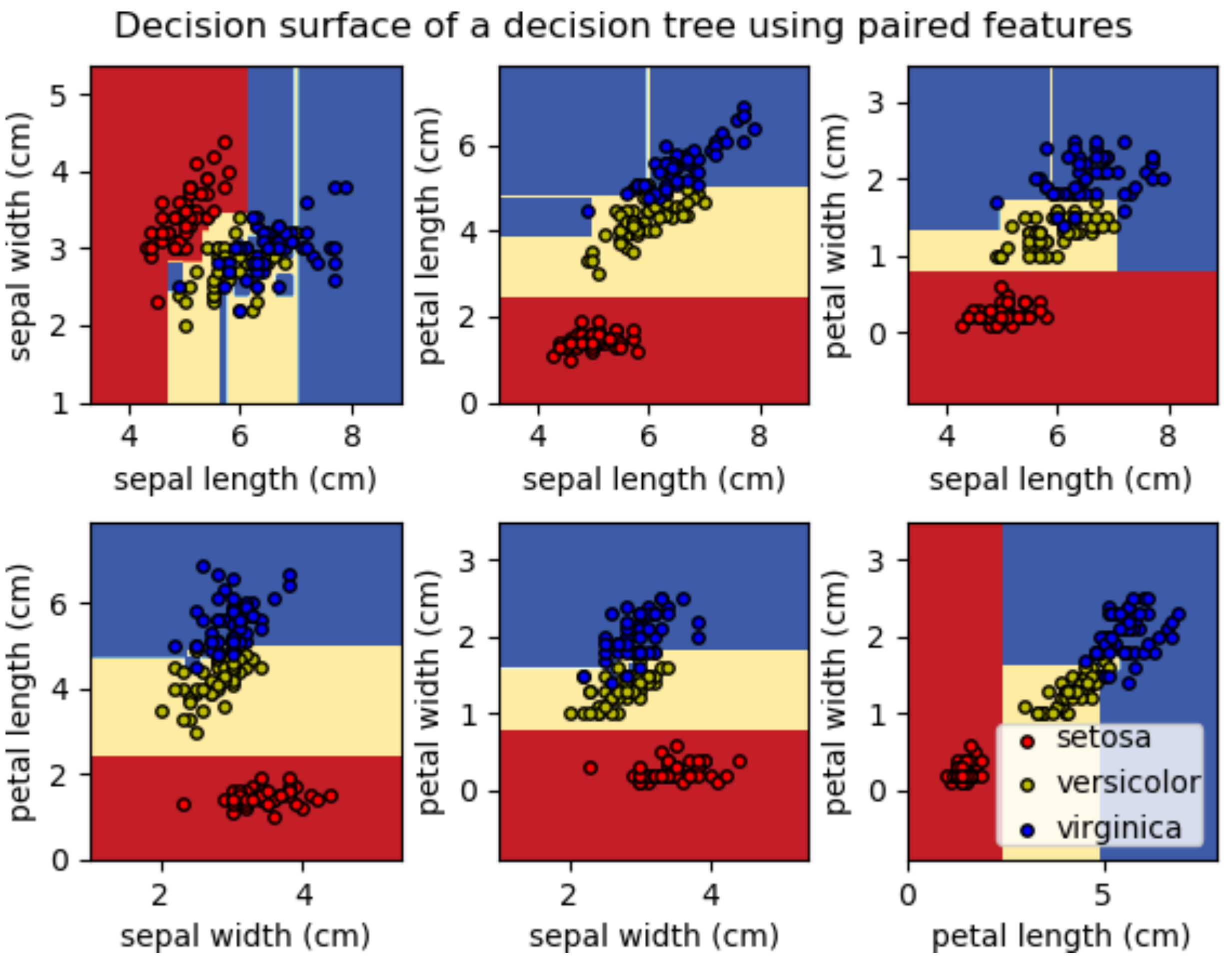
Ferns

- Ferns are a simpler form of tree.
- Here the function of the data for nodes at the same level of the tree are the same.
  - Although the thresholds can be different.
- This leads to more efficient prediction, as a more limited set of functions of the data need to be evaluated.

Problems with Decision Trees

- Decision trees are tricky to get to predict very accurately compared to other kinds of machine learning models.
  - Learning good trees requires a lot of experimentation.
- **Unstable**: small changes to the input data can have large effects on the structure of the tree → decision trees are high variance models.

Example



University of Sussex

Example Code:

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.tree import DecisionTreeClassifier
>>> from sklearn.tree import export_text
>>> iris = load_iris()
>>> decision_tree = DecisionTreeClassifier(random_state=0, max_depth=2)
>>> decision_tree = decision_tree.fit(iris.data, iris.target)
>>> r = export_text(decision_tree, feature_names=iris['feature_names'])
>>> print(r)
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) > 0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- class: 1
|   |--- petal width (cm) > 1.75
|   |   |--- class: 2
```

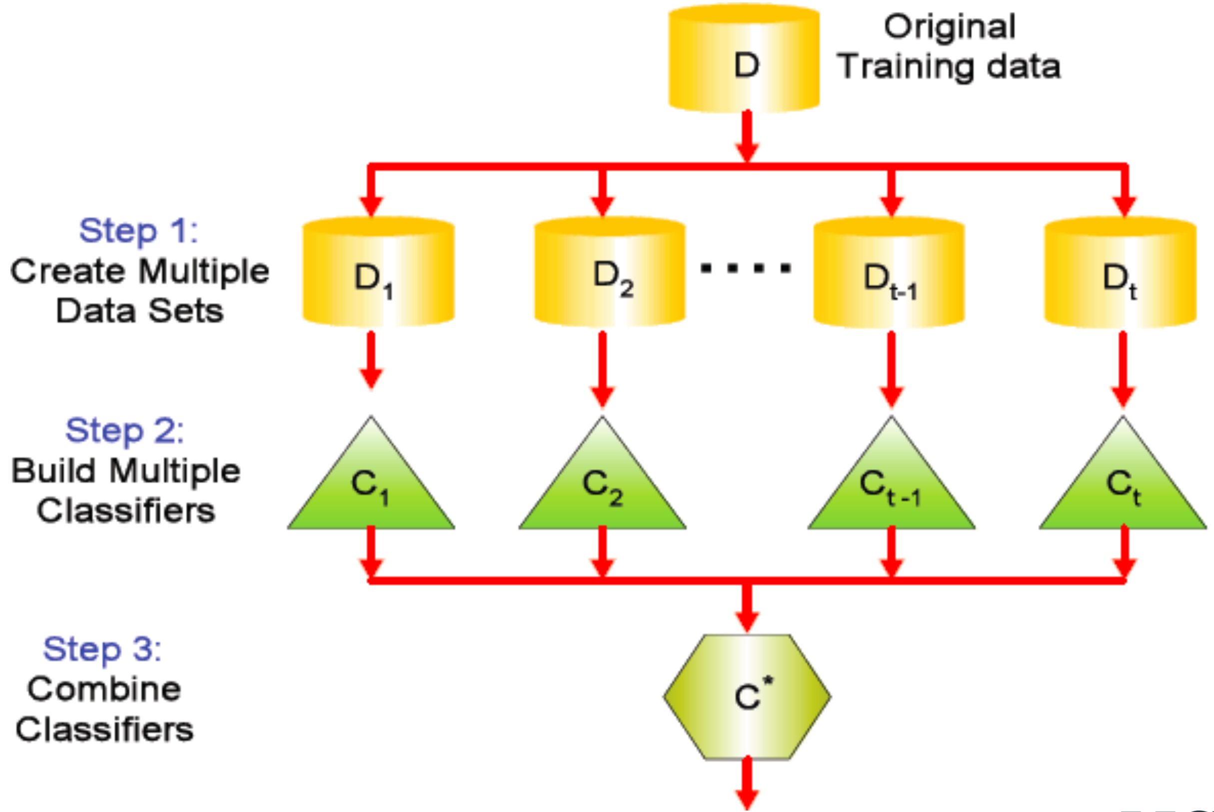
Code from https://scikit-learn.org/stable/modules/tree.html

So how can we make these more useful?

- By building lots of them and combining them together!

Ensemble Models – General Idea

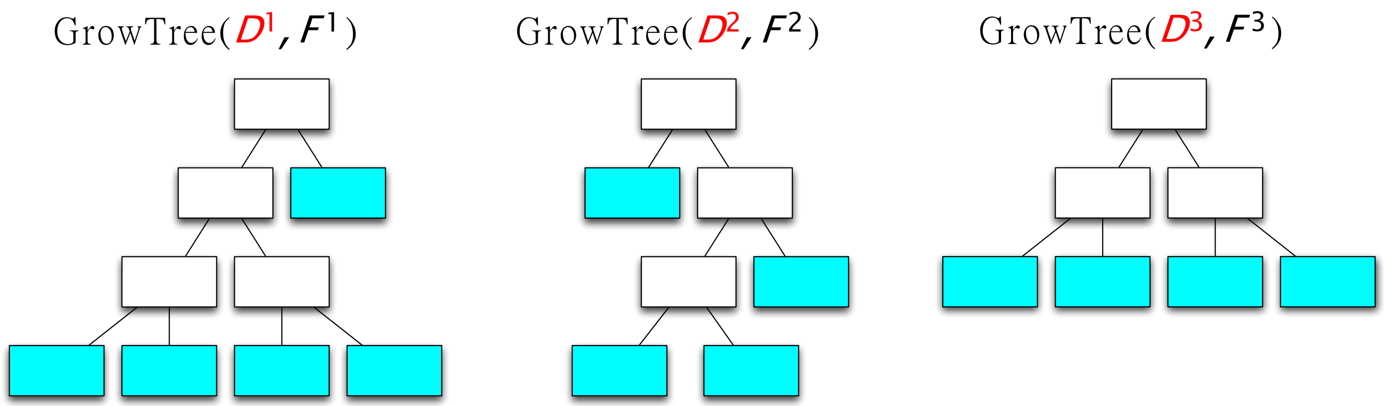
Ensemble models to reduce model variance.



University of Sussex

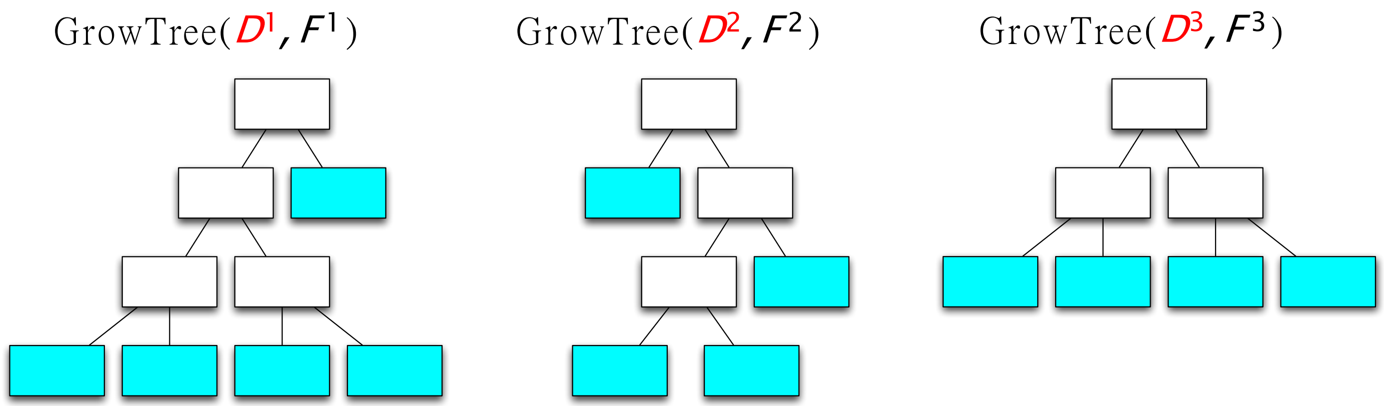
Random Forests

- A forest is an ensemble of trees.
- Each tree is slightly different from the others.
- Two sources of randomness in the trees:
  1. Random sampling of the training data: let D be the full training data, and let $D^t \subset D$ be the random subset of training data for tree t.



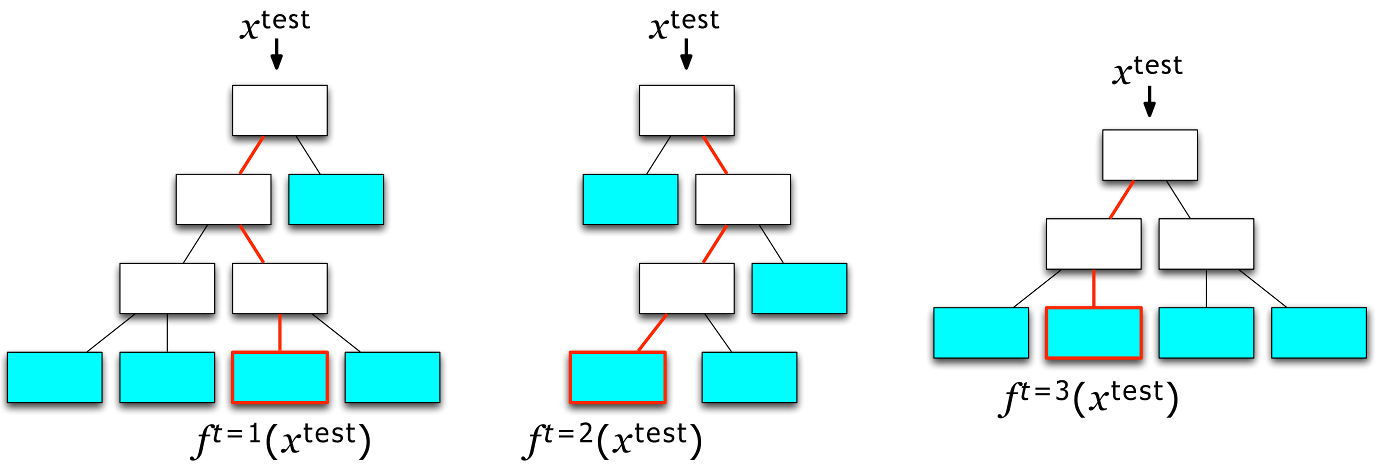
Random Forest

2. Random subset of data features: let F be the set of data features:
e.g. F = {Length, Gills, Beak, Teeth}
and let $F^t \subset F$ be a random subset of data features for tree t:
e.g. F^1 = {Length, Beak}



Random Forest Prediction

Prediction corresponds to an aggregation across trees by majority voting.



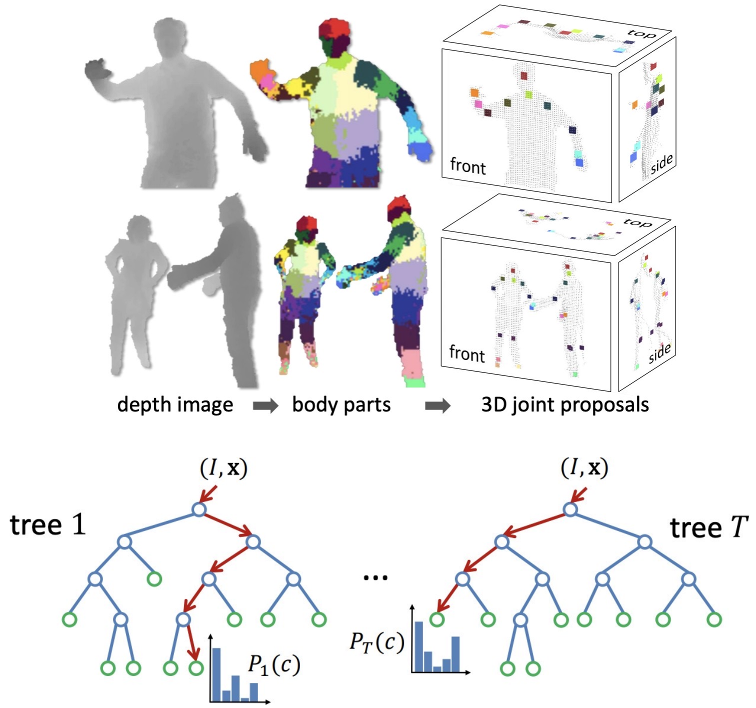
$$f^{RF}(x^{test}) = majority\{f^{t=i}(x^{test})\}_{i=1}^{3}$$

University of Sussex

Random Forests

- Combining all these trees together leads to substantially improved performance and reduces overfitting.
  - This is because we aggregate over these high-variance models.
- Random forests were state of the art for many applications until quite recently, and they are still very much in use.

Random Regression Forests on the Xbox Kinect



- Classify each pixel as a body part.
- Features are differences in the depth map around a pixel of interest: e.g. depth(x,y) - depth(x+1,y)
- Leaves contain class probabilities rather than single classes.

Real-Time Human Pose Recognition in Parts from Single Depth Images, Shotton et al. CVPR 2011

Summary

- Decision trees are powerful models, but they need to be used carefully to avoid overfitting.
- There are several heuristics that you need to define to make them work well.
- Random forests can work very well, but again experimentation is required!

What’s next?

Semi-supervised learning