Data Science: Principles and Practice

Lecture 4: Ensemble Learning

Ekaterina Kochmar

UNIVERSITY OF CAMBRIDGE

Wisdom of the crowd

- The collective opinion of a group of individuals rather than that of a single expert

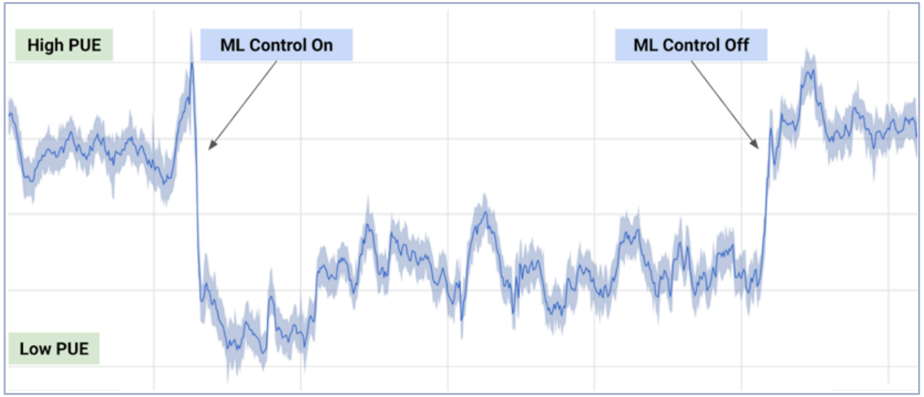
Wisdom of the crowd

- The collective opinion of a group of individuals rather than that of a single expert
- *Classic example*: point estimation of a continuous quantity
- 1906 country fair in Plymouth: 800 people participated in a contest to estimate the weight of an ox. Median guess of 1207 pounds accurate within 1% of the true weight of 1198 pounds (Francis Galton)

Wisdom of the crowd

- The collective opinion of a group of individuals rather than that of a single expert
- *Classic example*: point estimation of a continuous quantity
- 1906 country fair in Plymouth: 800 people participated in a contest to estimate the weight of an ox. Median guess of 1207 pounds accurate within 1% of the true weight of 1198 pounds (Francis Galton)
- Crowd's individual judgments can be modelled as a probability distribution of responses with the median centred near the true value of the quantity to be estimated
- **Applications**: crowdsourcing, social information sites (Wikipedia, Quora, Stack Overflow), decision-making (trial by jury), sharing economy self-regulating platforms (Uber, Airbnb)

Ensemble-based models in practice



Data centres control by DeepMind using ensembles of neural networks

A number of top teams in the competition (https://netflixprize.com) used ensembles

Data Science: Principles and Practice

01 Simple voting classifiers using hard and soft voting strategies

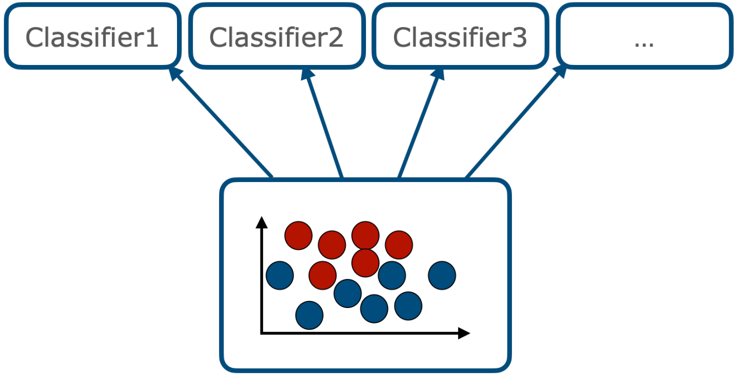
02 Bagging and pasting ensembles (Random Forests)

03 Boosting (AdaBoost, Gradient Boosting)

04 Application to classification and regression problems

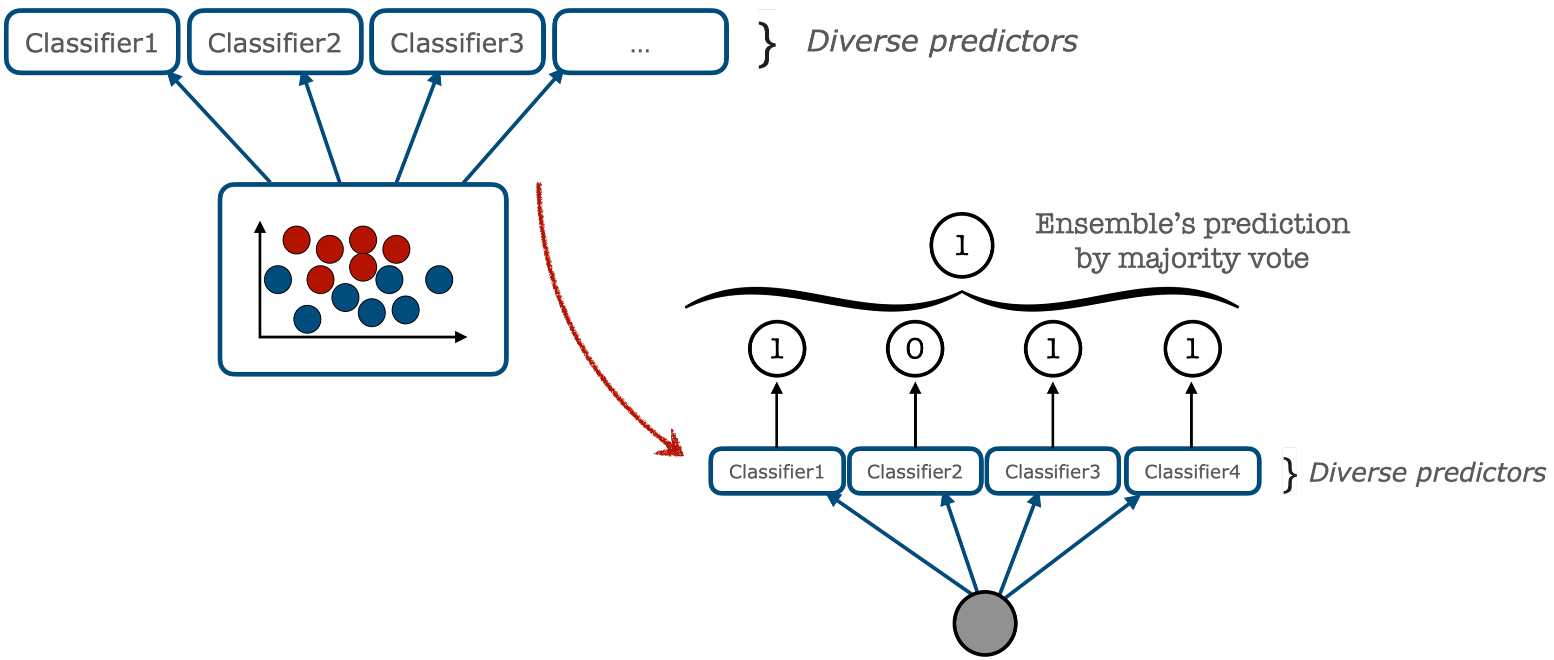
05 Practical 3

Voting classifiers



} *Diverse predictors*

Hard voting strategy

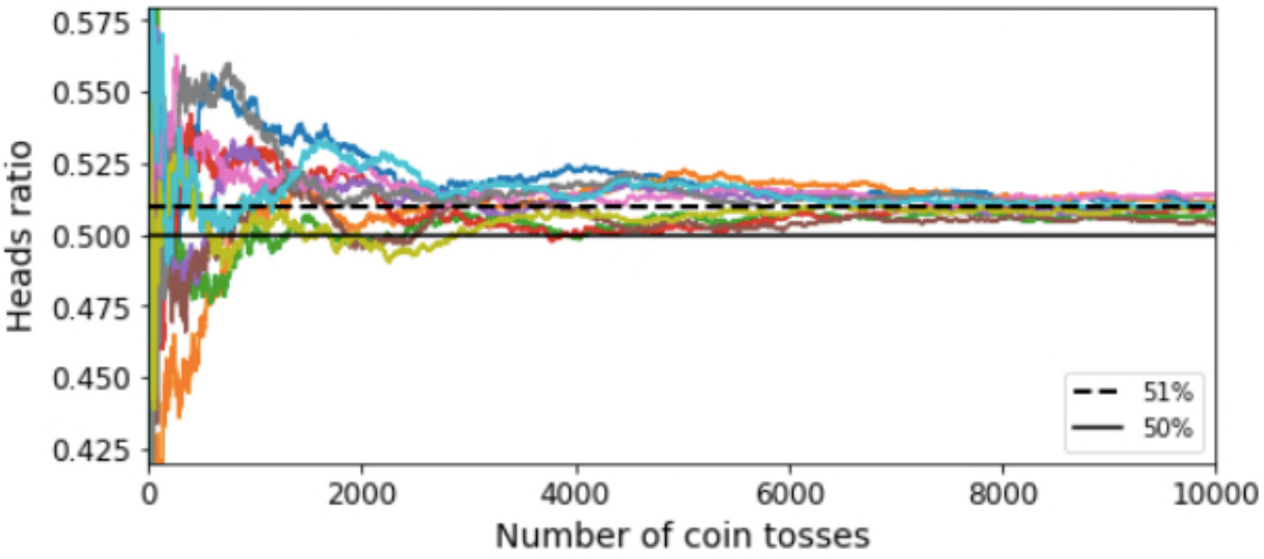


Voting classifiers

- Even when individual voters are *weak learners* (hardly above random baseline), the ensemble can still be a *strong learner*
- **Condition**: individual voters should be sufficiently diverse, i.e. make different (uncorrelated) errors
- Hard to achieve in practice as classifiers are usually trained on the same data
- Why does this work?

Coin example

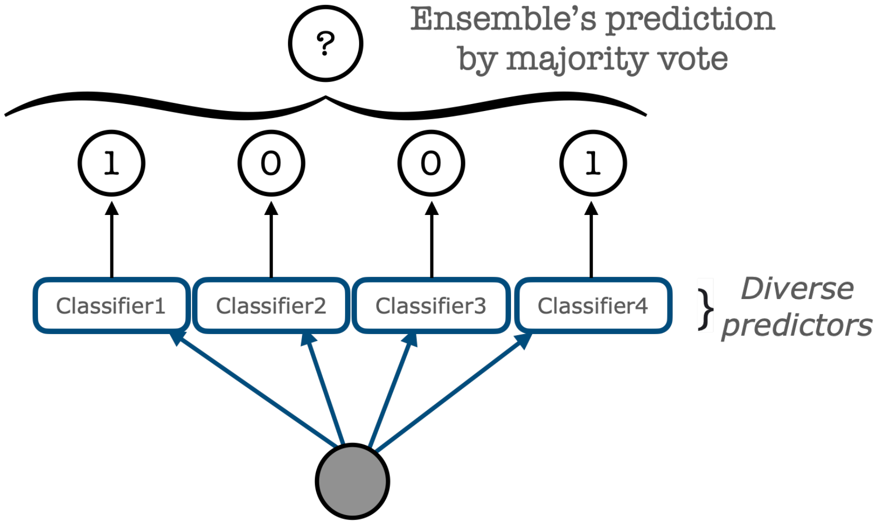
A slightly biased coin: 51% chance of heads



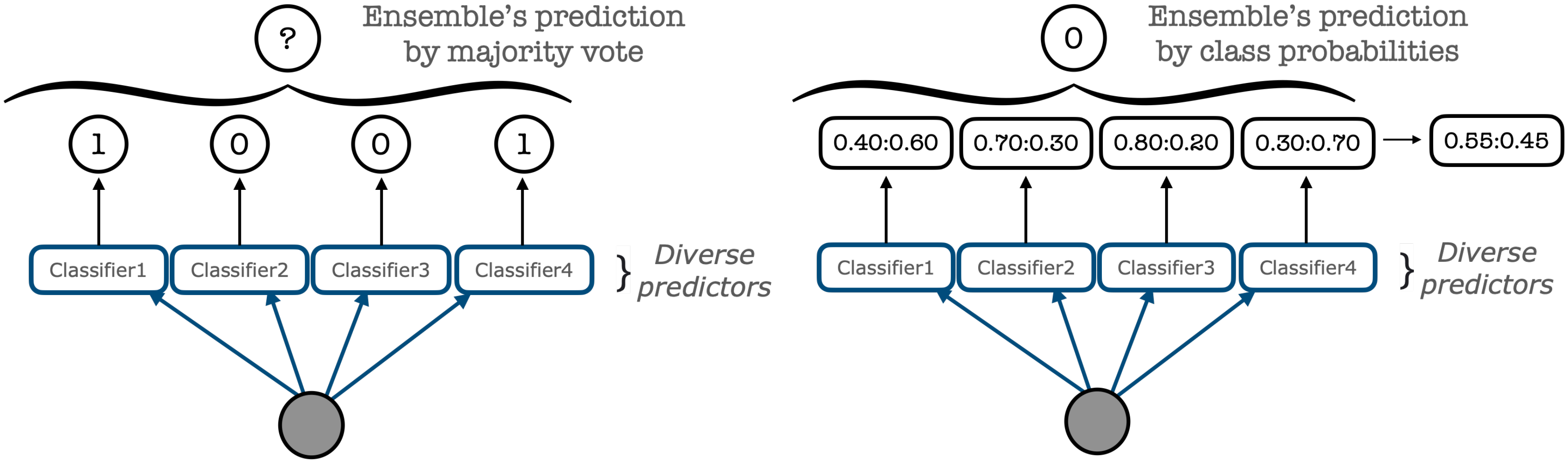
Coin example

- *Law of large numbers*: over the large number of tosses the ratio of heads gets closer to the probability of heads (51%)
- The probability of obtaining the majority of heads after 1,000 tosses of this coin approaches 73%; after 10,000 tosses – 97%
- $\Rightarrow$ If you had 1,000 independent classifiers, each of which is only slightly more accurate than random guessing, you can hope to achieve ~73%

Hard vs soft voting



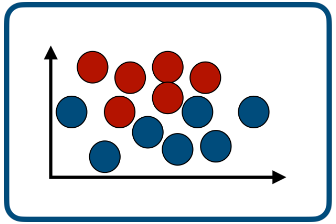
Hard vs soft voting



Bagging and Pasting

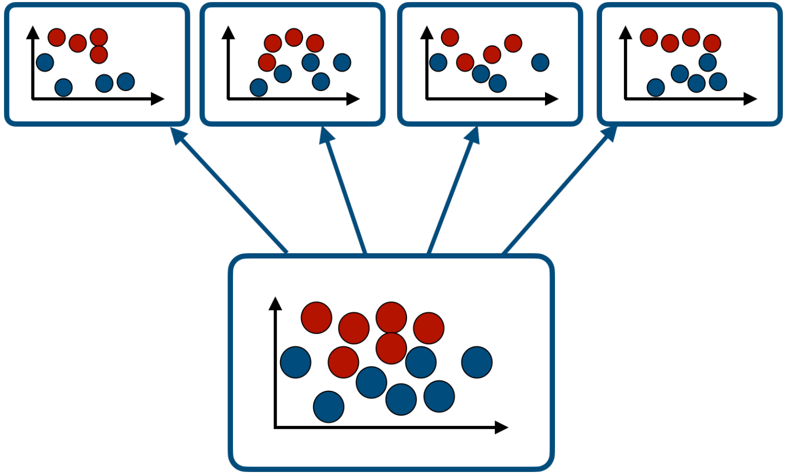
- One way to ensure that the classifiers' decisions are independent is to use very different training algorithms
- Another way is to train the predictor algorithms on different random subsets of the training data:
  - with **bagging** (**bootstrap aggregating**) you are sampling *with* replacement
  - with **pasting** you are sampling *without* replacement
- Both strategies allow the predictors to be trained in parallel

Ensembles using bagging



Full training set

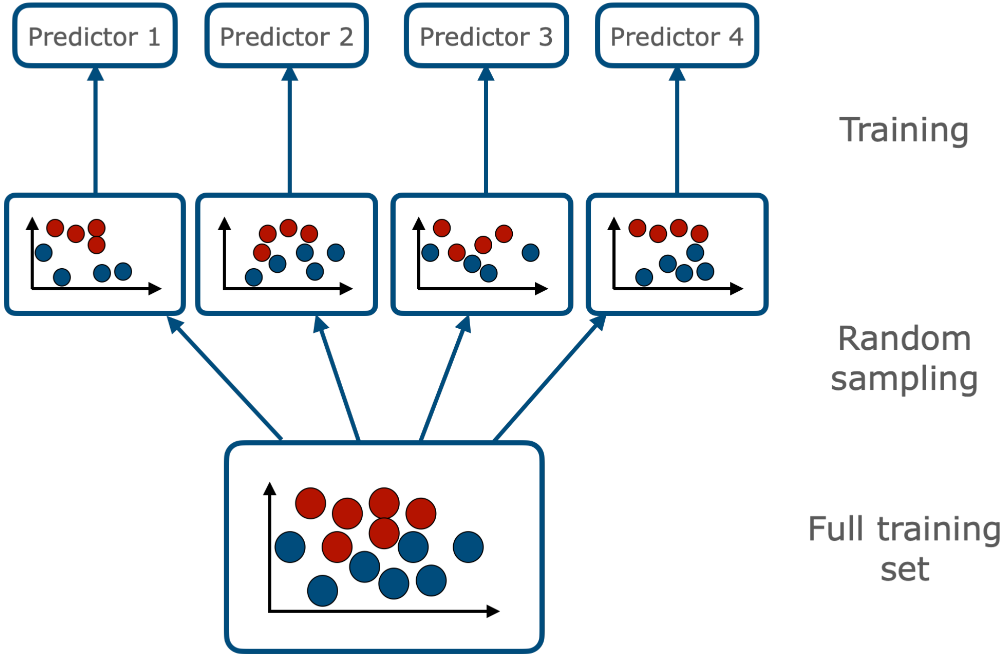
Ensembles using bagging



Random sampling

Full training set

Ensembles using bagging



At prediction time, the ensemble makes a prediction, e.g. by taking the statistical mode (the most frequent prediction) from the individual predictors

Out-of-bag evaluation

- Bagging samples *m* training instances, where *m* is the size of the training set
- Only about 63% of the instances are sampled on average for each predictor
- ⇒ The other 37% are called **out-of-bag** (**oob**) instances
- Each predictor can be evaluated on the oob instances without any need for a separate validation set or cross-validation

The bias / variance trade-off

Model’s generalisation error can be expressed as the sum of three different errors:

01 **Bias** is due to wrong assumptions about the data: e.g. linear instead of quadratic. A high bias model is likely to underfit the training data

The bias / variance trade-off

Model's generalisation error can be expressed as the sum of three different errors:

01 **Bias** is due to wrong assumptions about the data: e.g. linear instead of quadratic. A high bias model is likely to underfit the training data

02 **Variance** is due to the model's excessive sensibility to small variations in the training data. A model with many degrees of freedom (e.g., a high-degree polynomial) is likely to have high variance ⟶ overfit the training data

The bias / variance trade-off

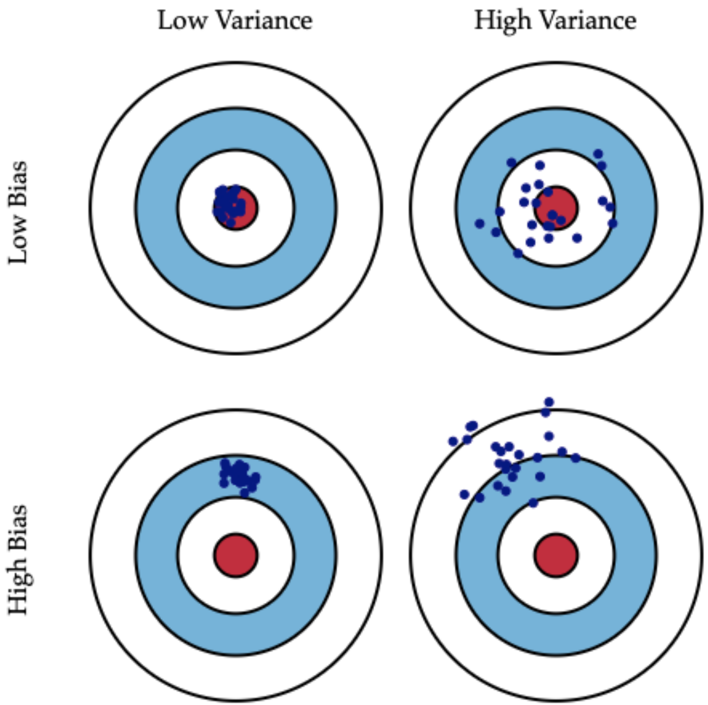
Model's generalisation error can be expressed as the sum of three different errors:

01 **Bias** is due to wrong assumptions about the data: e.g. linear instead of quadratic. A <u>high bias</u> model is likely to <u>underfit</u> the training data

02 **Variance** is due to the model's excessive sensibility to small variations in the training data. A model with many degrees of freedom (e.g., a high-degree polynomial) is likely to have <u>high variance</u> $\longrightarrow$ <u>overfit</u> the training data

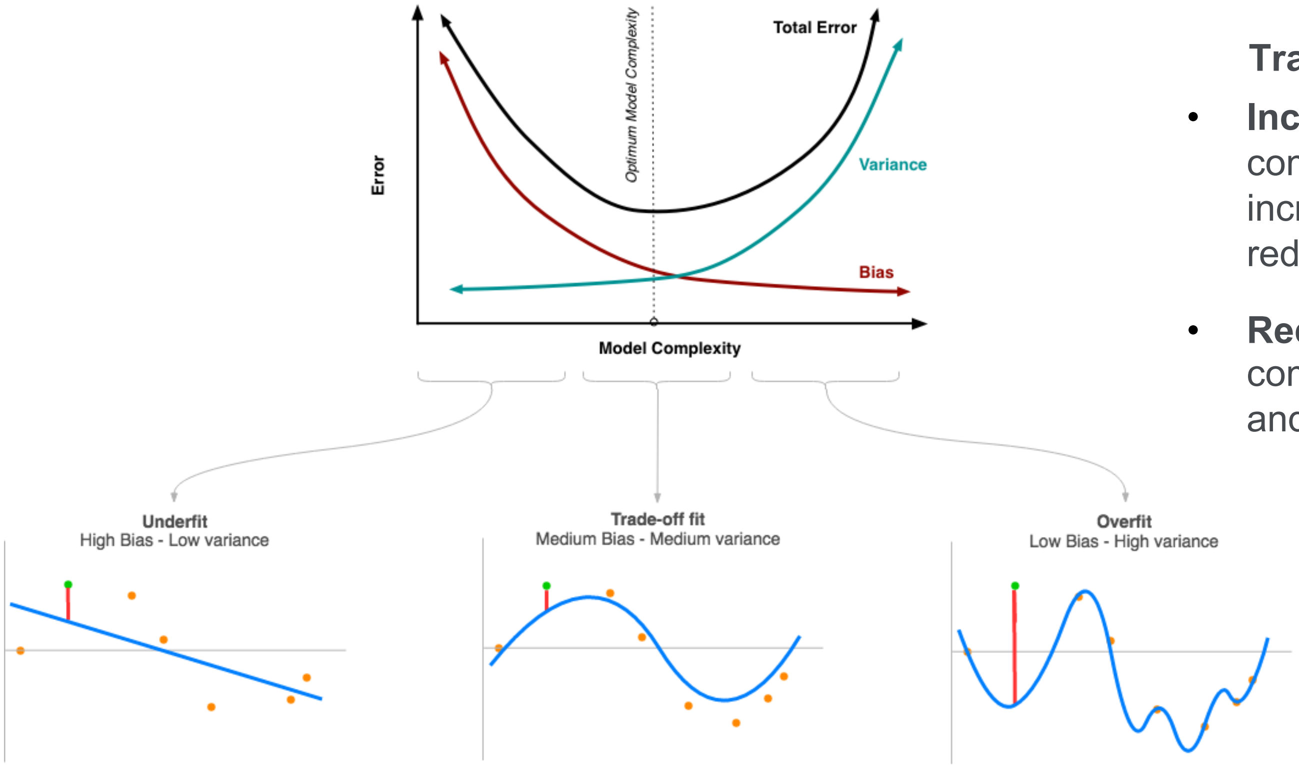
03 **Irreducible error** is due to the noisiness in the data itself (solution: clean the data, remove outliers, etc.)

The bias / variance trade-off



http://scott.fortmann-roe.com/docs/BiasVariance.html

The bias / variance trade-off



Trade-off:

- **Increasing** model's complexity will typically increase its variance and reduce bias.
- **Reducing** model's complexity increases bias and reduces variance.

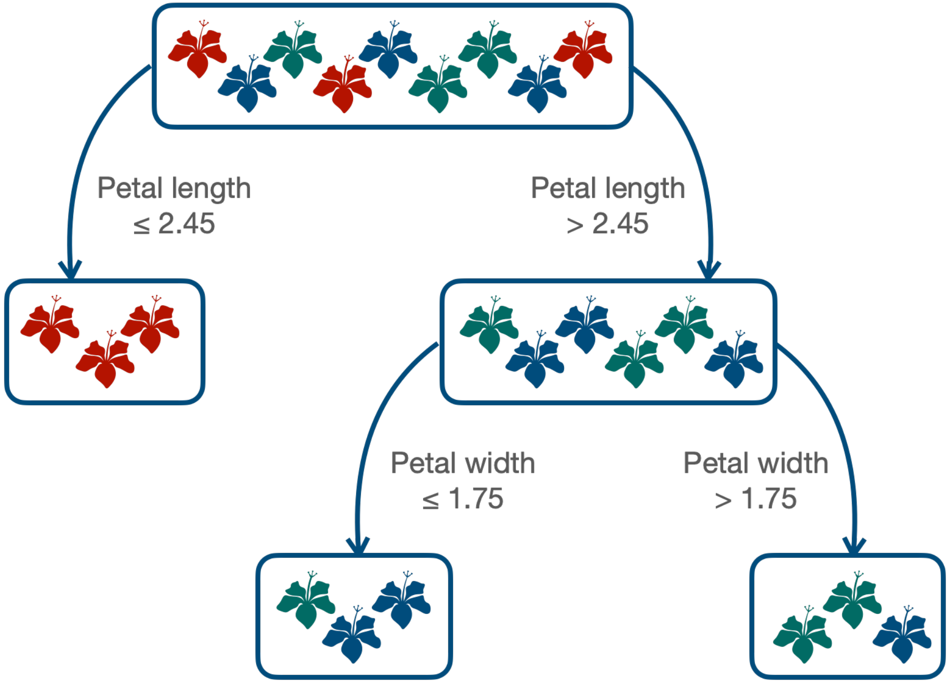
http://www.ebc.cat/wp-content/uploads/2017/02/right_fited_model.png

What about ensemble models?

- Each individual predictor may have a higher bias than if it were trained on the whole dataset
- Bagging: aggregation reduces variance while retaining the bias
- Predictors end up being less correlated, so the ensemble’s variance is reduced
- Bagging is generally preferred as it usually results in better models

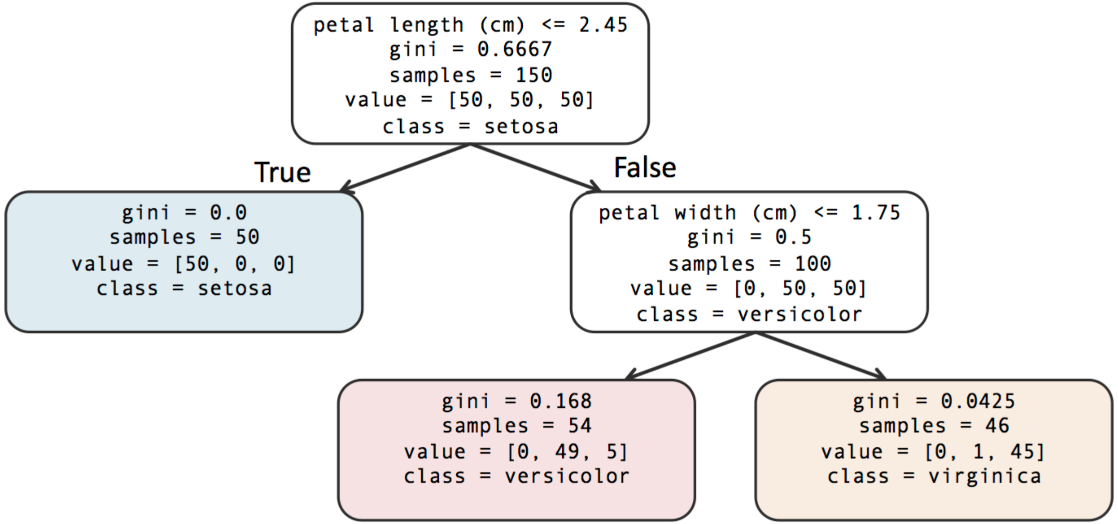
Decision Trees on the *Iris* dataset

Classifying 3 types of irises by petal length and width



Decision Trees on the *Iris* dataset

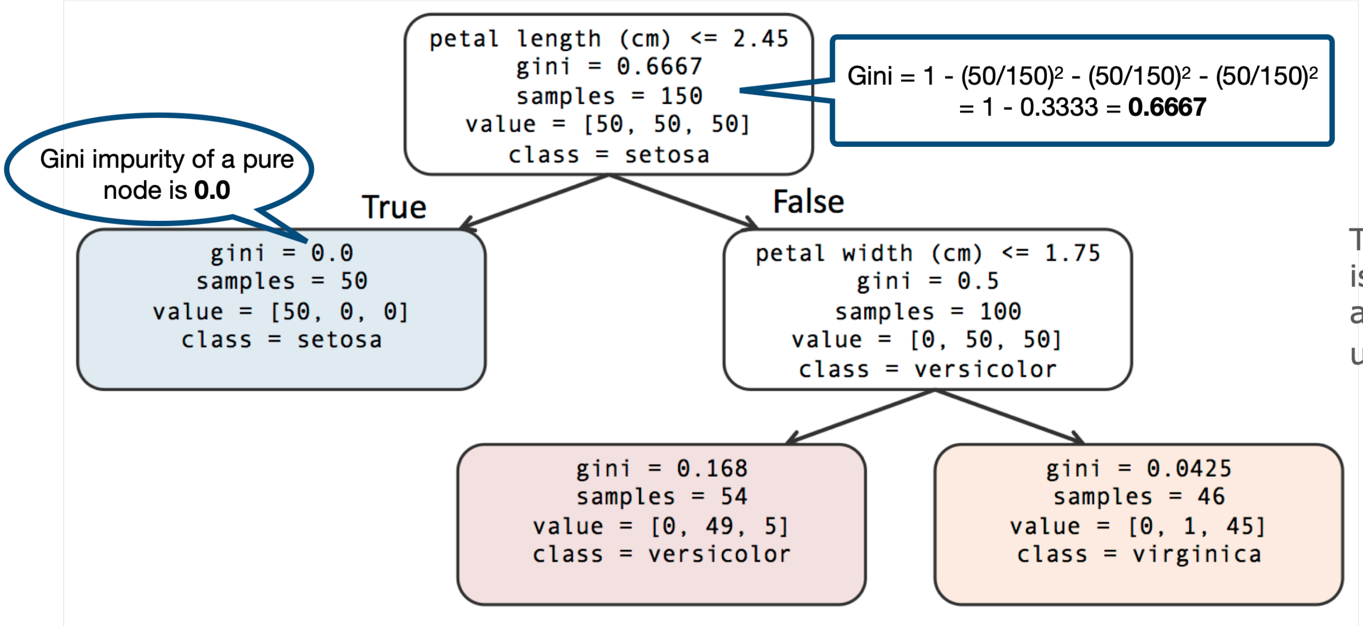
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X, y)
```



Decision Trees on the *Iris* dataset

Gini impurity (impurity of the node) = $1 - \sum_{k=1}^{n} p_{i,k}^2$

where $p_{i,k}$ is the ratio of class ***k*** instances among the training instances of the ***i***-th node



The goal of the algorithm is to minimise the impurity at every split, minimising uncertainty about the data

Decision Trees training

CART (**Classification and Regression Tree**)[1] algorithm:

- **Start:** Split the training set in two subsets using a single feature ***k*** and a threshold $\boldsymbol{t_k}$
- To select the **(*k*, $\boldsymbol{t_k}$)** pair, search for the purest subsets weighted by size
- **Cost function:** $$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}$$

where $\begin{cases} G_{left/right} & \text{measures the impurity of the left/right subset, and} \\ m_{left/right} & \text{is the number of instances in the left/right subset.} \end{cases}$

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone (1984). "Classification and Regression Trees"

Decision Trees training

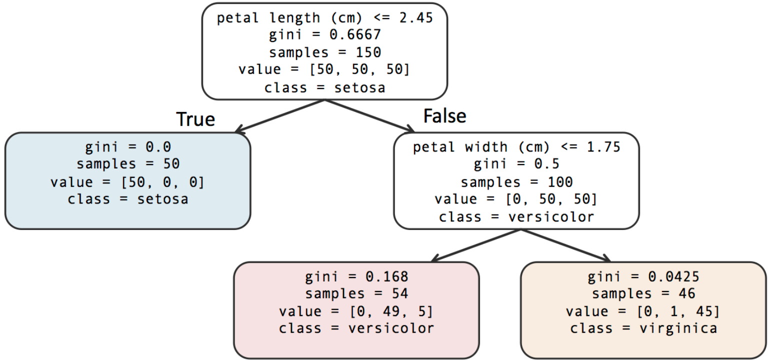
CART (**Classification and Regression Tree**)[1] algorithm:

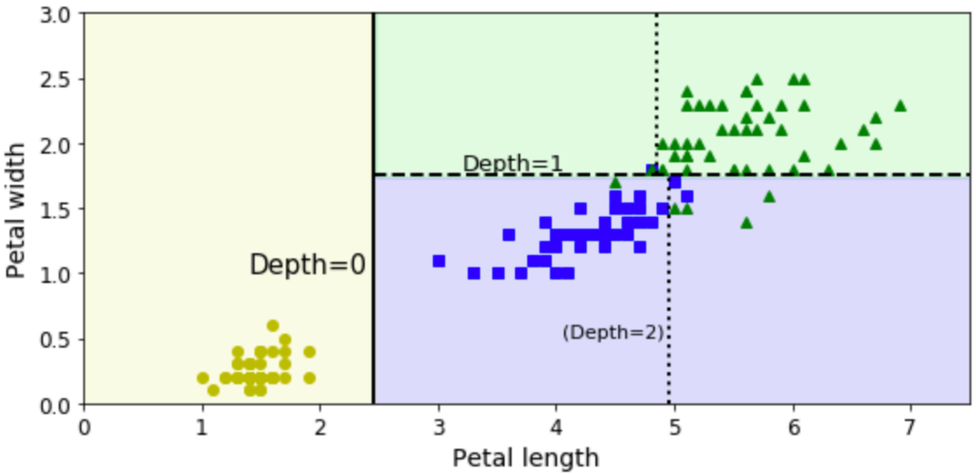
- **Start:** Split the training set in two subsets using a single feature $\boldsymbol{k}$ and a threshold $\boldsymbol{t_k}$
- To select the **($\boldsymbol{k}$, $\boldsymbol{t_k}$)** pair, search for the purest subsets weighted by size
- **Cost function:** $$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}$$

  where $\begin{cases} G_{left/right} & \text{measures the impurity of the left/right subset, and} \\ m_{left/right} & \text{is the number of instances in the left/right subset.} \end{cases}$
- **Recursion:** Apply to all subsets recursively
- **Stopping criteria:** Max depth reached, or no more splits that reduce impurity

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone (1984). “Classification and Regression Trees”

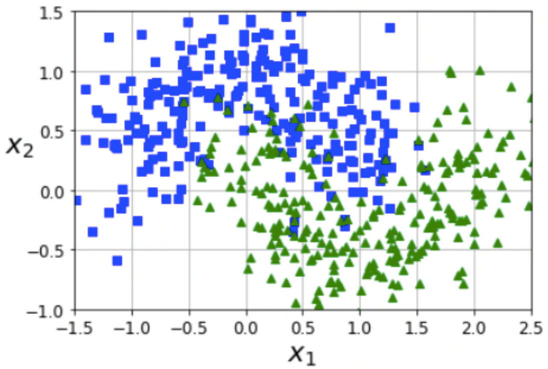
Decision Trees decision boundaries

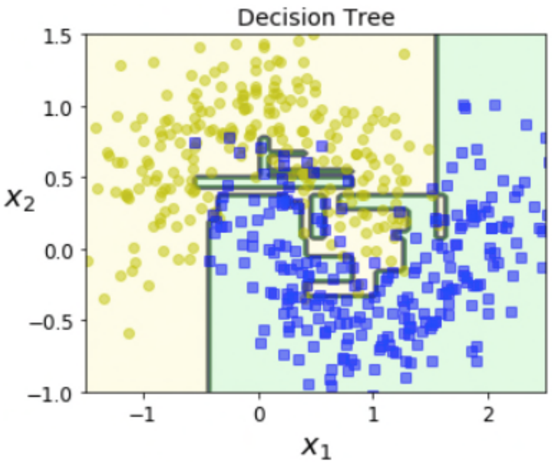




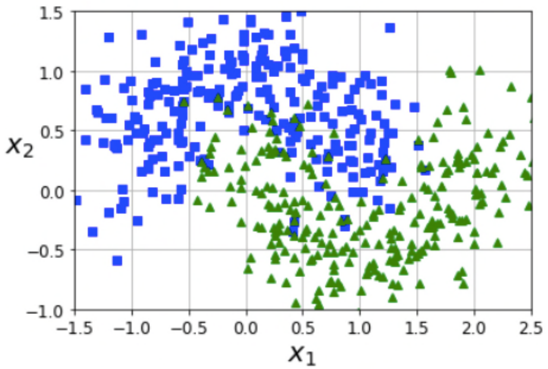
https://github.com/ageron/handson-ml

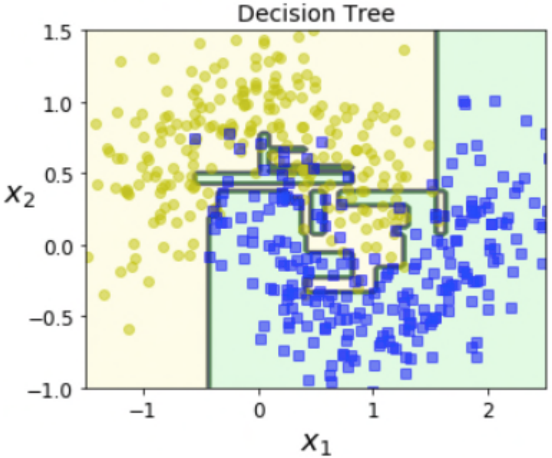
From a single tree

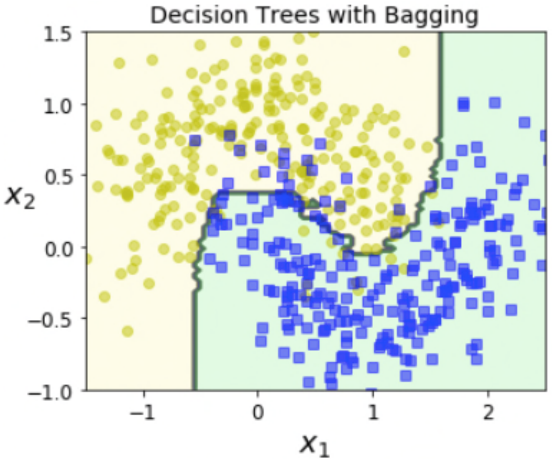




From a single tree to a forest







Random Forests classifier

- Allows you to control both how the trees are grown (i.e., the usual hyperparameters for Decision Trees) and how the ensemble is built
- **Extra randomness**: instead of searching for the very best feature to split a node on, it searches for the best feature among a random subset of features
- Trading higher bias for lower variance ⟶ overall, more generalisable
- **Extremely Randomised Trees** (**Extra-Trees**): use random thresholds for features rather than searching for the best possible thresholds ⟶ trains much faster

Feature importance

- **Importance** of each feature can be measured by looking at how much the nodes that are using a particular feature reduce impurity on the average, i.e. across all trees in the forest

- This can be used for quick assessment of which features matter most, i.e. **feature selection**

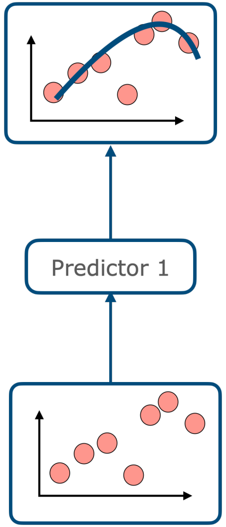
- Alternatively, further randomness can be introduced by training on random subsets of the features (supported in `sklearn`) using:

    - **Random Patches** method when sampling both training instances and features
    - **Random Subspaces** method when keeping all training instances but sampling features

Boosting

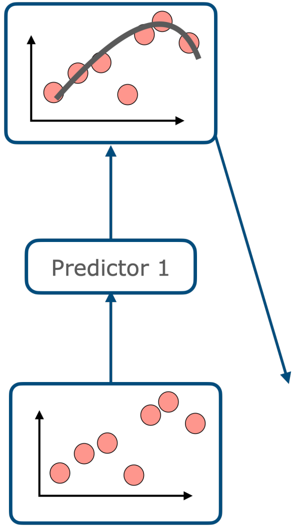
- **Boosting** (or **hypothesis boosting**) is an approach that can combine several weaker learners into a stronger learner
- Train predictors **sequentially**, so that each next classifier tries to correct the errors from its predecessor
- Most popular approaches – `AdaBoost` and `Gradient Boosting`

AdaBoost



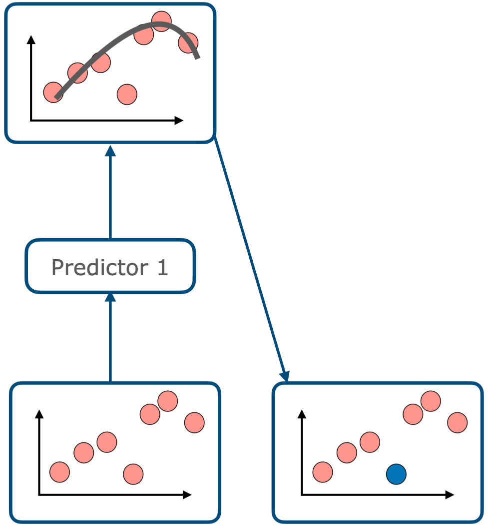
- Start with the first predictor

AdaBoost



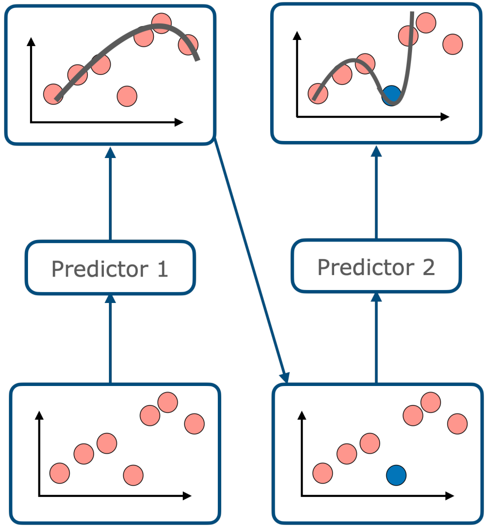
- Start with the first predictor
- Train and estimate its performance

AdaBoost



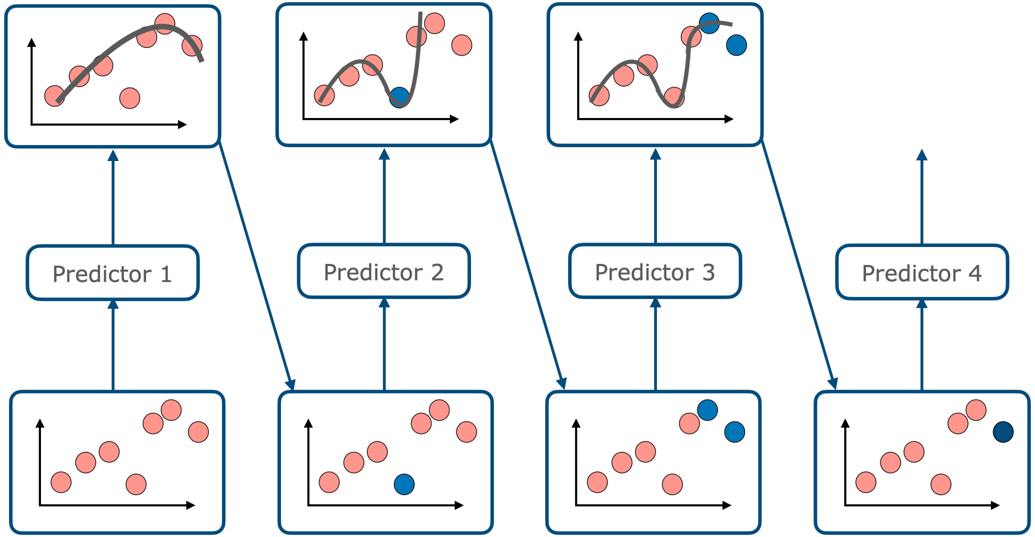
- Start with the first predictor
- Train and estimate its performance
- Increase relative weight of misclassified training instances

AdaBoost



- Start with the first predictor
- Train and estimate its performance
- Increase relative weight of misclassified training instances
- Train a new predictor on updated weights and make new predictions

AdaBoost



- Start with the first predictor
- Train and estimate its performance
- Increase relative weight of misclassified training instances
- Train a new predictor on updated weights and make new predictions
- Repeat until stopping criteria are satisfied

AdaBoost

- **Initialisation:** $w^{(i)} = \frac{1}{m}$ for each instance, where ***m*** is the number of instances
- **Error rate:** $r_j = \frac{\sum_{\hat{y}_j^{(i)} \neq y^{(i)}} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$ where $\hat{y}_j^{(i)}$ is the ***j***-th classifier prediction on ***i***-th instance
- **Predictor's weight:** $\alpha_j = \eta log \frac{1-r_j}{r_j}$ (higher for more accurate ones), where η is the learning rate
- **Update:** $$w^{(i)} = \begin{cases} w^{(i)}, & \text{if } \hat{y}_j^{(i)} = y_j^{(i)} \\ w^{(i)} exp(\alpha_j), & \text{if } \hat{y}_j^{(i)} \neq y_j^{(i)} \end{cases}$$

  all instances' weights normalised by $\sum_{i=1}^{m} w^{(i)}$

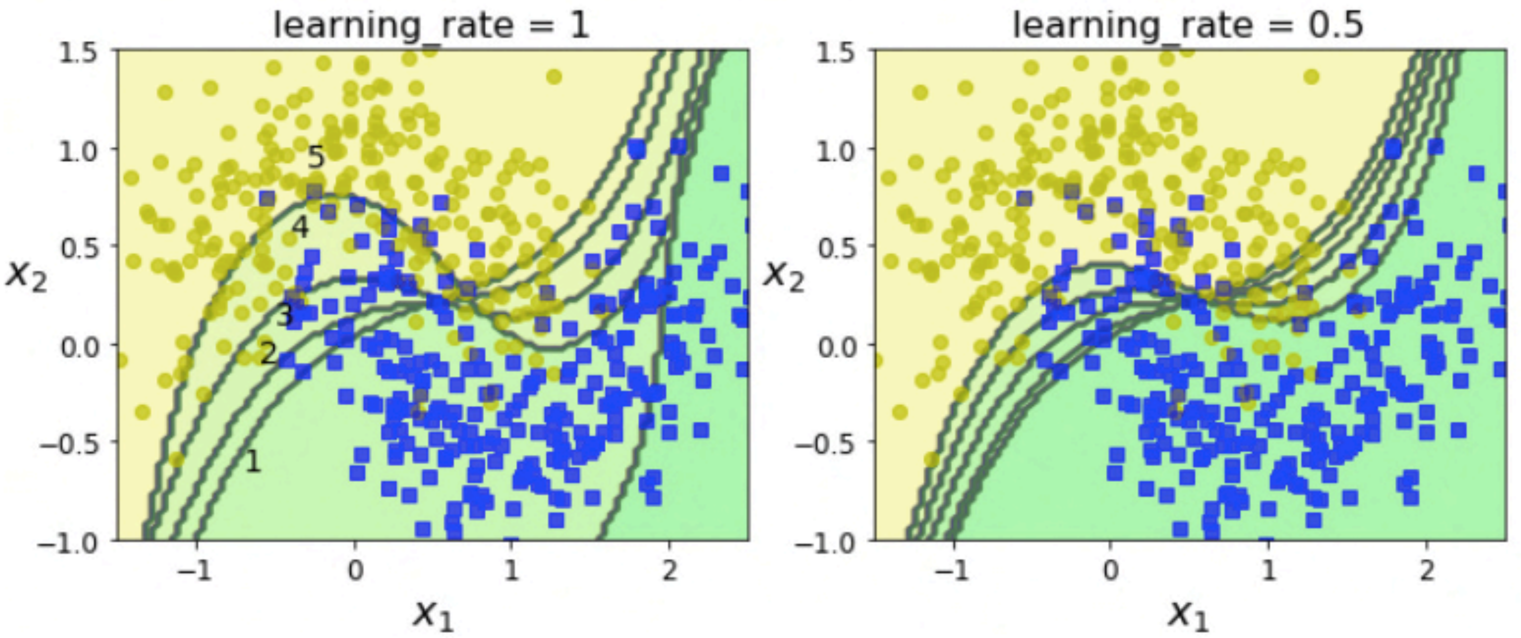
AdaBoost

- **Stopping criteria:** a perfect predictor is found, or the predefined number of predictors in the ensemble is reached

- **At prediction time:** $$\hat{y}(x) = argmax_k \sum_{j=1;\hat{y}_j(x)=k}^{N} \alpha_j$$

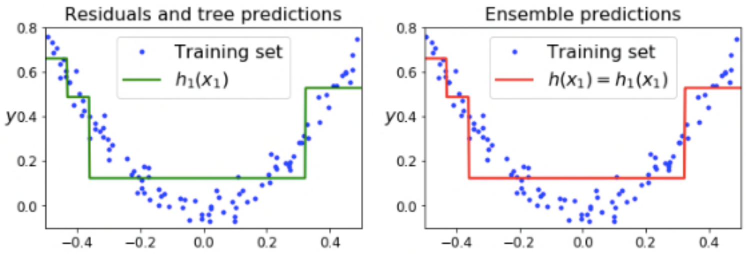
where ***N*** is the number of predictors

AdaBoost with different learning rates



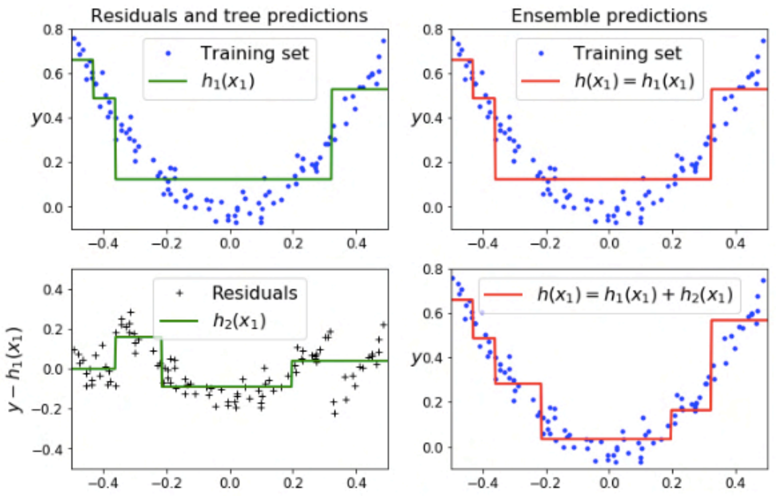
Gradient Boosting

Underlying idea: train predictors on the predecessor's residual errors



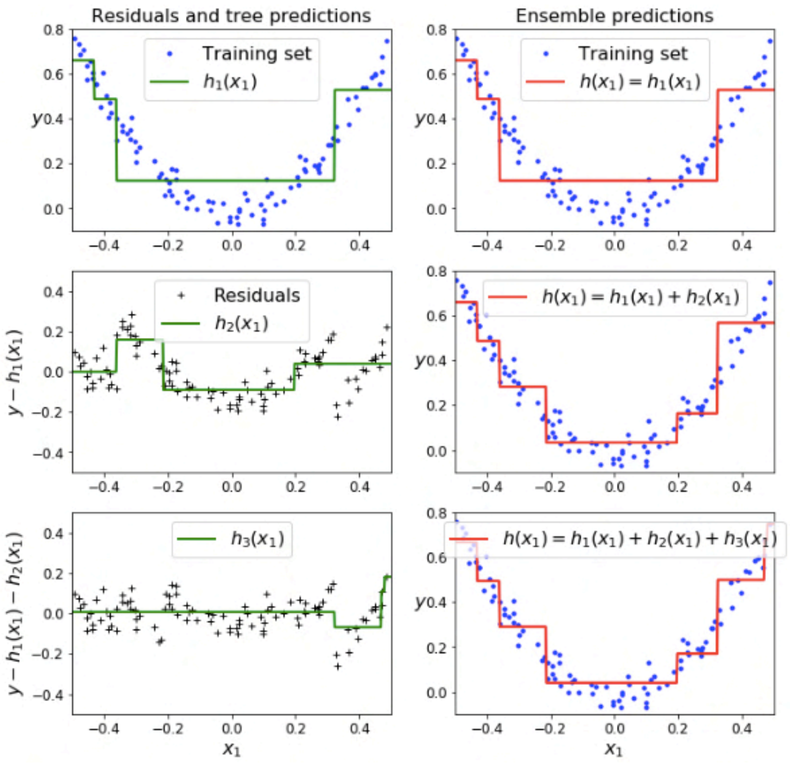
Gradient Boosting

Underlying idea: train predictors on the predecessor's residual errors



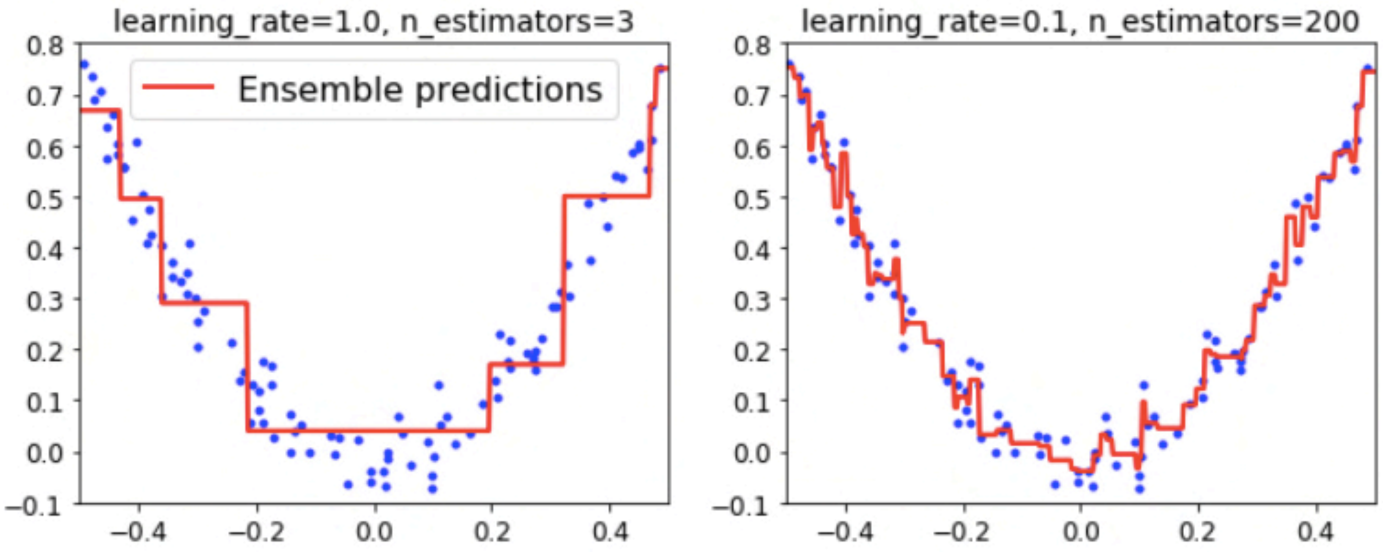
Gradient Boosting

Underlying idea: train predictors on the predecessor's residual errors



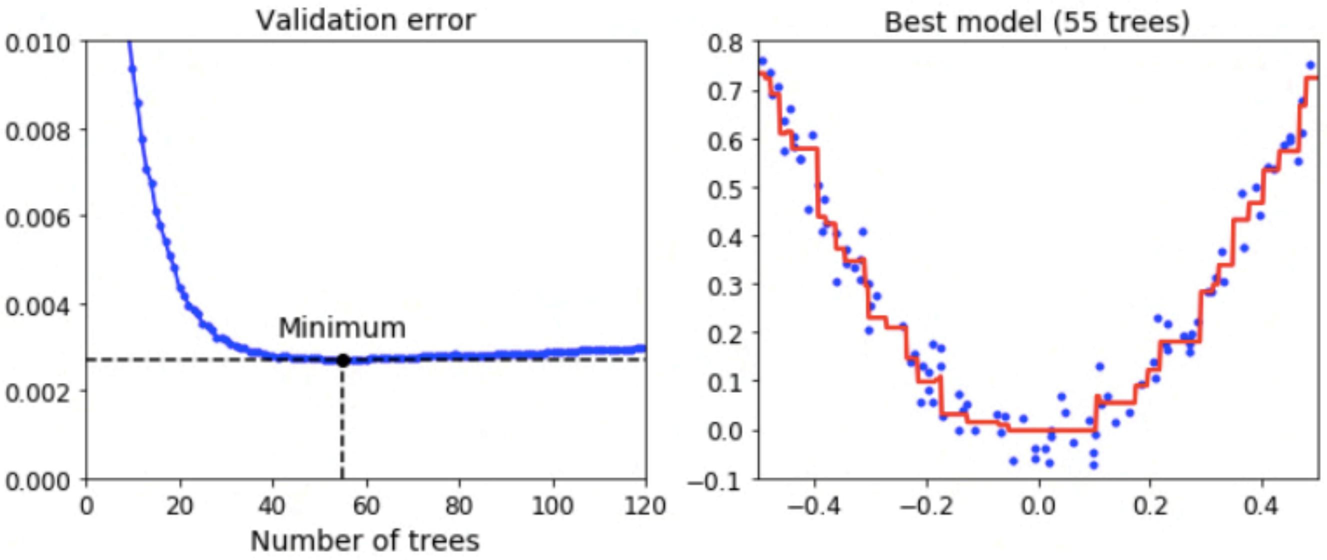
Learning rate

The learning rate scales the contribution of each tree: the lower the rate, the more trees you will need to include in the ensemble (but the predictions will usually generalise better)



Early stopping

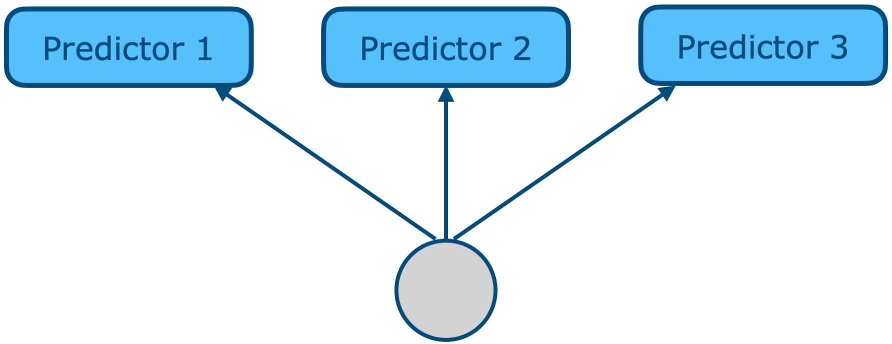
How do we know when to stop? – Estimate validation error and stop when it reached a minimum (or does not improve for a number of iterations)



Stacking

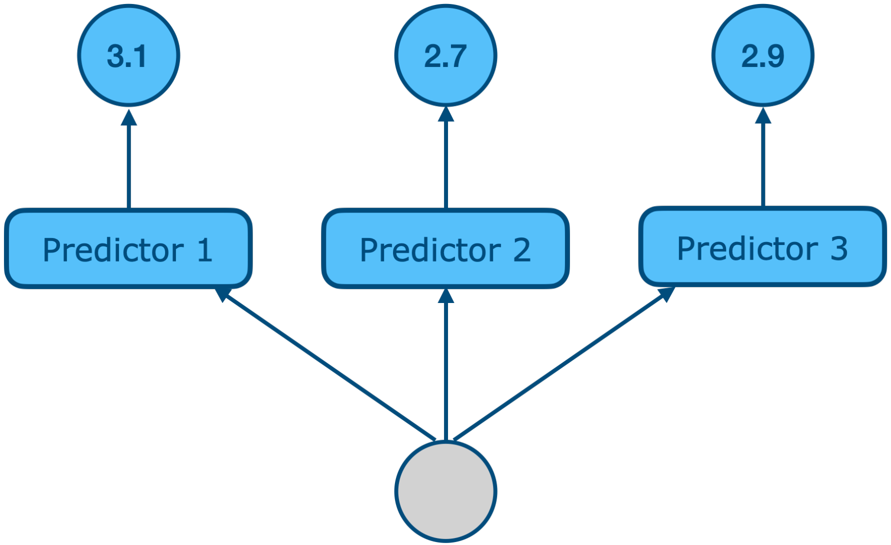
- **Stacking** (or **stacked generalisation**): instead of using a trivial function like hard voting to aggregate predictions, why not *train a model to learn* such aggregation function?
- Such a model is called **blender** or **meta-learner**

Stacking



New instance

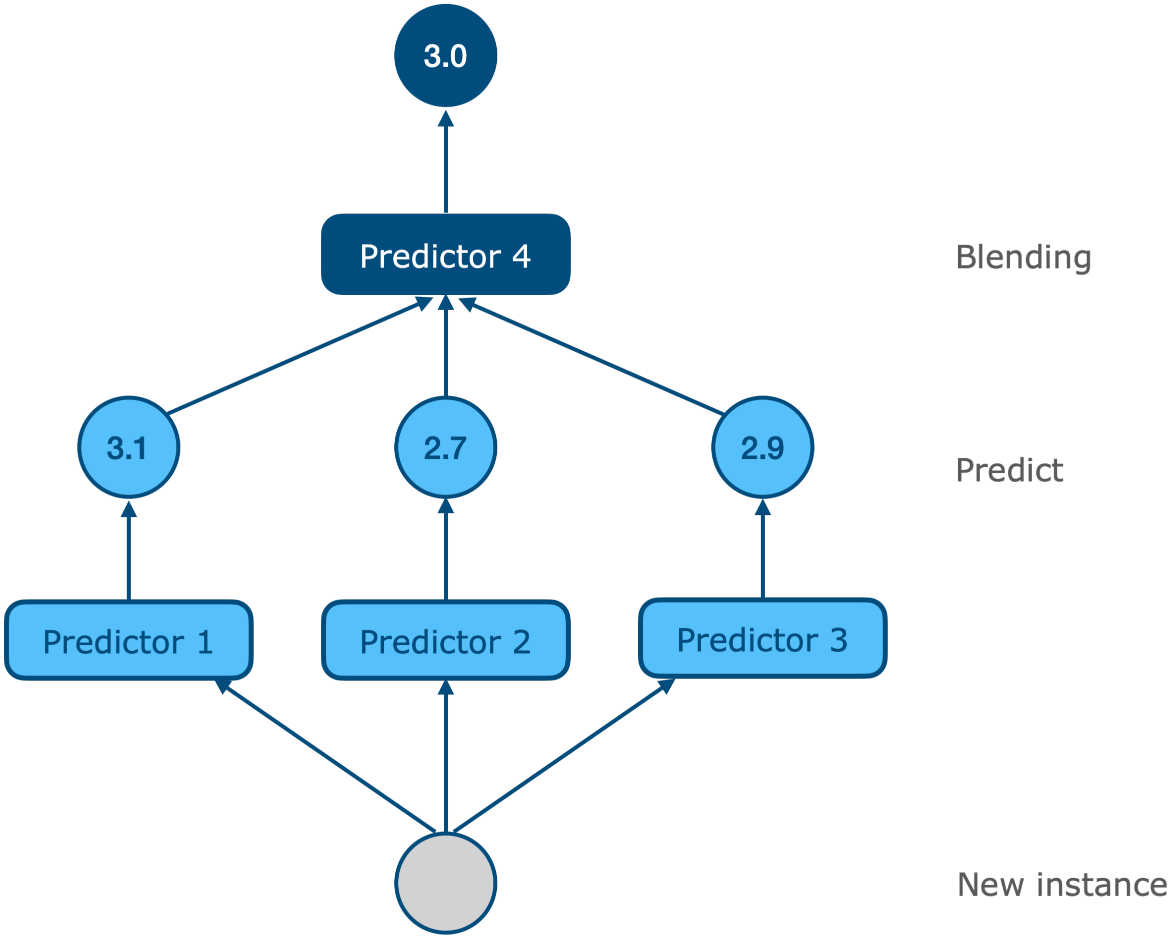
Stacking



Predict

New instance

Stacking

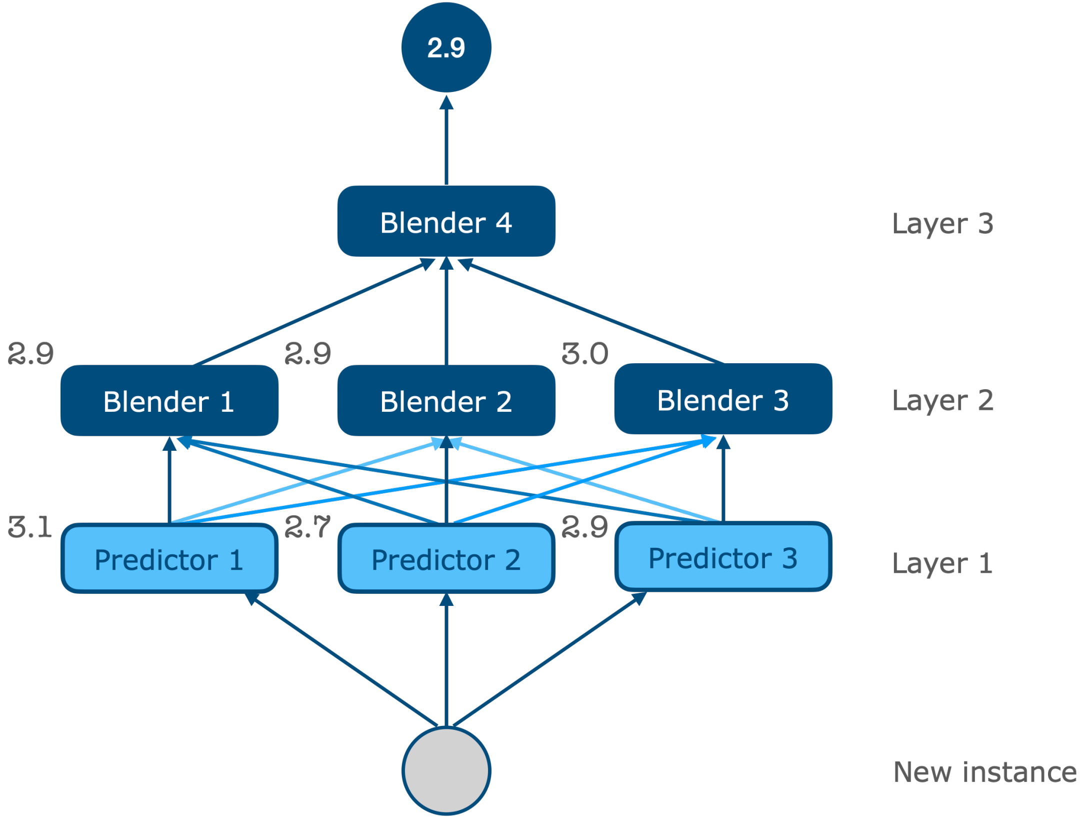


3.0
Predictor 4
Blending
3.1
2.7
2.9
Predict
Predictor 1
Predictor 2
Predictor 3
New instance

Training a stacking ensemble

- **Step 1:** Split the training set in two subsets – *subset*$_1$ and *subset*$_2$
- **Step 2**: Use *subset*$_1$ to train the predictors in the first layer
- **Step 3:** Use the first-layer predictors to make predictions on *subset*$_2$ (note that there is no “data leakage” here as predictors never saw *subset*$_2$ during training)
- **Step 4:** Use these predictions from the first-layer predictors and the original target values as your *new training set* to train the blender

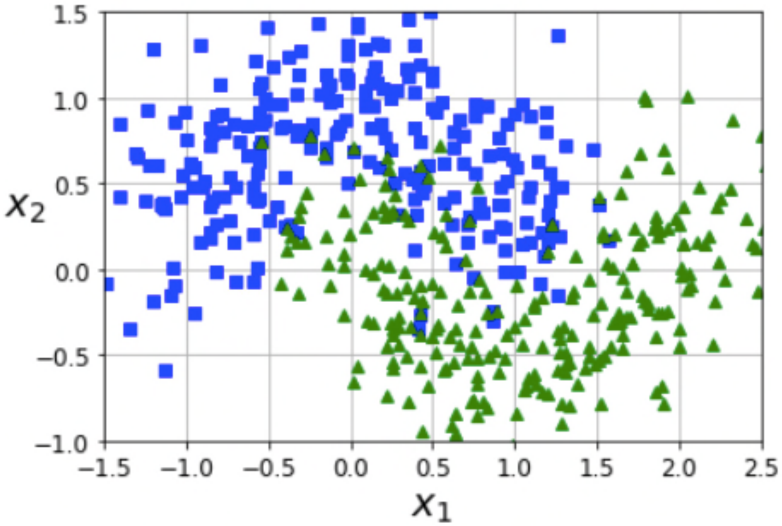
Multi-layer stacking ensemble



Practical 3

Data

- **Artificially generated moons dataset:** 500 data points, two interleaving half circles providing a good “toy” example for testing classification strategies



Your task: Learning objectives

- Learn about simple voting classifiers using hard and soft voting strategies
- Learn about bagging and pasting ensembles
- Learn about boosting and early stopping
- Apply popular ensemble-based learning algorithms, e.g. `RandomForests` and `AdaBoost`
- Apply ensemble techniques of your choice to another dataset (of your choice)
- **Optional**: implement a stacking algorithm

Practical 2 Logistics

- Data and code for Practical 3 can be found on: Github (https://github.com/ekochmar/cl-datasci-pnp-2021/tree/master/DSPNP_practical3)

- Practical ('ticking') session over Zoom at the time allocated by your demonstrator

- At the practical, be prepared to discuss the task and answer the questions about the code to get a 'pass'

- Upload your solutions (Jupyter notebook or Python code) to Moodle by the deadline (Tuesday 17 November, 4pm)