Machine Learning

FIB, Master in Innovation and Research in Informatics

Marta Arias, Computer Science @ UPC

Topic 7: Random forests

Ensemble methods

Wisdom of the crowd

From wikpedia:

```
The classic wisdom-of-the-crowds finding involves point
estimation of a continuous quantity. At a 1906 country fair in
Plymouth, 800 people participated in a contest to estimate the
weight of a slaughtered and dressed ox. Statistician Francis
Galton observed that the median guess, 1207 pounds, was accurate
within 1% of the true weight of 1198 pounds.
```

Ensemble methods

What are they?

An ensemble method is a method that **combines** two or more (better-than-chance) predictors in making its predictions instead of using a single predictor

When can we use ensemble methods?

When we have several models that are slightly better-than-chance and, more importantly, are *independent*

How can we combine individual models?

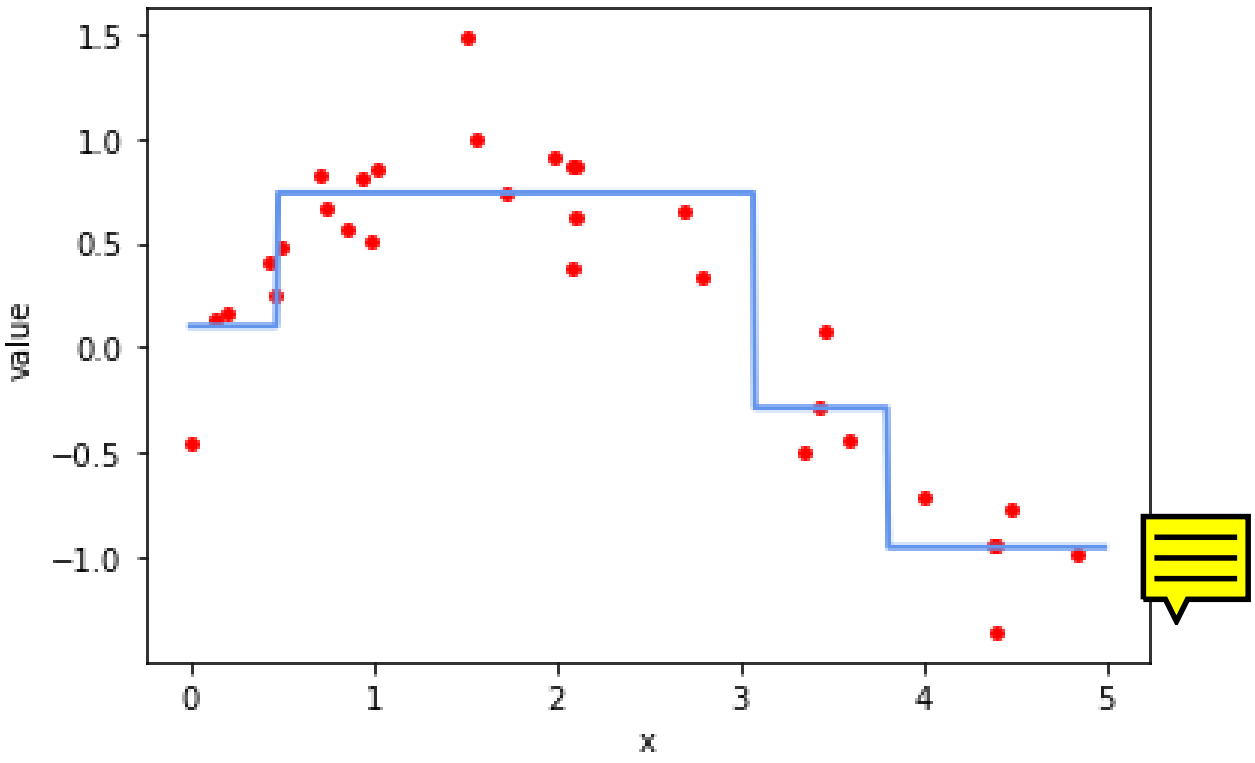
In regression, by averaging their predictions; in classification, by taking majority vote or computing probability class distributions.

In this lecture, we'll cover:

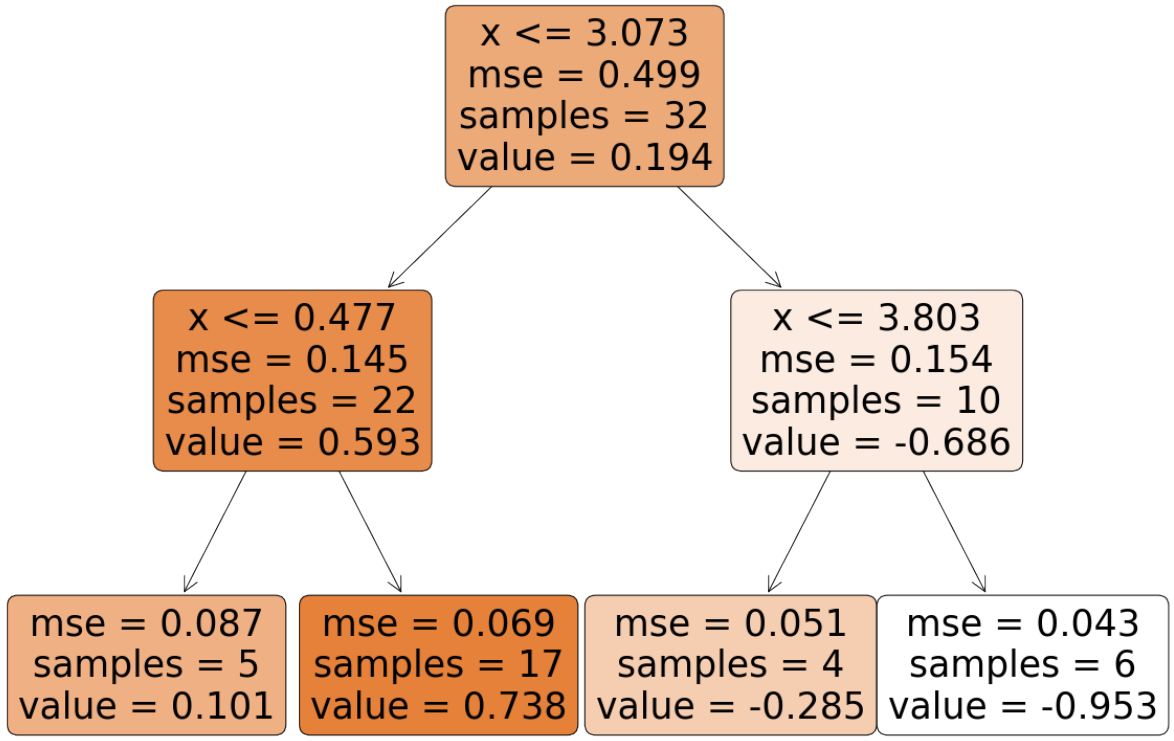
1. **Classification and regression trees**, and how to build them
2. **Random Forests**: an ensemble method that *combines* individual classification or regression trees

Regression tree

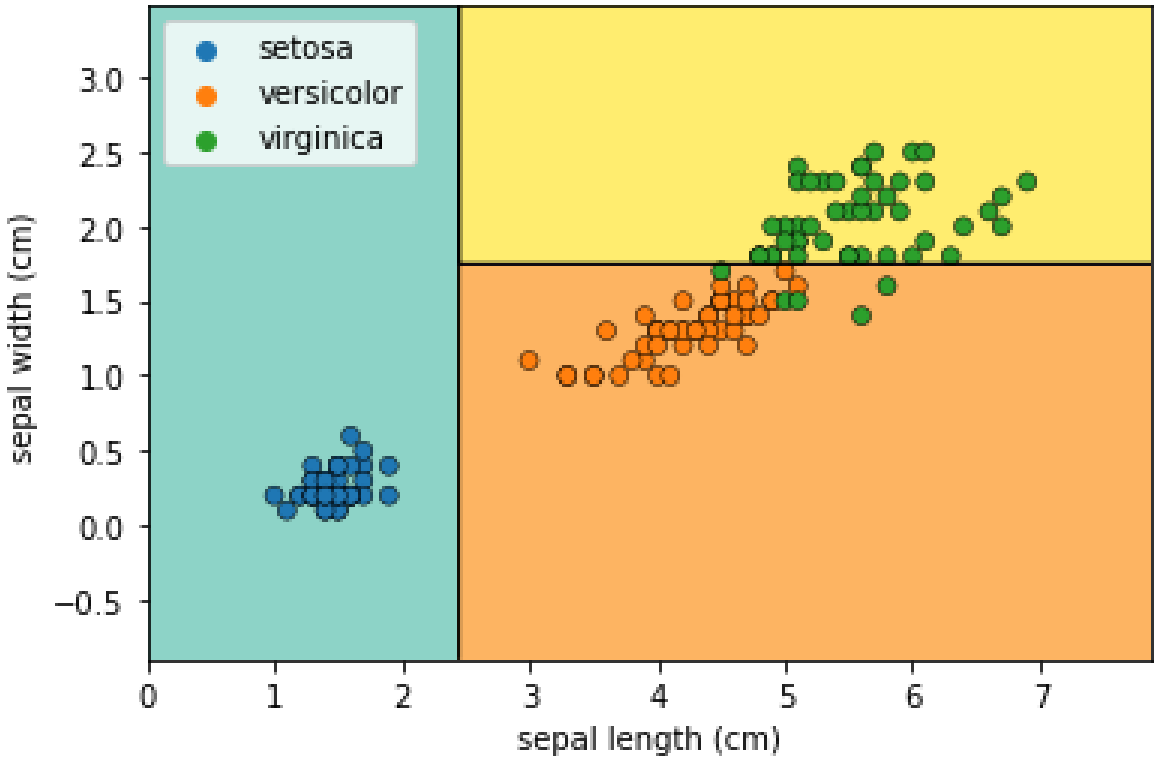
A regression tree partitions the feature space into axis-parallel regions and predicts using the average of training points that fall into that region



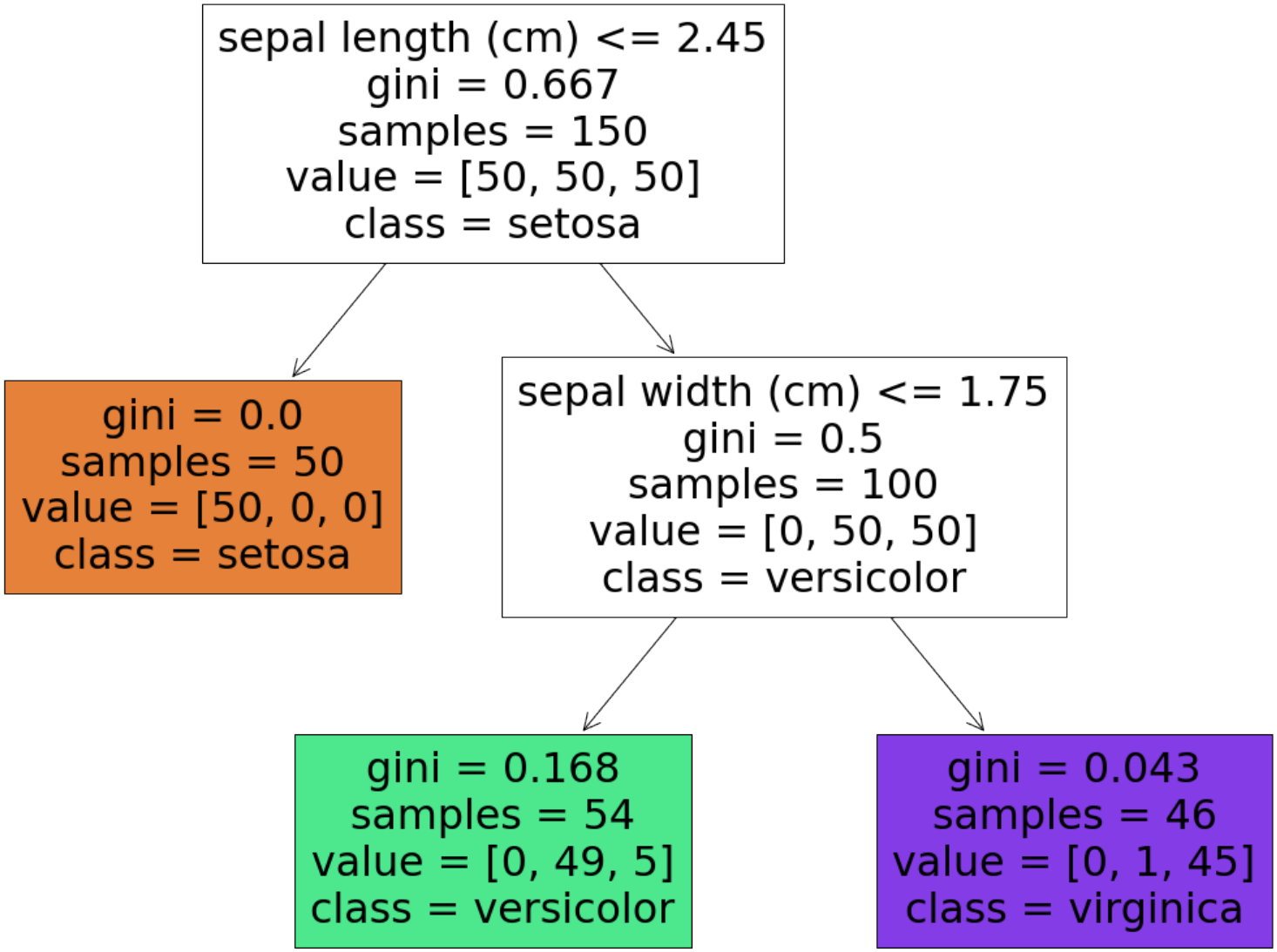
Regression tree, cont.



Classification tree



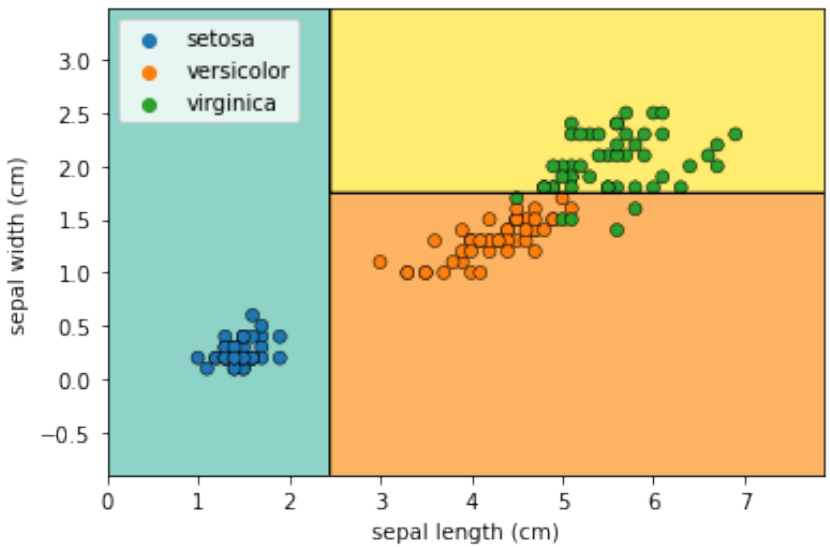
Classification tree, cont.



Growing a classification tree

It is NP-hard to grow an "optimal tree" (of minimum size), so greedy heuristics are typically used when learning trees. We show here how to grow a tree using the Gini impurity metric.

The general idea is to apply a *divide-and-conquer* approach, so first find node at the root, and then split and recurse.



Growing a classification tree, cont

Gini impurity value

Let S be a subset of examples in our input data $S \subseteq \{(\mathbf{x}_1, y_1), .., (\mathbf{x}_n, y_n)\}$, for $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathcal{Y}$ with $|\mathcal{Y}| = K \geq 2$. In what follows, $p_k(S)$ is the proportion of examples in S that belong to class k.

$$Gini(S) = \sum_{k=1}^{K} p_k(S)(1 - p_k(S)) = 1 - \sum_{k} (p_k(S))^2$$

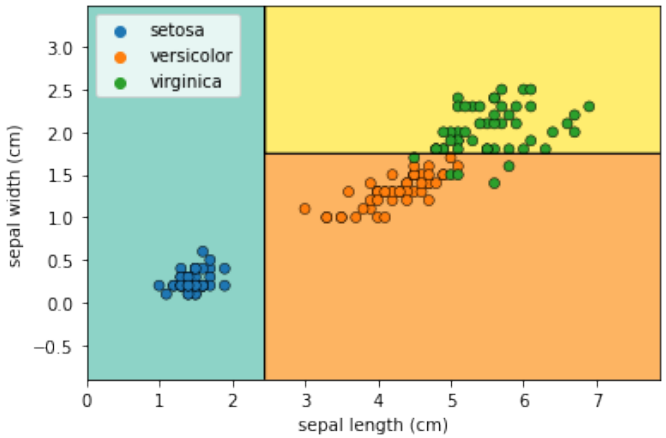
If $K = 3$, then:

Distribution	Gini impurity value
$(1.0, 0.0, 0.0)$	$0 = 1 * 0 + 0 * 1 + 0 * 1$
$(0.5, 0.3, 0.2)$	$0.62 = 0.5^2 + 0.3 * 0.7 + 0.2 * 0.8$
$(0.3, 0.3, 0.3)$	$0.6667 = 1/3 * 2/3 + 1/3 * 2/3 + 1/3 * 2/3$

- The more "pure" the distribution, the lower the Gini impurity value
- The more "uniform", the higher the Gini impurity value

Growing a classification tree, cont

Finding greedy-optimal split node



Find pair (v^*, s^*) where v^* is an input feature, and $s^* \in \mathbb{R}$ s.t.

$$v^*, s^* := \arg\min_{v,s} \left[\frac{|S_{v \leq s}|}{|S|} Gini(S_{v \leq s}) + \frac{|S_{v > s}|}{|S|} Gini(S_{v > s}) \right]$$

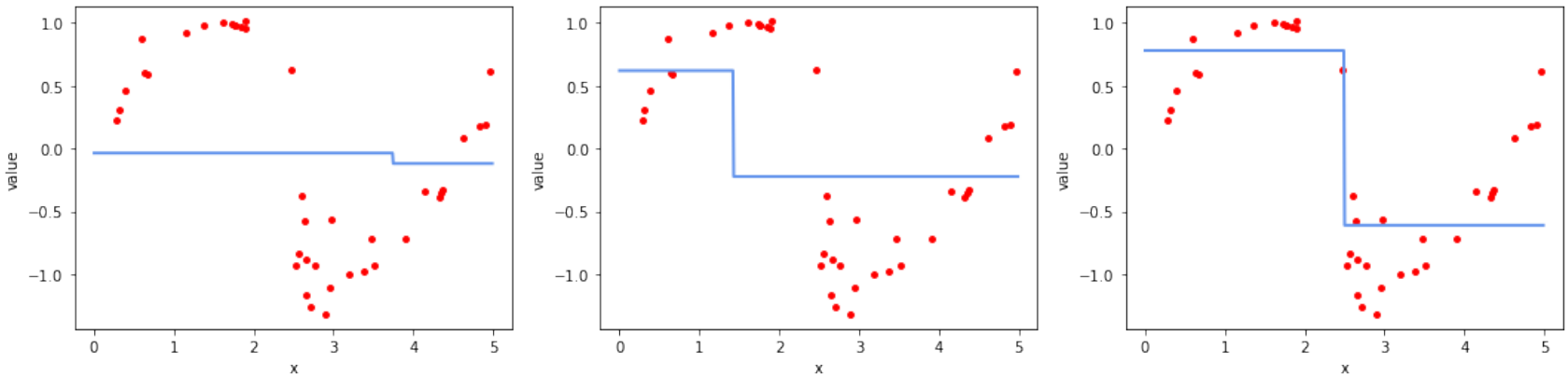
In this case a good choice is v^*=`sepal_length`, $s^* = 2.5$ for example.

Growing a regression tree

Similar to classification, but minimize the *sum of squared errors* relative to the average target value in each induced partition; namely find pair (v^*, s^*) where v^* is an input feature, and $s^* \in \mathbb{R}$ s.t.

$$v^*, s^* := \arg\min_{v,s} \left[SSE(S_{v \leq s}) + SSE(S_{v > s}) \right]$$

where $SSE(S) := \sum_{i:(\mathbf{x}_i, y_i) \in S} (y_i - avg(S))^2$



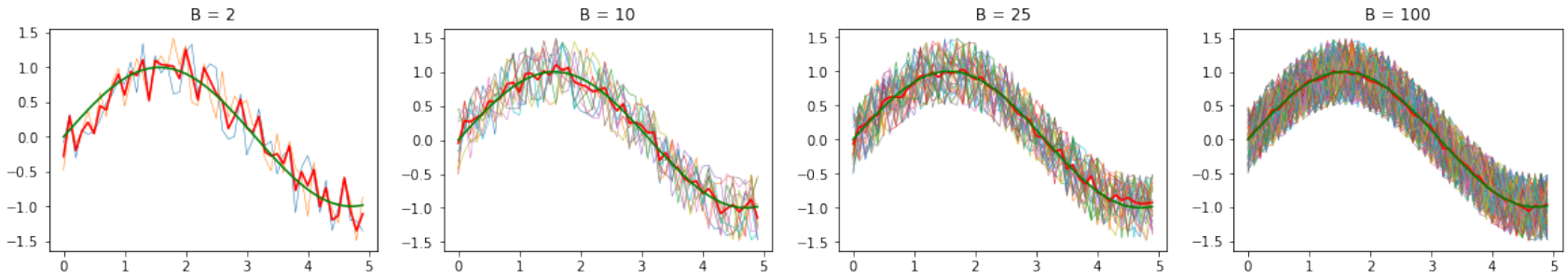
Growing trees, other considerations

- Categorical variables, multi-way splits
- When to stop?
  - Hyperparameters: `max_depth`, `min_samples_split`, ...
- Overfitting
  - Pruning

From trees to forests

Bootstrap aggregation or "bagging"

To reduce the variance of an estimator, it is helpful to **average** estimates from **independent** draws from the data



Assuming each Y_b is an unbiased estimate of target value y:

$$\mathbb{E}[(y - Y_b)^2] = Var[Y_b]$$

$$\mathbb{E}[(y - \frac{1}{B}\sum_b Y_b)^2] = \frac{1}{B^2}\sum_b Var[Y_b] \qquad \textit{(if all } Y_b \textit{ independent)}$$

$$= \frac{1}{B} Var[Y_b] \qquad \textit{(if all } Y_b \textit{ have same variance)}$$

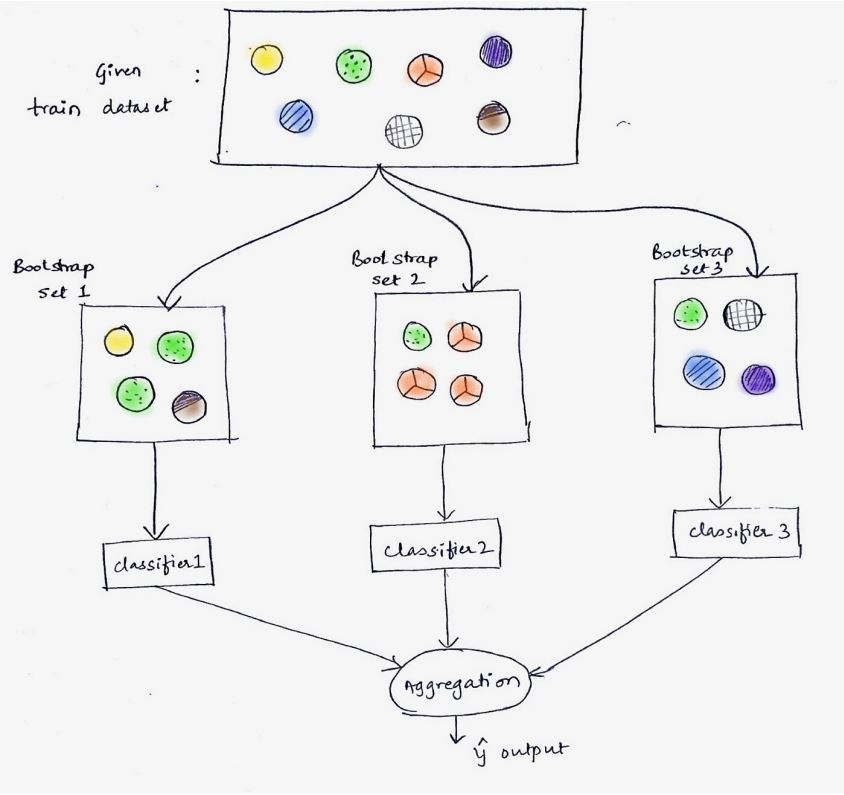
Bagging

So, the idea of **bagging** is to combine the predictions of a high-variance predictor trained on independent **bootstrap samples** from the same dataset, to make the combined predictions more robust (i.e. with lower variance) and, therefore, more accurate.

Trees typically suffer from high variance (= overfitting) so they seem an ideal candidate to be aggregated.

Bagging, cont.

The bootstrapped dataset is created by randomly selecting samples from the original dataset. Repeat selections are allowed (*sampling with replacement*).



Any samples that are not chosen for the bootstrapped dataset are placed in a separate dataset called the **out-of-bag dataset** (OOB).

Random forests

Generating diversity

We want to generate a diverse collection of trees that are slightly better-than-random, and are as indepent from each other as possible.

So, we "inject" stochasticity in the tree building process:

- Sample the training dataset (bootstrap samples)
- Manipulate features
- Manipulate targets
- Change learning parameters
- ...

or combinations of the above

Random forests

Pseudocode

Hyperparameters:

- B: nr of bootstrap samples (`n_estimators`)
- N: size of bootstrap samples (`max_samples`)
- m: nr of features to consider at each split (`max_features`)

Algorithm 1: Random forest

---

for $b \leftarrow 1$ **to** B **do**

  Draw bootstrap sample D_b of size N by sampling with replacement

  Grow tree T_b on D_b, by recursively adding nodes by

    - Select m variables at random from the d features available

    - Pick the best variable/split point according to Gini/SSE criterion

    - Split current node into two children according to variable/split found

end

Output forest $\{T_1, \ldots, T_B\}$

To make a **prediction** on new test example $\mathbf{x}$:

- For classification: output class probabilities or majority vote among $\{T_1(\mathbf{x}), \ldots, T_B(\mathbf{x})\}$
- For regression: output average prediction: $\frac{1}{B} \sum_b T_b(\mathbf{x})$

The **OOB error** is an estimation of generalization error that can be used as validation error to select appropriate values for the hyperparameters; as a direct consequence, **there is no need for cross-validation** for model selection (hyperparameter tuning).

- Each tree is constructed using a different bootstrap sample from the original data. About one-third[1] of the cases are left out of the bootstrap sample and not used in the construction of the b-th tree.
- Put each case left out in the construction of the b-th tree down the b-th tree to get a classification. In this way, a test set classification is obtained for each case in about one-third of the trees.
- Average all these predictions (regression) or take the majority vote (classification) to compute the oob error for each example, and average accross examples.

[1] In a dataset with n examples, each has a probability $p(n) = 1 - (1 - 1/n)^n$ of being selected at least once; now $\lim_{n\to\infty} p(n) = 1 - e^{-1} \approx 0.632$; in fact for small values e.g. $n = 20$, $p(20) = 0.641$ already.

If a random forest contains many trees, it can be difficult to comprehend what the model is doing (not interpretable by a person). - Variable importance plot add interpretability to the model

1. Gini-based variable importance

   Add gini impurity gains for variables in splits in each tree in the forest, sort variables by their sum.

2. Permutation-based variable importance

   For each variable, permute values and compute difference in OOB error metrics before and after permutation. If variable is important, then accuracy in the permuted copy should decrease. Sort variables by this difference.

Permutation-based more reliable, but slower; gini-based is biased towards categorical variables with many splits.[2]

[2] If interested, you can read this article.

Main idea: two examples falling into the same leaf of a decision tree supports the fact that the examples are *somewhat similar*

Expanding this idea to a large and diverse set of trees (i.e. a **random forest**), we can build a $n \times n$ similarity matrix; that is, we build a matrix where each entry (i, j) corresponds to the fraction of trees in the forest such that examples $\mathbf{x}_i, \mathbf{x}_j$ end up in the same leaf.

The resulting matrix can be used in distance/similarity-based methods e.g. clustering or kernel methods, or apply PCA to obtain numeric representation of your data (see Multidimensional scaling).

Random forests

Imbalanced data (classification)

In classification, it can be the case that classes are very imbalanced. In this case, there are several techniques to deal with this:

1. Use of F-score or related metric (other than accuracy) for model selection
2. Under-sampling majority class when building bootstrap samples
3. Over-sampling minority class when building bootstrap samples
4. Use of weights (loss matrix) when computing errors; explore the `class_weight` parameter of a random forest

You can also explore the imblearn library that contains `scikit-learn`-compatible models for imbalanced classification.

Random forests

In summary ..

- Random forests are fast, accurate, and *relatively* interpretable
- Can be easily parallelized
- They offer methods for imbalanced classification problems
- No need for cross-validation for tuning hyperparameters, use OOB instead
- Can naturally incorporate both continuous and categorical variables
- No scaling necessary