

Machine Learning

João Catalão Fernandes, FCUL




Ciências
ULisboa

Deteção Remota Multiespectral, MEGeospatial, MSIG-TA

Tópicos

5. Machine Learning

- What is machine learning?
- Bibliography and software
- Tasks for machine learning
- Machine learning models
- Generalization, Overfitting
- k-NN algorithm
- Linear Models
- Decision Trees
- Neural Network

A large, 3D-rendered graphic of the words 'BIG DATA' in white, bold, sans-serif capital letters. The text is surrounded by a shower of colorful confetti in shades of red, yellow, blue, and pink, creating a celebratory or dynamic effect.

What is Machine Learning?

Remote sensing multispectral image data, behavioural geography data (person location and trip), transportation network data... BIG DATA of geography.

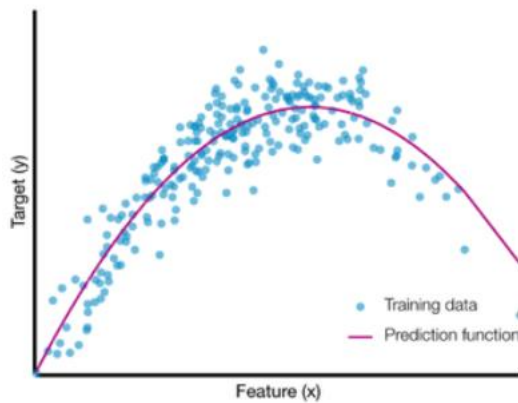
Machine learning is believed to be the powerful tool to explore and analyze the geography big data.

What is machine learning?

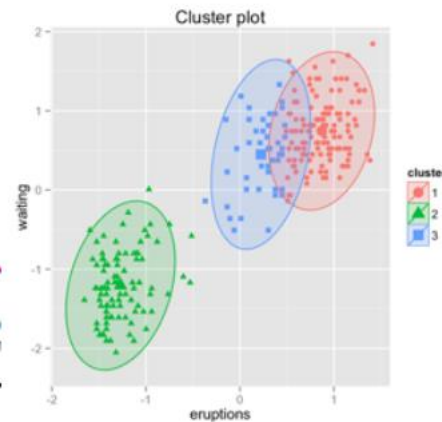
Machine learning evolved from the study of **pattern recognition** and **computational learning theory** in **artificial intelligence (AI)**.

Machine Learning

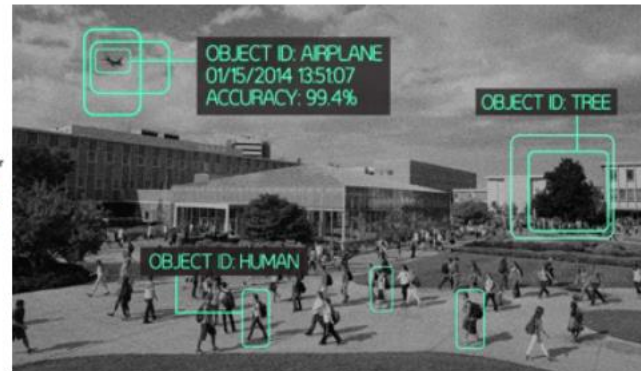
“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E ” — *T. Michell (1997)*



regression



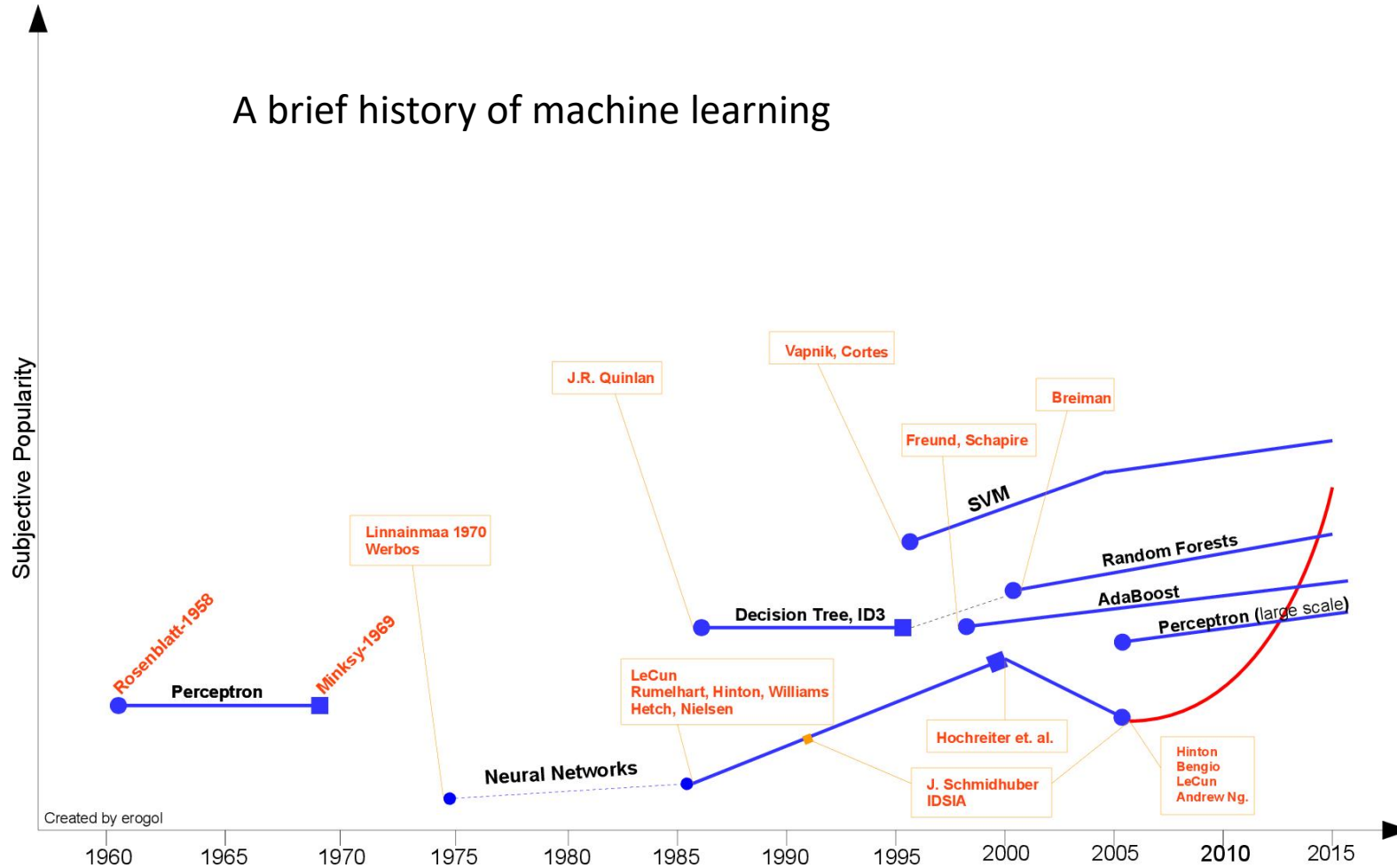
clustering



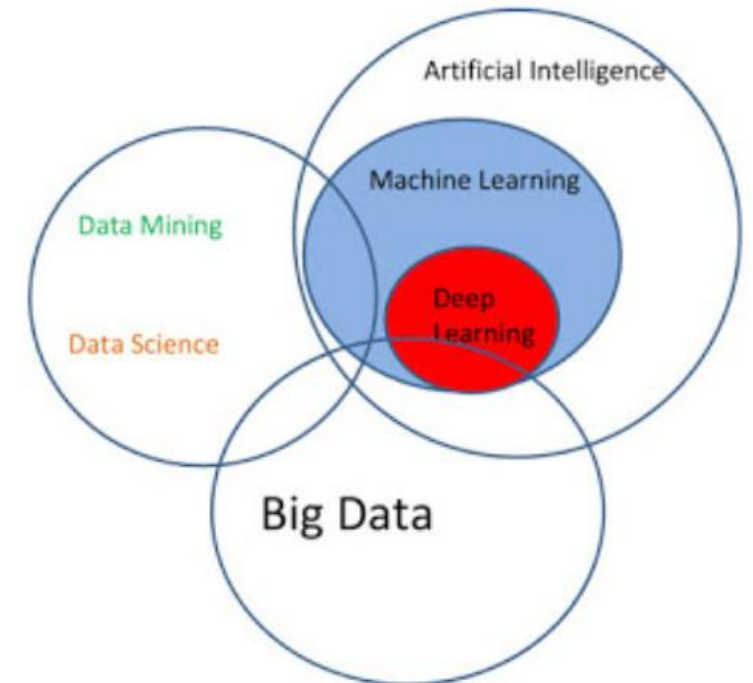
classification

T: Playing checkers
P: Games won
E: playing games against itself

A brief history of machine learning



<http://www.erogol.com/wp-content/uploads/2014/05/test.jpg>

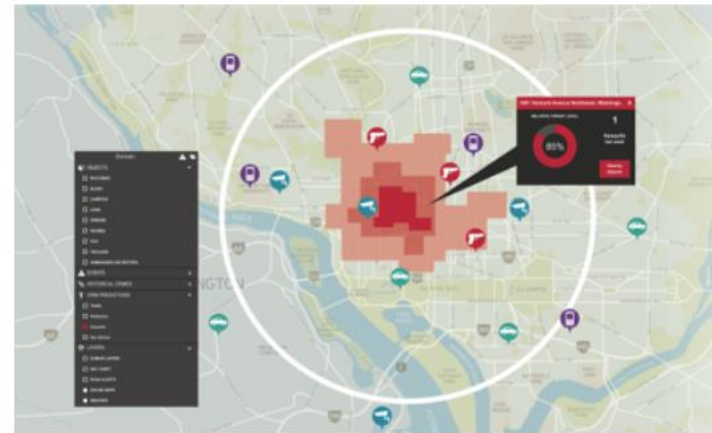


It is all about machine learning...



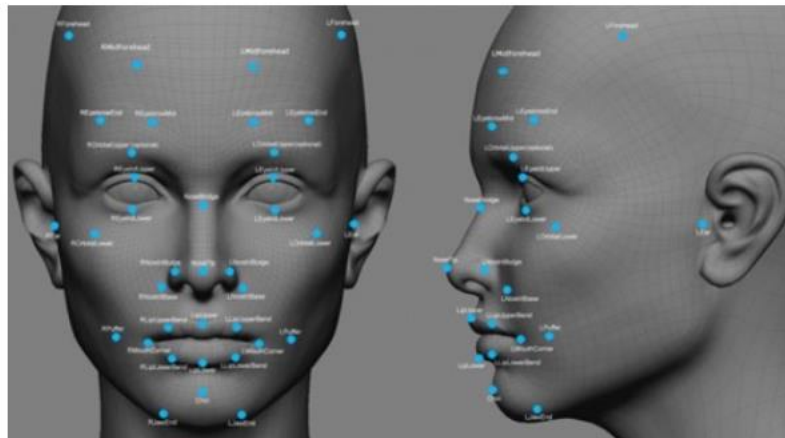
Intelligent voice assistant

<http://www.apple.com/ios/siri/>



Predictive policing

<http://www.predpol.com/>



Facial recognition

<http://www.face-rec.org/>

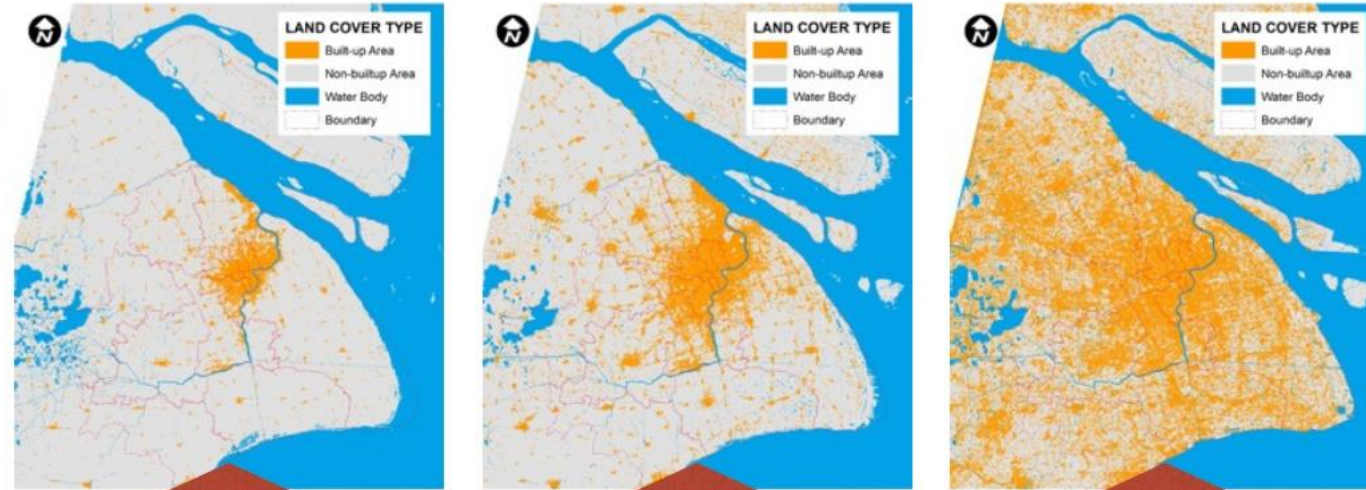


Self-driving car

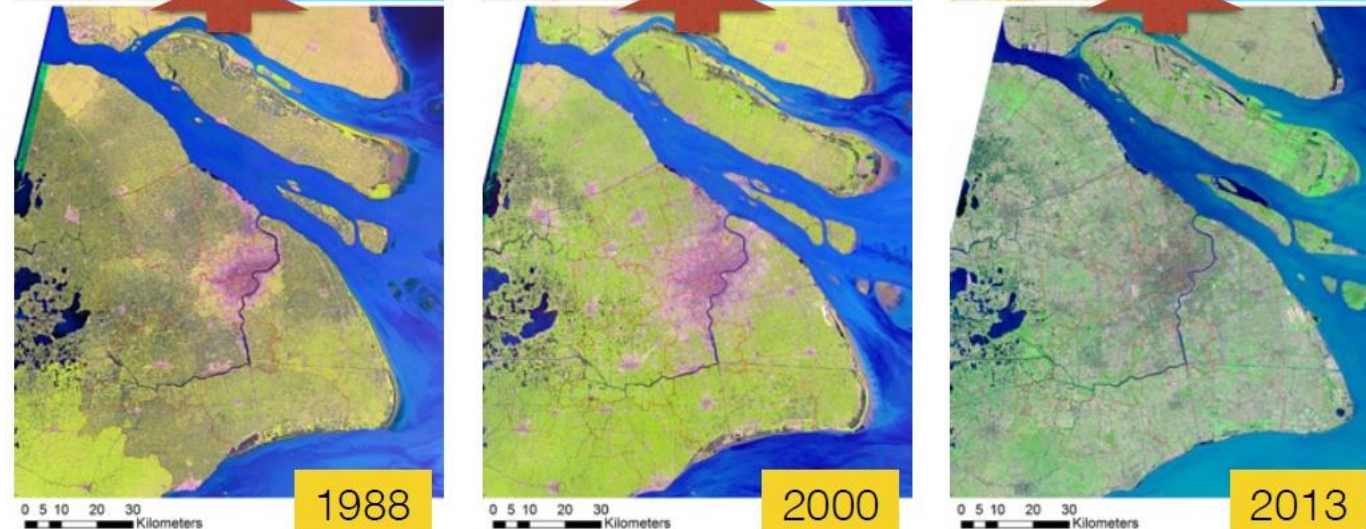
<https://www.google.com/selfdrivingcar/>

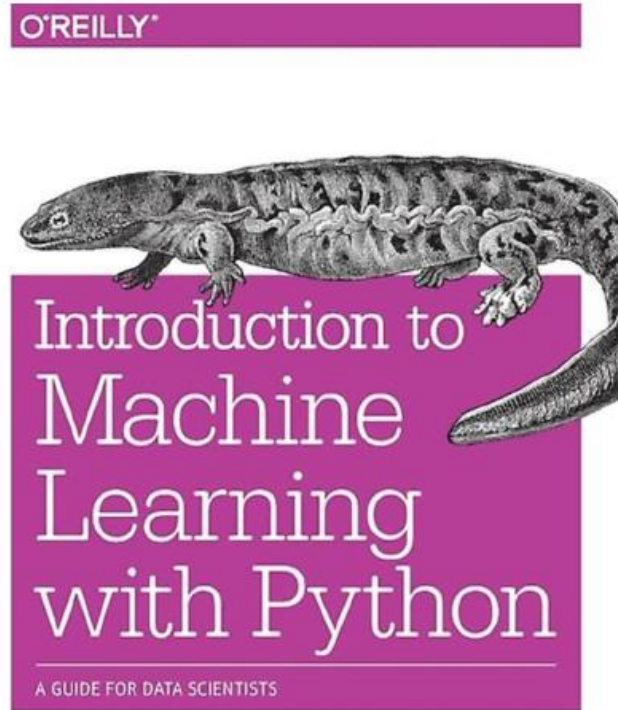
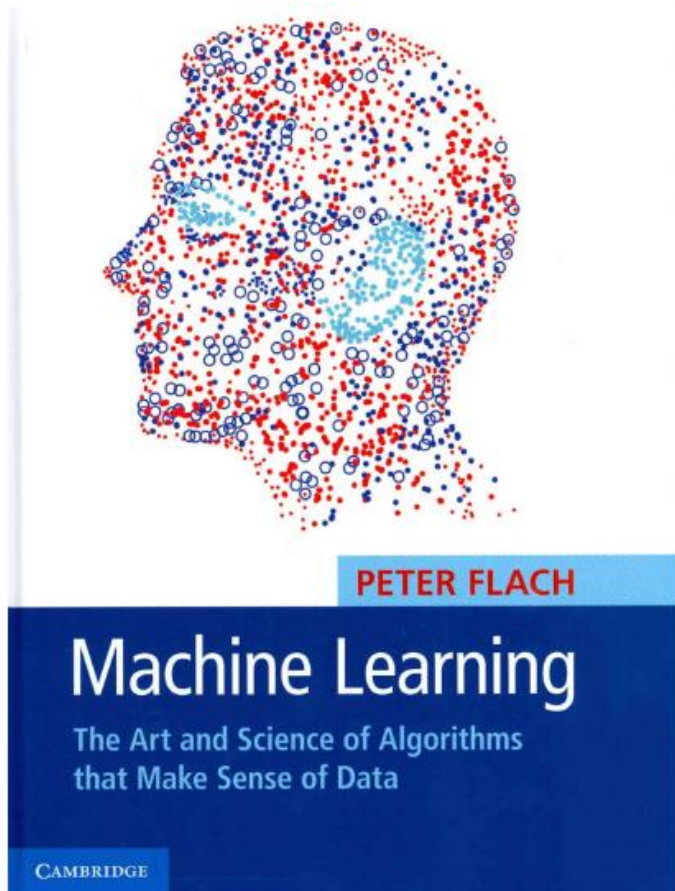
Machine Learning in Remote Sensing

Machine Learning Classification Results

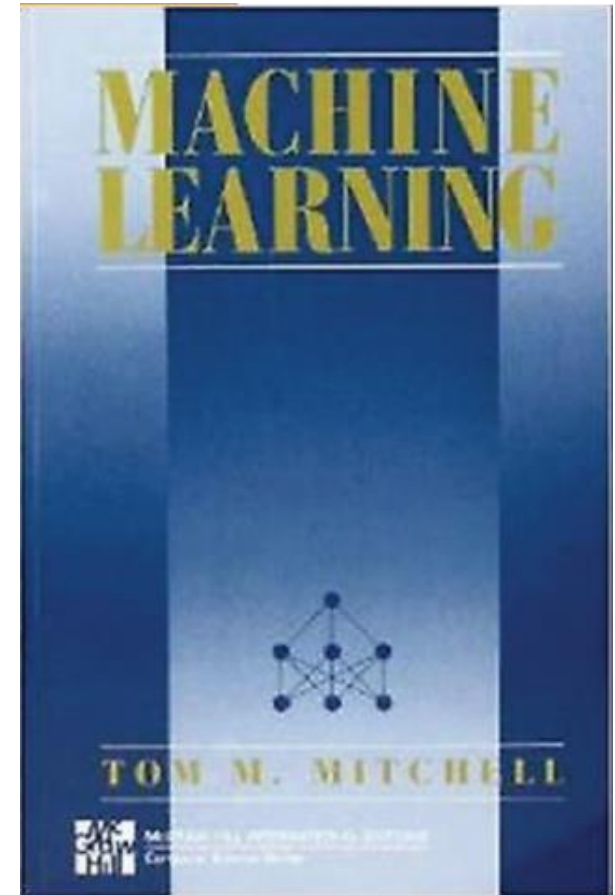


Landsat Satellite Images





Andreas C. Müller & Sarah Guido





"The Most Popular Python Data Science Platform"



Python 3



orange3

3.13.0



"Interactive computational environment, in which you can combine code execution, rich text, mathematics, plots and rich media."



"Component based data mining framework. Data visualization and data analysis for novice and experts. Interactive workflows with a large toolbox."



Orfeo ToolBox

Orfeo ToolBox is not a black box

[Forum](#) [Download](#) [Documentation](#) [Blog](#) [Community](#)

Orfeo ToolBox is an open-source project for state-of-the-art remote sensing, including a fast image viewer, apps callable from Bash, Python or QGIS, and a powerful C++ API.

Open Source processing of remote sensing images



[Start using OTB](#)



[OTB features](#)



[Documentation](#)



[OTB community](#)



[Developers corner](#)



[Media](#)



[External projects](#)



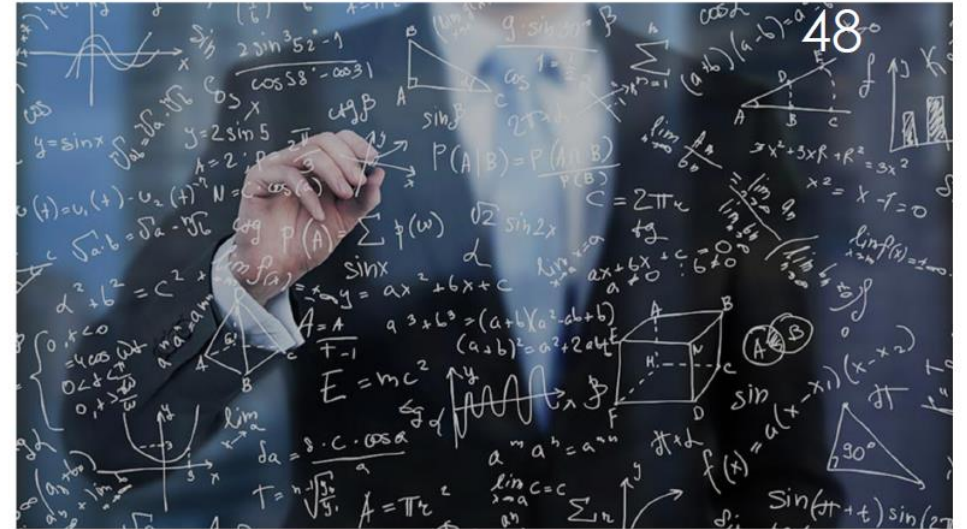
[Blog](#)



List of Common Machine Learning Algorithms

1. Naïve Bayes Classifier Algorithm
2. K Means Clustering Algorithm
3. Support Vector Machine Algorithm
4. Apriori Algorithm
5. Linear Regression
6. Logistic Regression
7. Artificial Neural Networks
8. Random Forests
9. Decision Trees
10. Nearest Neighbours

The 10 Algorithms Machine Learning Engineers Need to Know



1. Linear Regression
2. Logistic Regression
3. Decision Trees
4. SVM (Support Vector Machine)
5. Naive Bayes
6. KNN (K- Nearest Neighbors)
7. K-Means
8. Random Forests
9. Dimensionality Reduction Algorithms
10. Gradient Boosting & AdaBoost

The most common machine learning tasks are *predictive*, in the sense that they concern predicting a target variable from features. .

- 👉 Binary and multi-class classification: categorical target
- 👉 Regression: numerical target
- 👉 Clustering: hidden target

Descriptive tasks are concerned with exploiting underlying structure in the data.

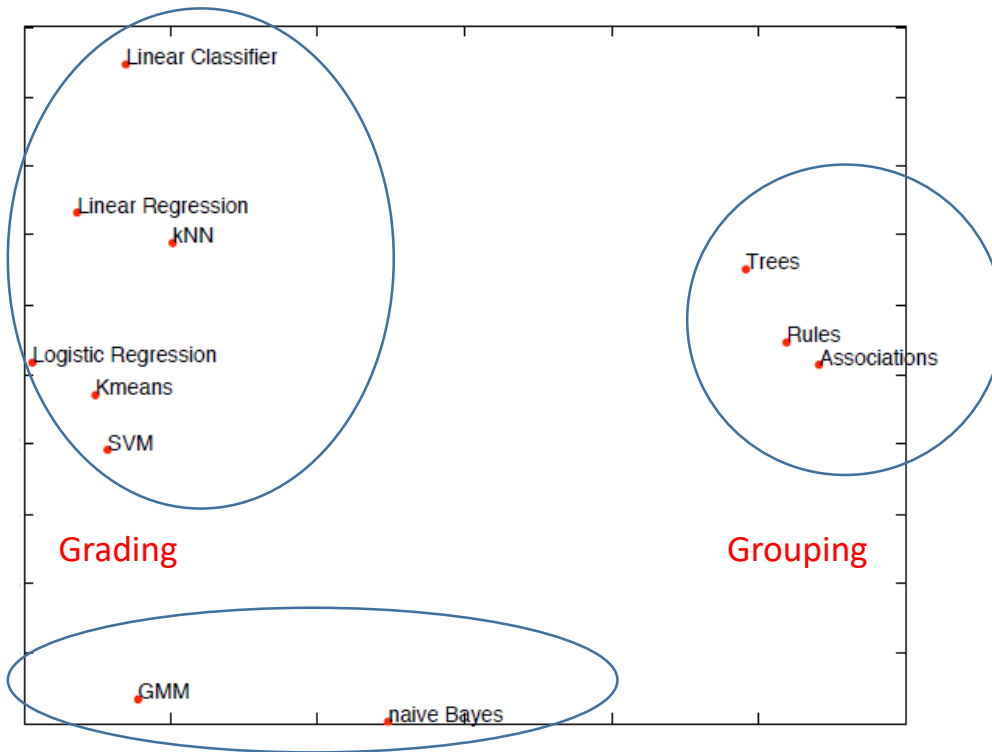
	<i>Predictive model</i>	<i>Descriptive model</i>
<i>Supervised learning</i>	classification, regression	subgroup discovery
<i>Unsupervised learning</i>	predictive clustering	descriptive clustering, association rule discovery

Machine learning models can be distinguished according to their main intuition:

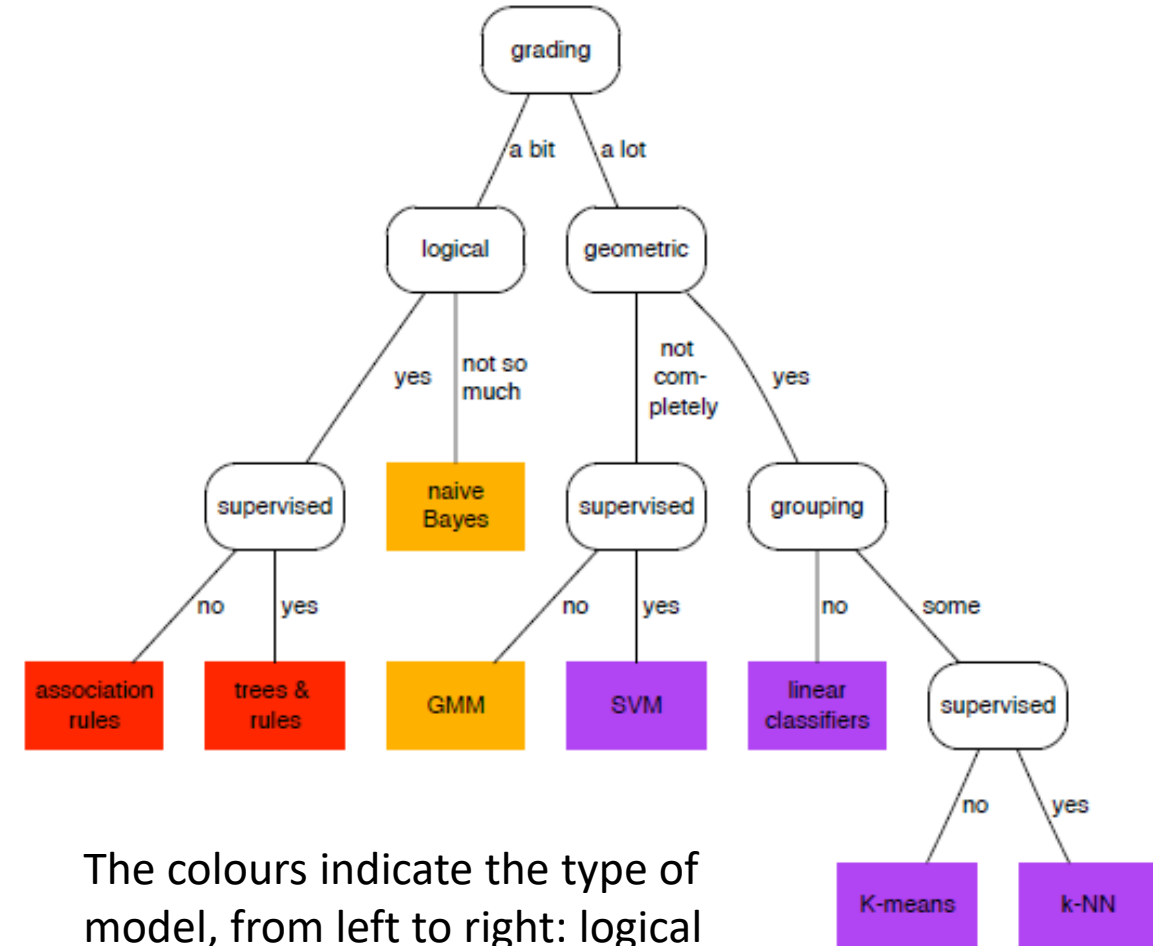
- 👉 *Geometric* models use intuitions from geometry such as separating (hyper-)planes, linear transformations and distance metrics.
- 👉 *Probabilistic* models view learning as a process of reducing uncertainty, modelled by means of probability distributions.
- 👉 *Logical* models are defined in terms of easily interpretable logical expressions.

Alternatively, they can be characterised by their *modus operandi*:

- 👉 *Grouping models* divide the instance space into segments; in each segment a very simple (e.g., constant) model is learned.
- 👉 *Grading models* learning a single, global model over the instance space.



Models that share characteristics are plotted closer together: logical models to the right, geometric models on the top left and probabilistic models on the bottom left. The horizontal dimension roughly ranges from grading models on the left to grouping models on the right.



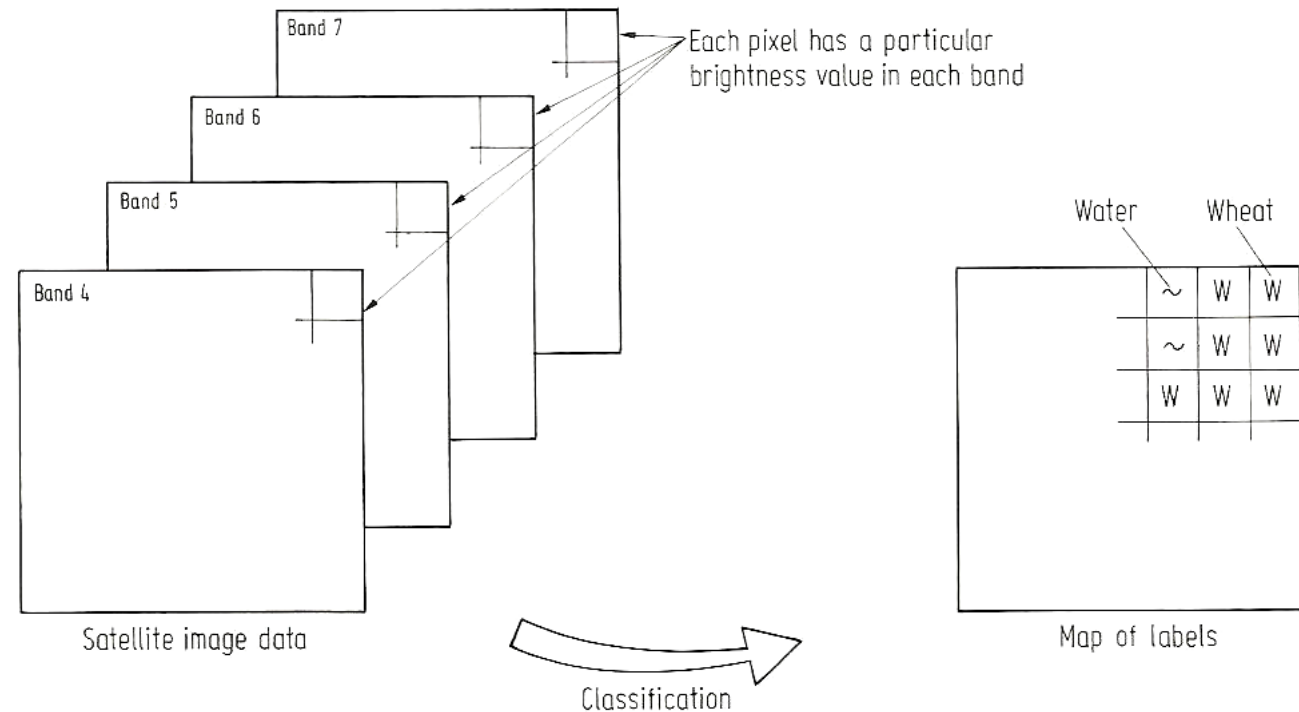
The colours indicate the type of model, from left to right: logical (red), probabilistic (orange) and geometric (purple).

<i>Task</i>	<i>Label space</i>	<i>Output space</i>	<i>Learning problem</i>
Classification	$\mathcal{L} = \mathcal{C}$	$\mathcal{Y} = \mathcal{C}$	learn an approximation $\hat{c} : \mathcal{X} \rightarrow \mathcal{C}$ to the true labelling function c
Scoring and ranking	$\mathcal{L} = \mathcal{C}$	$\mathcal{Y} = \mathbb{R}^{ \mathcal{C} }$	learn a model that outputs a score vector over classes
Probability estimation	$\mathcal{L} = \mathcal{C}$	$\mathcal{Y} = [0, 1]^{ \mathcal{C} }$	learn a model that outputs a probability vector over classes
Regression	$\mathcal{L} = \mathbb{R}$	$\mathcal{Y} = \mathbb{R}$	learn an approximation $\hat{f} : \mathcal{X} \rightarrow \mathbb{R}$ to the true labelling function f

A *classifier* is a mapping $\hat{c} : \mathcal{X} \rightarrow \mathcal{C}$, where $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ is a finite and usually small set of *class labels*. We will sometimes also use C_i to indicate the set of examples of that class.

We use the ‘hat’ to indicate that $\hat{c}(x)$ is function $c(x)$. Examples for a classifier (an instance and $c(x)$ is the true class by noise).

Learning a classifier involves constructing as closely as possible (and not just on instance space \mathcal{X}).

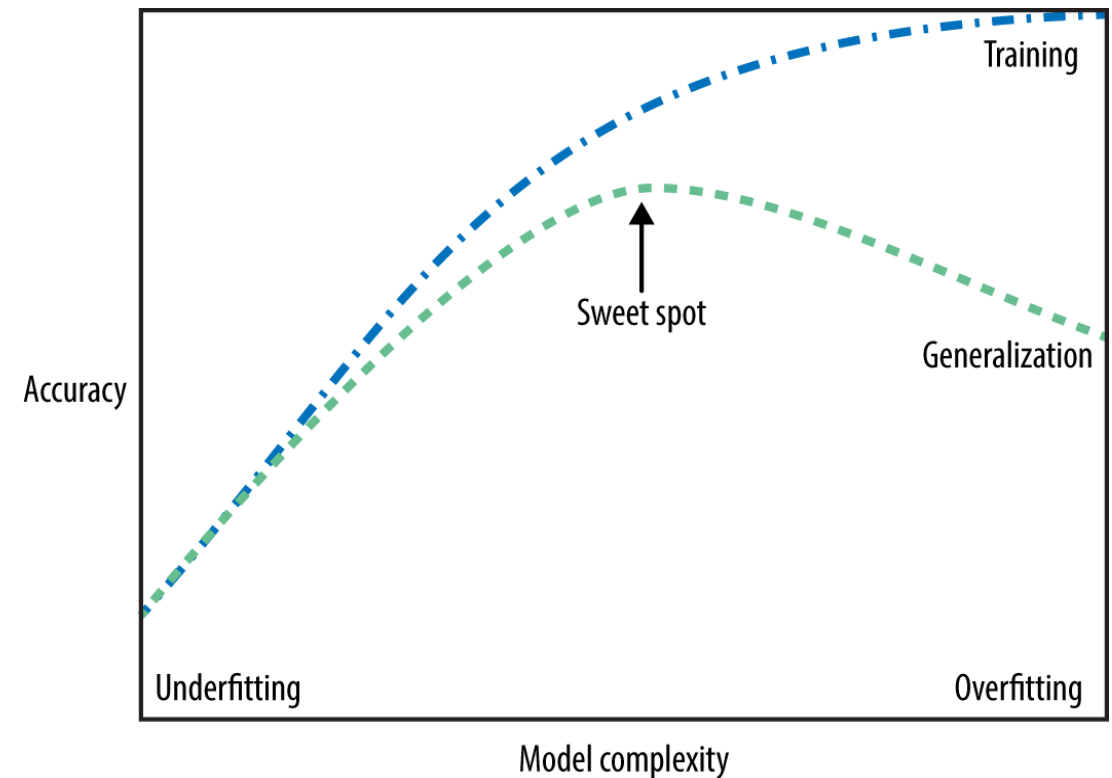


If a model is able to make accurate predictions on unseen data, we say it is able to **generalize** from the training set to the test set. We want to build a model that is able to generalize as accurately as possible.

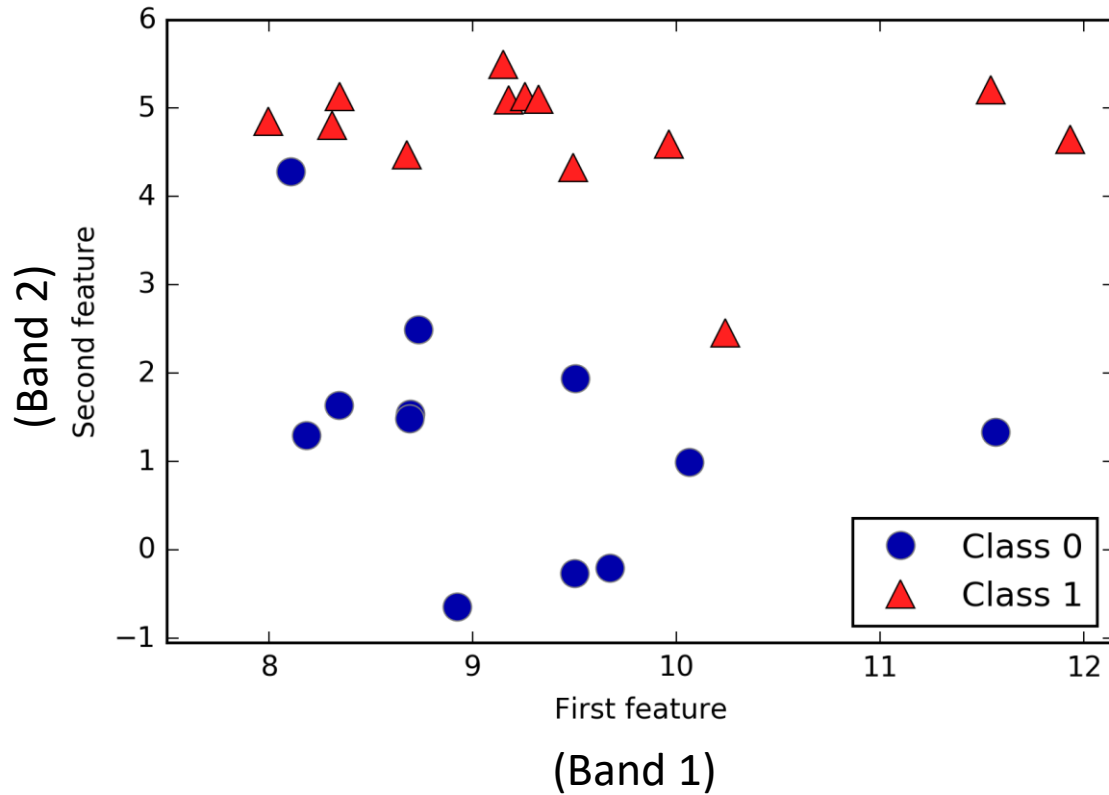
Overfitting occurs when you fit a model too closely to the particularities of the training set and obtain a model that works well on the training set but is not able to generalize to new data.

More complex the model => better we will be able to predict on the training data.
 However : Too complex => focusing too much in our training set => not generalize well to new data.
 There is a sweet spot in between that will yield the best generalization performance.

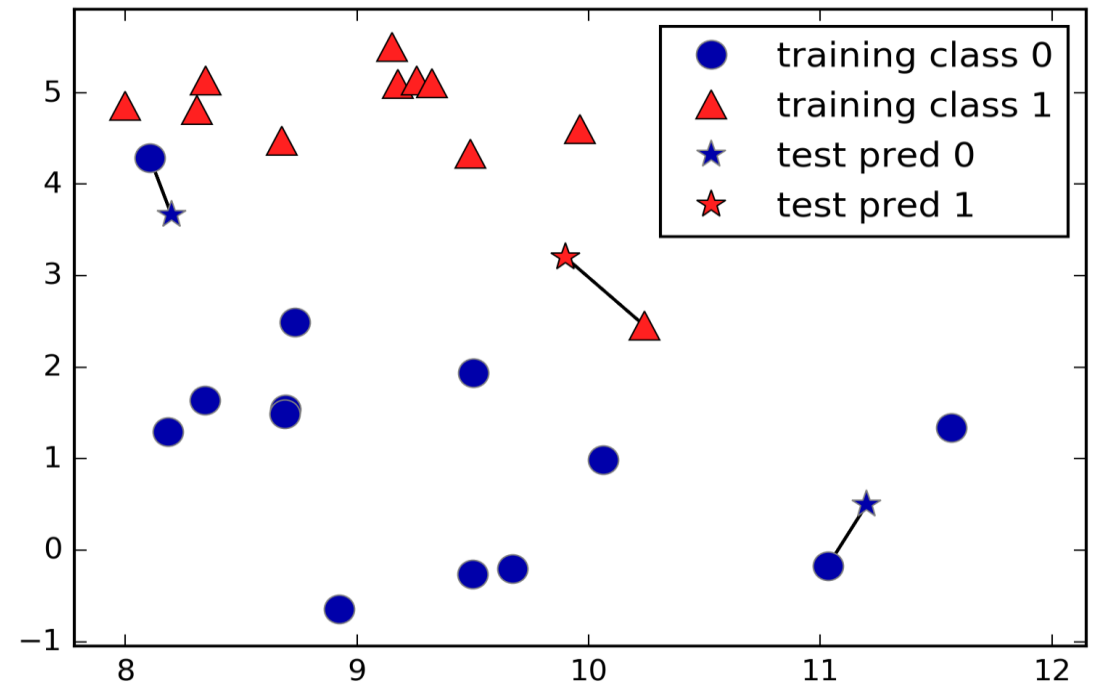
Trade-off of model complexity against training and test accuracy



Scatter plot of training dataset
2 bands and 2 classes



Predictions made by the one-nearest-neighbour model on the dataset



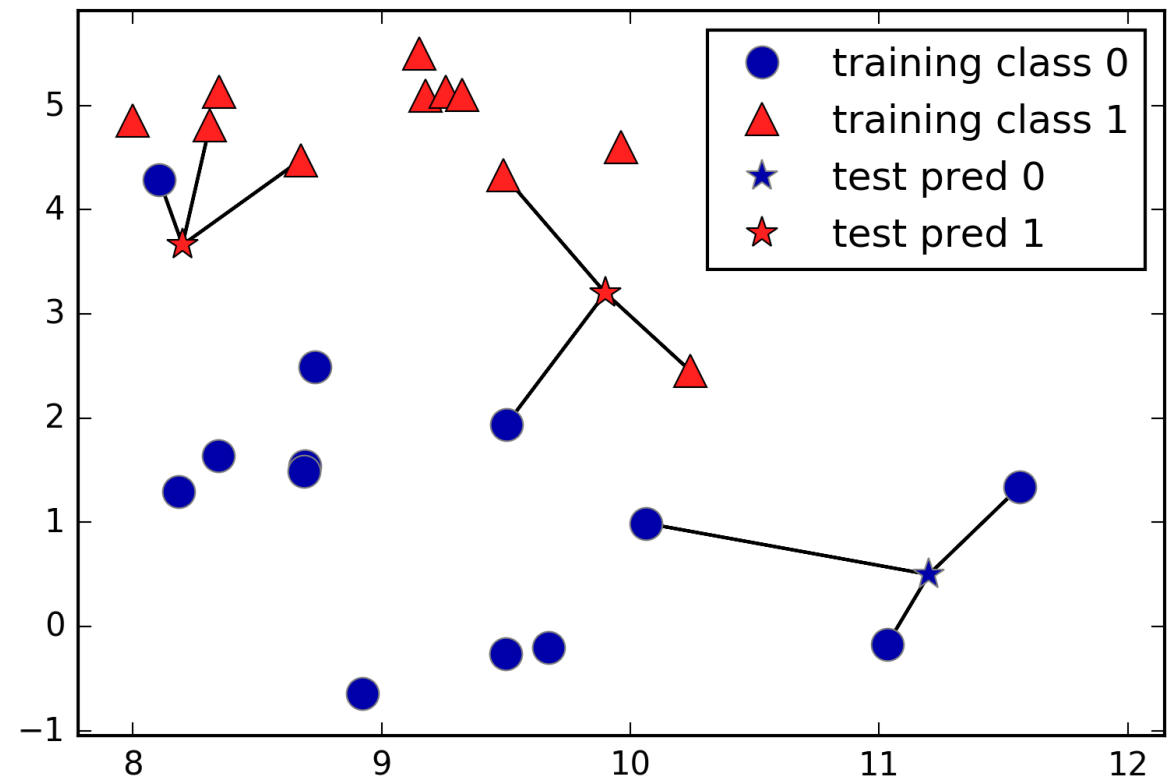
Instead of considering only the closest neighbour, we can also consider an arbitrary number, k , of neighbours.

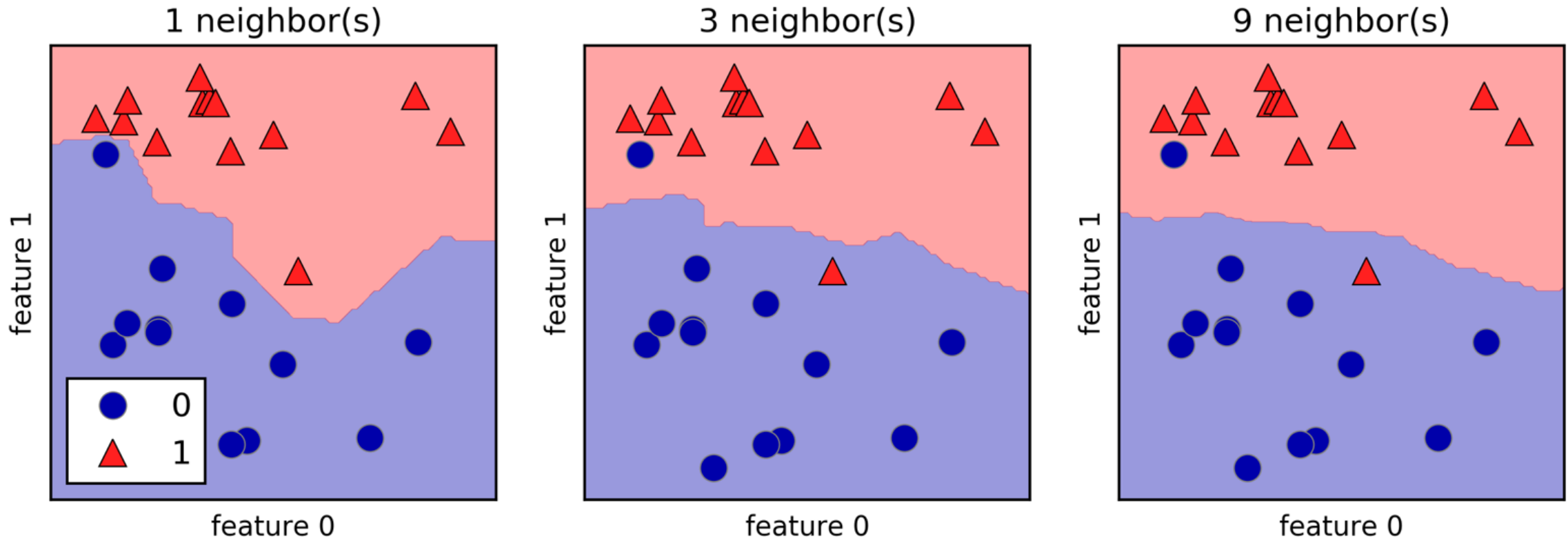
This is where the name of the k -nearest neighbours algorithm comes from.

When considering more than one neighbour, we use **voting** to assign a label. This means that for each test point, we count how many neighbours belong to class 0 and how many neighbours belong to class 1.

We then assign the class that is more frequent: in other words, the majority class among the k -nearest neighbours.

Predictions made by the three-nearest-neighbours model on the dataset





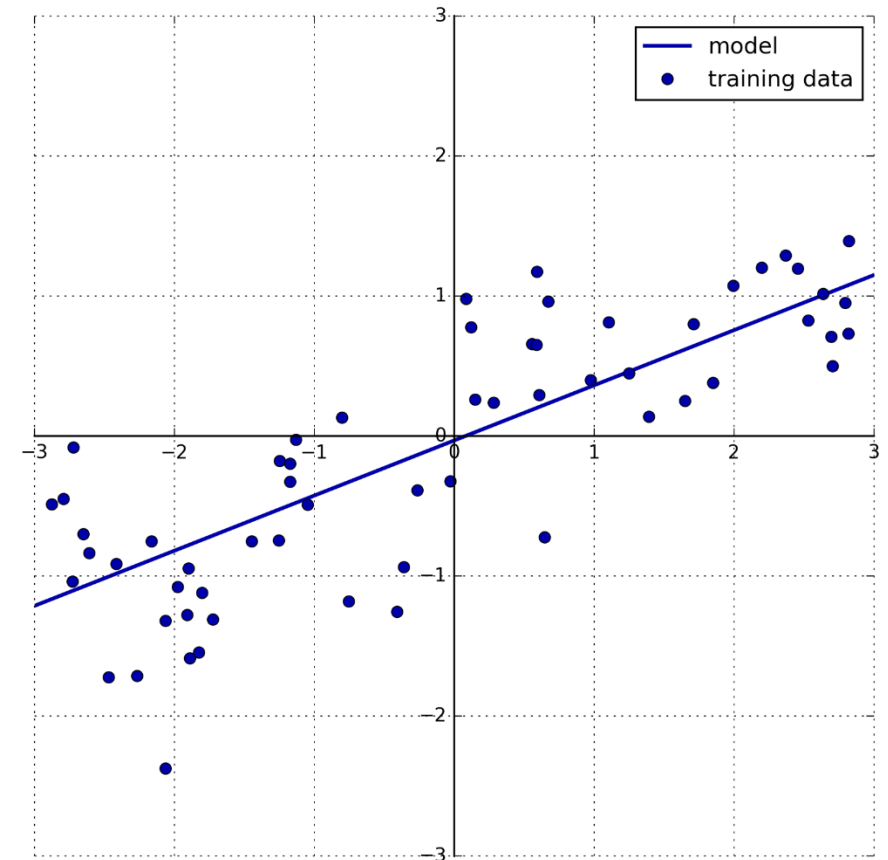
Decision boundaries created by the nearest neighbours model for different values of $k_{\text{neighbours}}$

Linear models are a class of models that are widely used in practice and have been studied extensively in the last few decades, with roots going back over a hundred years.

Linear models make a prediction using a *linear function* of the input features, which we will explain shortly. For regression:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b$$

Here, $x[0]$ to $x[p]$ denotes the features (in our case, the spectral bands, $p+1$) of a single pixel (or set of pixels), w and b are parameters of the model that are learned, and \hat{y} is the prediction the model makes.



For a dataset with a single feature, this is:

$$\hat{y} = w[0] * x[0] + b$$

Linear models are also extensively used for classification.

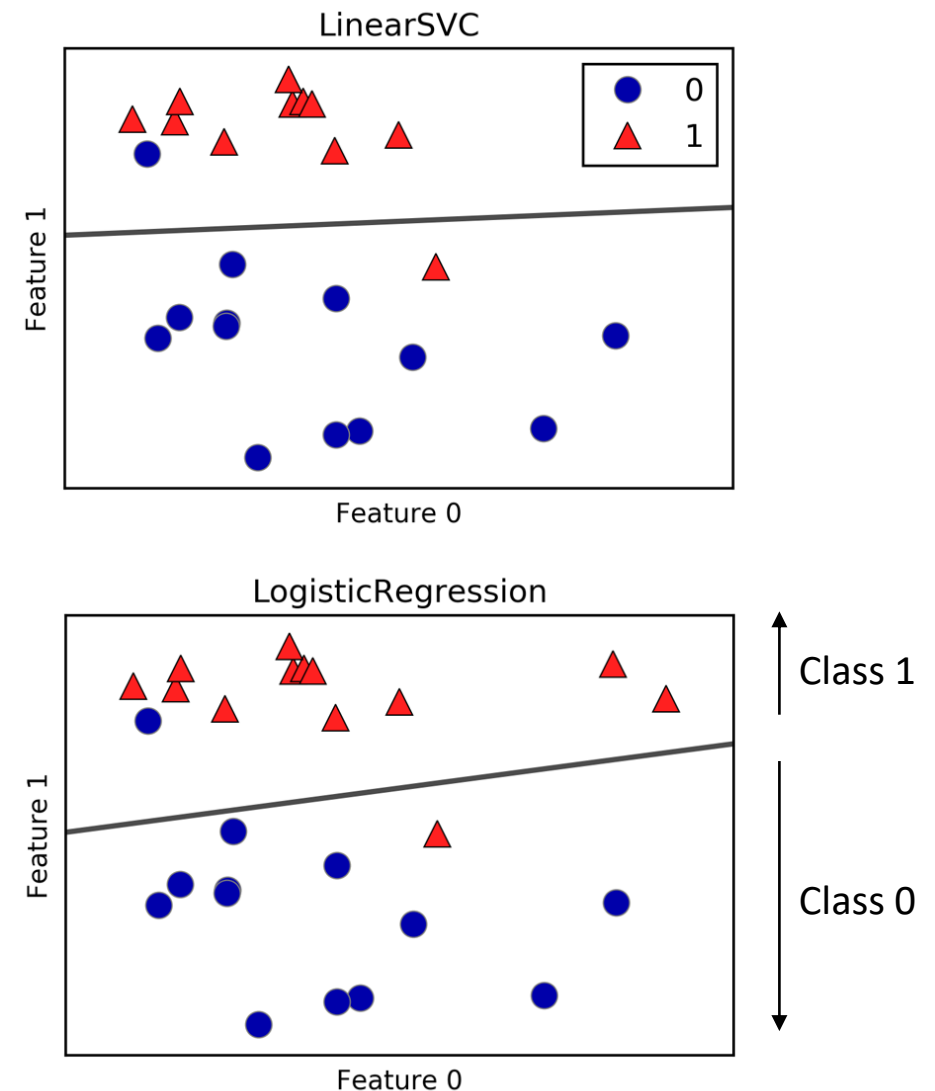
In this case, a prediction is made using the following formula:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b > 0$$

The formula looks very similar to the one for linear regression, but instead of just returning the weighted sum of the features, we threshold the predicted value at zero.

If the function is smaller than zero, we predict the class -1; if it is larger than zero, we predict the class +1.

This prediction rule is common to all linear models for classification. Again, there are many different ways to find the coefficients (w) and the intercept (b).



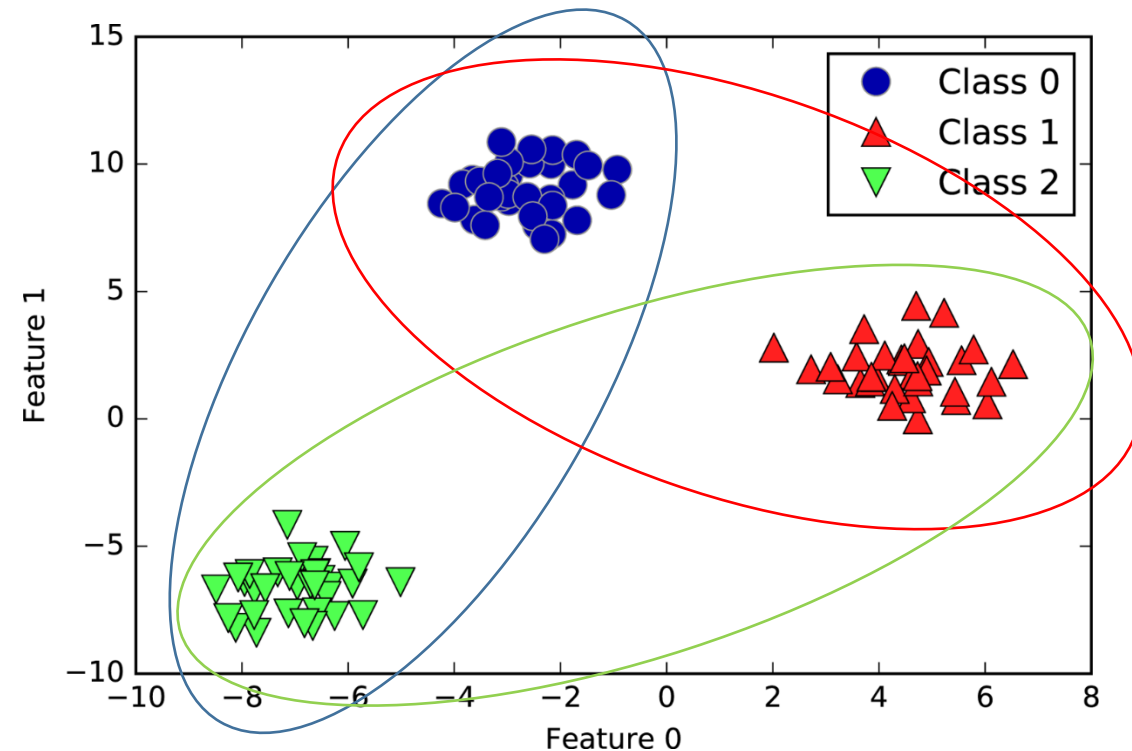
A common technique to extend a binary classification algorithm to a multiclass classification algorithm is the **one-vs.-rest approach**.

In the **one-vs.-rest** approach, a binary model is learned for each class that tries to separate that class from all of the other classes, resulting in as many binary models as there are classes.

Having one binary classifier per class results in having one vector of coefficients (w) and one intercept (b) for each class.

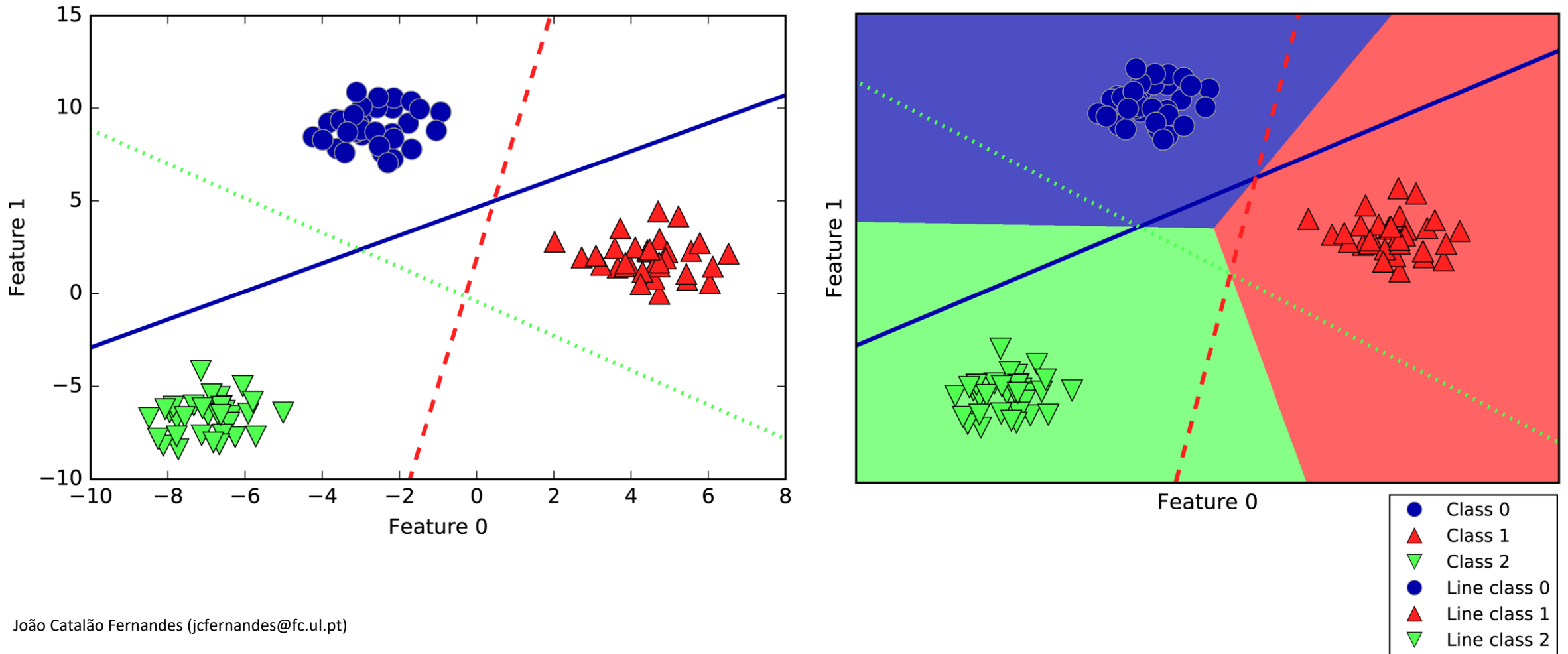
The class for which the result of the classification confidence formula given here is highest is the assigned class label:

$$w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b$$

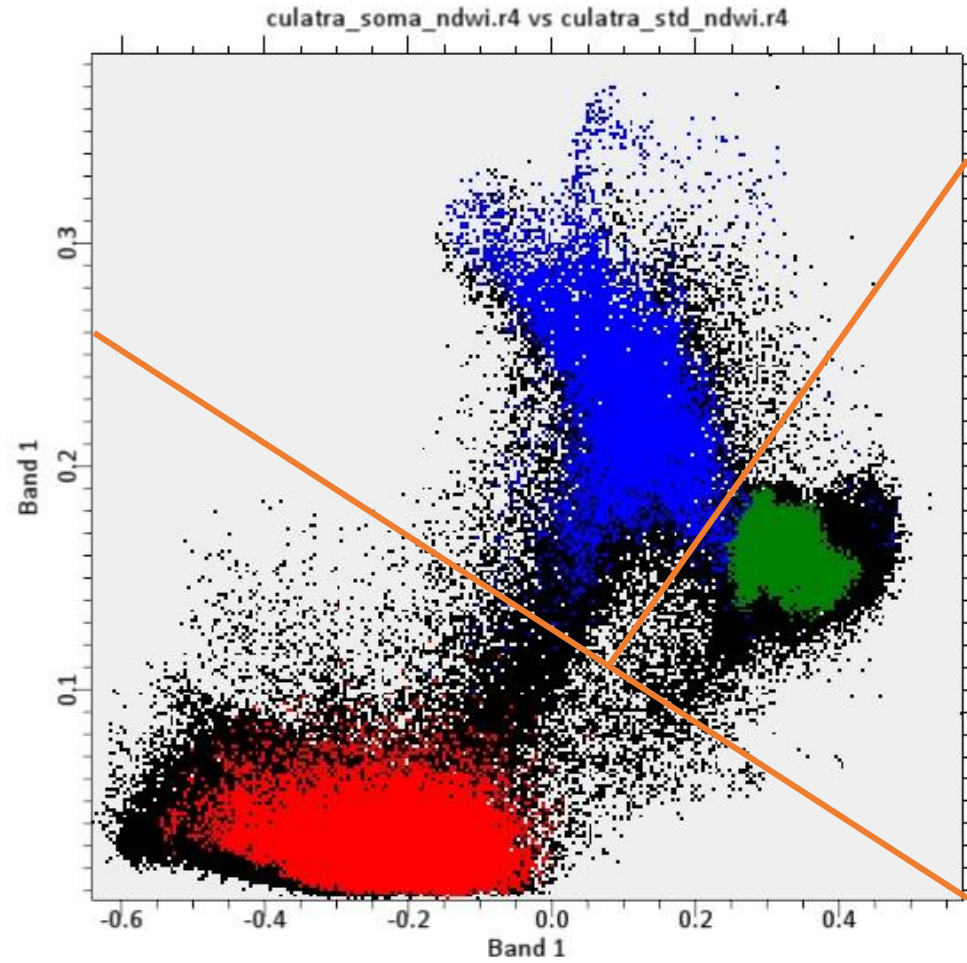


The classifier that has the highest score on its single class “wins,” and this class label is returned as the prediction.

Multiclass decision boundaries derived from the three one-vs.-rest classifiers



Blue: water
 Red: Land
 Green: intertidal

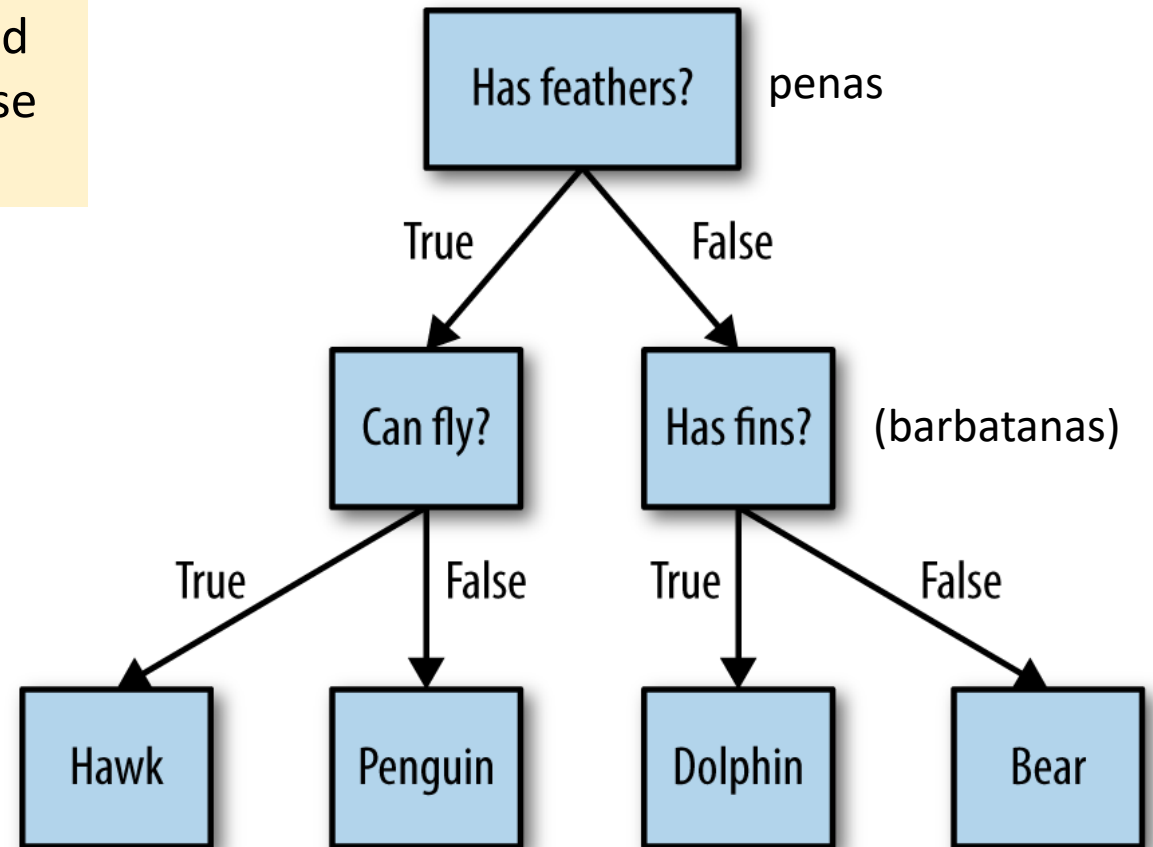


Decision trees are widely used models for classification and regression tasks. Essentially, they learn a hierarchy of if/else questions, leading to a decision.

Imagine you want to distinguish between the following four animals:

bears, hawks, penguins, and dolphins.

Your goal is to get to the right answer by asking as few if/else questions as possible.



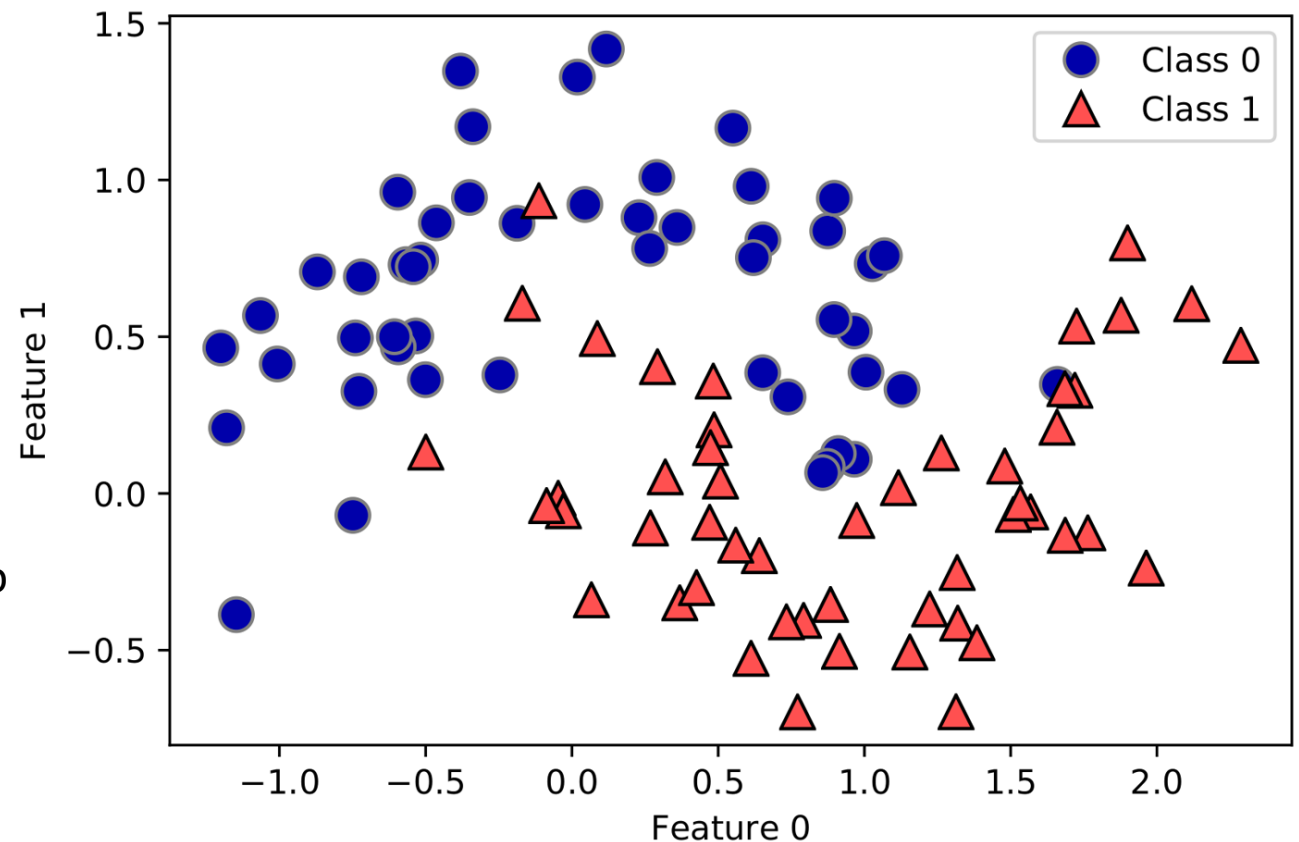
In this illustration, each node in the tree either represents a question or a terminal node (also called a *leaf*) that contains the answer. The edges connect the answers to a question with the next question you would ask.

Learning a decision tree means learning the sequence of **if/else** questions that gets us to the true answer most quickly.

In the machine learning setting, these questions are called **tests** (not to be confused with the test set, which is the data we use to test to see how generalizable our model is).

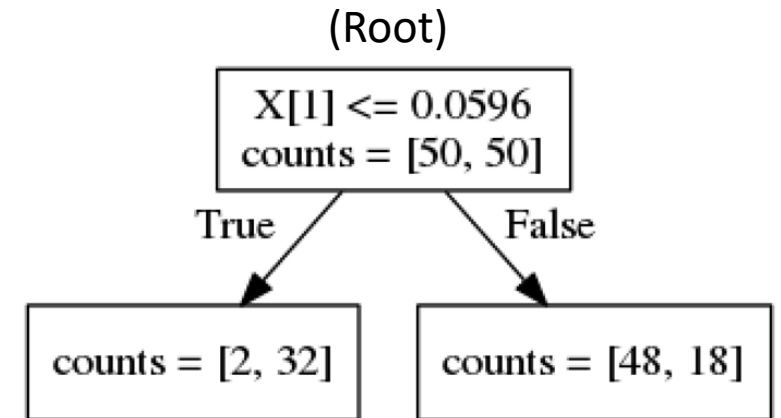
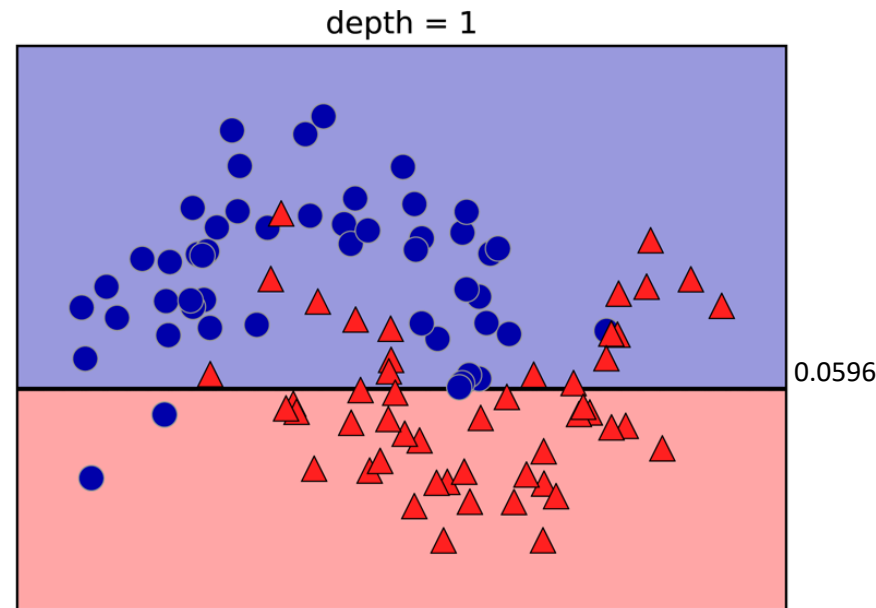
Usually data does not come in the form of binary yes/no features as in the animal example, but is instead represented as continuous features such as in the 2D dataset shown in figure.

The tests that are used on continuous data are of the form “Is feature i larger than value a ?”



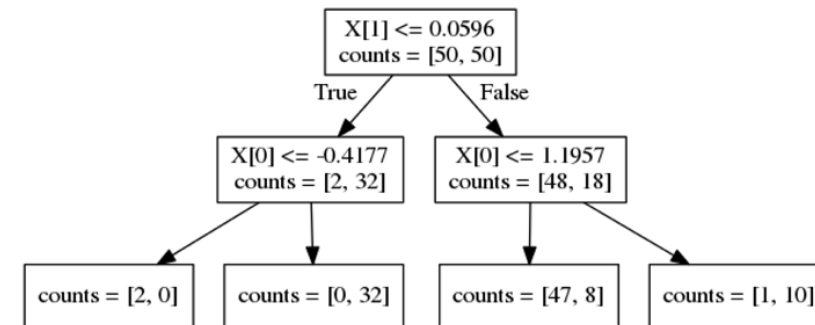
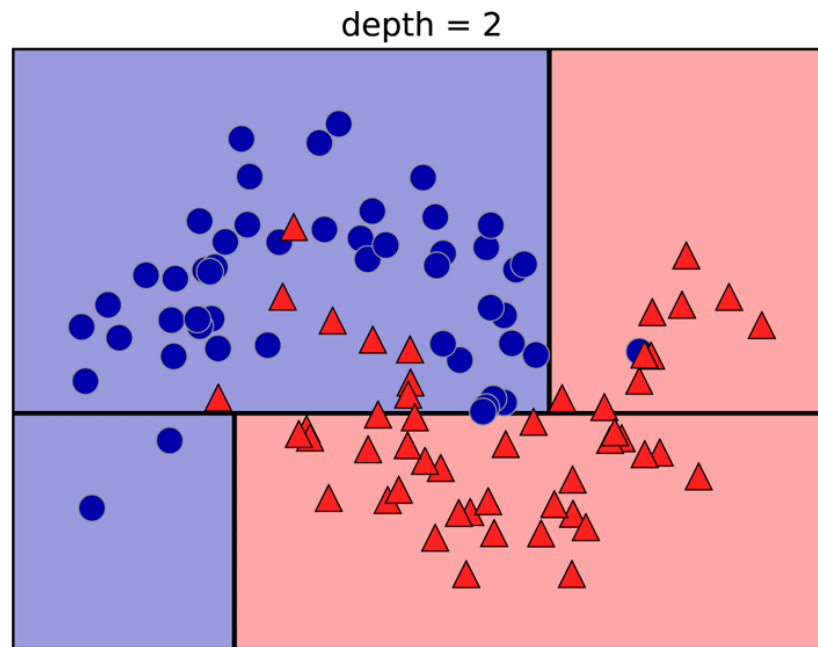
Two-moons dataset on which the decision tree will be built

To build a tree, the algorithm searches over all possible tests and finds the one that is most informative about the target variable.

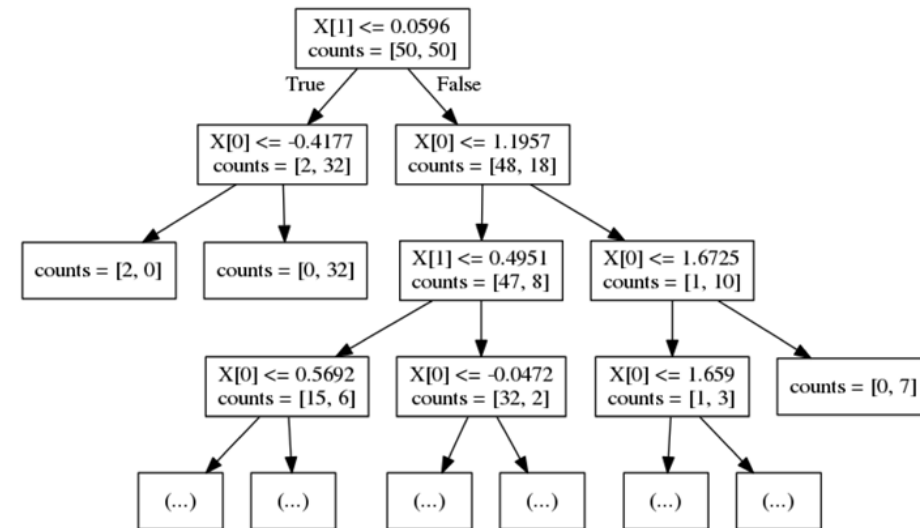
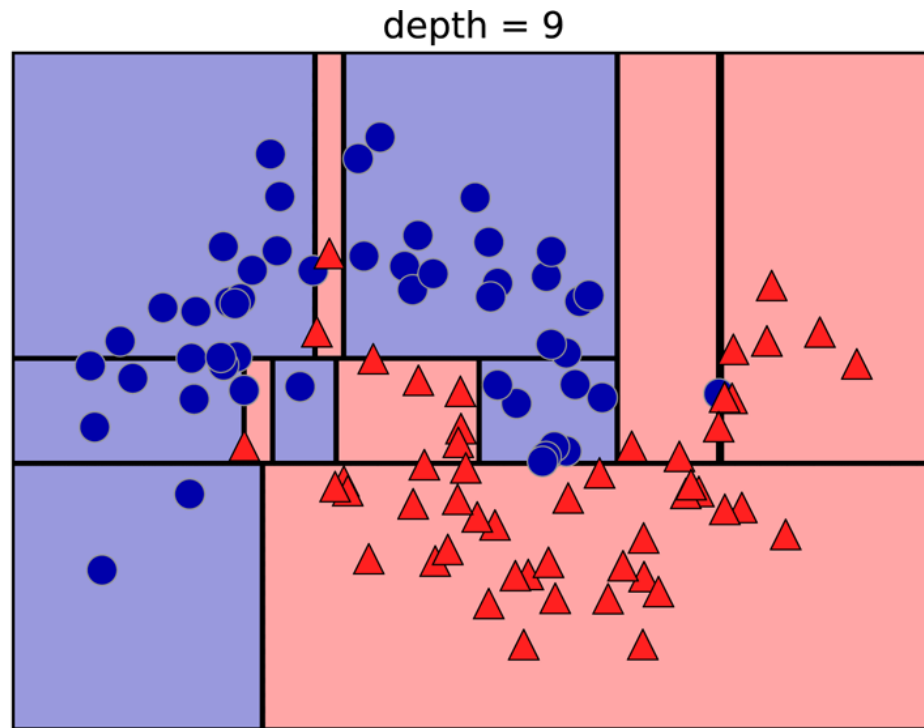


Splitting the dataset horizontally at $x[1]=0.0596$ yields the most information; it best separates the points in class 0 from the points in class 1. The top node, also called the *root*, represents the whole dataset, consisting of 50 points belonging to class 0 and 50 points belonging to class 1. The split is done by testing whether $x[1] \leq 0.0596$, indicated by a black line. If the test is true, a point is assigned to the left node, which contains 2 points belonging to class 0 and 32 points belonging to class 1.

Even though the first split did a good job of separating the two classes, the bottom region still contains points belonging to class 0, and the top region still contains points belonging to class 1. We can build a more accurate model by repeating the process of looking for the best test in both regions.



This recursive process yields a binary tree of decisions, with each node containing a test.

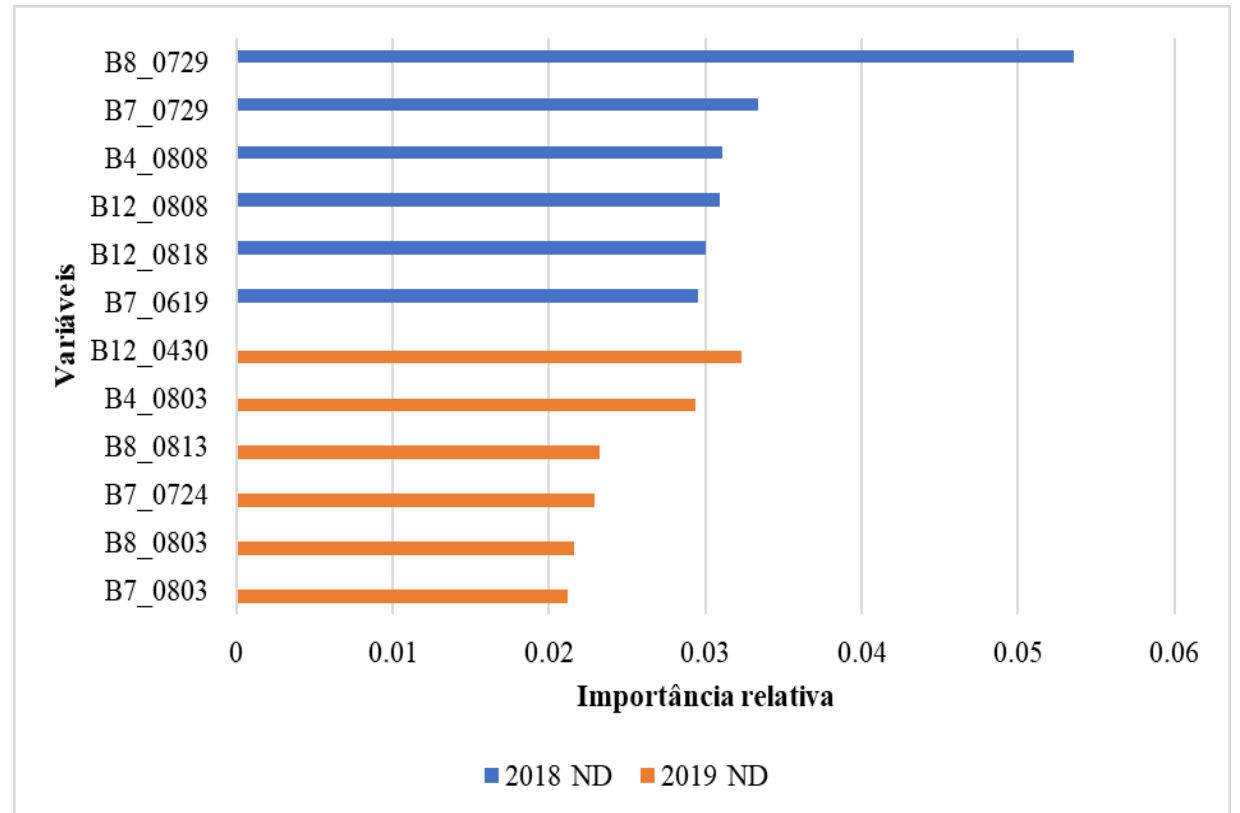


Typically, building a tree as described here and continuing until all leaves are pure leads to models that are very complex and highly overfit to the training data. The presence of pure leaves mean that a tree is 100% accurate on the training set; each data point in the training set is in a leaf that has the correct majority class.

Instead of looking at the whole tree, there are some useful properties that we can derive to summarize the workings of the tree.

The most commonly used summary is **feature importance**, which rates how important each feature is for the decision a tree makes.

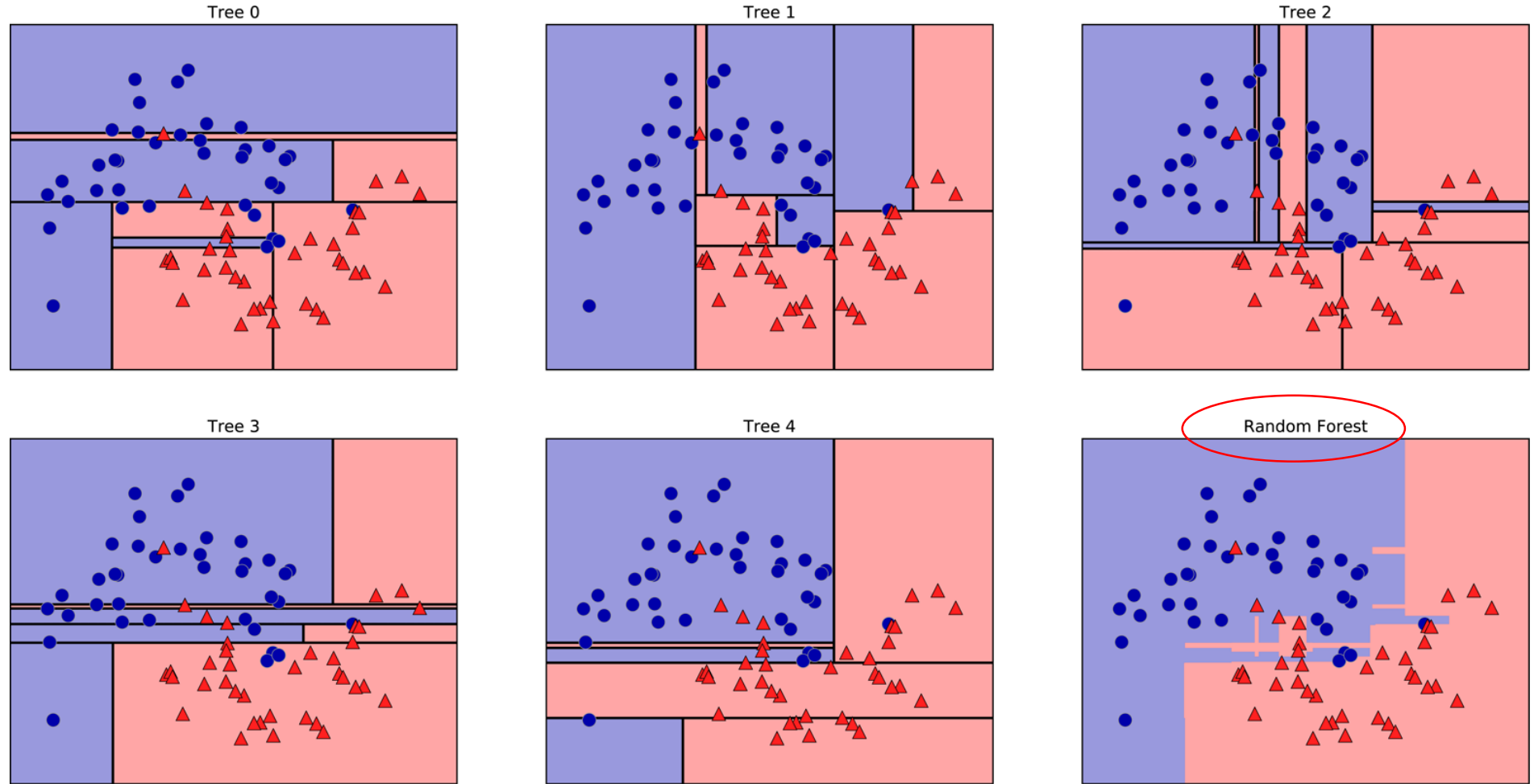
It is a number between 0 and 1 for each feature, where 0 means “not used at all” and 1 means “perfectly predicts the target.”



Importância relativa das variáveis na classificação com RF para dados de 2018 (a azul) e de 2019 (a laranja). As denominações das variáveis dizem respeito à banda, mês e dia de aquisição da imagem, respetivamente.

The **random forest** overfits less than any of the trees individually

In any real application, we would use many more trees (often hundreds or thousands), leading to even smoother boundaries.



Decision boundaries found by five randomized decision trees and the decision boundary obtained by averaging their predicted probabilities

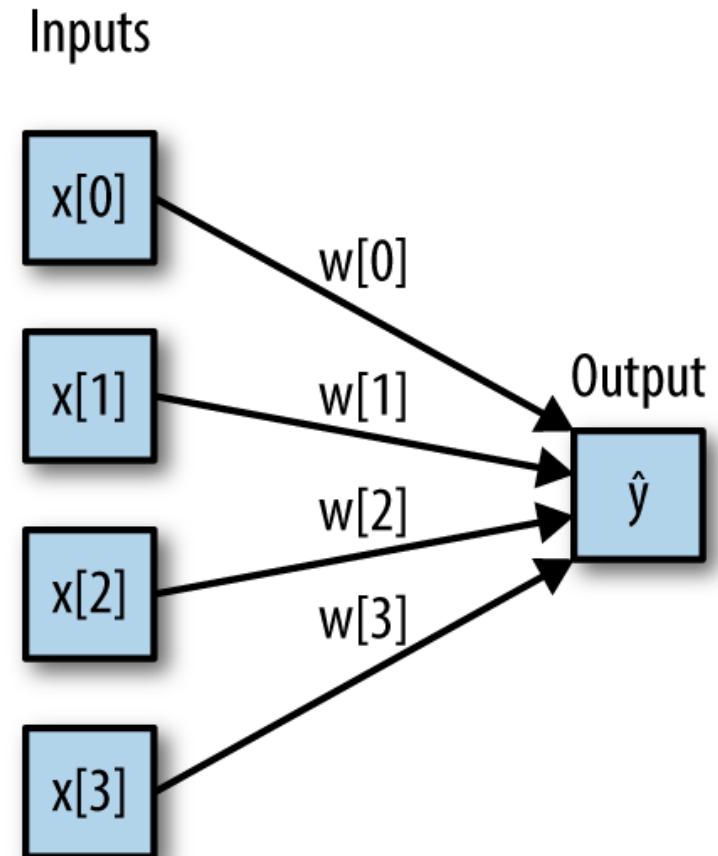
Multilayer perceptrons (MLPs) are also known as feed-forward neural networks, or sometimes just **neural networks**.

MLPs can be viewed as generalizations of linear models that perform multiple stages of processing to come to a decision.

Remember that the prediction by a linear regressor is given as:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b$$

in plain English, \hat{y} is a weighted sum of the input features $x[0]$ to $x[p]$ (our spectral bands), weighted by the learned coefficients $w[0]$ to $w[p]$ (classes de ocupação do solo).



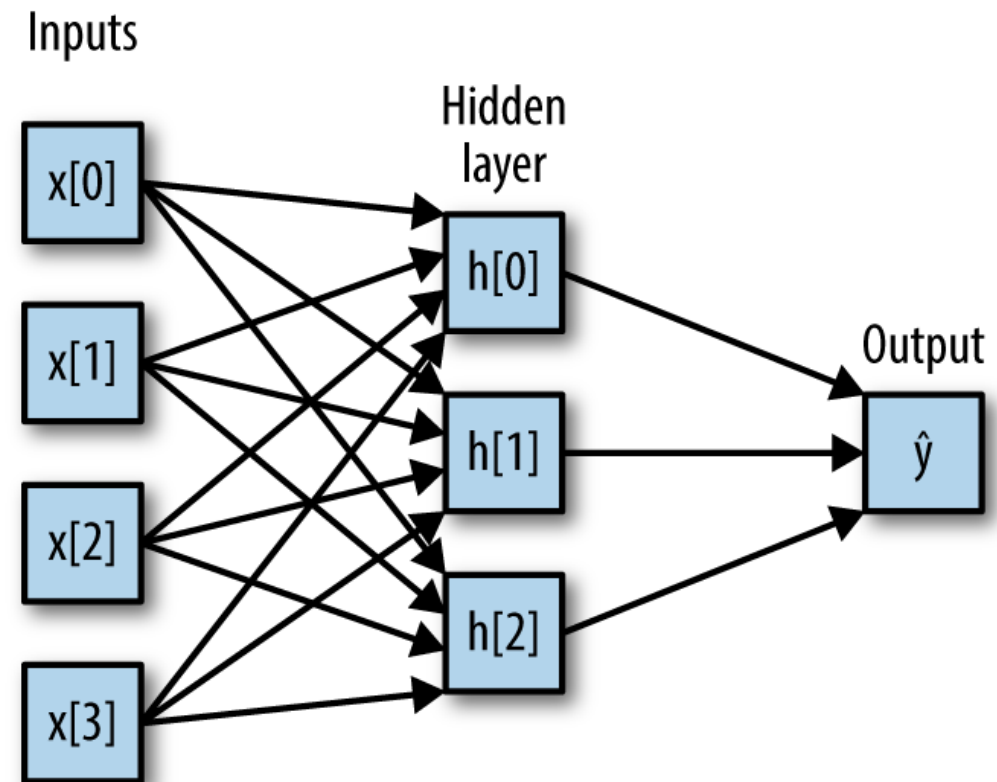
(“deep learning” are a revival of the neural networks tailored very carefully to a specific use case)

Here, each node on the left represents an input feature, the connecting lines represent the learned coefficients, and the node on the right represents the output, which is a weighted sum of the inputs.

In an MLP this process of computing weighted sums is repeated multiple times,

first computing **hidden units** that represent an intermediate processing step, which are again combined using weighted sums to yield the final result.

Multilayer perceptron with a single hidden layer

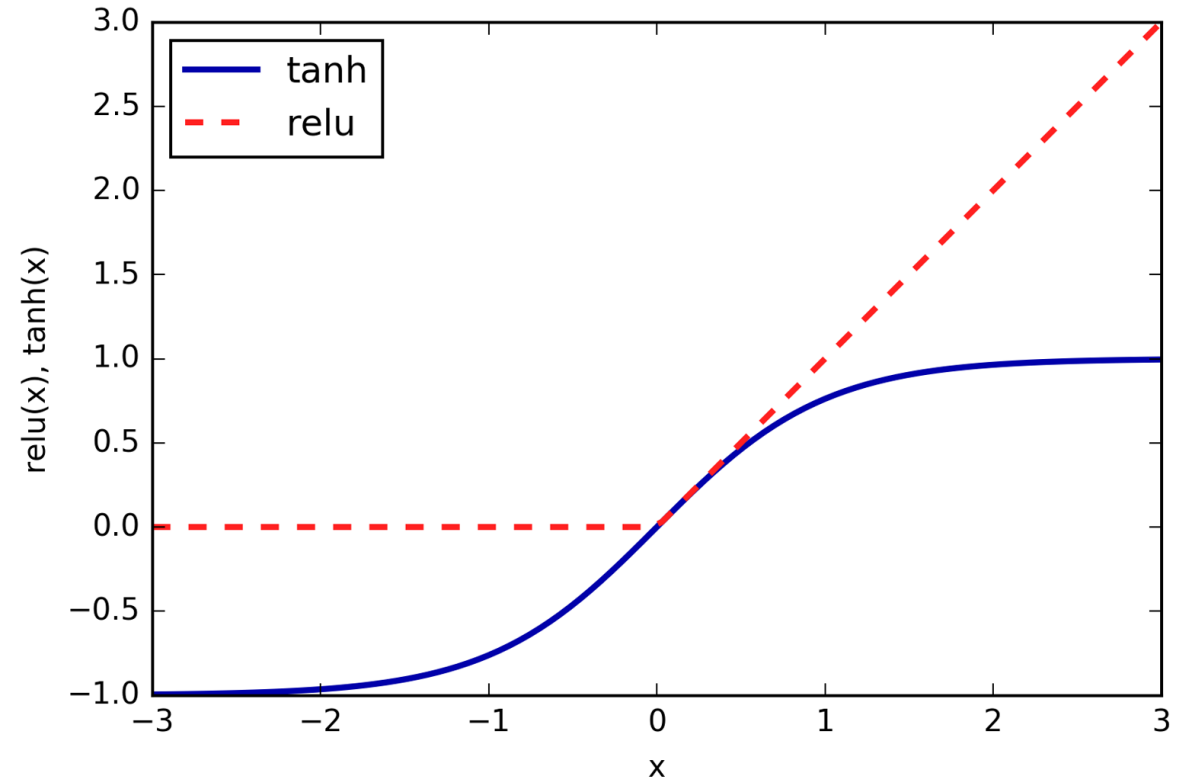


This model has a lot more coefficients (also called weights) to learn: there is one between every input and every hidden unit (which make up the hidden layer), and one between every unit in the hidden layer and the output.

Computing a series of weighted sums is mathematically the same as computing just one weighted sum, so to make this model truly more powerful than a linear model, we need one extra trick.

After computing a weighted sum for each hidden unit, a nonlinear function is applied to the result—usually the *rectifying nonlinearity* (also known as rectified linear unit or relu) or the *tangens hyperbolicus* (tanh).

The result of this function is then used in the weighted sum that computes the output, \hat{y} .



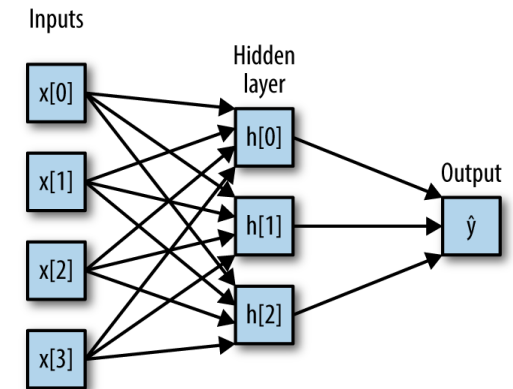
For the small neural network the full formula for computing \hat{y} in the case of regression would be (when using a tanh nonlinearity):

$$h[0] = \tanh(w[0, 0] * x[0] + w[1, 0] * x[1] + w[2, 0] * x[2] + w[3, 0] * x[3] + b[0])$$

$$h[1] = \tanh(w[0, 1] * x[0] + w[1, 1] * x[1] + w[2, 1] * x[2] + w[3, 1] * x[3] + b[1])$$

$$h[2] = \tanh(w[0, 2] * x[0] + w[1, 2] * x[1] + w[2, 2] * x[2] + w[3, 2] * x[3] + b[2])$$

$$\hat{y} = v[0] * h[0] + v[1] * h[1] + v[2] * h[2] + b$$

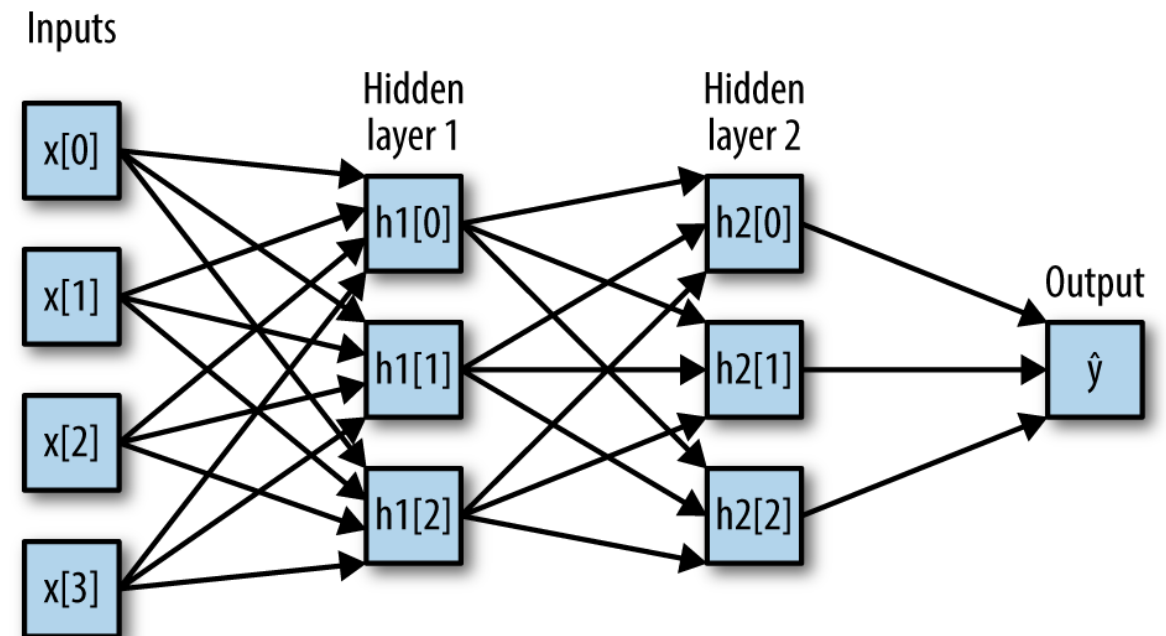


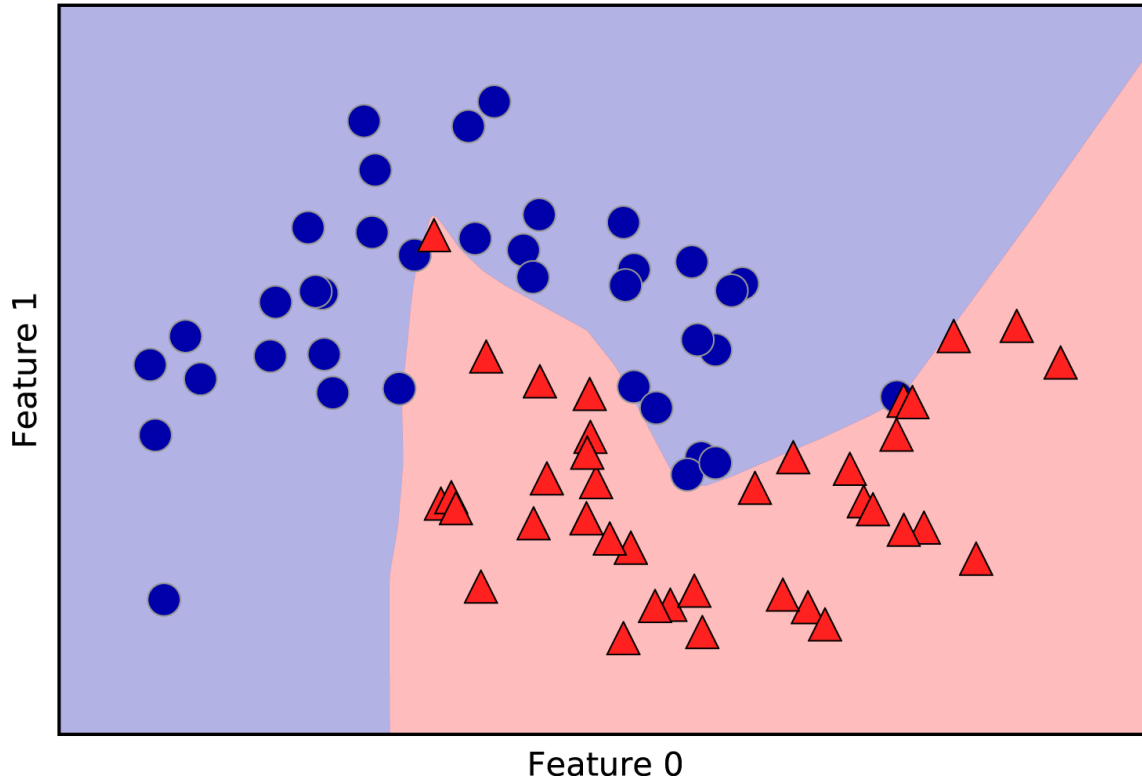
Here, w are the weights between the input x and the hidden layer h , and v are the weights between the hidden layer h and the output \hat{y} . The weights v and w are learned from data, x are the input features, \hat{y} is the computed output, and h are intermediate computations.

An important parameter that needs to be set by the user is the number of nodes in the hidden layer.

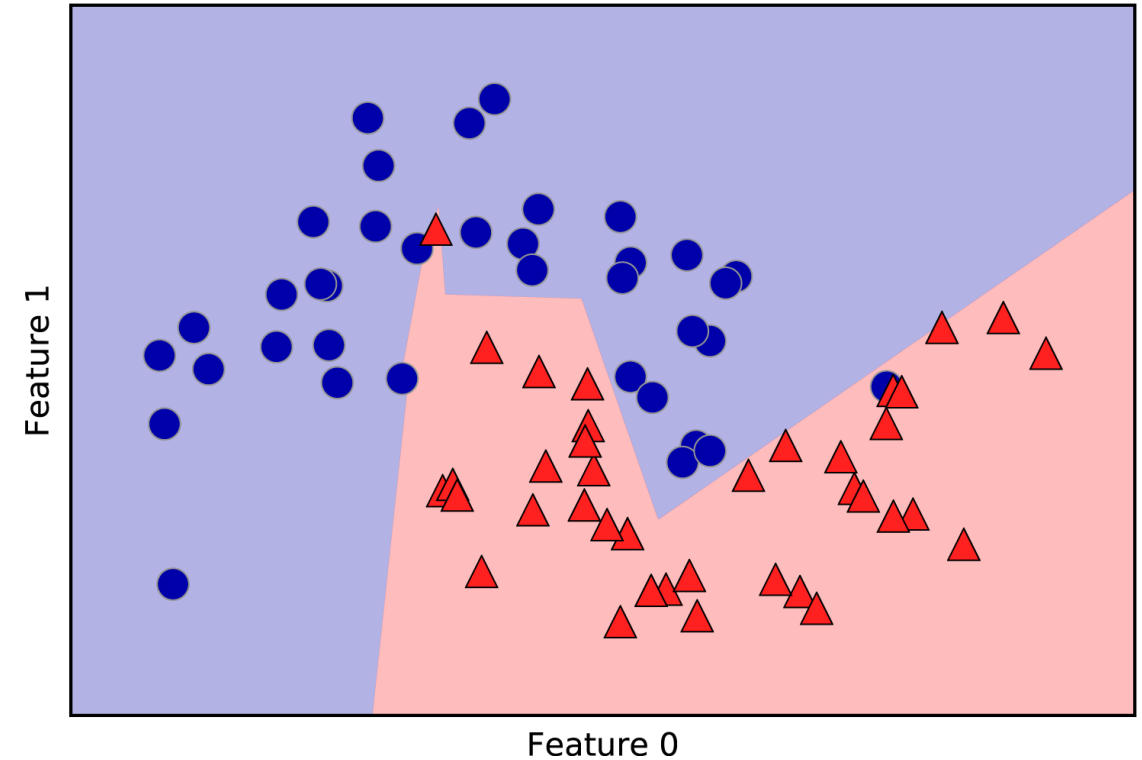
This can be as small as 10 for very small or simple datasets and as big as 10,000 for very complex data.

Having large neural networks made up of many of these layers of computation is what inspired the term **“deep learning.”**

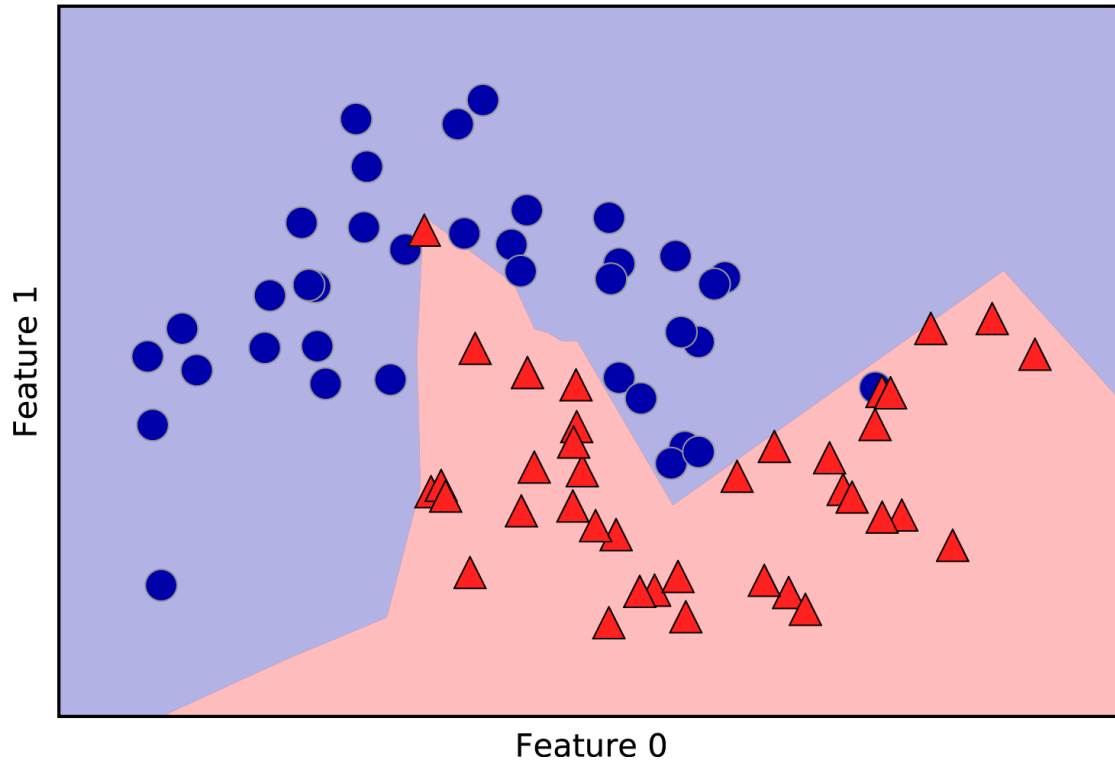




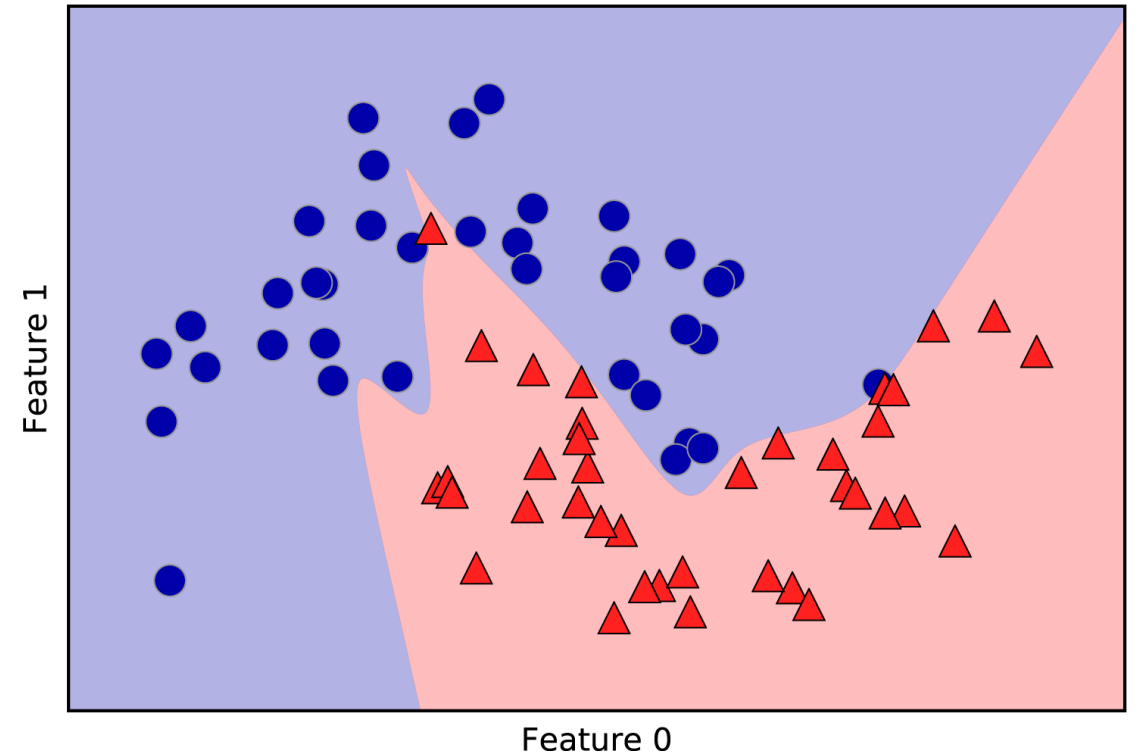
Decision boundary learned by a neural network with 100 hidden units on the two_moons dataset



Decision boundary learned by a neural network with 10 hidden units on the two_moons dataset



Decision boundary learned using 2 hidden layers with 10 hidden units each, with rect activation function



Decision boundary learned using 2 hidden layers with 10 hidden units each, with tanh activation function

A quick summary of when to use each model:

Algorithm	Characteristics
Nearest neighbors	For small datasets, good as a baseline, easy to explain.
Decision trees	Very fast, don't need scaling of the data, can be visualized and easily explained.
Random forests	Nearly always perform better than a single decision tree, very robust and powerful. Don't need scaling of data. Not good for very high-dimensional sparse data.
Support vector machines	Powerful for medium-sized datasets of features with similar meaning. Require scaling of data, sensitive to parameters.
Neural networks	Can build very complex models, particularly for large datasets. Sensitive to scaling of the data and to the choice of parameters. Large models need a long time to train.

Classification using Random Forest algorithm

