Machine Learning

João Catalão Fernandes, FCUL

Deteção Remota Multiespectral, MEGeoespacial, MSIG-TA

APPLICATIONS

ESA selects four new Earth Explorer mission ideas

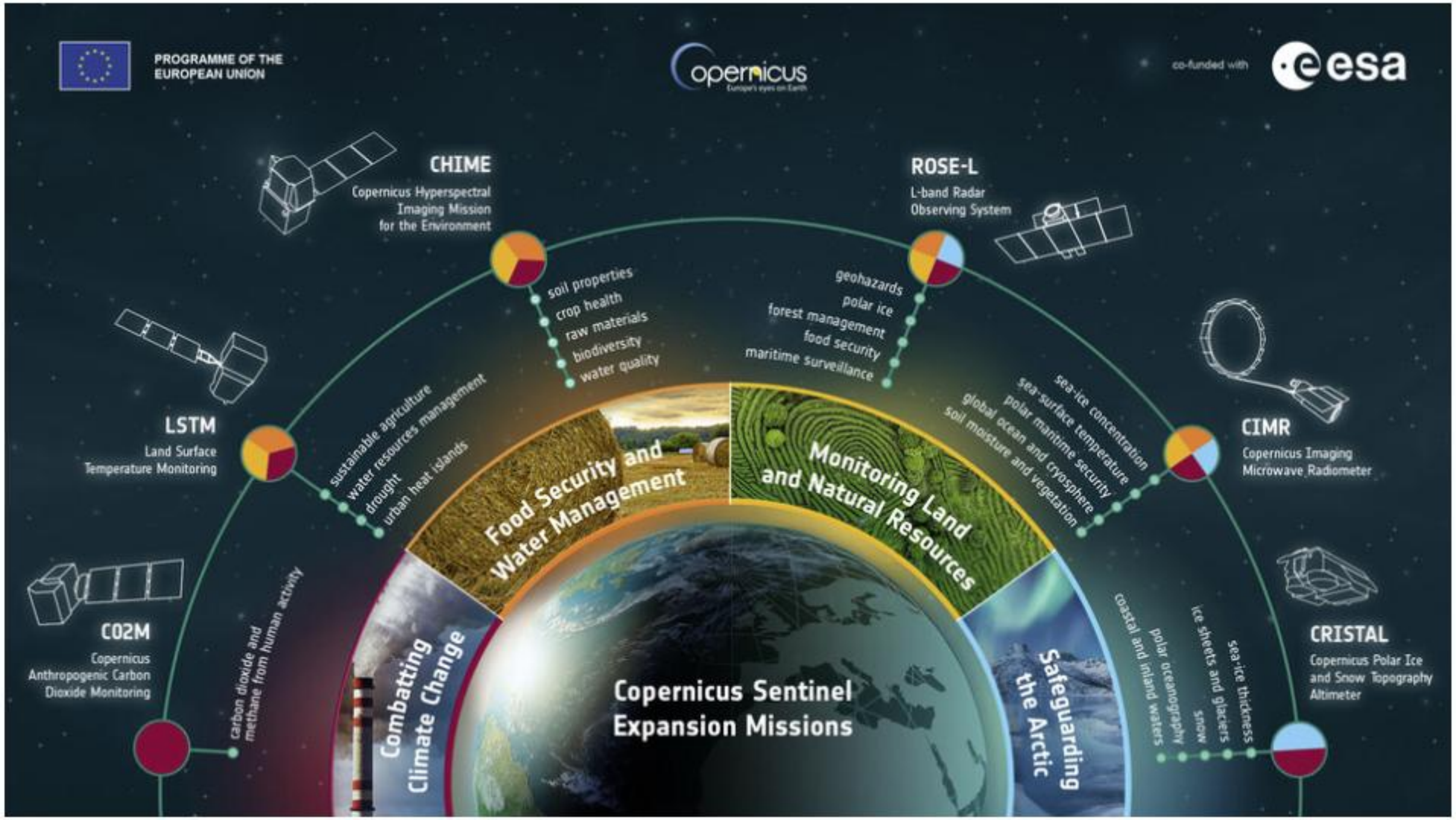
The four selected mission ideas

CryoRad would fill an important gap in observations of the cryosphere through the direct measurement of low-frequency passive-microwave brightness temperatures using a novel broadband radiometer. From these novel measurements key parameters such as the temperature profile of ice shelves, sea-ice thickness and sea-surface salinity in cold waters can be determined to improve our understanding of key processes in the polar regions. The mission would complement the upcoming Copernicus CIMR, CRISTAL and ROSE-L missions.

ECO would measure the difference between incoming solar radiation and outgoing radiation, which defines Earth's energy 'imbalance', and which fundamentally controls Earth's climate system. It would be the first time that this imbalance has been measured directly and would help reveal the future trajectory of the climate decades earlier than relying on monitoring global temperature and sea-level rise, as is currently the case. The unique concept envisages a satellite constellation, each carrying four wide field-of-view radiometers to ensure unprecedented coverage, accuracy and stability.

Hydroterra+ would be placed in geostationary orbit, which is unusual for an Earth-science radar mission. From this fixed position above the equator, the satellite's C-band synthetic aperture radar would deliver data twice a day over Europe, the Mediterranean and northern Africa to understand rapid processes tied to the water cycle and tectonic events in these regions.

Keystone would provide the first direct observations of atomic oxygen in the altitude range of 50–150 km using a unique combination of limb-sounding techniques. These measurements together with observations of composition, temperature and winds would allow scientists to study the processes that drive the variability and energy balance of the mesosphere-lower-thermosphere region of the atmosphere, also looking at the impact of solar cycles and space weather.


PROGRAMME OF THE EUROPEAN UNION
Copernicus
co-funded with esa
CHIME
Copernicus Hyperspectral Imaging Mission for the Environment
soil properties
crop health
raw materials
biodiversity
water quality
ROSE-L
L-band Radar Observing System
geohazards
polar ice
forest management
food security
maritime surveillance
LSTM
Land Surface Temperature Monitoring
sustainable agriculture
water resources management
drought
urban heat islands
CIMR
Copernicus Imaging Microwave Radiometer
sea-ice concentration
sea-surface temperature
polar maritime security
global ocean and cryosphere
soil moisture and vegetation
CO2M
Copernicus Anthropogenic Carbon Dioxide Monitoring
carbon dioxide and methane from human activity
CRISTAL
Copernicus Polar Ice and Snow Topography Altimeter
sea-ice thickness
ice sheets and glaciers
snow
polar oceanography
coastal and inland waters
Food Security and Water Management
Monitoring Land and Natural Resources
Combatting Climate Change
Safeguarding the Arctic
Copernicus Sentinel Expansion Missions

Tópicos

What is Machine Learning?

Machine Learning

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

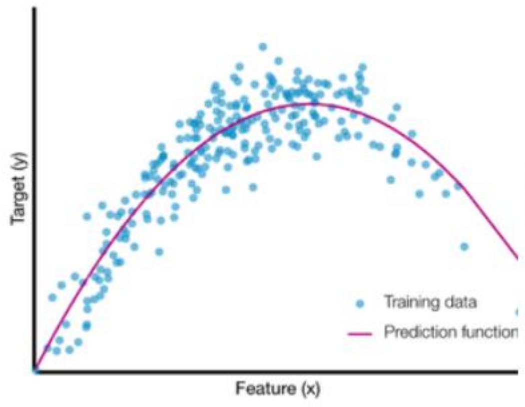
Machine learning is believed to be the powerful tool to explore and analyze the geography big data.

What is machine learning?

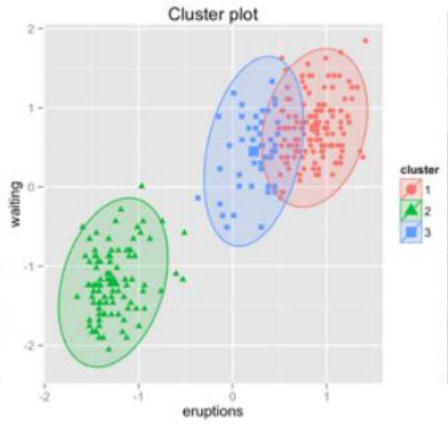
Machine learning evolved from the study of **pattern recognition** and **computational learning theory** in **artificial intelligence (AI).**

Machine Learning (Aprendizagem Automática)

"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" — *Tom Michell (1997)*



regression



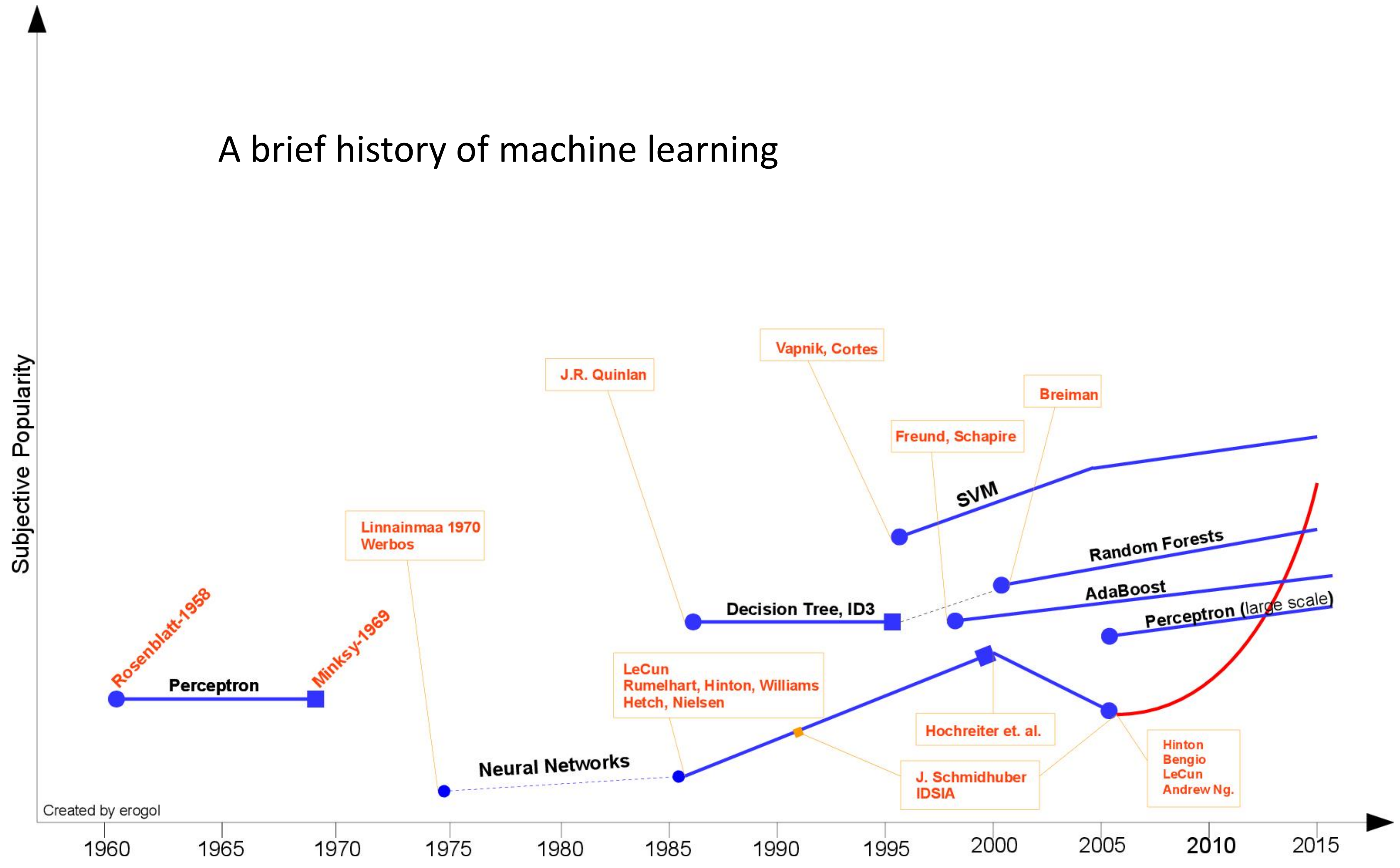
clustering

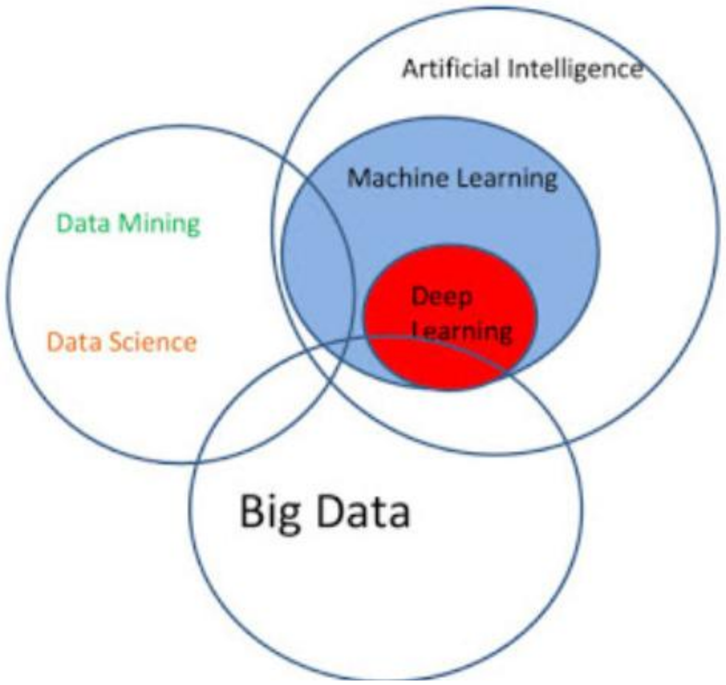
classification

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

Machine Learning

A brief history of machine learning



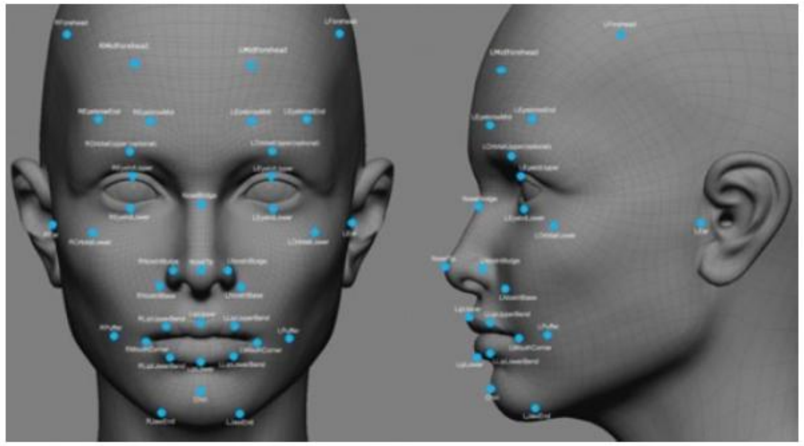


http://www.erogol.com/wp-content/uploads/2014/05/test.jpg

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/

How to connect the machine learning with geospatial data?

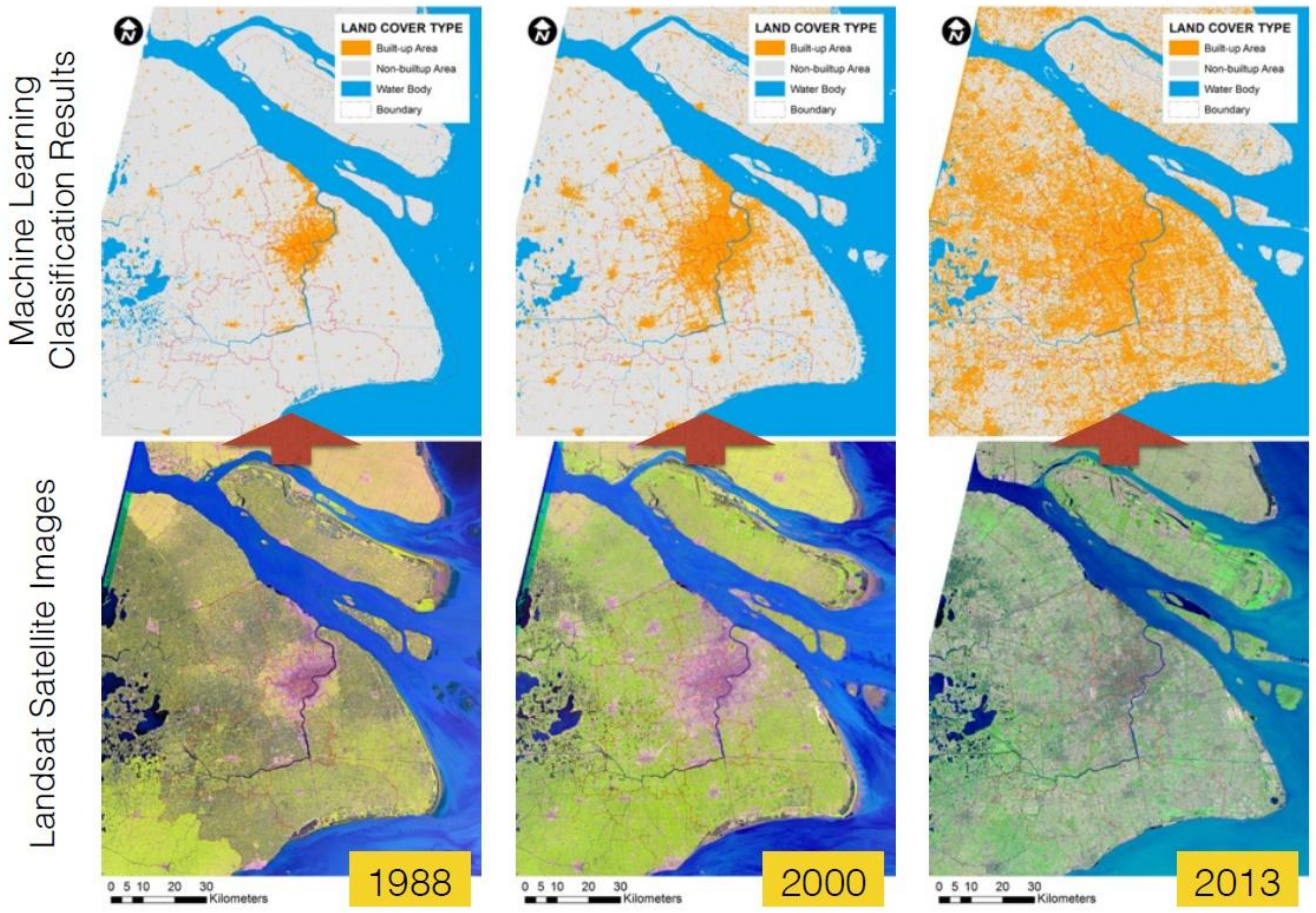
Geospatial Big Data

Remote sensing multispectral image data,
behavioral geography data (person trip),
transportation network data,

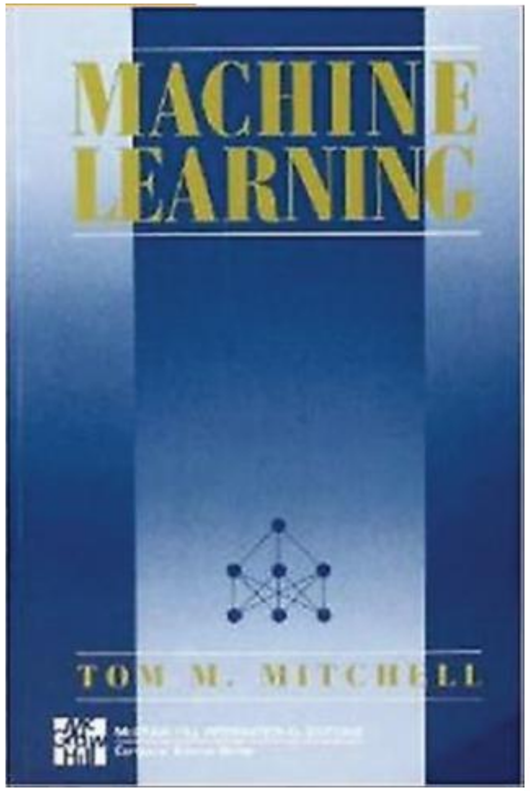
. . .

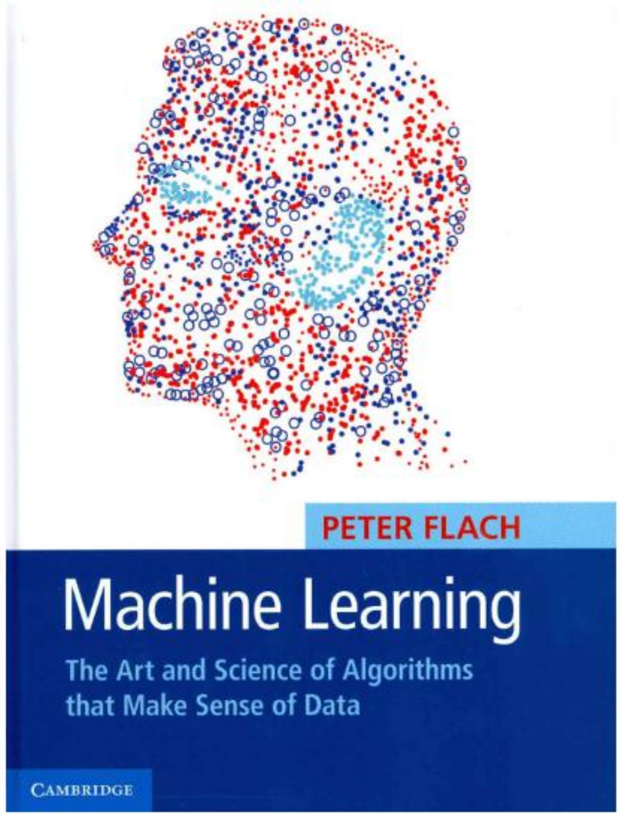
http://2012.bedreinnovation.dk/aktivitet/data-mining-og-machine-learning-i-praksis

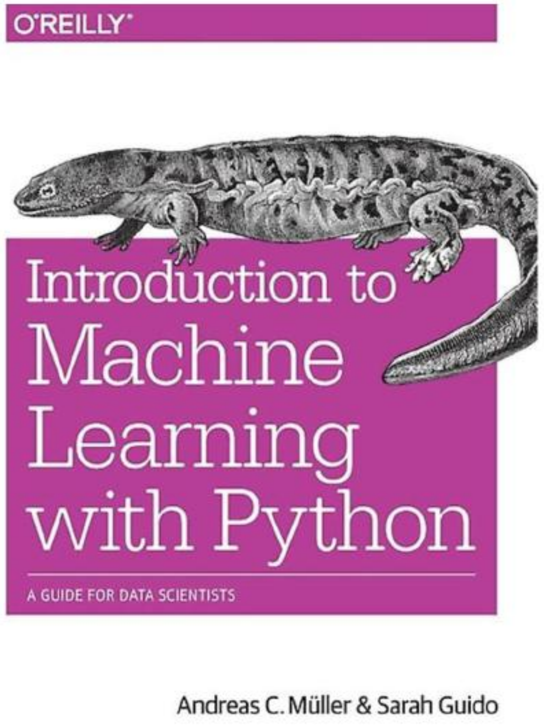
Machine Learning in Remote Sensing

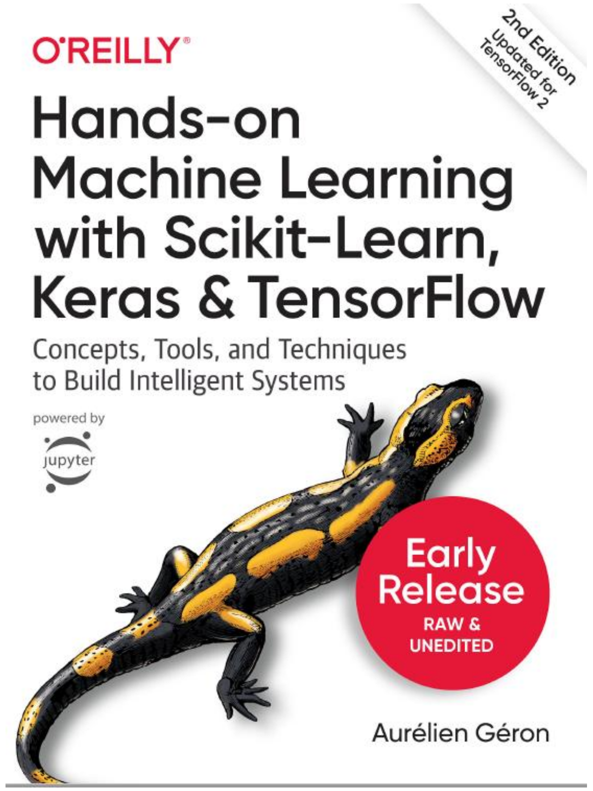


Bibliography and slides






O'REILLY
Introduction to Machine Learning with Python
A GUIDE FOR DATA SCIENTISTS
Andreas C. Müller & Sarah Guido


O'REILLY
2nd Edition Updated for TensorFlow 2
Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow
Concepts, Tools, and Techniques to Build Intelligent Systems
powered by jupyter
Early Release RAW & UNEDITED
Aurélien Géron

Software

ANACONDA
"The Most Popular Python Data Science Platform"

python™
Python 3

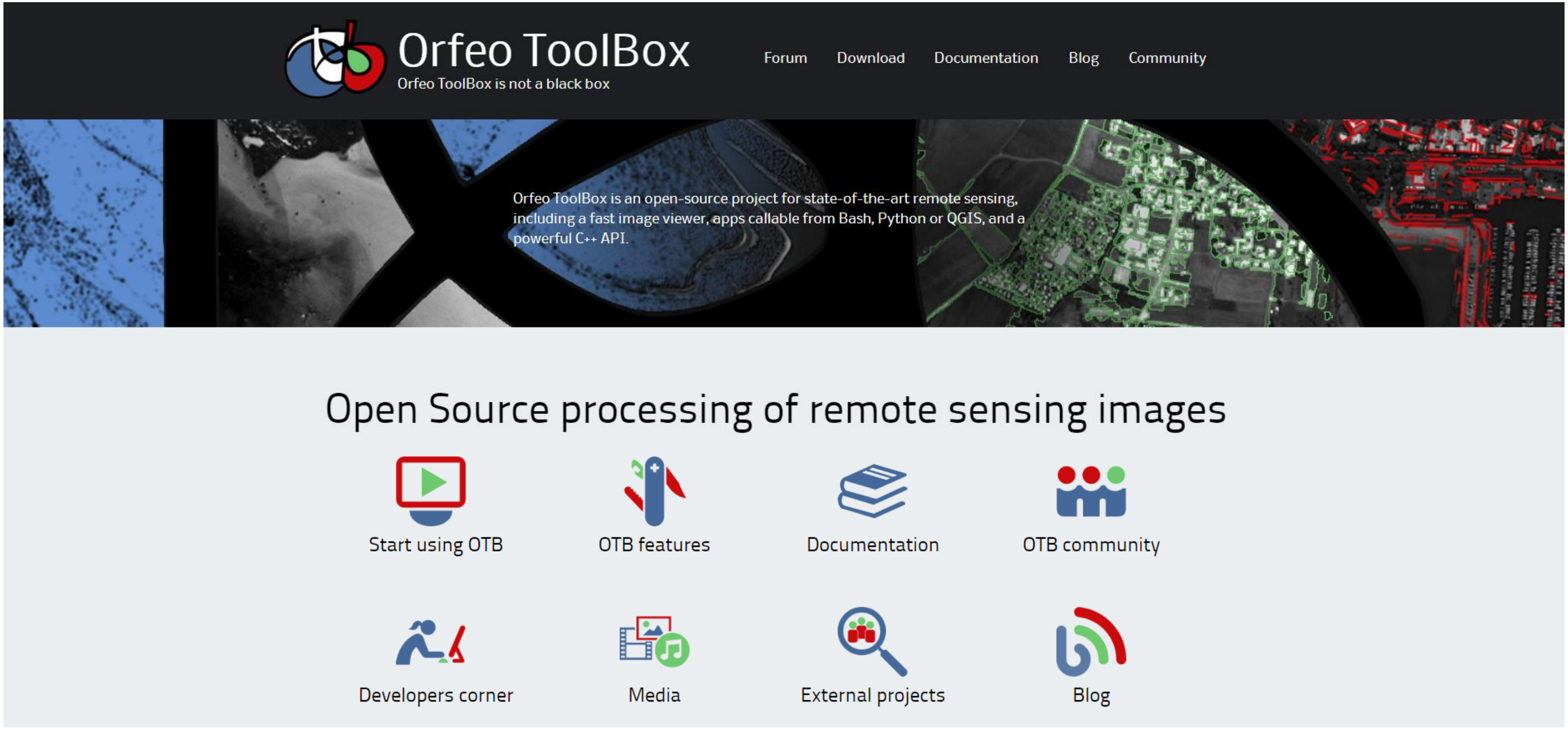
orange3
3.13.0
"Component based data mining framework. Data visualization and data analysis for novice and experts. Interactive workflows with a large toolbox.

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

scikits learn
machine learning in Python

TensorFlow 2.0

Software



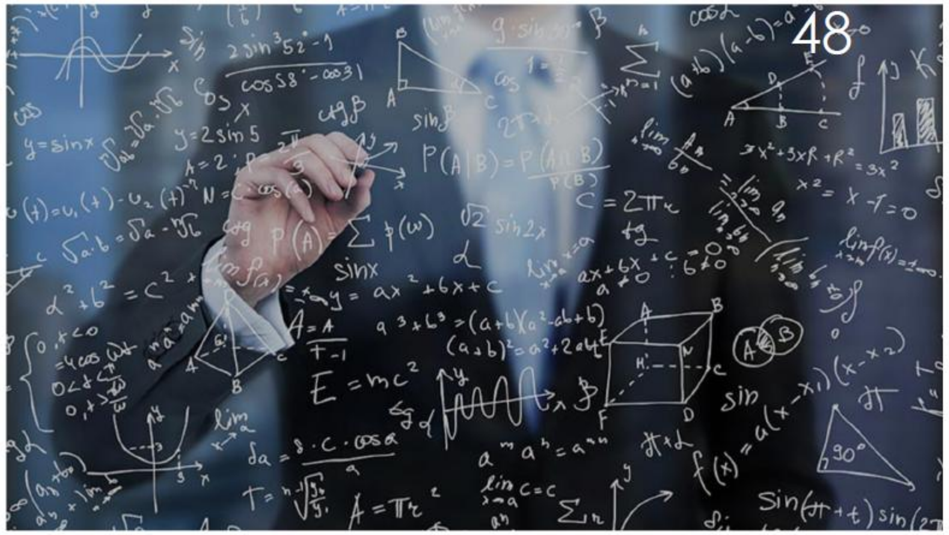
Orfeo ToolBox is an open-source project for state-of-the-art remote sensing, including a fast image viewer, apps callable from Bash, Python or QGIS, and a powerful C++ API.

Open Source processing of remote sensing images

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

Tasks for machine learning

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.

	Predictive model	*Descriptive model*
Supervised learning	classification, regression	subgroup discovery
Unsupervised learning	predictive clustering	descriptive clustering, association rule discovery

Machine learning Models

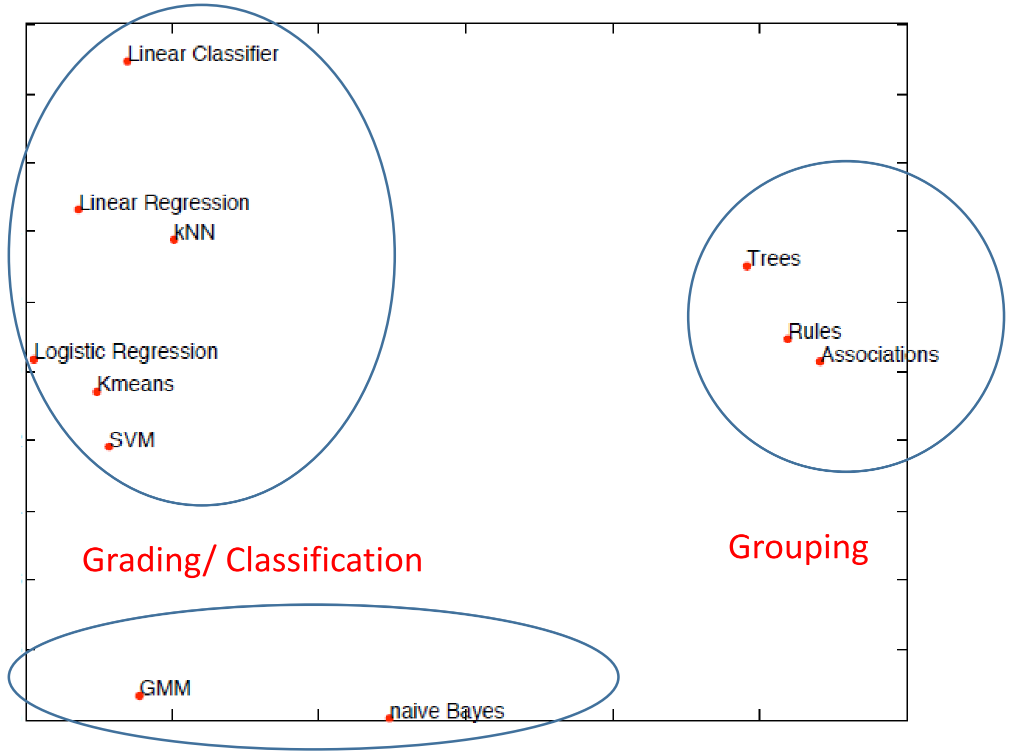
Machine learning models can be distinguished according to their main intuition:

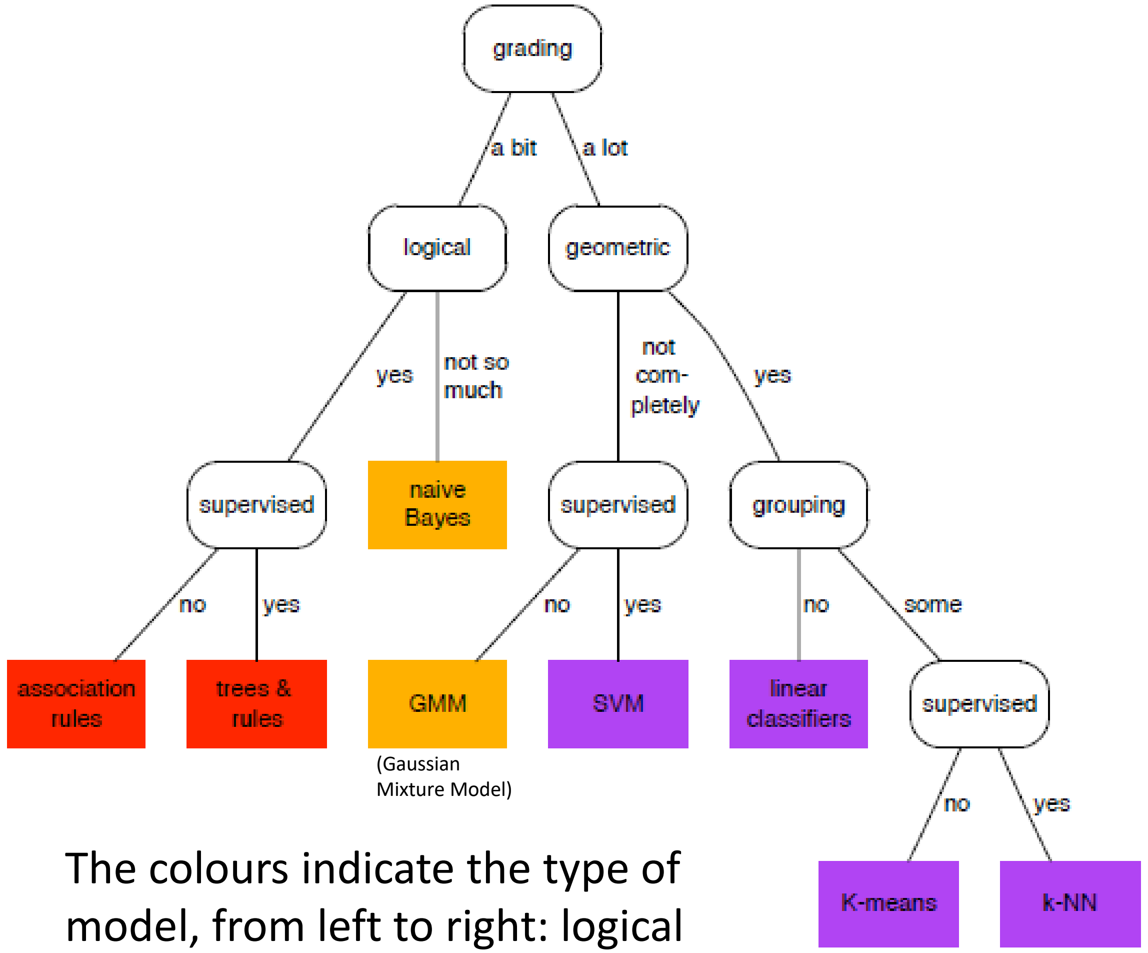
- ☞ *Geometric* models use intuitions from geometry such as separating (hyper-)planes, linear transformations and distance metrics.
- ☞ *Probabilistic* models view learning as a process of reducing uncertainty, modelled by means of probability distributions.
- ☞ *Logical* models are defined in terms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

Machine Learning Taxonomy





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).

Predictive machine learning scenarios

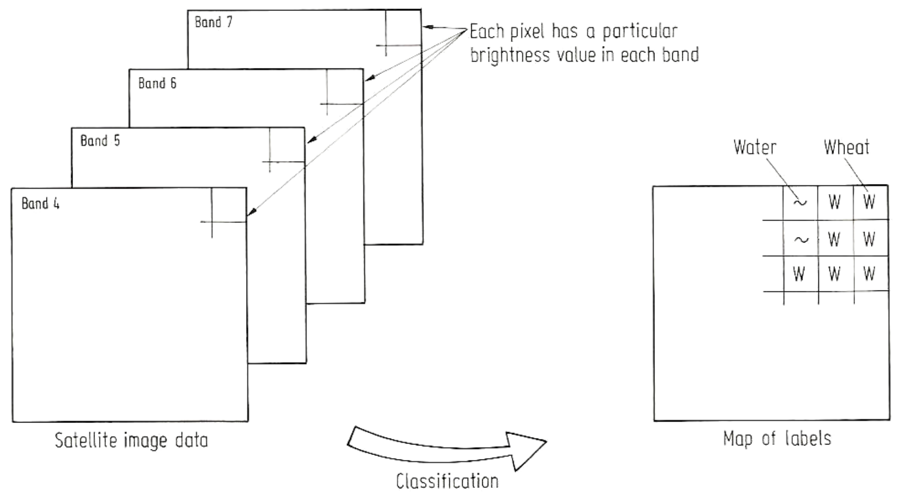
Task	*Label space*	*Output space*	*Learning problem*
Classification	$\mathscr{L} = \mathscr{C}$	$\mathscr{Y} = \mathscr{C}$	learn an approximation $\hat{c} : \mathscr{X} \rightarrow \mathscr{C}$ to the true labelling function c
Scoring and ranking	$\mathscr{L} = \mathscr{C}$	$\mathscr{Y} = \mathbb{R}^{\lvert\mathscr{C}\rvert}$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

Classification

A *classifier* is a mapping $\hat{c} : \mathscr{X} \rightarrow \mathscr{C}$, where $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ 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 an estimate of the true but unknown function $c(x)$. Examples for a classifier take the form $(x, c(x))$, where $x \in \mathscr{X}$ is an instance and $c(x)$ is the true class of the instance (sometimes contaminated by noise).

Learning a classifier involves constructing the functic
as closely as possible (and not just on the training se
instance space $\mathscr{X}$).

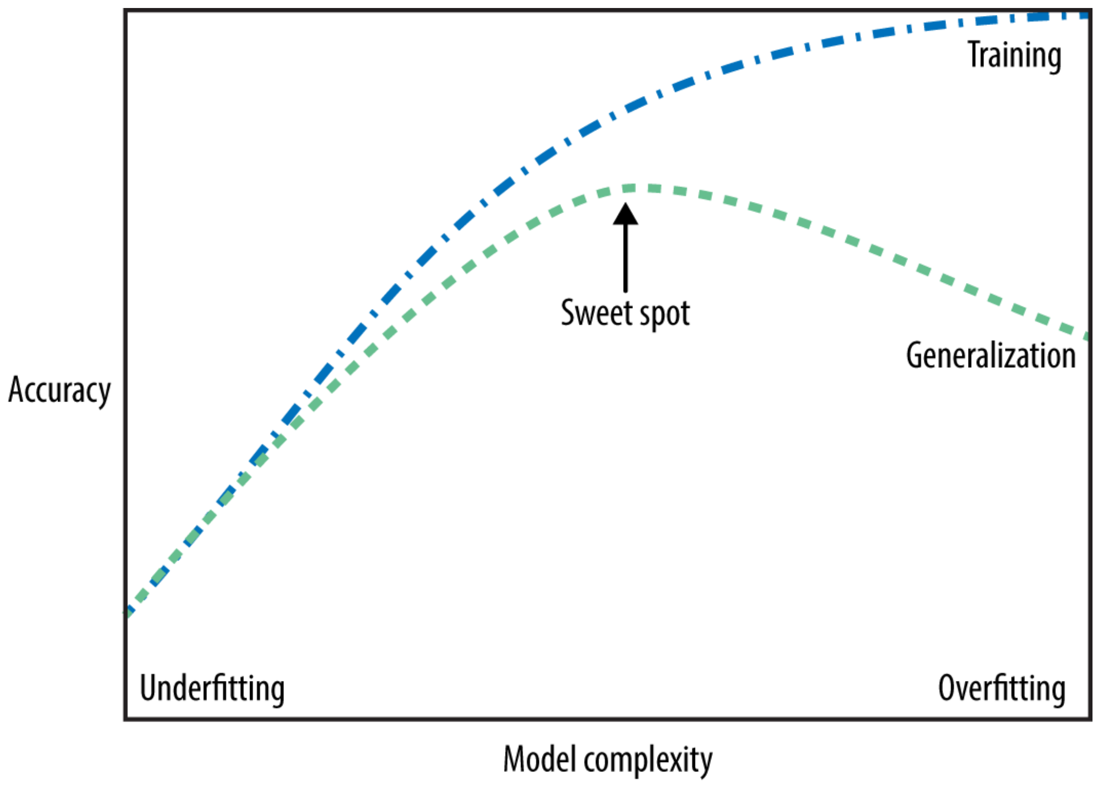


Generalization, Overfitting, and Underfitting

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.

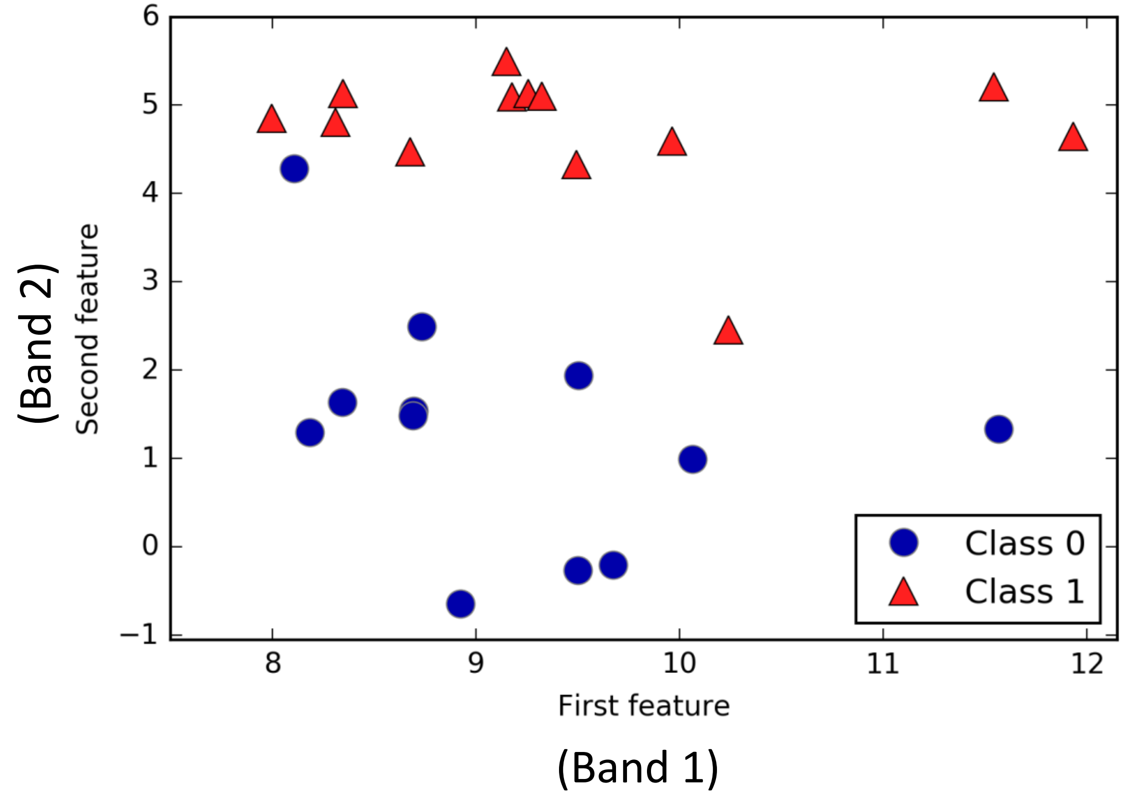
Trade-off of model complexity against training and test accuracy



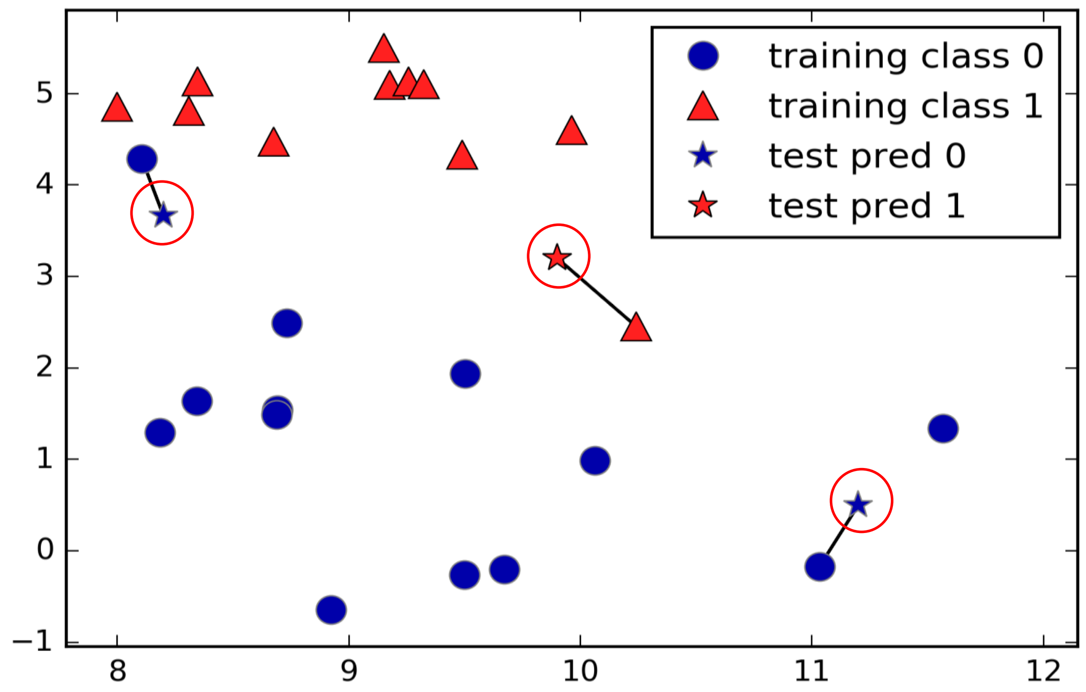
More complex the model => better we will be able to predict on the training data.
However : Too complex => focusing too much in our training set => not generalize well to new data.
There is a sweet spot in between that will yield the best generalization performance.

K-NN (K-Nearest Neighbours)

Scatter plot of training dataset
2 bands and 2 classes



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



K-NN (K-Nearest Neighbours)

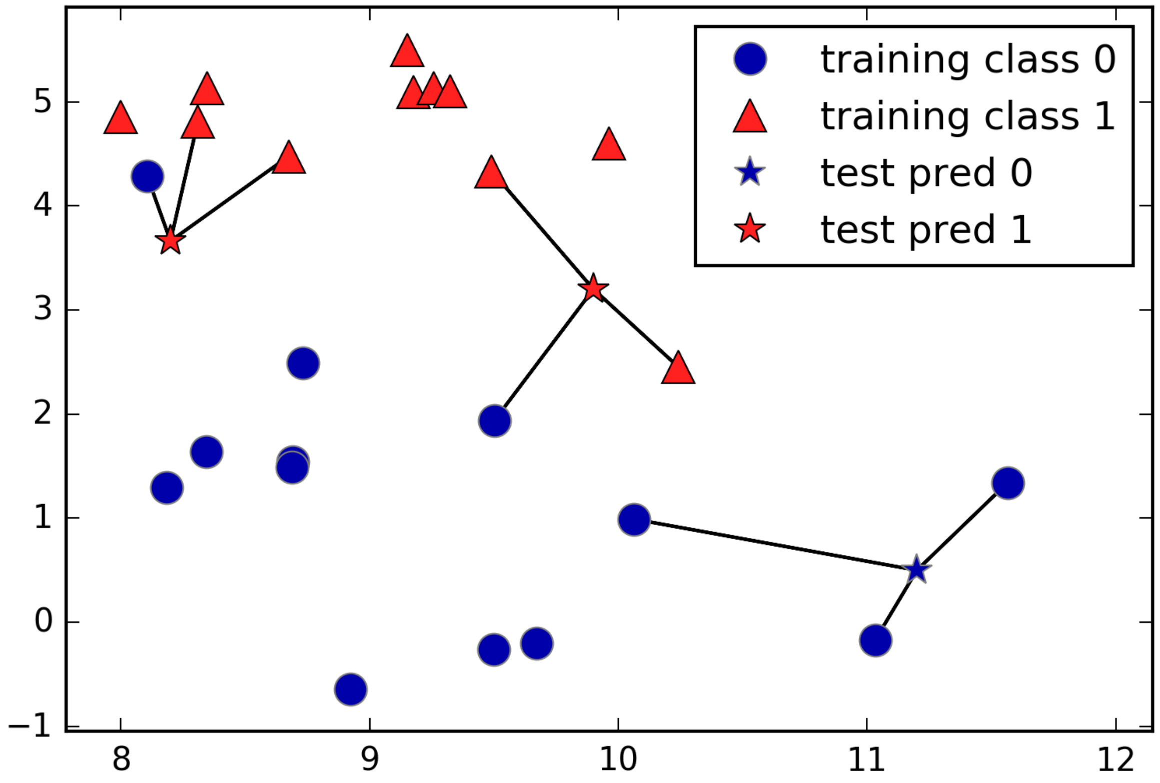
Instead of considering only the closest neighbour, we can also consider an arbitrary number, *k*, of neighbours.

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

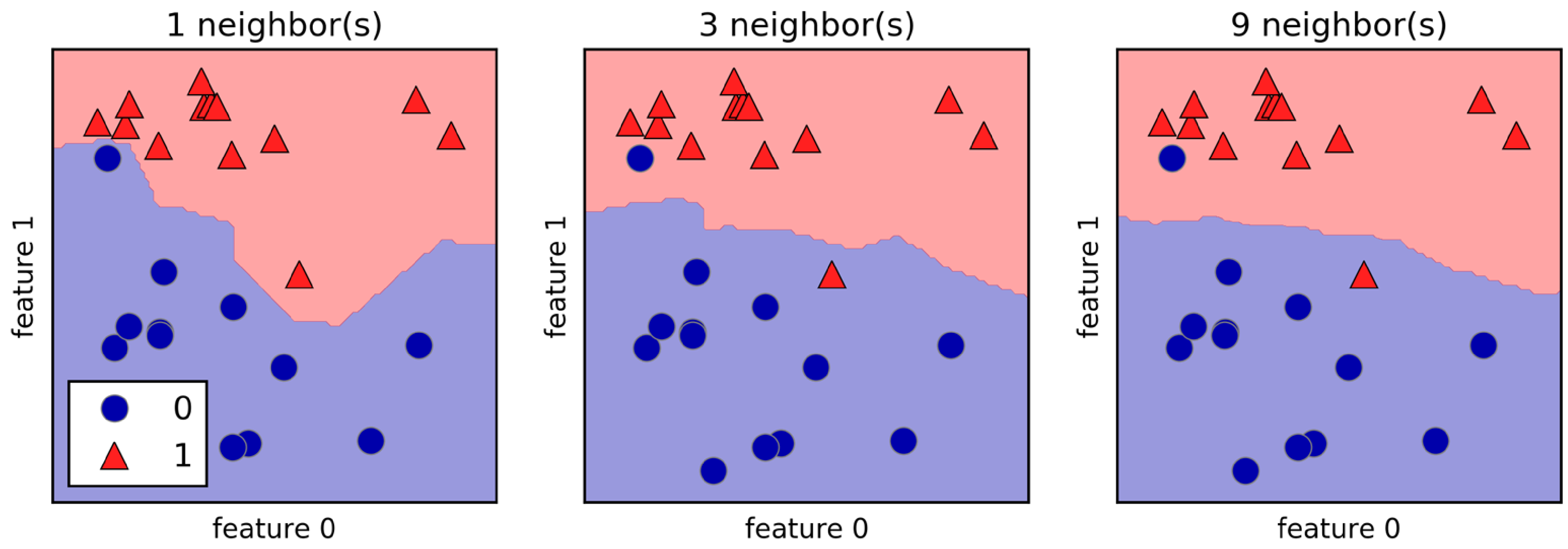
When considering more than one neighbour, we use ***voting*** to assign a label. This means that for each test point, we count how many neighbours belong to class 0 and how many neighbours belong to class 1.

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



K-NN (K-Nearest Neighbours)



Decision boundaries created by the nearest neighbours model for different values of k_neighbours

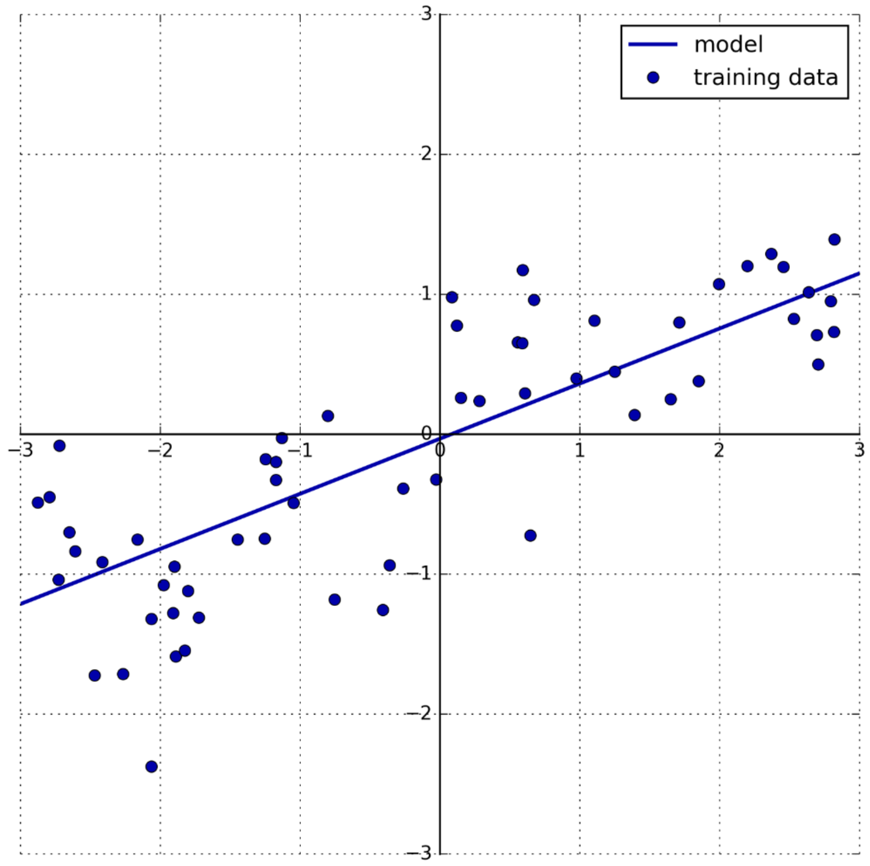
Linear Models

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:

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

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.



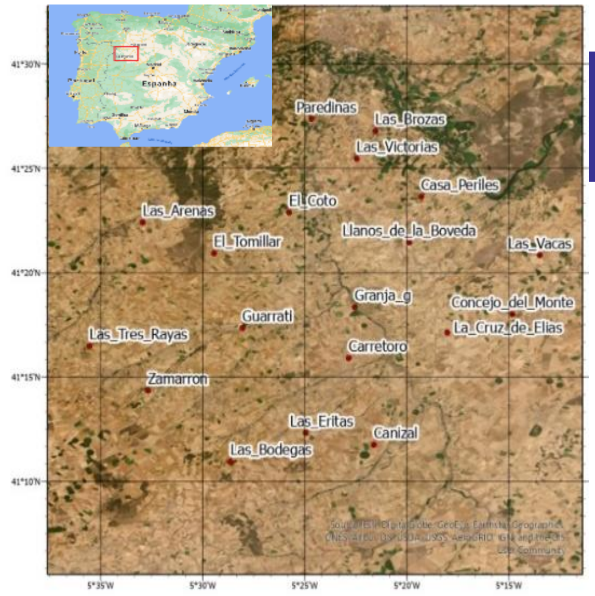
For a dataset with a single feature, this is:

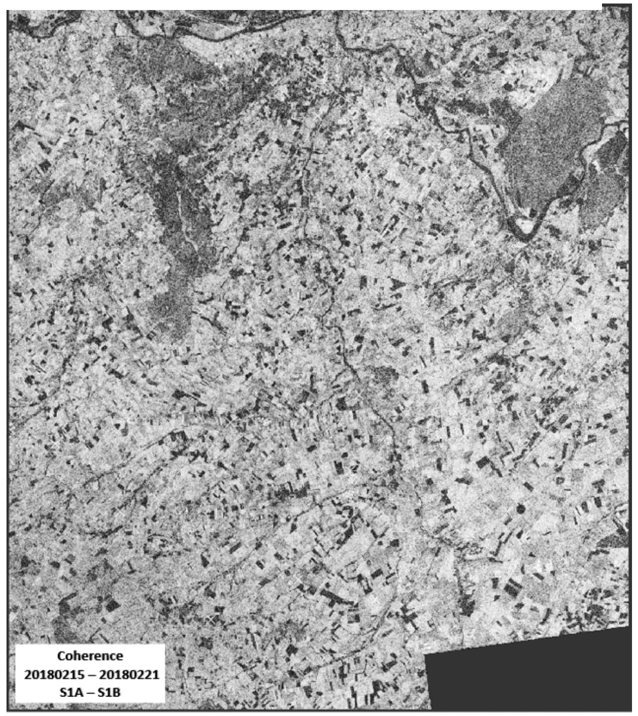
$$\hat{y} = w[0] * x[0] + b$$

Assess the usefulness of the temporal coherence matrix on the estimation of the soil moisture changes

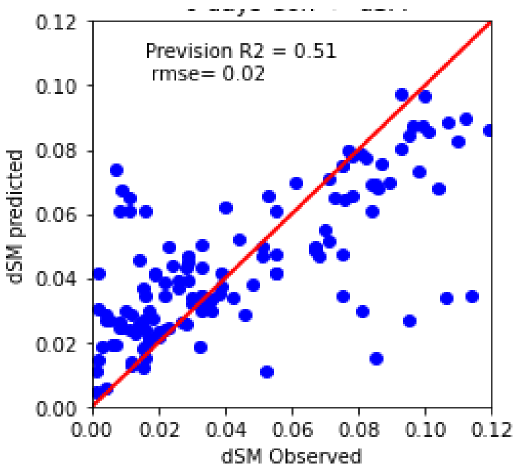
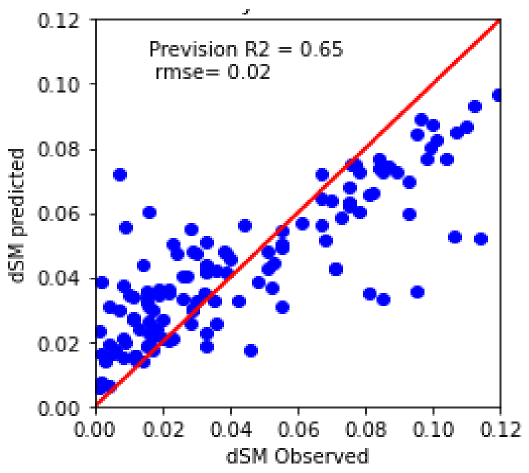
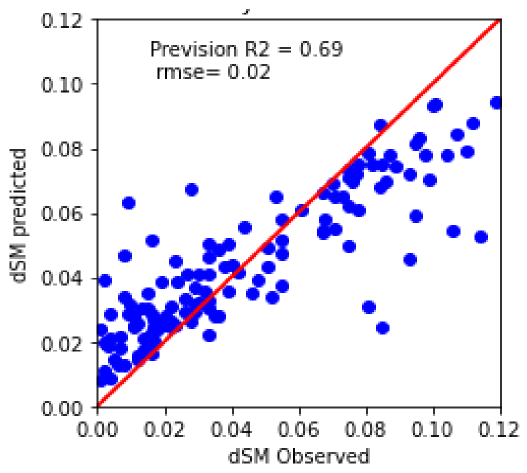
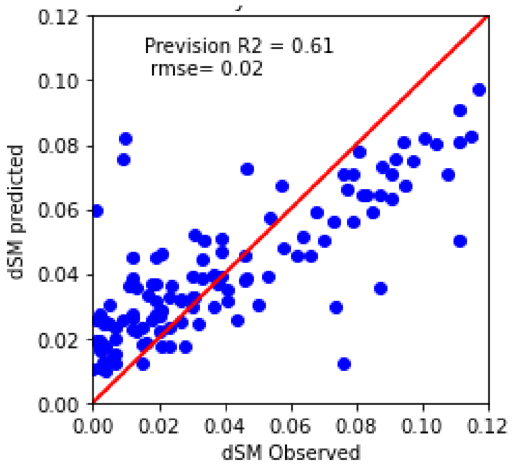
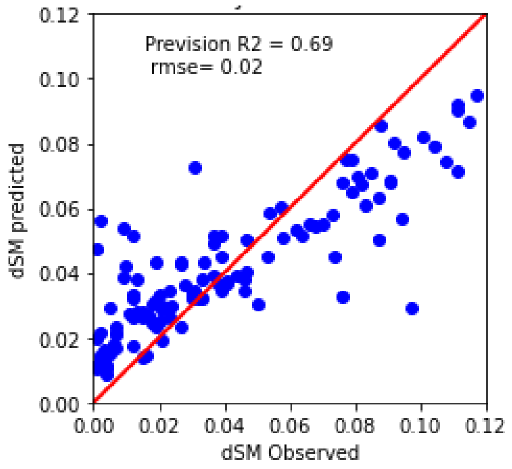
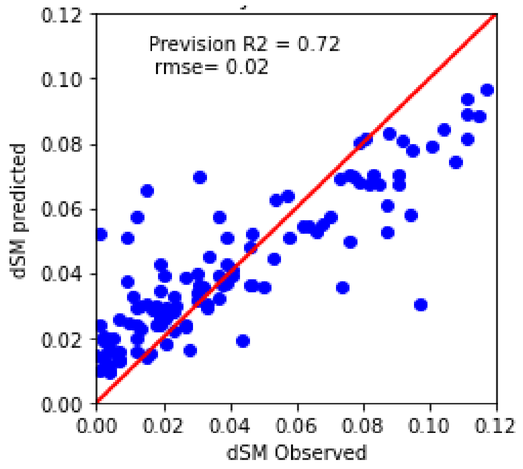
- Machine Learning Regression Techniques
  - Linear Regression (LR)
  - Random Forest Regressor (RFR)
  - ExtraTree + Bagging Regressor (ETBR)

- Data inputs: InSAR coherence, phase and soil type
- Data output: soil moisture change between two dates





SM estimation vs SM observation

	Coh → dSM	Coh & Phase → dSM	Coh, Phase & Soil → dSM
Orbit 74			
Orbit 154			

Linear Models for Classification

Linear models are also extensively used for classification.

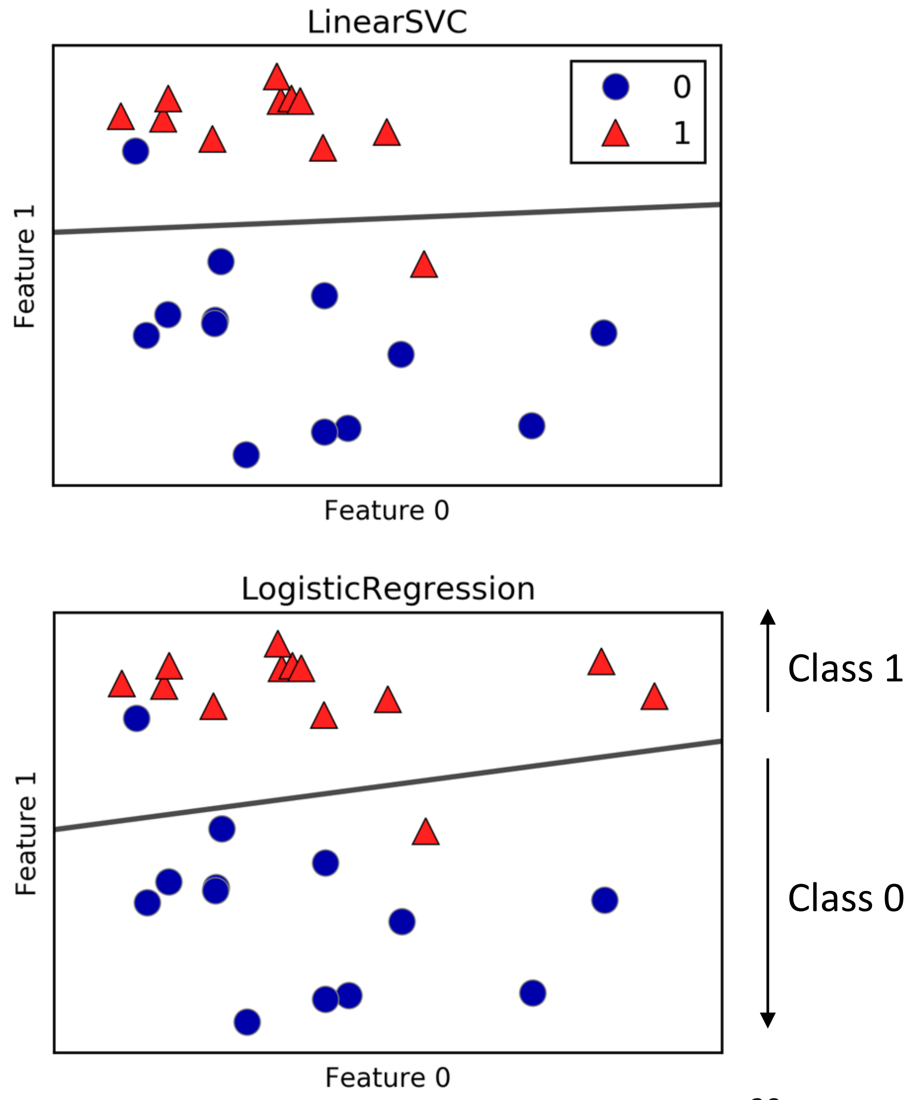
In this case, a prediction is made using the following formula:

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

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).



Linear Models for multiclass Classification

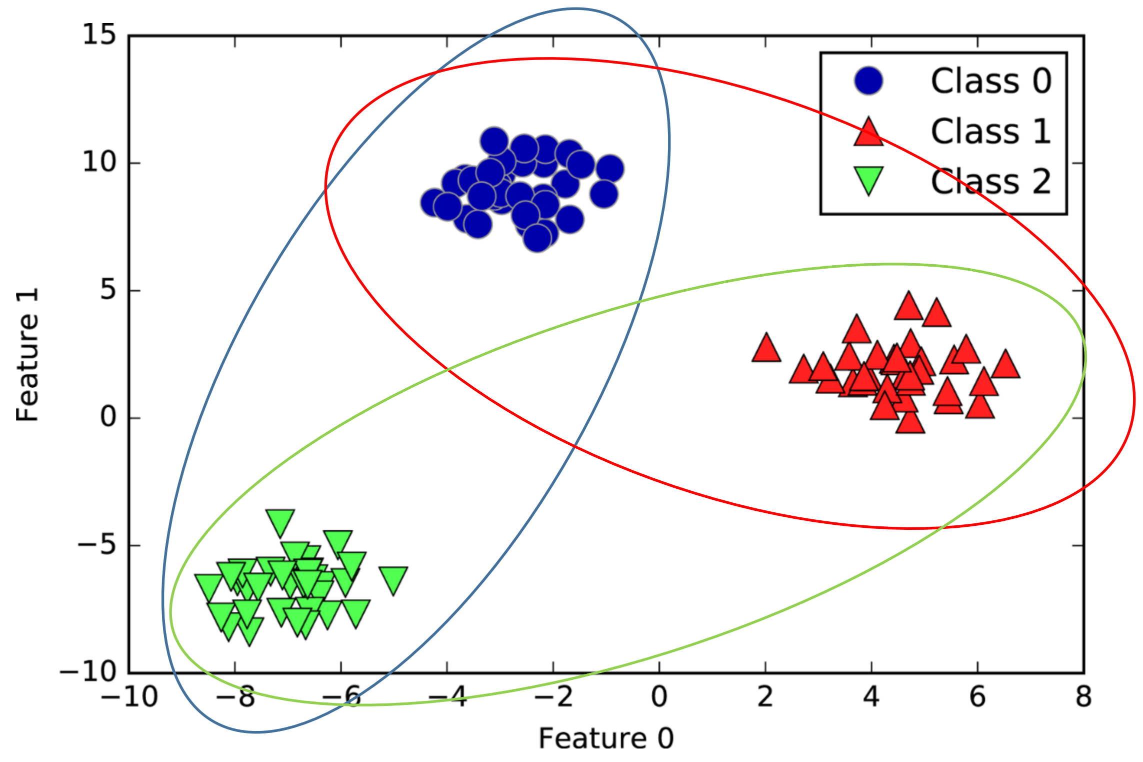
A common technique to extend a binary classification algorithm to a multiclass classification algorithm is the ***one-vs.-rest* approach.**

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:

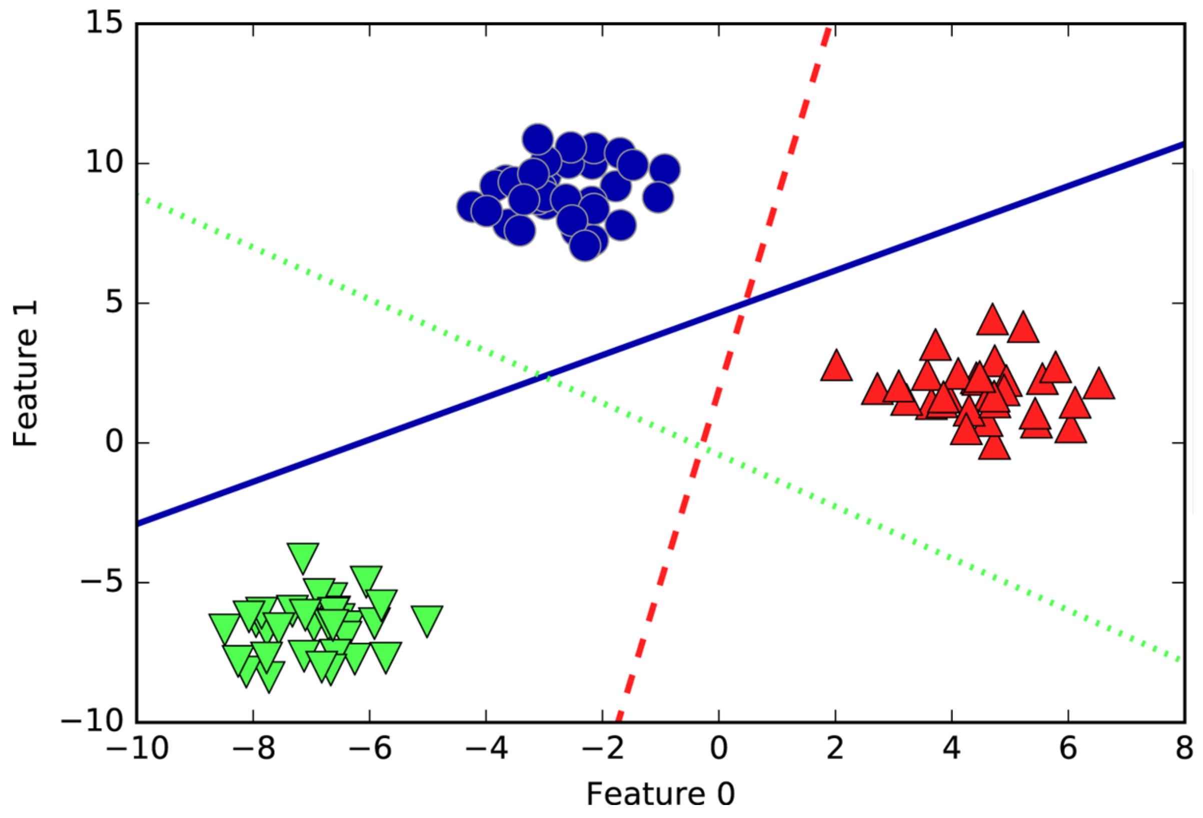
$w[0] * x[0] + w[1] * x[1] + \ldots + w[p] * x[p] + b$

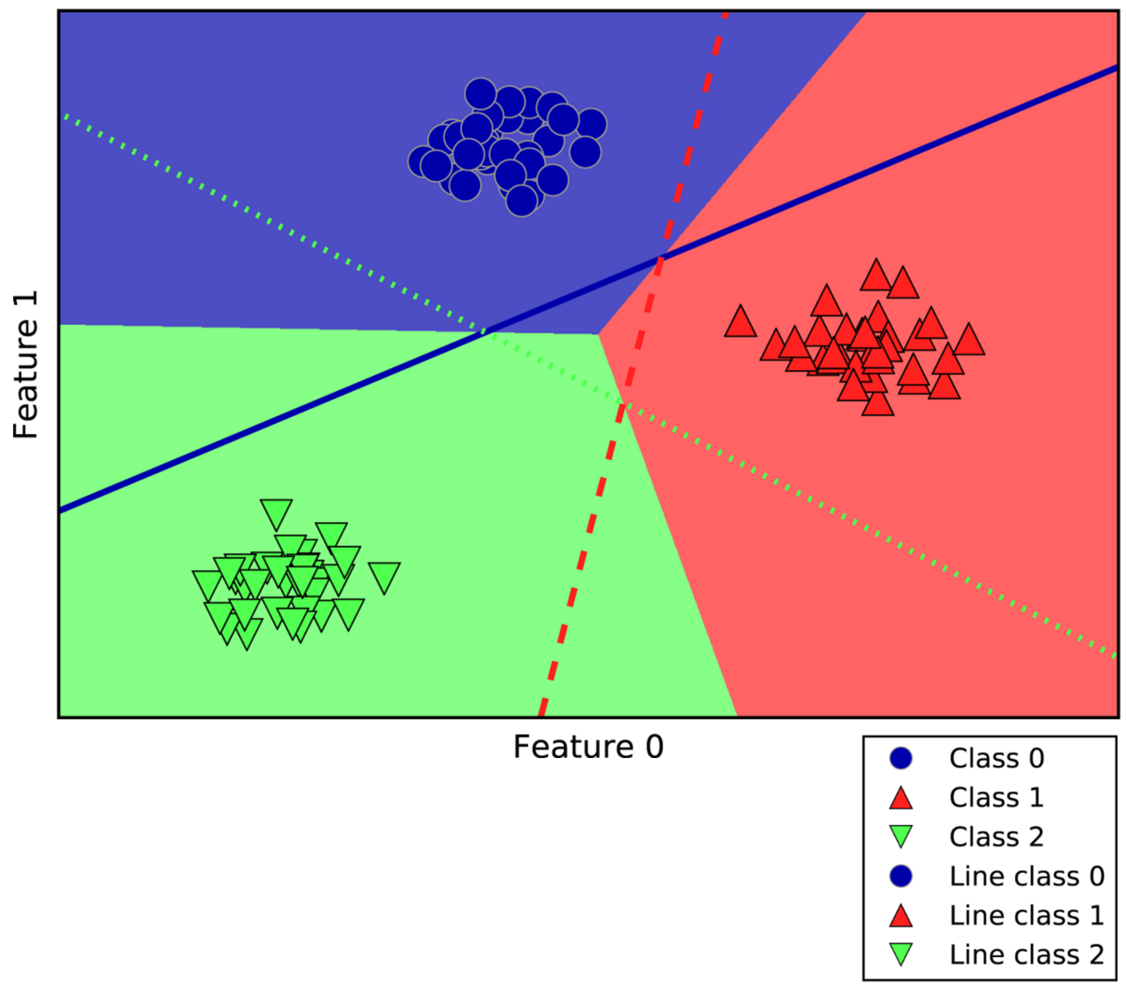


The classifier that has the highest score on its single class "wins," and this class label is returned as the prediction.

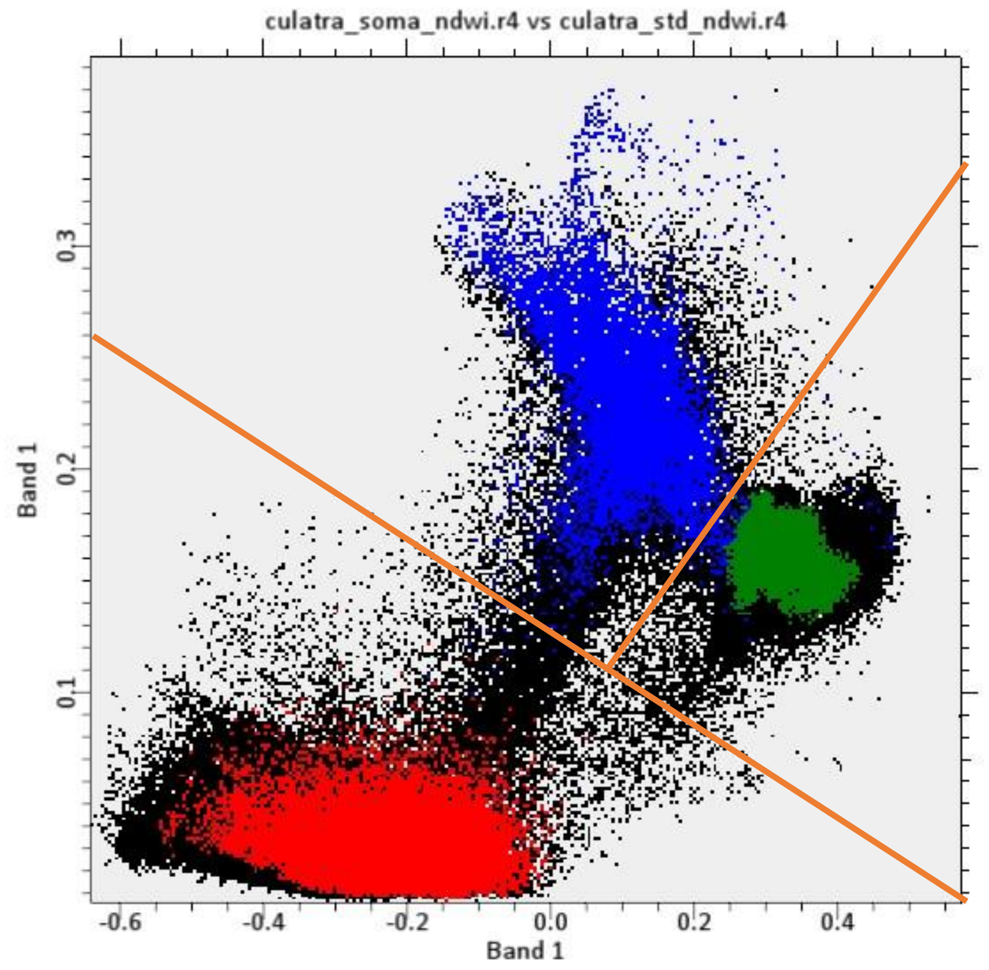
Linear Models for multiclass Classification

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





Blue: water
Red:Land
Green: intertidal



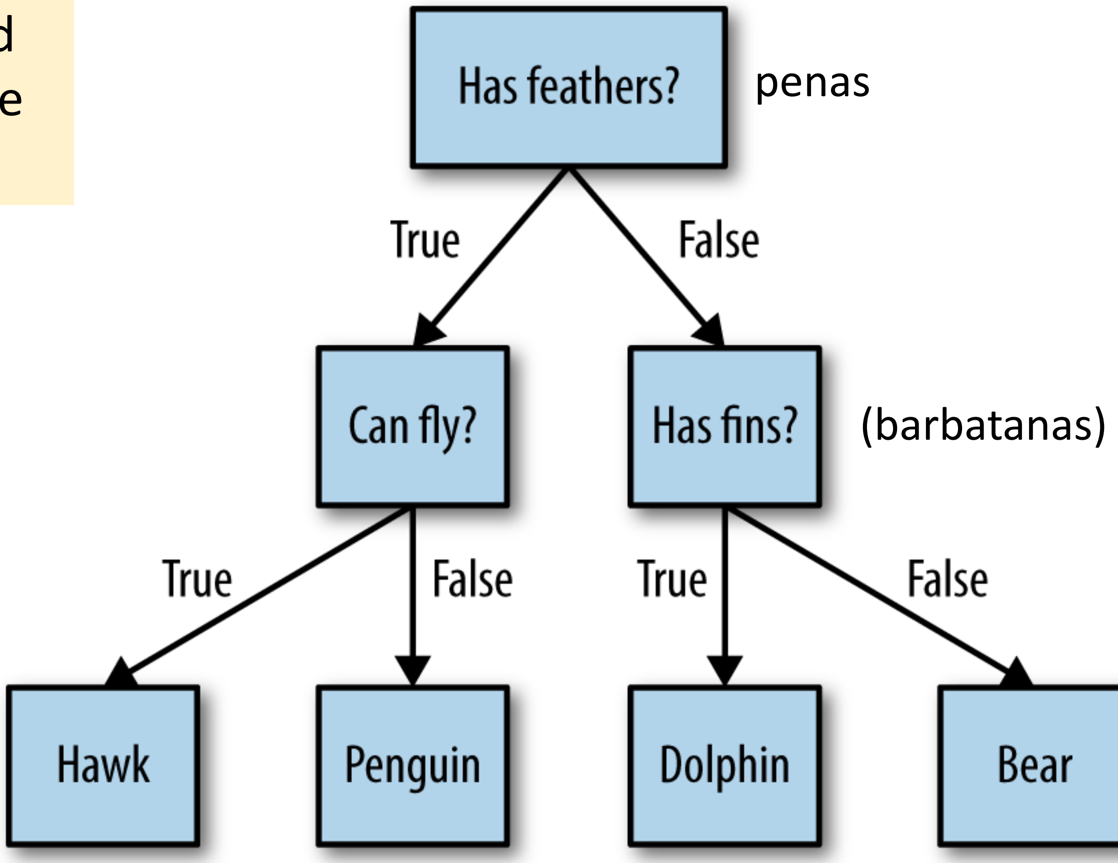
Decision trees

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.



penas

(barbatanas)

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.

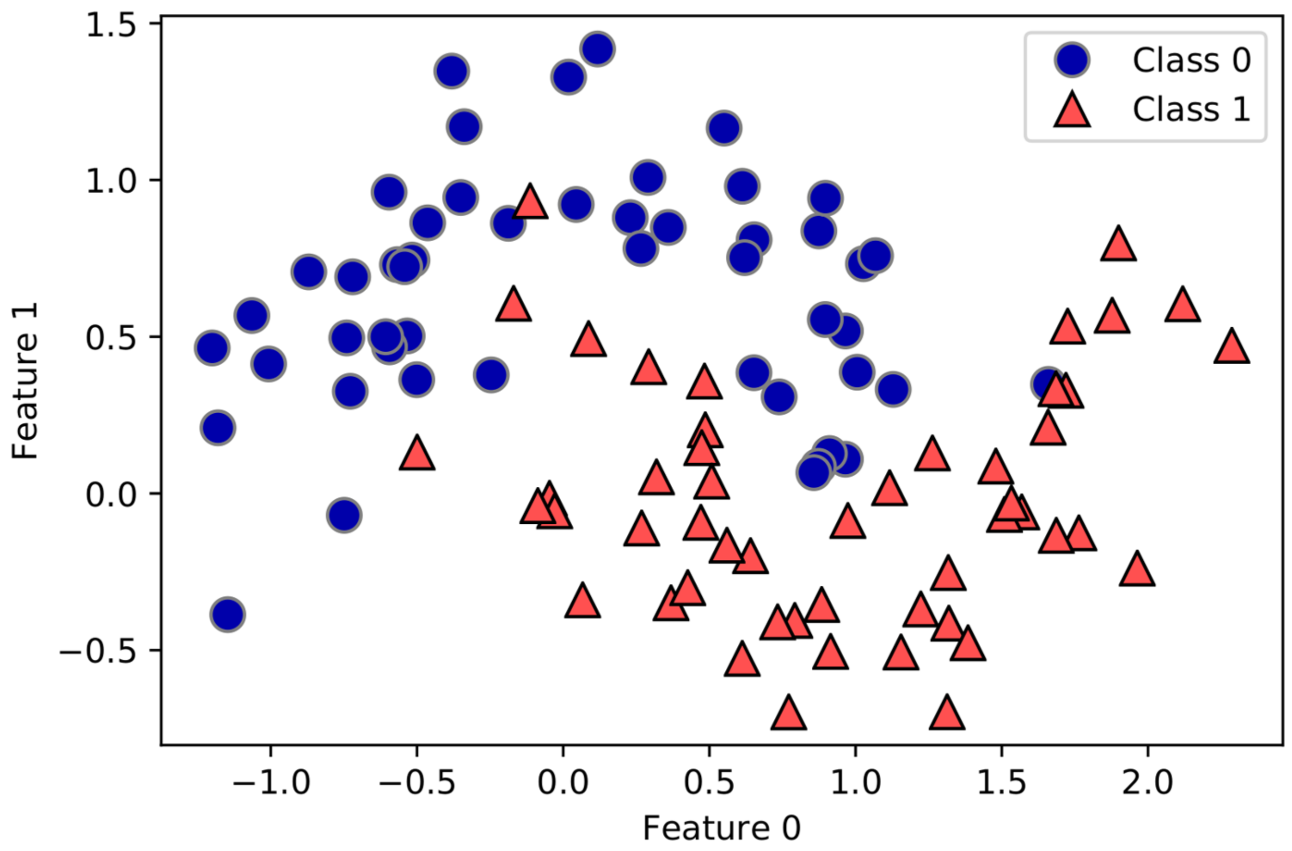
Decision trees

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.

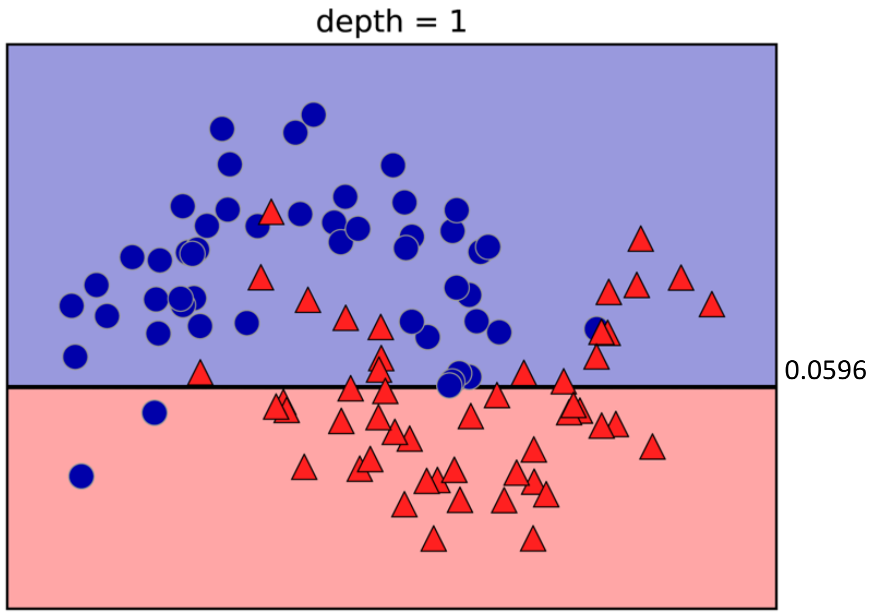
The tests that are used on continuous data are of the form "Is feature *i* larger than value *a*?"

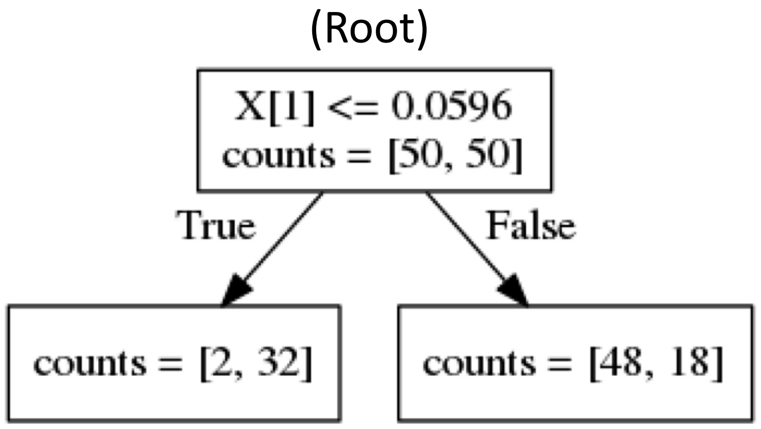


Two-moons dataset on which the decision tree will be built

Decision trees

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

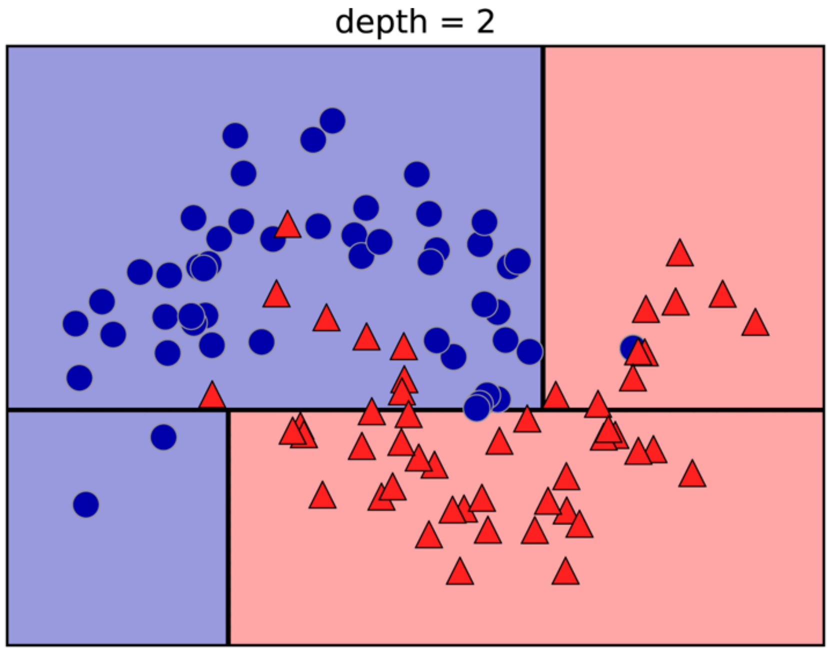


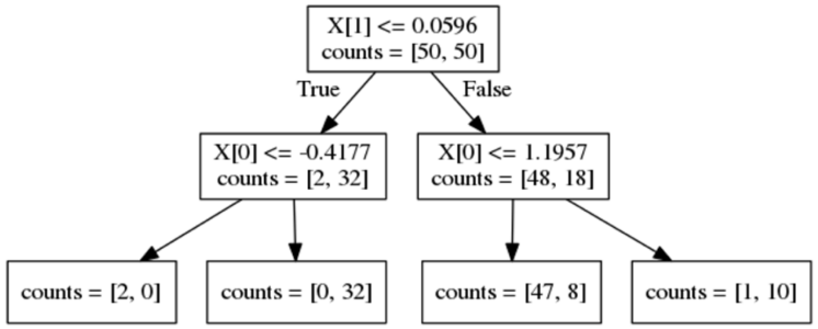


Splitting the dataset horizontally at x[1]=0.0596 yields the most information; it best separates the points in class 0 from the points in class 1. The top node, also called the *root*, represents the whole dataset, consisting of 50 points belonging to class 0 and 50 points belonging to class 1. The split is done by testing whether x[1] <= 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.

Decision trees

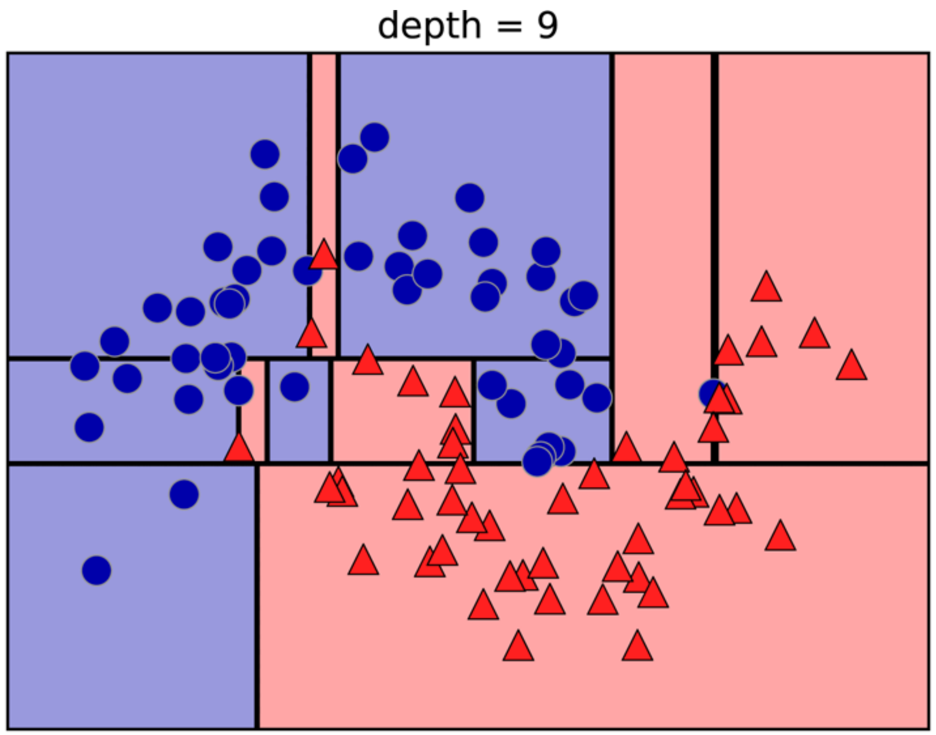
Even though the first split did a good job of separating the two classes, the bottom region still contains points belonging to class 0, and the top region still contains points belonging to class 1. We can build a more accurate model by repeating the process of looking for the best test in both regions.

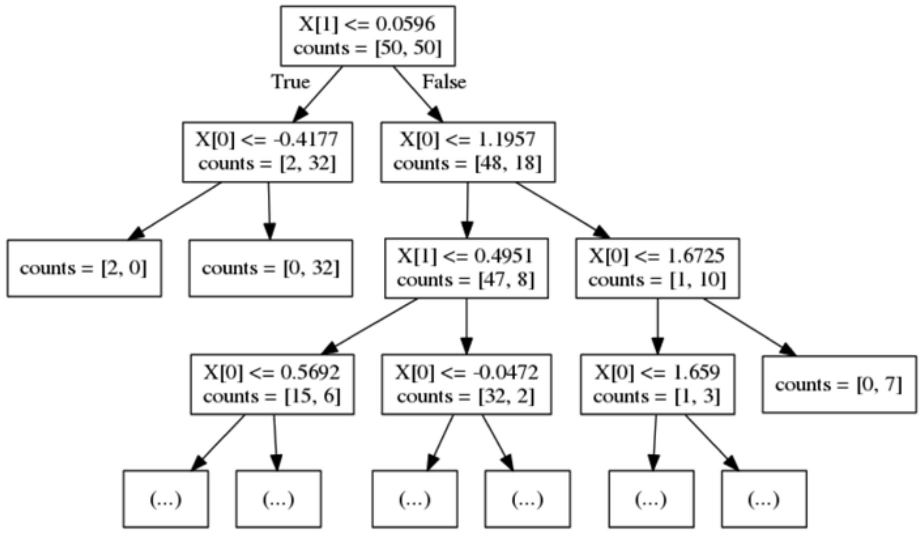




This recursive process yields a binary tree of decisions, with each node containing a test.

Decision trees





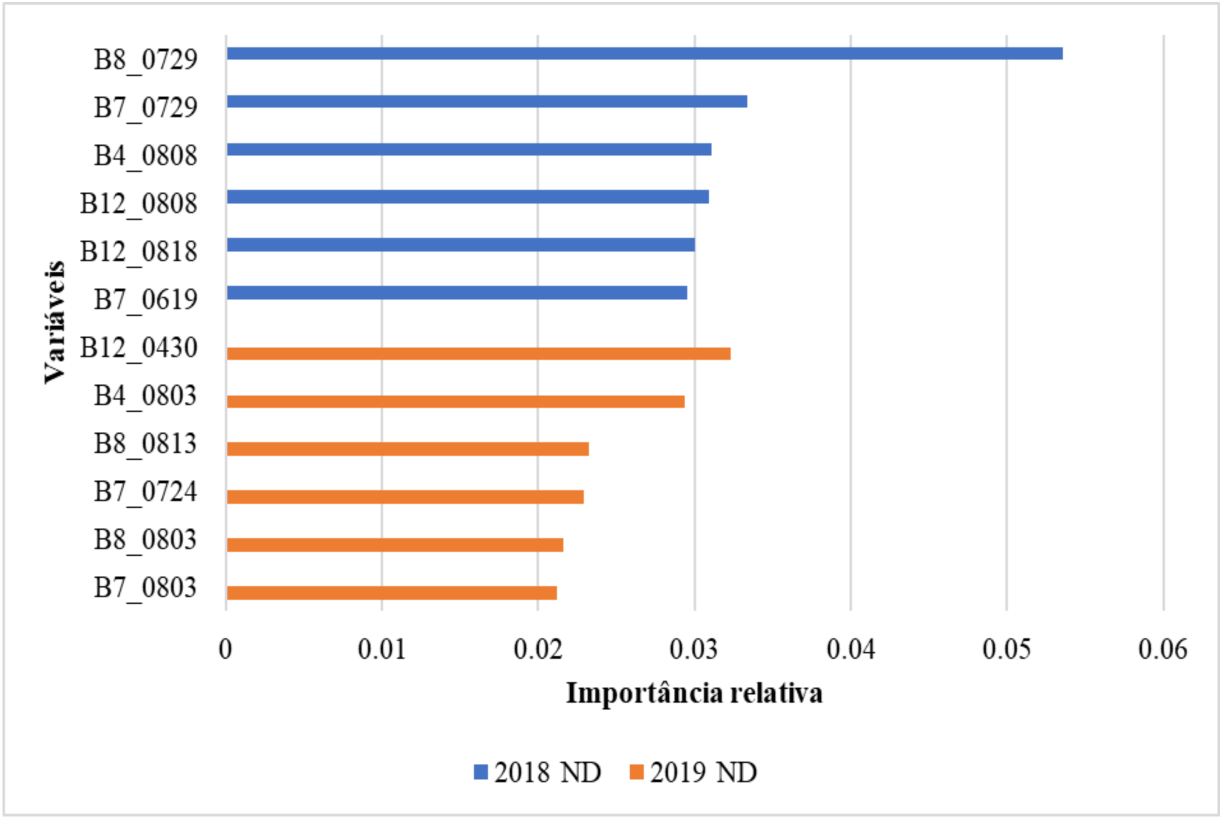
Typically, building a tree as described here and continuing until all leaves are pure leads to models that are very complex and highly overfit to the training data. The presence of pure leaves mean that a tree is 100% accurate on the training set; each data point in the training set is in a leaf that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

Feature Importance

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."

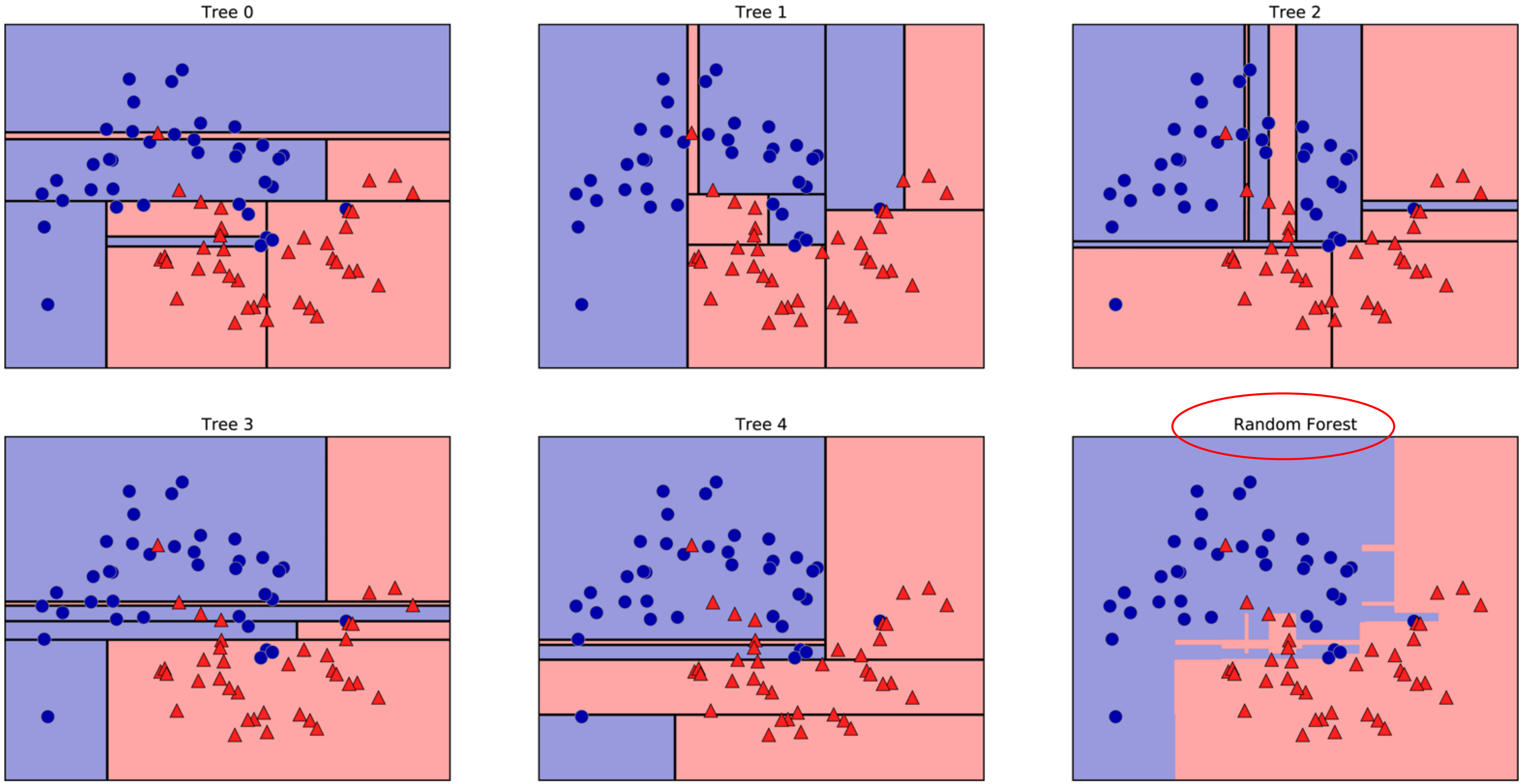


Importância relativa das variáveis na classificação com RF para dados de 2018 (a azul) e de 2019 (a laranja). As denominações das variáveis dizem respeito à banda, mês e dia de aquisição da imagem, respetivamente.

Ensembles of decision trees

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

Neural Network

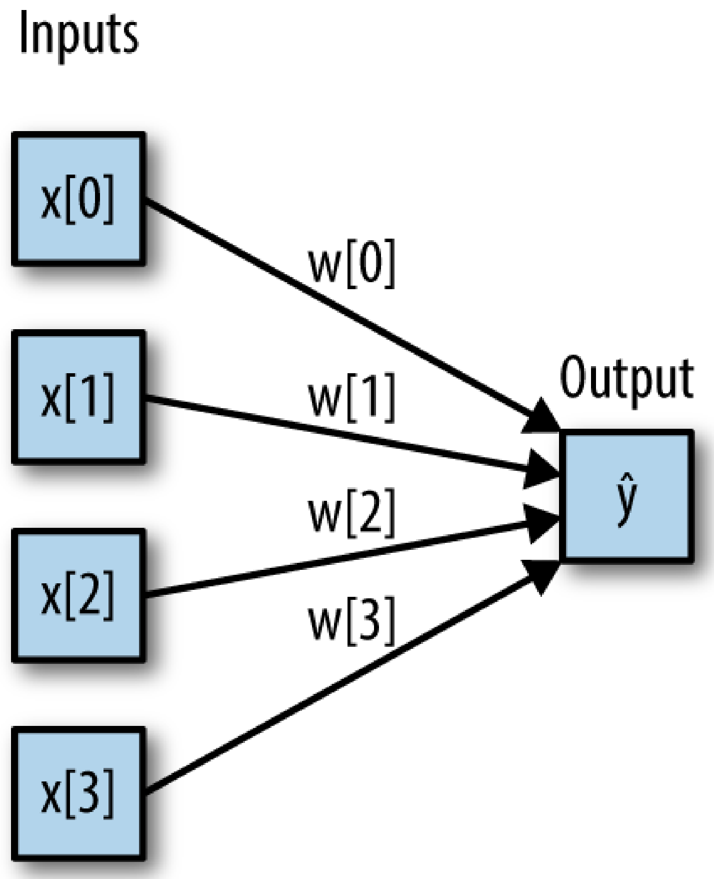
Multilayer perceptrons (MLPs) are also known as feed-forward neural networks, or sometimes just **neural networks**.

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:

ŷ = w[0] * x[0] + w[1] * x[1] + ... + w[p] * x[p] + b

in plain English, ŷ 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].



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

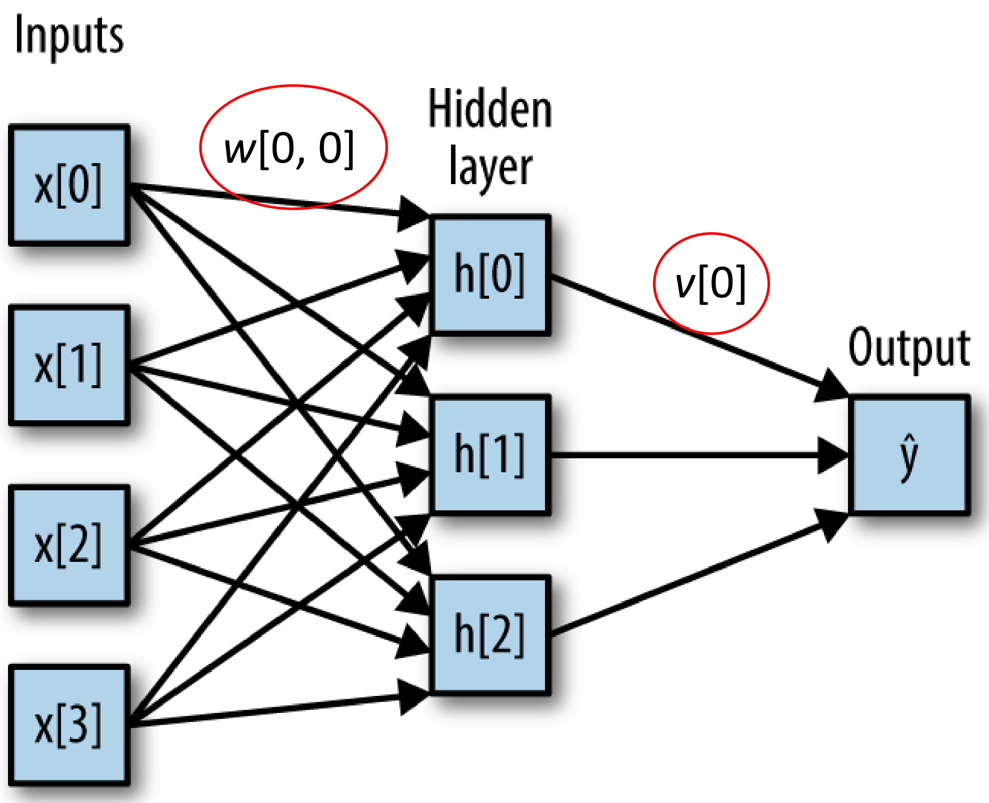
Neural Network

Multilayer perceptron with a single hidden layer

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.



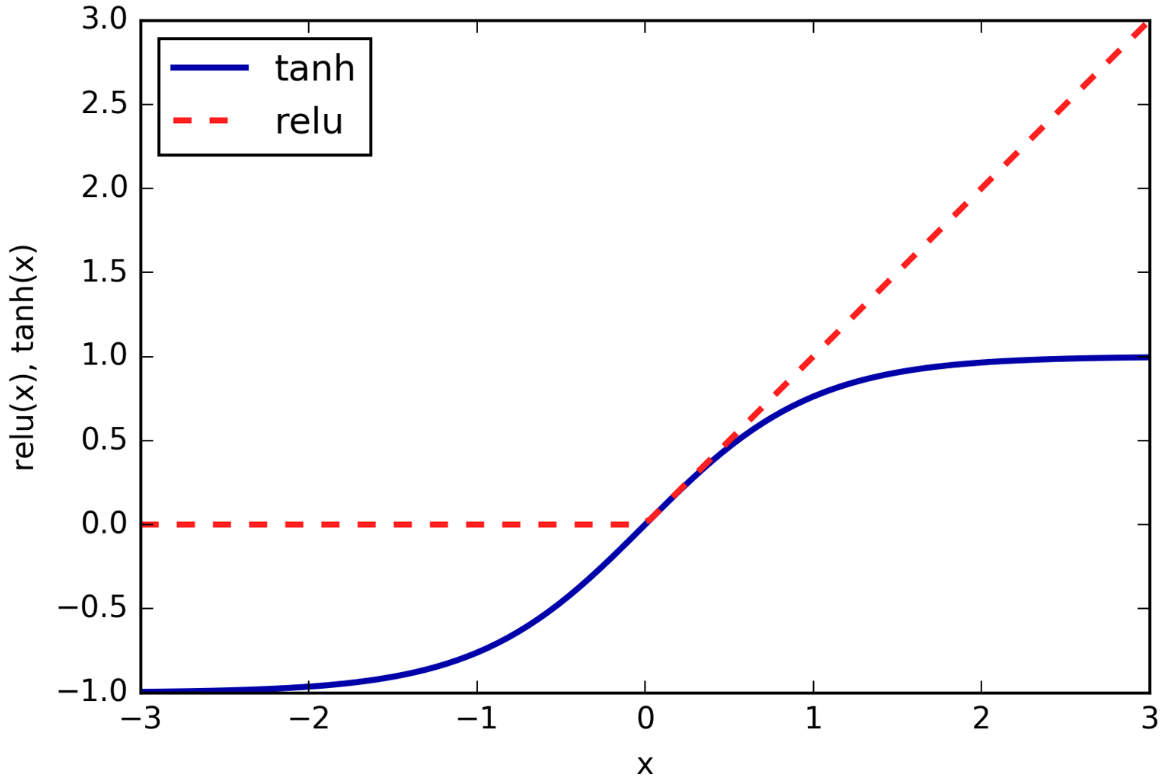
This model has a lot more coefficients (also called weights) to learn: there is one between every input and every hidden unit (which make up the hidden layer), and one between every unit in the hidden layer and the output.

Neural Network

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, $\hat{y}$.



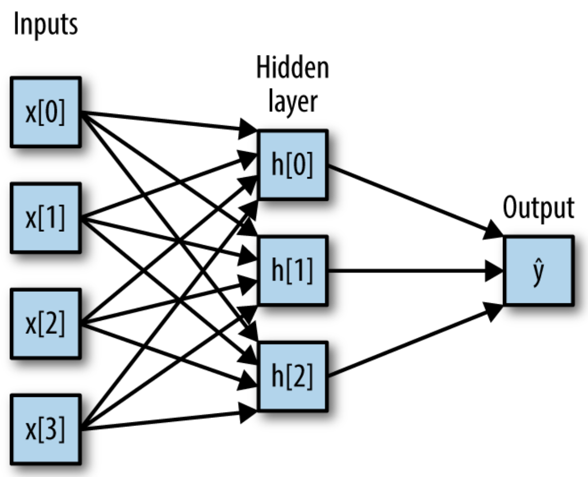
h[0] = **tanh**(*w*[0, 0] * *x*[0] + *w*[1, 0] * *x*[1] + *w*[2, 0] * *x*[2] + *w*[3, 0] * *x*[3] + *b*[0])

Neural Network

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 *ŷ*. The weights *v* and *w* are learned from data, *x* are the input features, *ŷ* is the computed output, and *h* are intermediate computations.

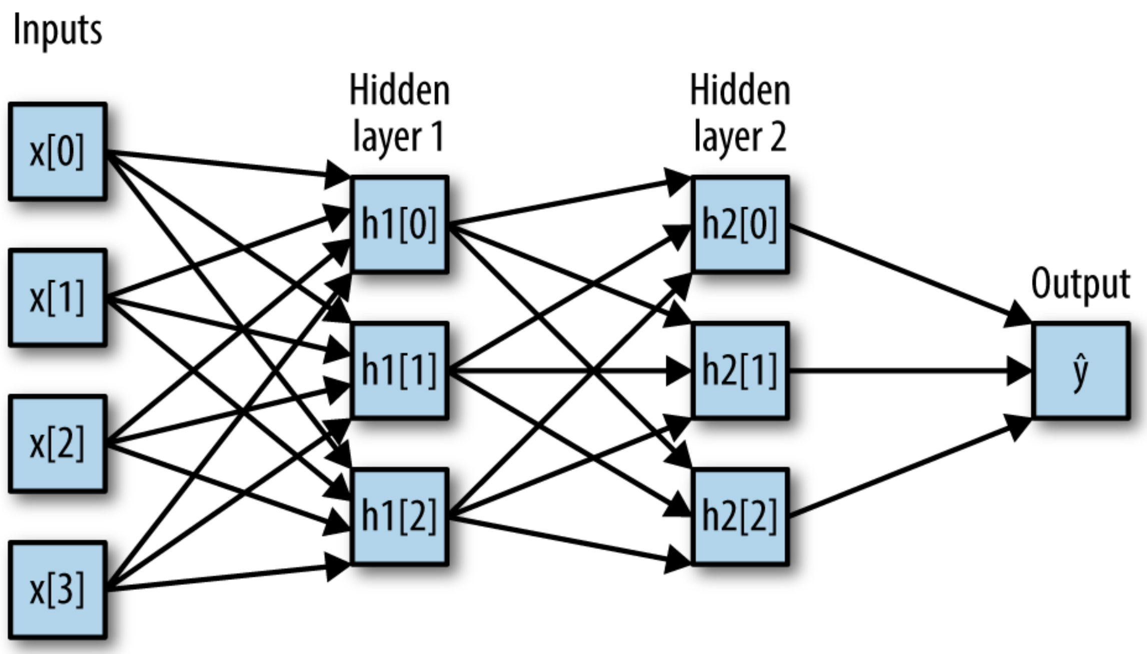
1 pixel = 19 weights
we know (x,y) for a sample of pixels

Neural Network

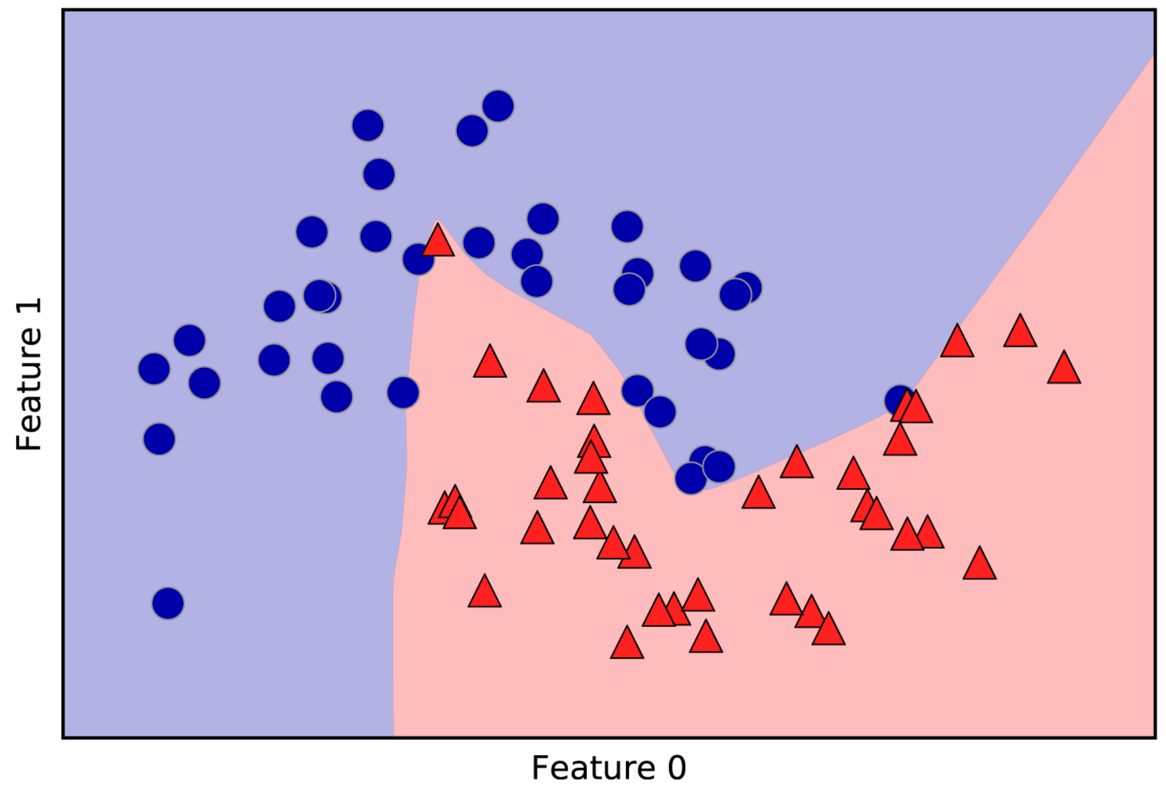
An important parameter that needs to be set by the user is the number of nodes in the hidden layer.

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

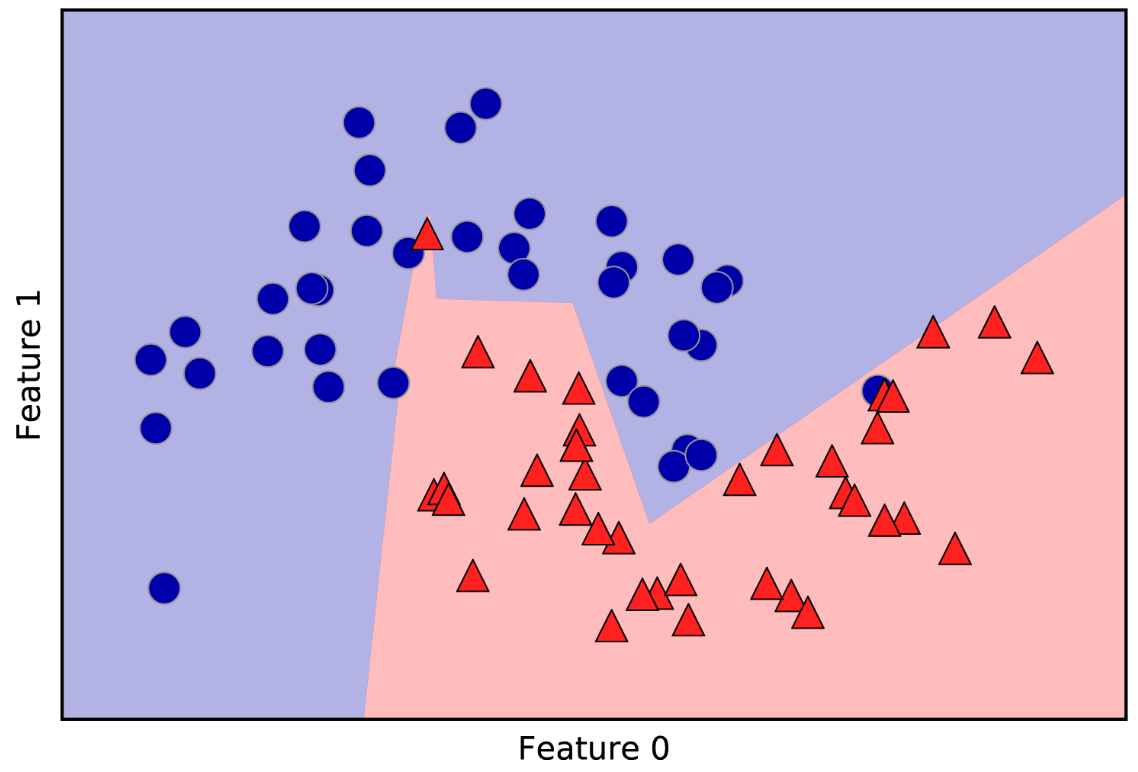
Having large neural networks made up of many of these layers of computation is what inspired the term "**deep learning.**"



Neural Network

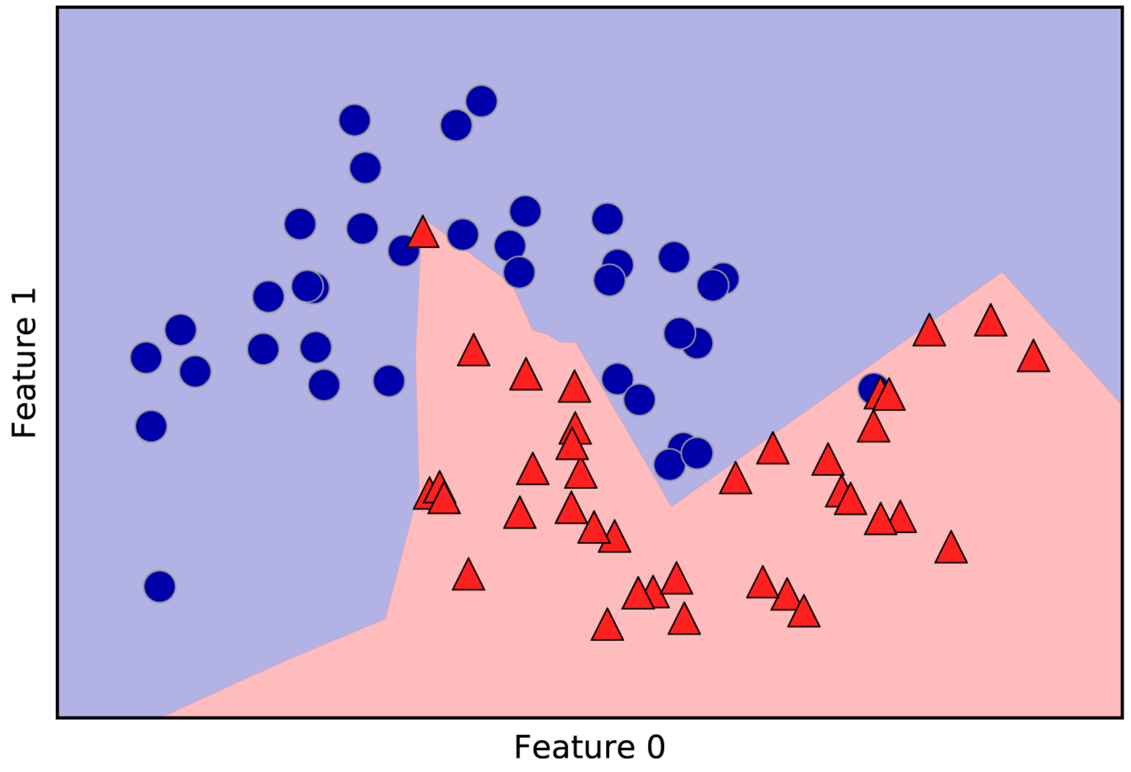


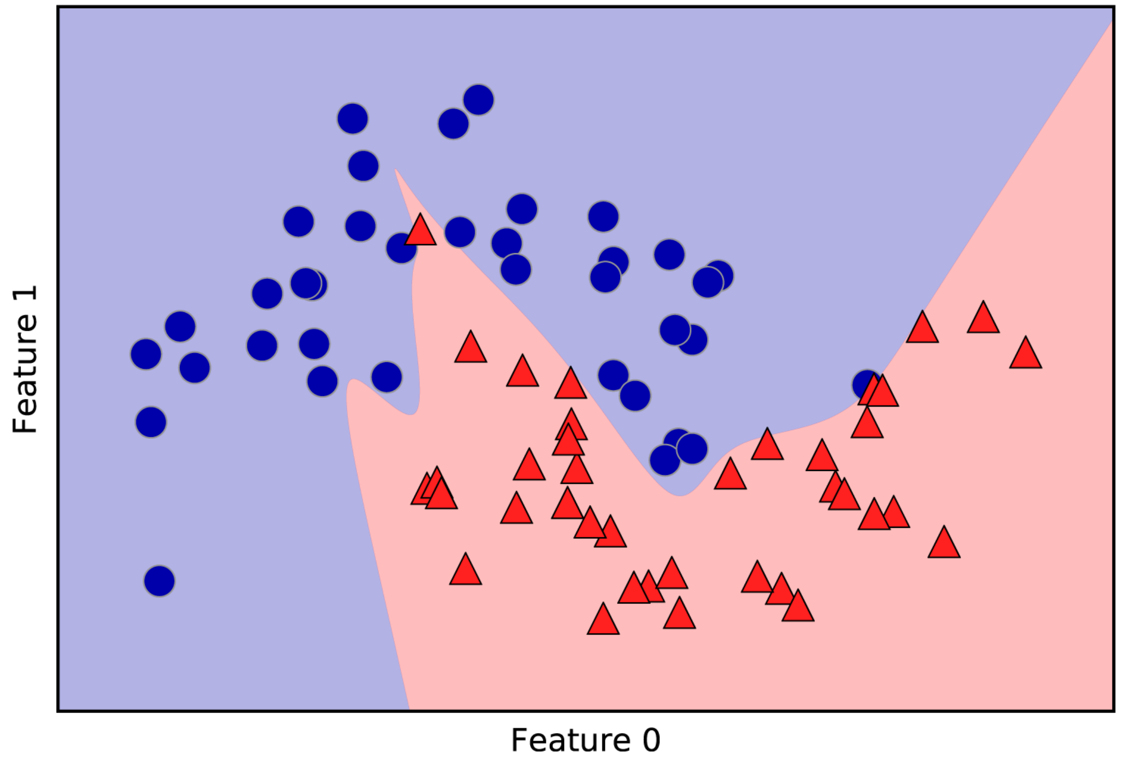
Decision boundary learned by a neural network with <u>100 hidden</u> units on the two_moons dataset



Decision boundary learned by a neural network with <u>10 hidden</u> units on the two_moons dataset

Neural Network





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

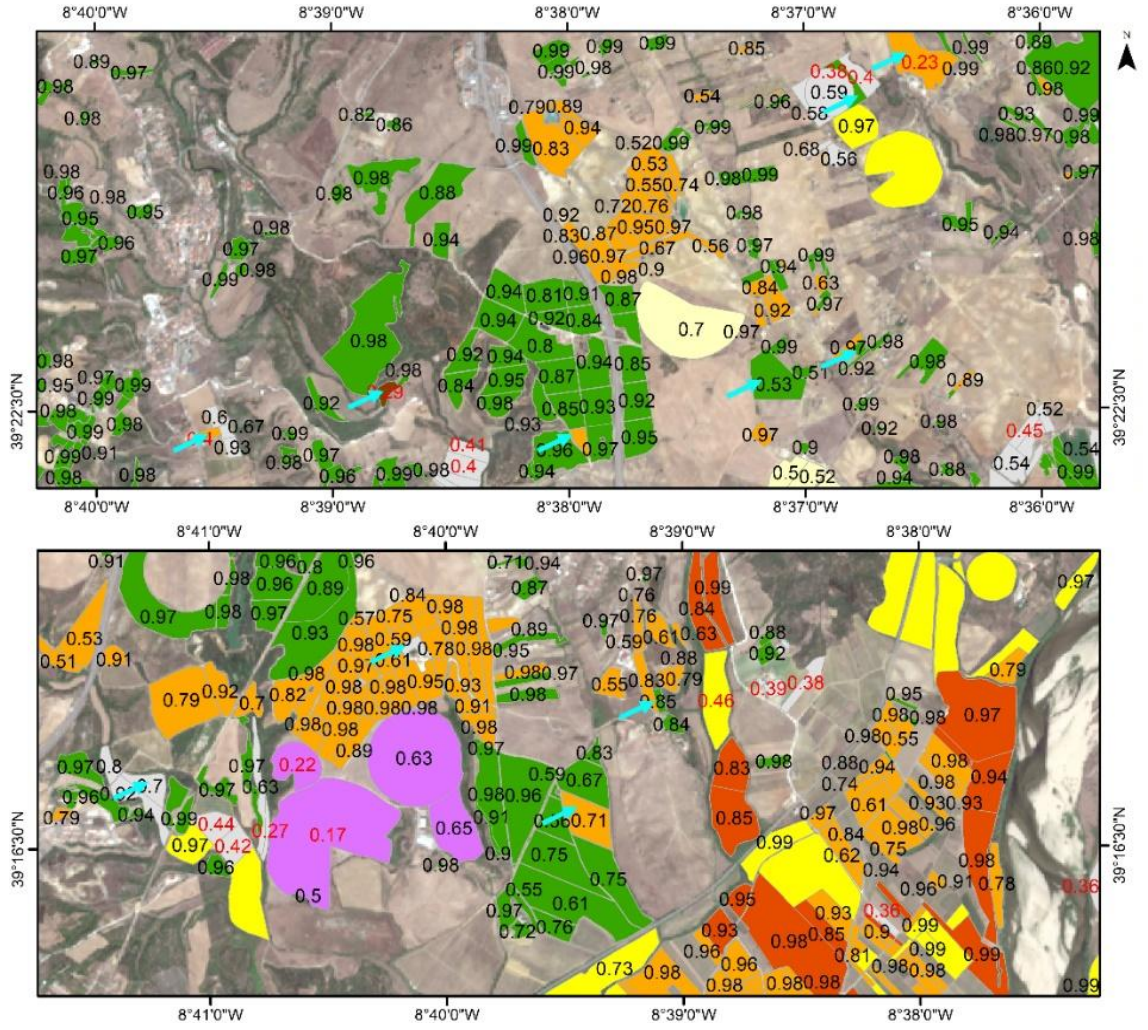
Decision boundary learned using 2 hidden layers with 10 hidden units each, with **tanh** activation function

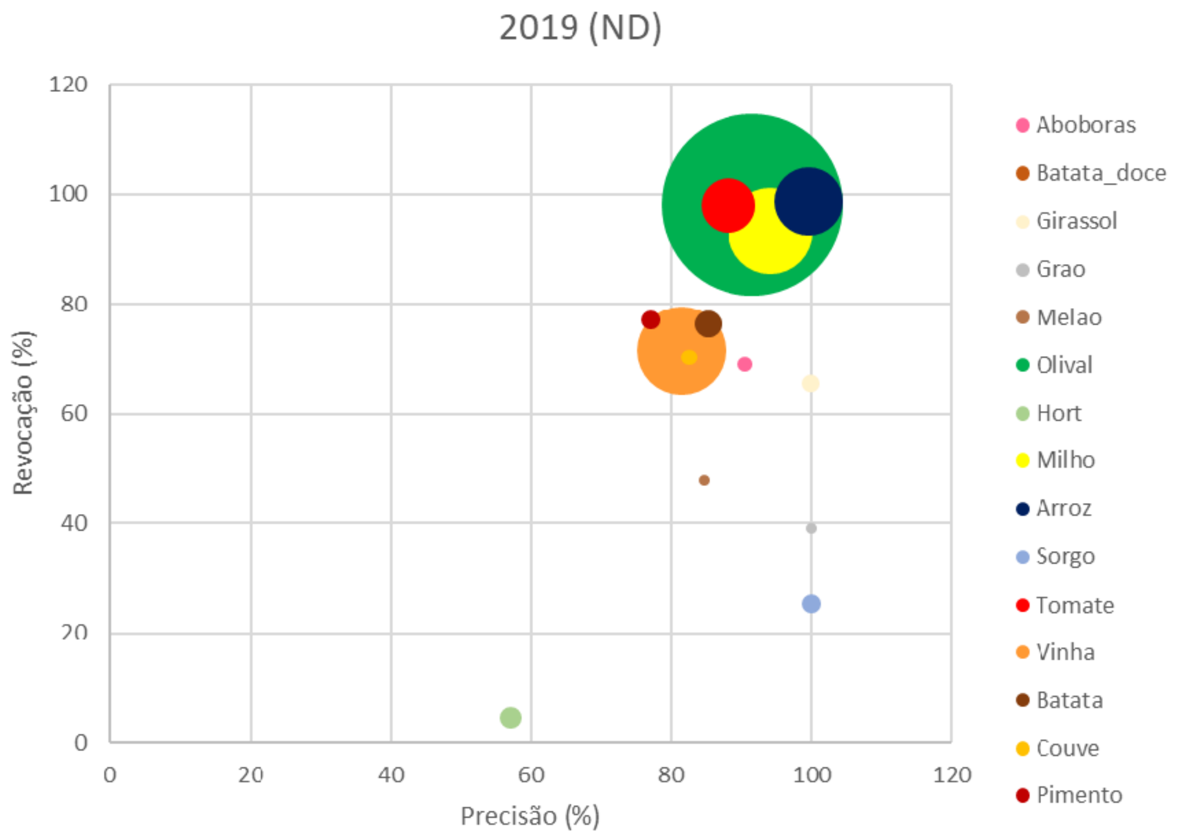
Summary

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

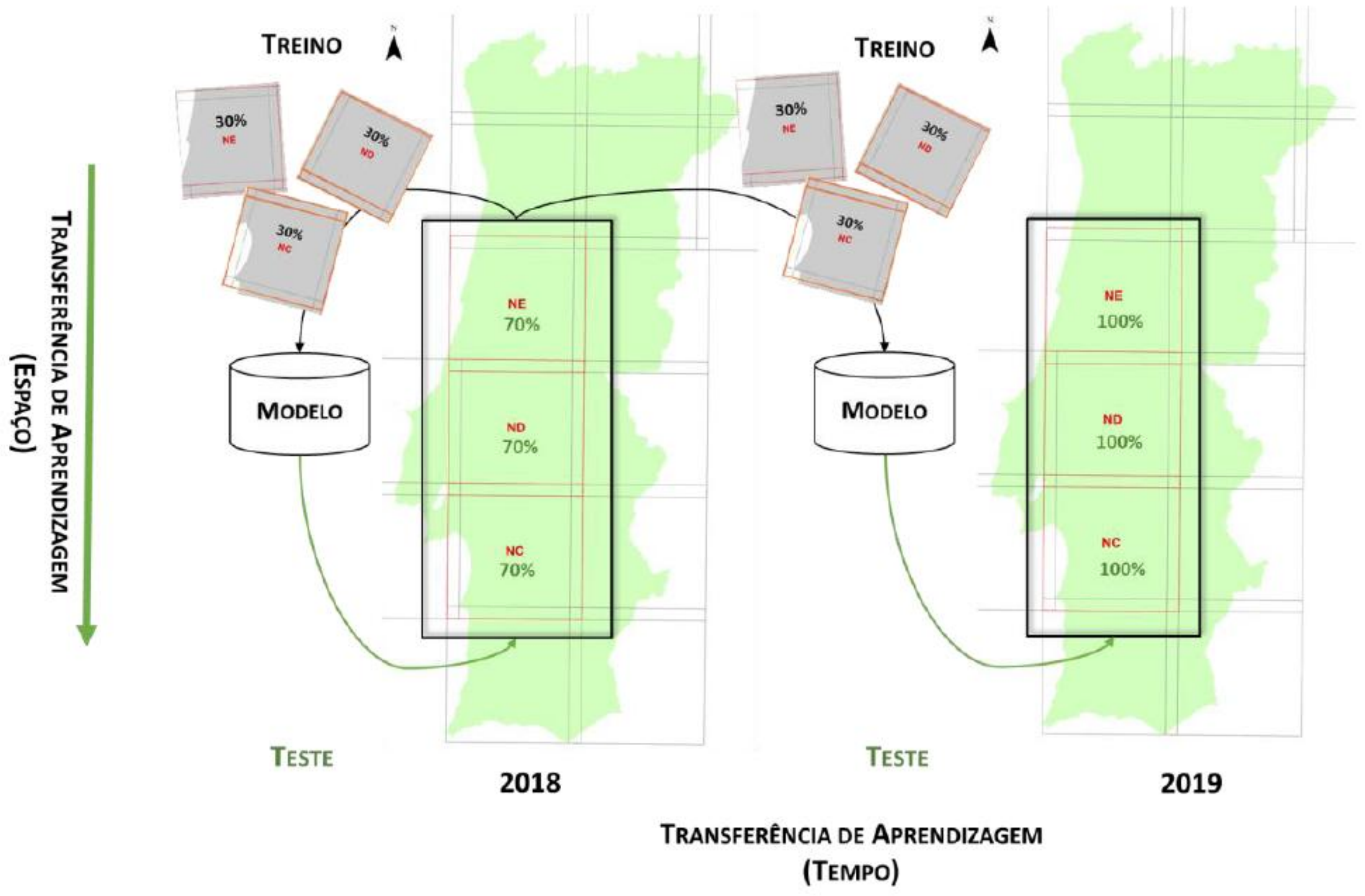




A dimensão dos círculos ilustra a frequência de cada classe nos dados.

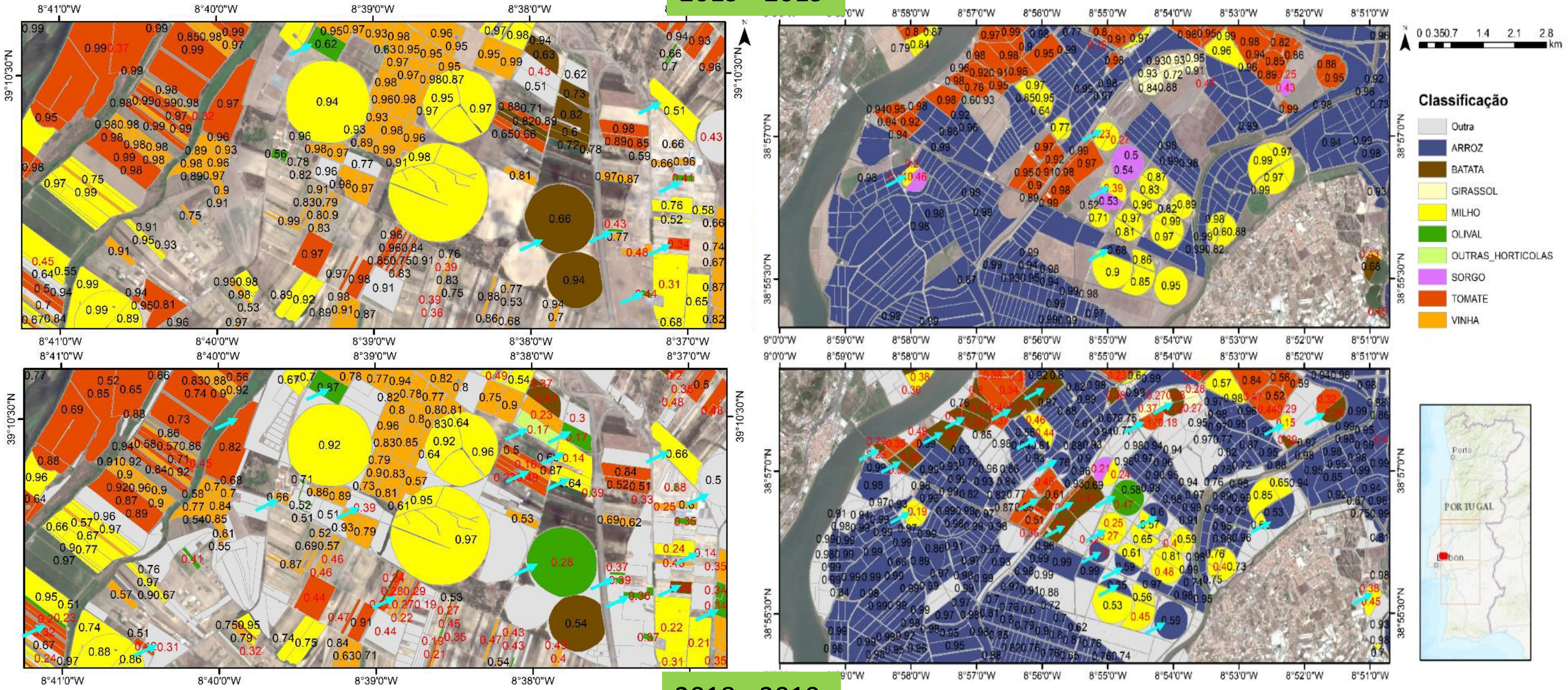
Inês Silva (2020)

Transfer Learning



Transfer Learning in time

2019 - 2019



2018 - 2019

Convolutional Neural Network

David H. Hubel and Torsten Wiesel performed a series of experiments on cats in 1958 and 1959 (and a few years later on monkeys), giving crucial insights on the structure of the visual cortex.

The authors showed that some neurons react only to images of horizontal lines, while others react only to lines with different orientations.

These observations led to the idea that the higher-level neurons are based on the outputs of neighbouring lower-level neurons.

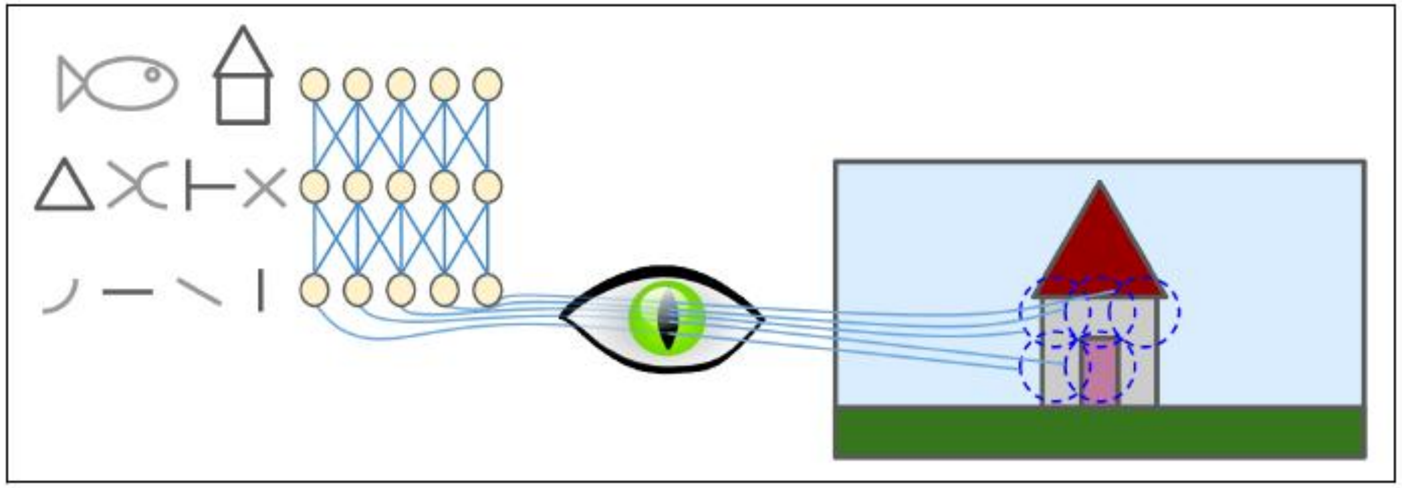


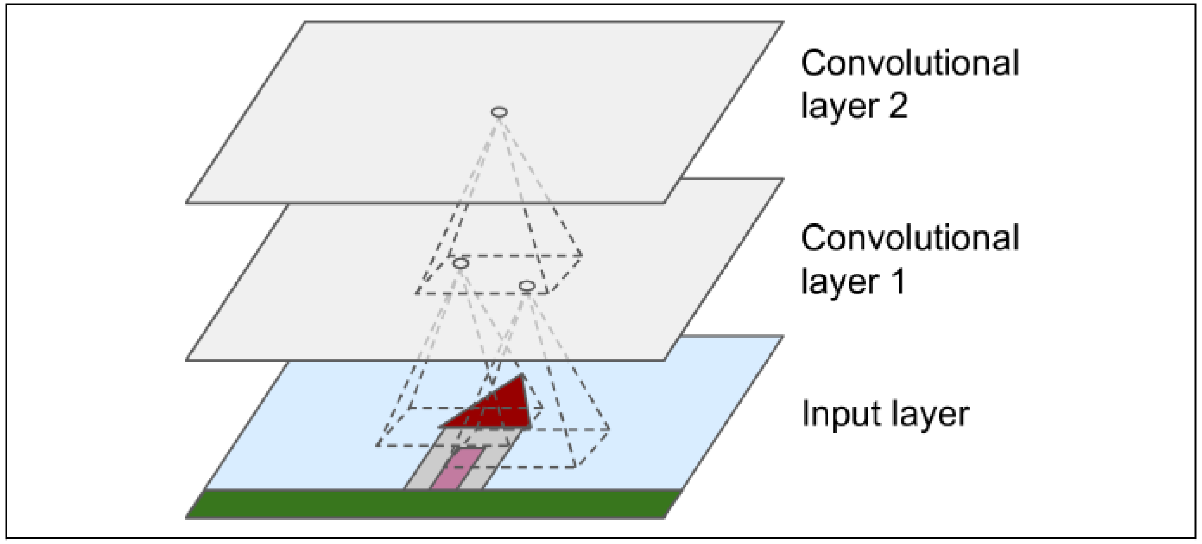
Figure 14-1. Local receptive fields in the visual cortex

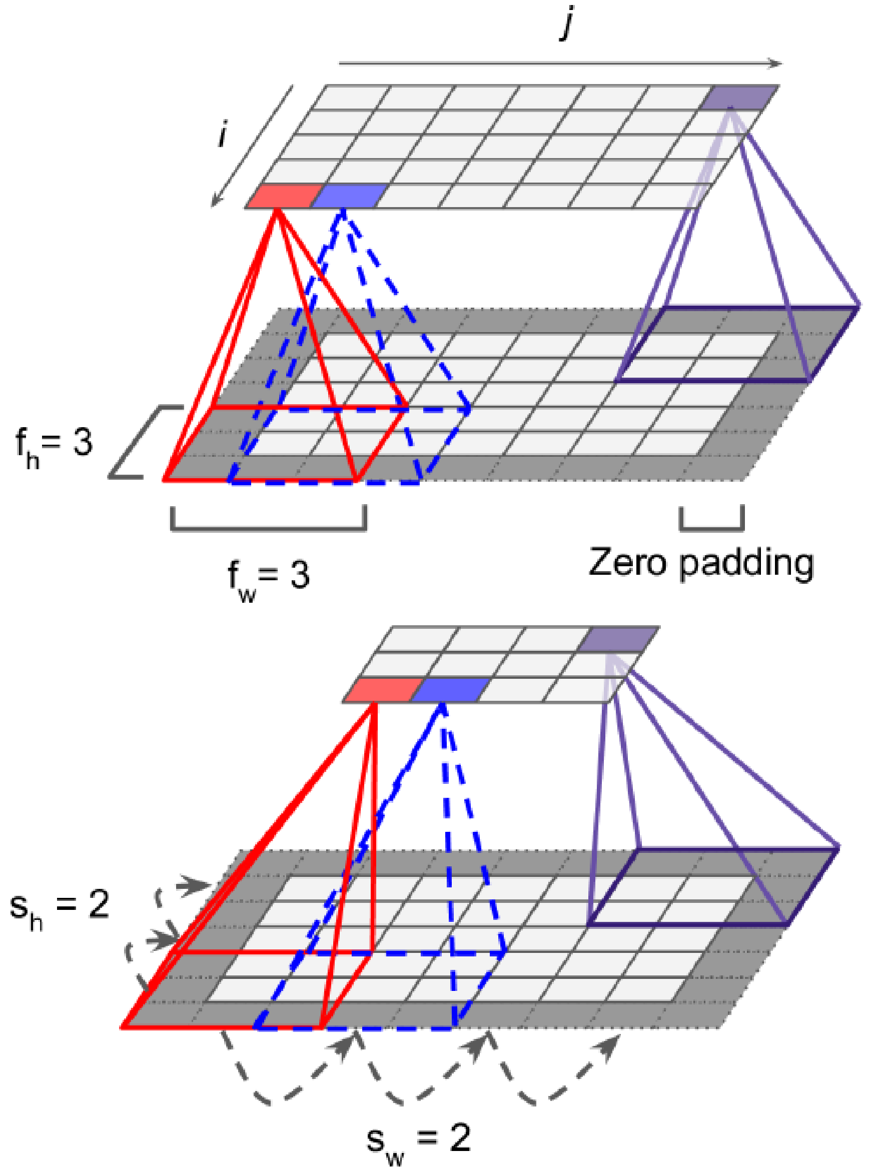
These studies of the visual cortex inspired the neocognitron, introduced in 1980, which gradually evolved into what we now call ***convolutional neural networks***.

An important milestone was a 1998 paper by Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner, which introduced the famous *LeNet-5* architecture, widely used to recognize handwritten check numbers.

Convolutional Neural Network

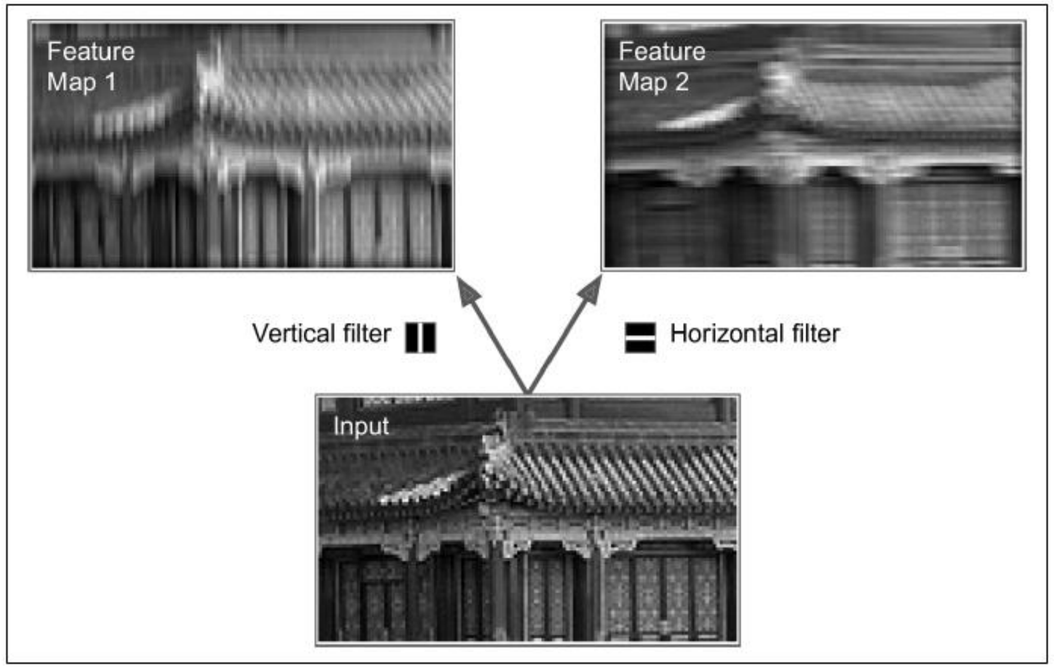
CNN layers with rectangular local receptive fields



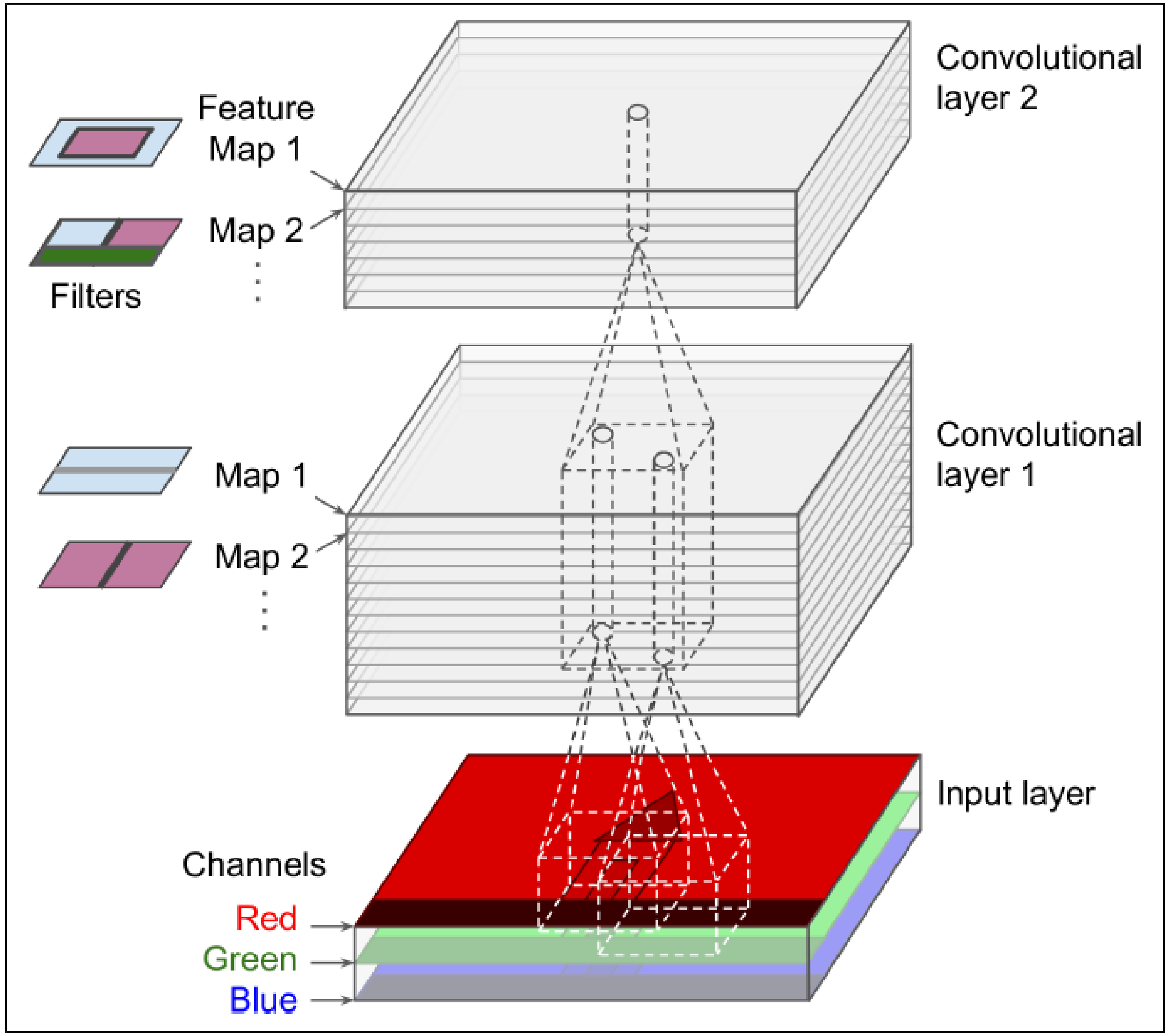


A neuron located in position *(i,j)* in the upper layer is connected to the outputs of the neurons in the previous layer located in
Rows: [*i* × *sh* to *i* × *sh* + *fh* – 1]
Column: [*j* × *sw* to *j* × *sw* + *fw* –1],

where *sh* and *sw* are the **vertical and horizontal strides**.

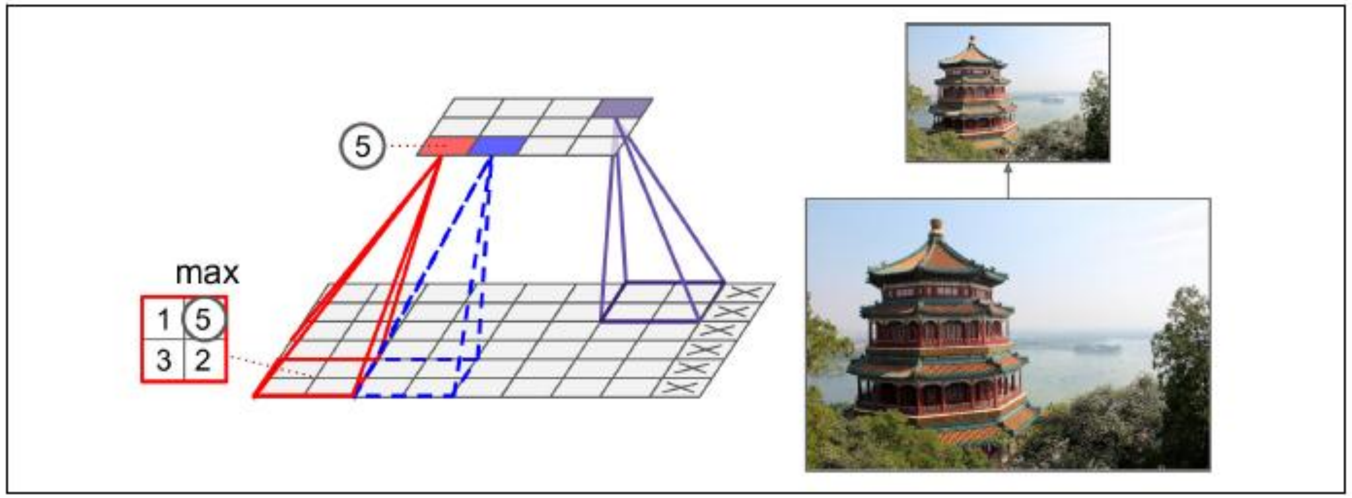


Aplicação de um filtro vertical e um filtro horizontal

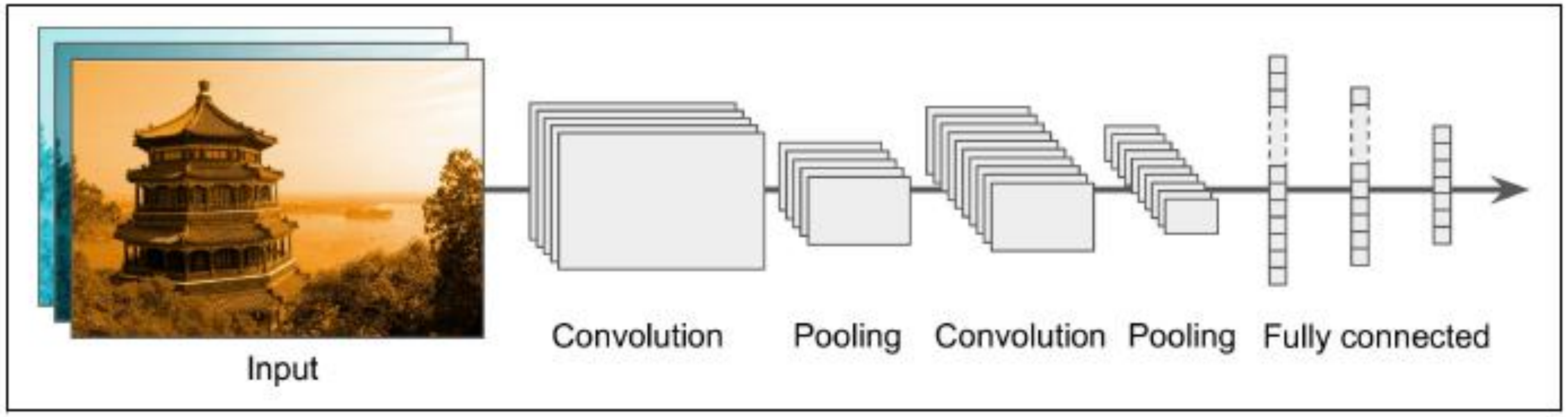


Aplicação de multiplos filtros em cada layer

Max pooling layer (2 × 2 pooling kernel, stride 2, no padding)



Typical CNN architecture



Convolutions

Imagem 4x4 e filtro 3x3

0	50	0	29
0	80	31	2
33	90	0	75
0	9	0	95

-1	0	1
-2	0	2
-1	0	1

Sobreposição do filtro na imagem

0	50	0	29
0	80	31	2
33	90	0	75
0	9	0	95

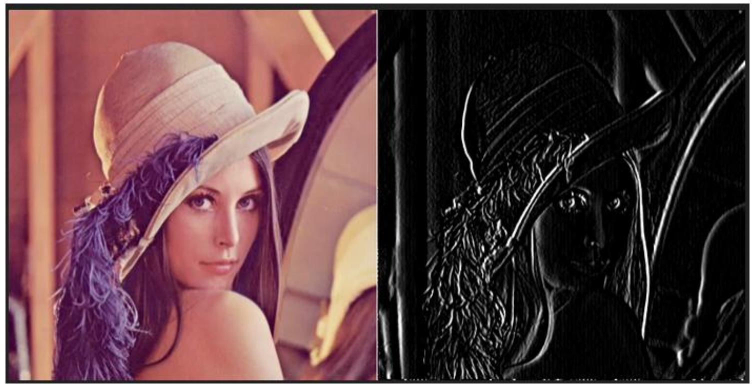
-1	0	1
-2	0	2
-1	0	1

Cálculo do filtro

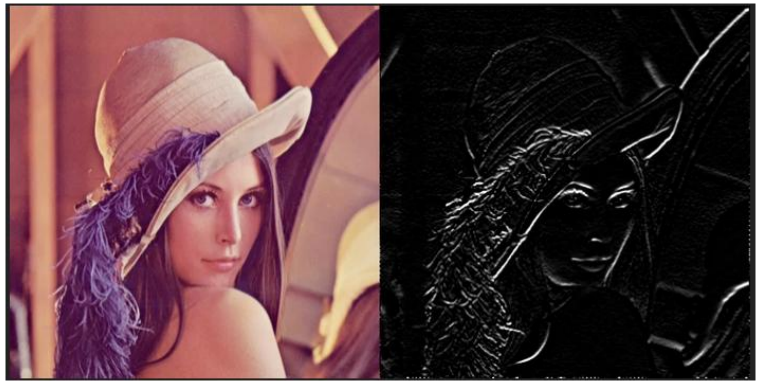
Image Value	Filter Value	Result
0	-1	0
50	0	0
0	1	0
0	-2	0
80	0	0
31	2	62
33	-1	-33
90	0	0
0	1	0

0	50	0	29
0	80	31	2
33	90	0	75
0	9	0	95

29	?
?	?



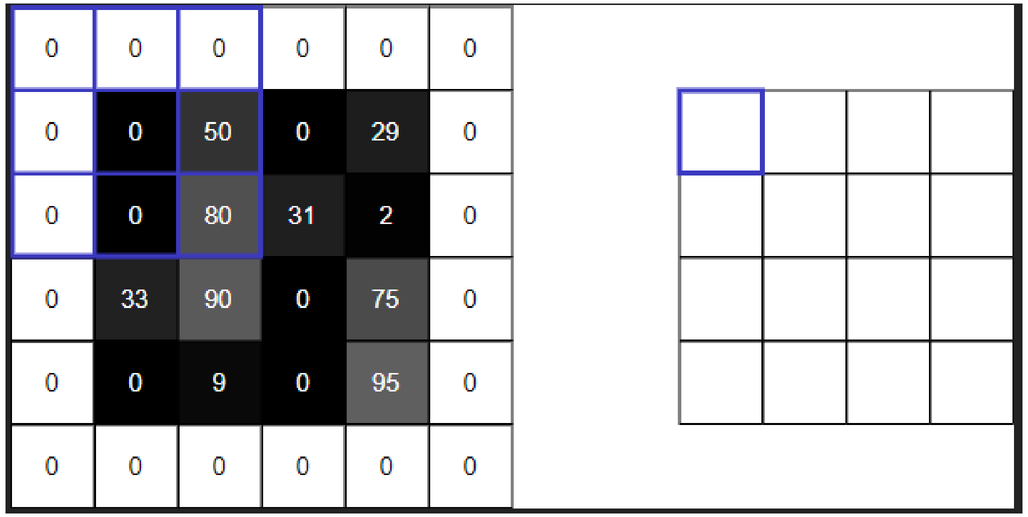
Sobel vertical filter



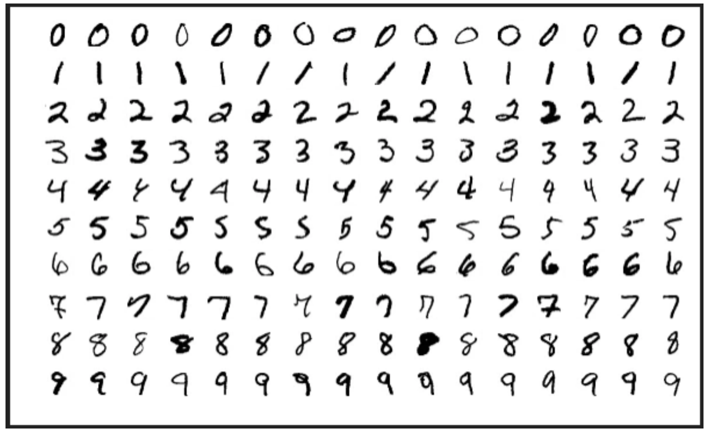
Sobel horizontal filter

Padding (preencher)

Using padding to keep the output image with the same size as the input image. To do this, we add zeros around the image so we can overlay the filter in more places. A 3x3 filter requires 1 pixel of padding. This is called **“same” padding.**

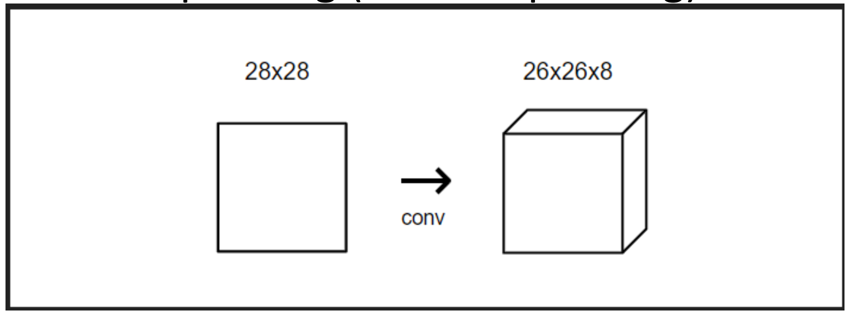


0	0	0	0	0	0
0	0	50	0	29	0
0	0	80	31	2	0
0	33	90	0	75	0
0	0	9	0	95	0
0	0	0	0	0	0



MNIST dataset

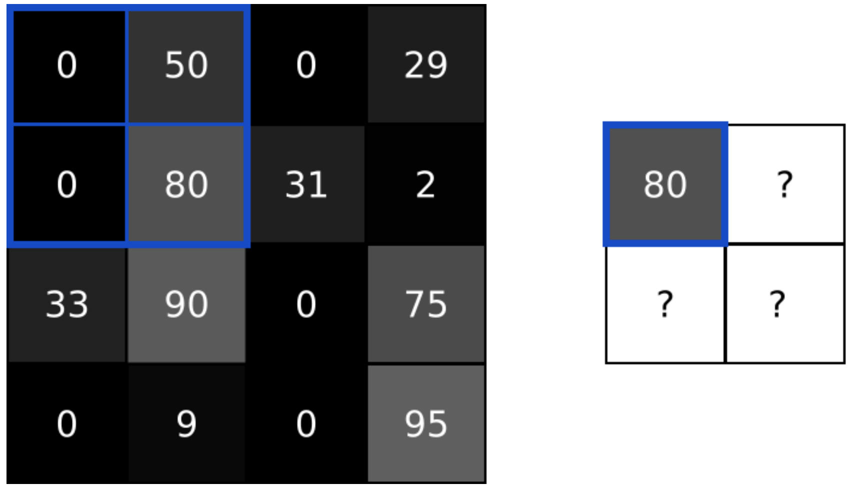
With valid padding (without padding) and 8 filters

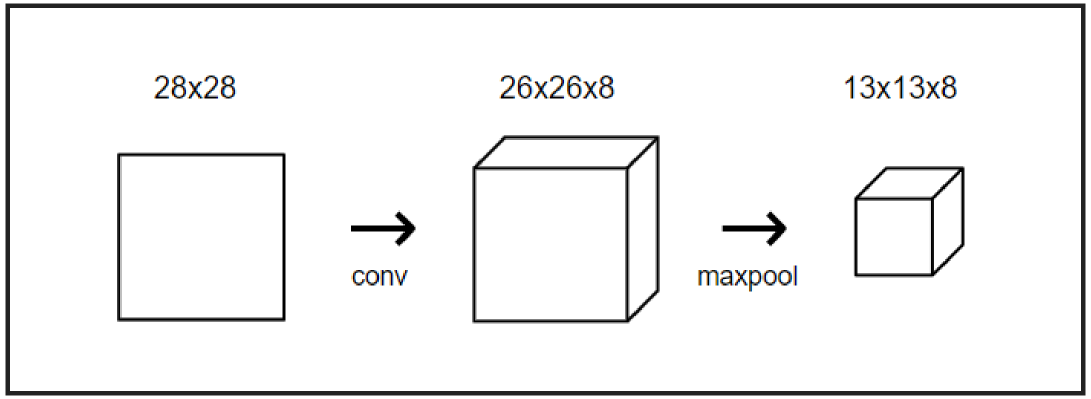


Each of the 8 filters in the conv layer produces a 26x26 output, so stacked together they make up a 26x26x8 volume. All of this happens because of 3 ×× 3 (filter size) ×× 8 (number of filters) = **only 72 weights**!

Pooling (agrupar)

Neighboring pixels in images tend to have similar values, so conv layers will typically also produce similar values for neighboring pixels in outputs. As a result, **much of the information contained in a conv layer's output is redundant**.

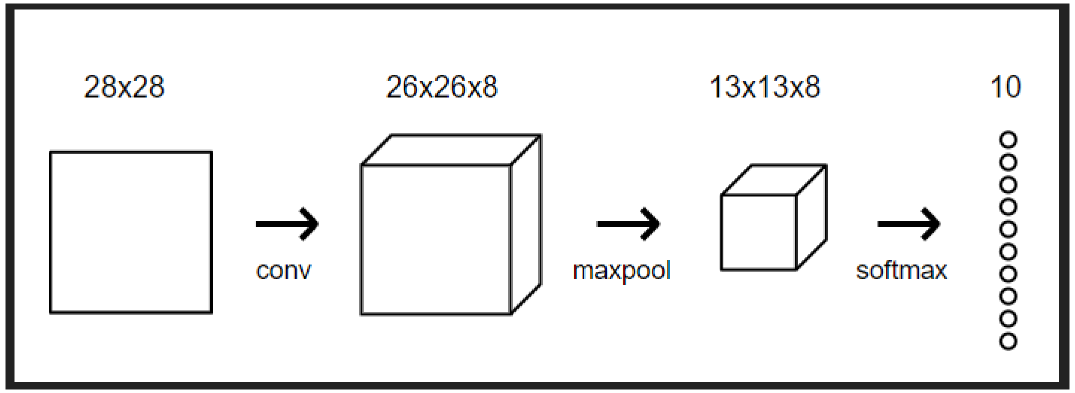




Pooling layers solve this problem. All they do is reduce the size of the input it's given by (you guessed it) *pooling* values together in the input. The pooling is usually done by a simple operation like MAX, MIN, AVERAGE

Softmax

To complete our CNN, we need to give it the ability to actually make predictions. We'll do that by using the standard final layer for a multiclass classification problem: the **Softmax** layer, a fully-connected (dense) layer that uses the Softmax function as its activation.



We'll use a softmax layer with 10 nodes, one representing each digit (0 a 9), as the final layer in our CNN.
The digit represented by the node with the highest probability will be the output of the CNN!

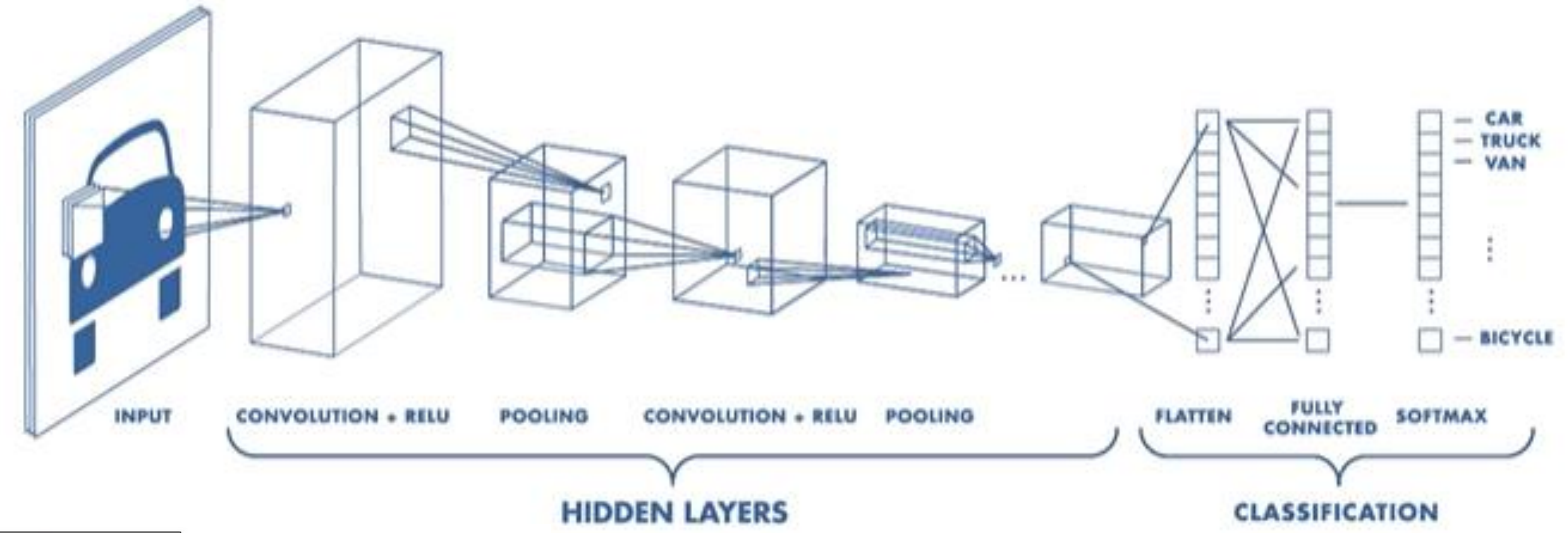
Training a neural network typically consists of two phases:

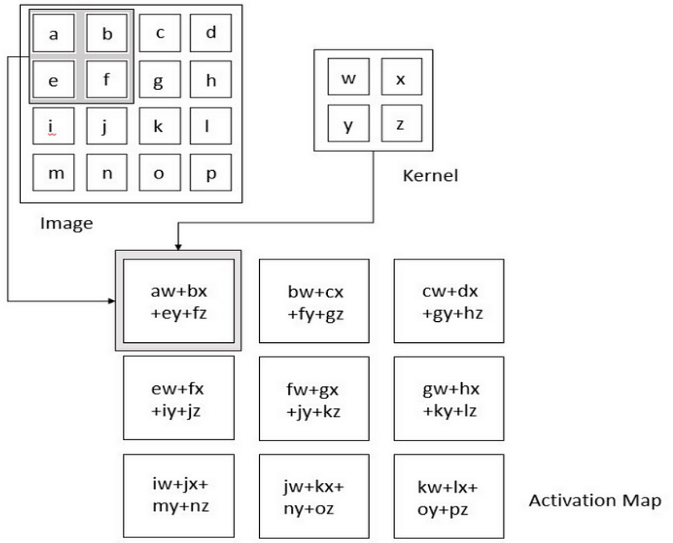
1. A **forward** phase, where the input is passed completely through the network.

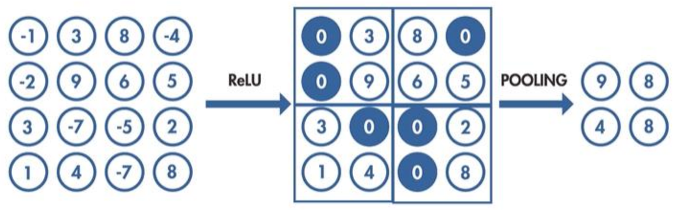
2. A **backward** phase, where gradients are backpropagated (backprop) and weights are updated.

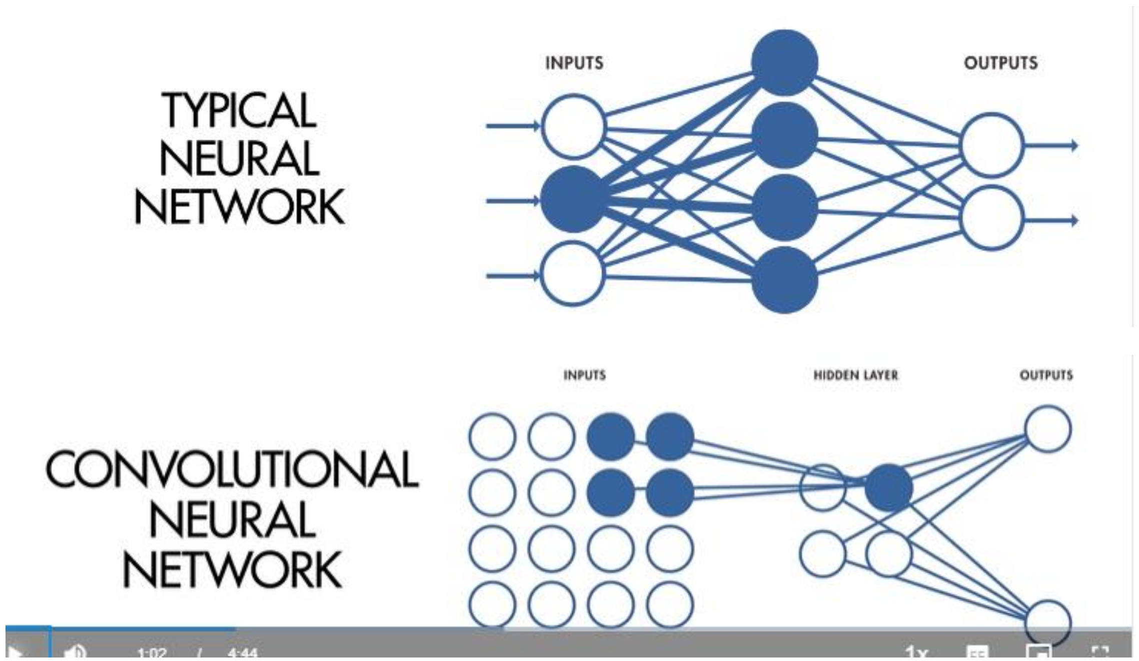
•During the forward phase, each layer will **cache** any data (like inputs, intermediate values, etc) it'll need for the backward phase. This means that any backward phase must be preceded by a corresponding forward phase.

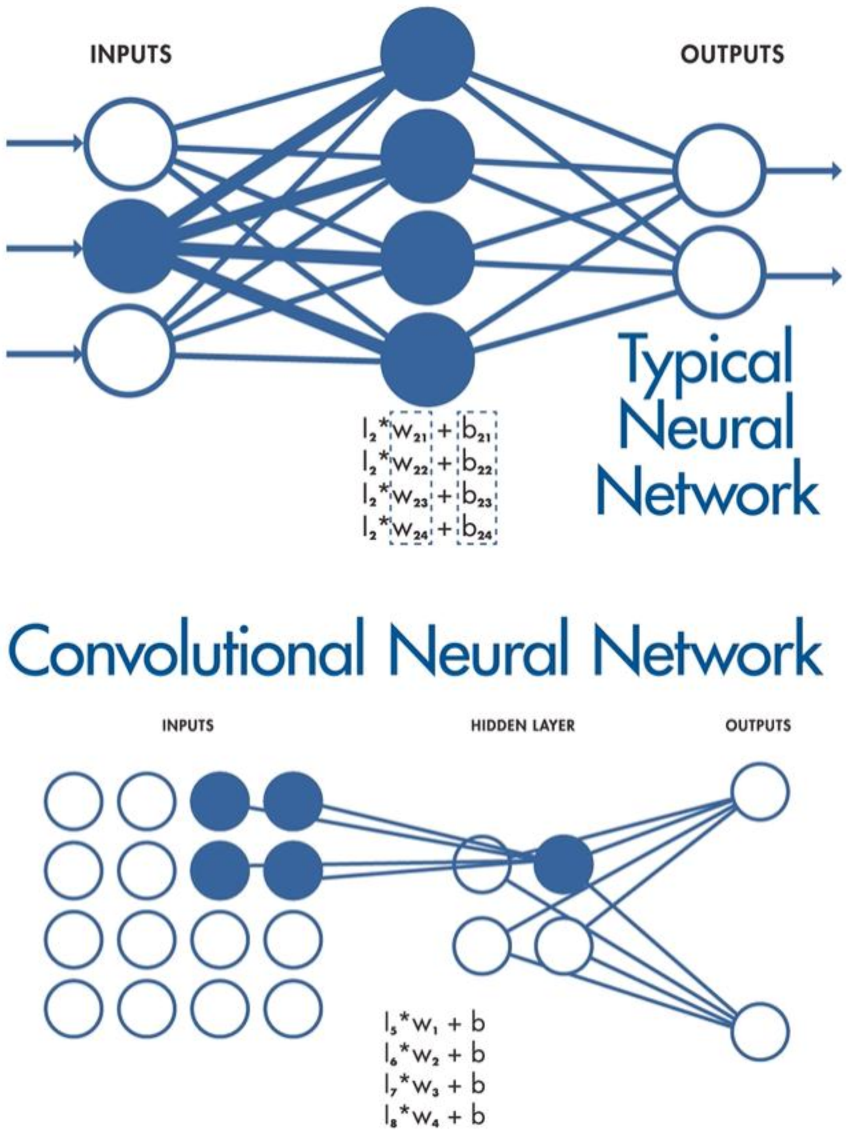
•During the backward phase, each layer will **receive a gradient** and also **return a gradient**. It will receive the gradient of loss with respect to its *outputs* and return the gradient of loss with respect to its *inputs* .


CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
HIDDEN LAYERS
CLASSIFICATION


a b c d
e f g h
i j k l
m n o p
Image
w x
y z
Kernel
aw+bx +ey+fz
bw+cx +fy+gz
cw+dx +gy+hz
ew+fx +iy+jz
fw+gx +jy+kz
gw+hx +ky+lz
iw+jx+ my+nz
jw+kx+ ny+oz
kw+lx+ oy+pz
Activation Map


-1 3 8 -4
-2 9 6 5
3 -7 -5 2
1 4 -7 8
ReLU
0 3 8 0
0 9 6 5
3 0 0 2
1 4 0 8
POOLING
9 8
4 8

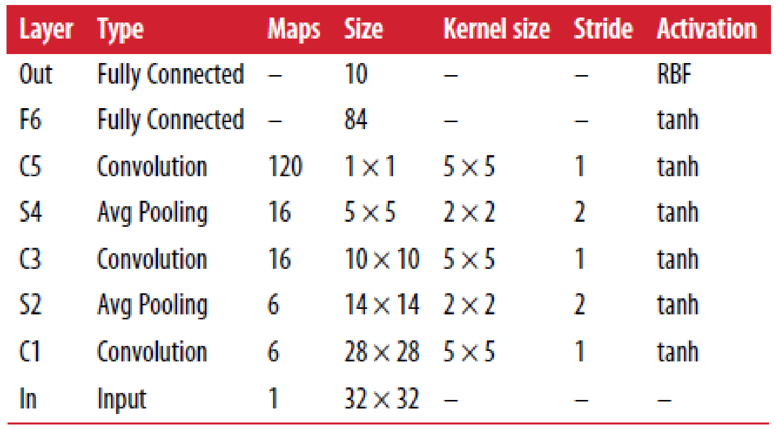




In a typical neural network, each neuron in the input layer is connected to a neuron in the hidden layer. However, in a CNN, only a small region of input layer neurons connects to neurons in the hidden layer. These regions are referred to as local receptive fields.

LeNet-5

The LeNet-5 architecture is the most widely known CNN architecture. It was created by Yann LeCun in 1998 and widely used for handwritten digit recognition

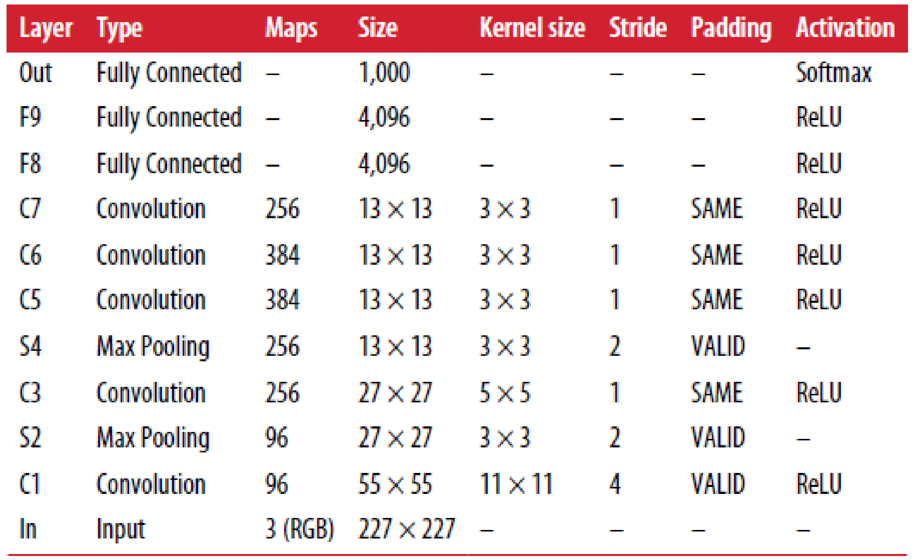


Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

AlexNet

The *AlexNet* CNN architecture was developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton.

It is quite similar to LeNet-5, only much larger and deeper, and it was the first to stack convolutional layers directly on top of each other, instead of stacking a pooling layer on top of each convolutional layer.



Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	VALID	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

Outras Arquiteturas:

GoogLeNet

VGGNet

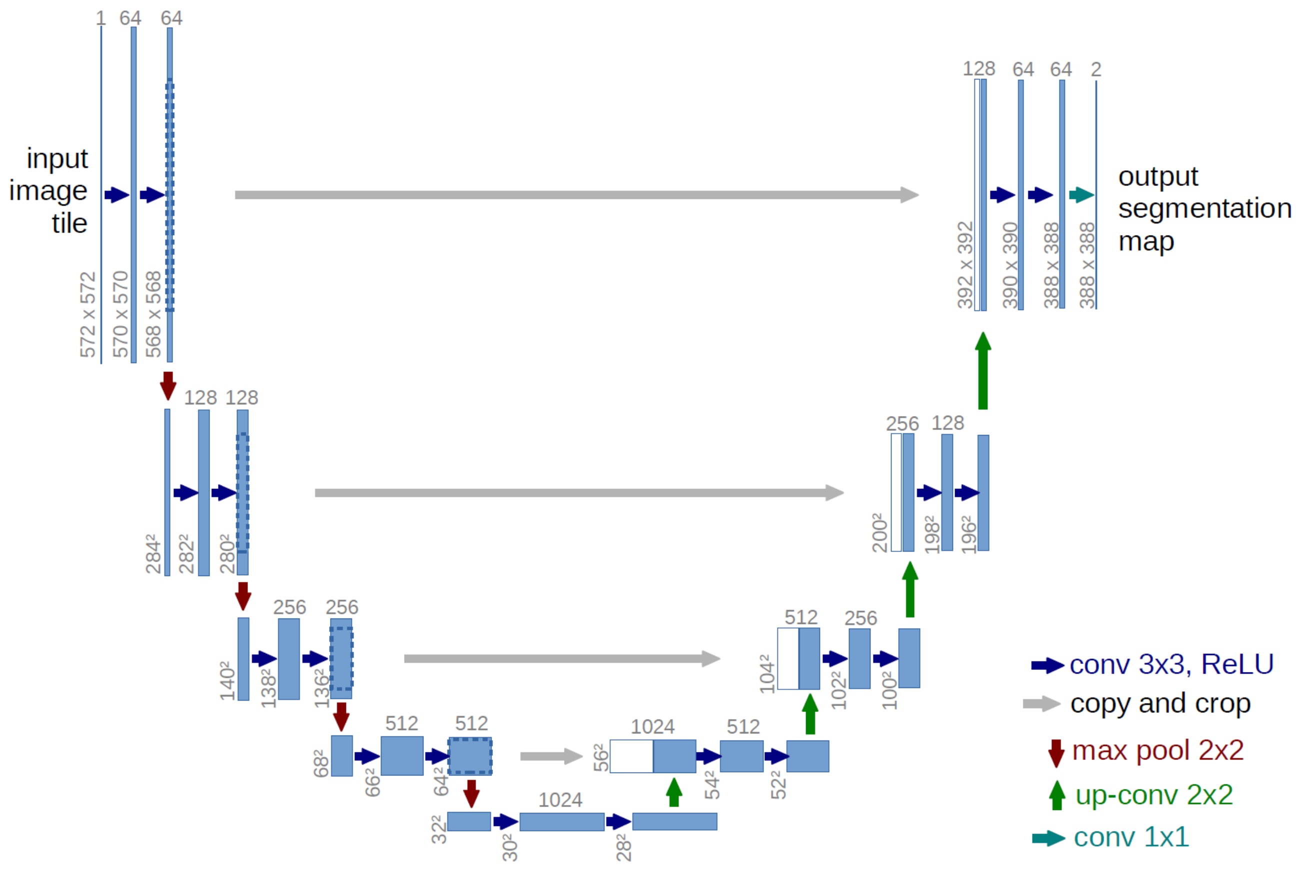
ResNet

Xception (variante da GoogLeNet)

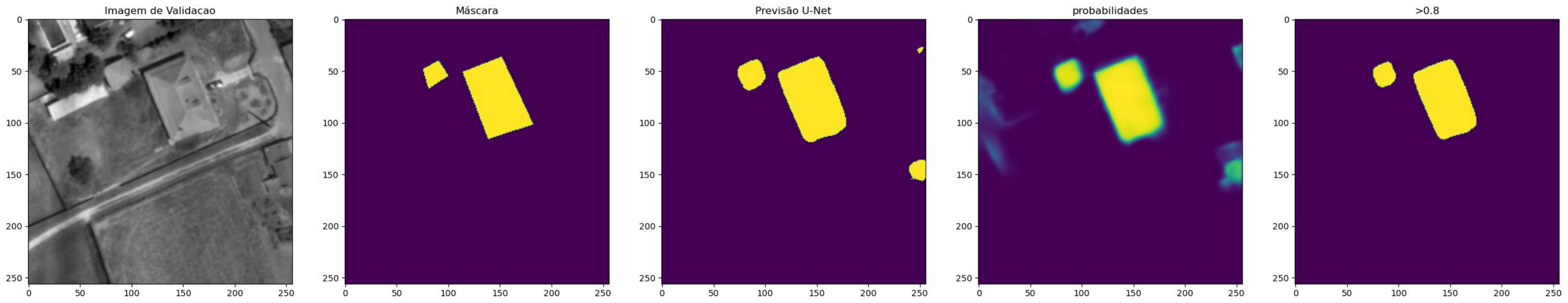
SENet

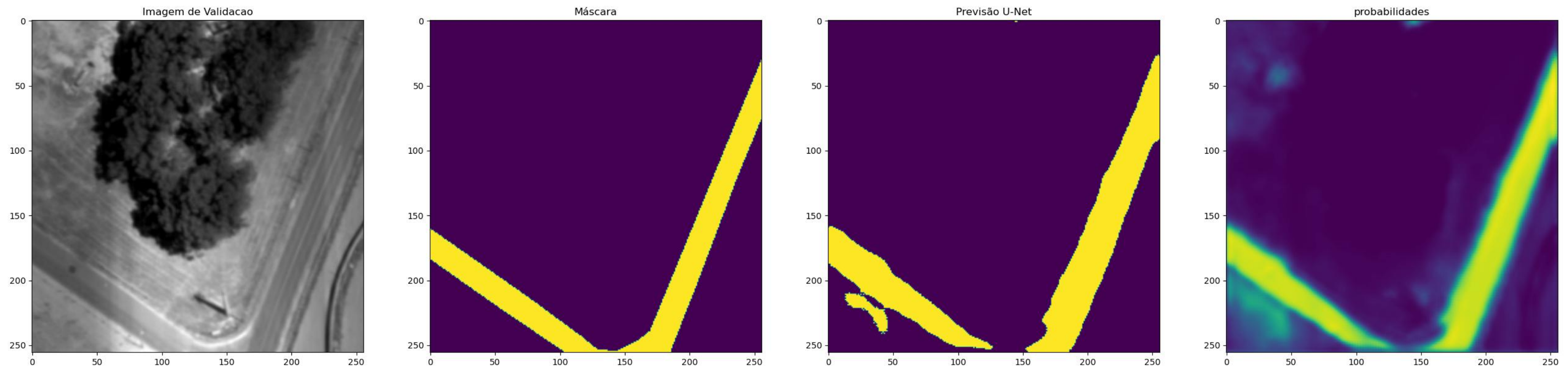
Desenvolvidas nos ultimos 5 anos

U-Net (Olaf Ronneberger, 2015)



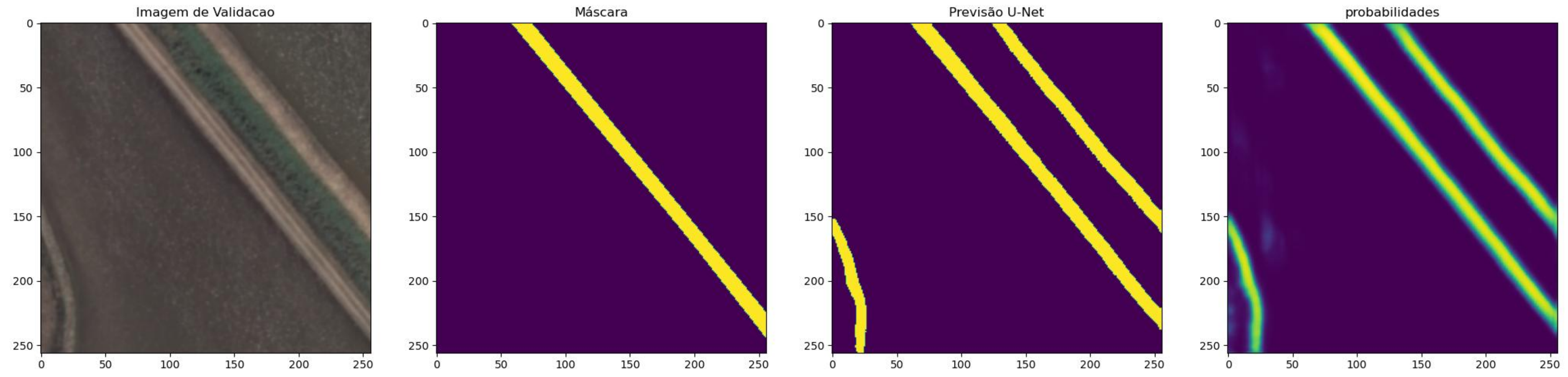
```
def modelo_unet(IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS):
    inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
    s = inputs

    #Descida
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(s)
    c1 = Dropout(0.1)(c1)
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c1)
    p1 = MaxPooling2D((2, 2))(c1)

    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p1)
    c2 = Dropout(0.1)(c2)
    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c2)
    p2 = MaxPooling2D((2, 2))(c2)

    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p2)
    c3 = Dropout(0.2)(c3)
    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c3)
    p3 = MaxPooling2D((2, 2))(c3)

    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p3)
    c4 = Dropout(0.2)(c4)
    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c4)
    p4 = MaxPooling2D((2, 2))(c4)
    #Fundo
    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c5)

#Subida
    u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u6)
    c6 = Dropout(0.2)(c6)
    c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c6)

    u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u7)
    c7 = Dropout(0.2)(c7)
    c7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c7)

    u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u8)
    c8 = Dropout(0.1)(c8)
    c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c8)

    u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u9)
    c9 = Dropout(0.1)(c9)
    c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c9)

    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
    model = Model(inputs=[inputs], outputs=[outputs])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    return model
```



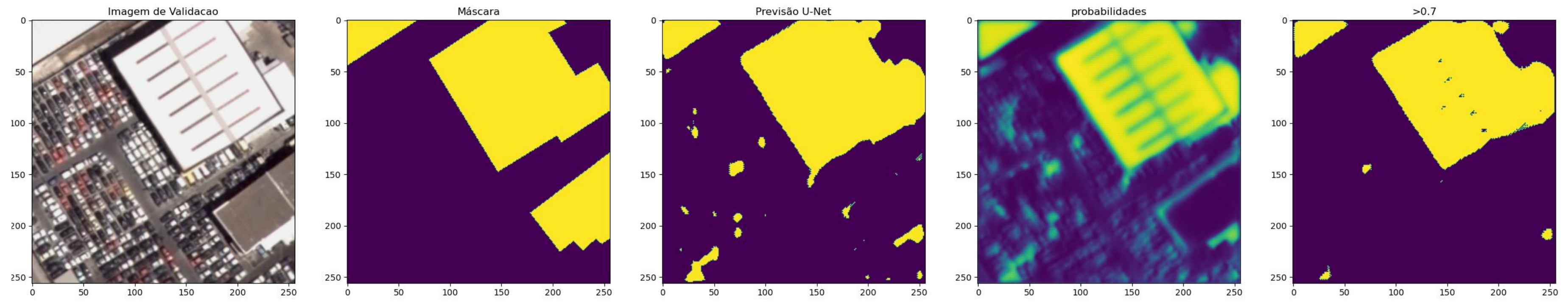


Vias

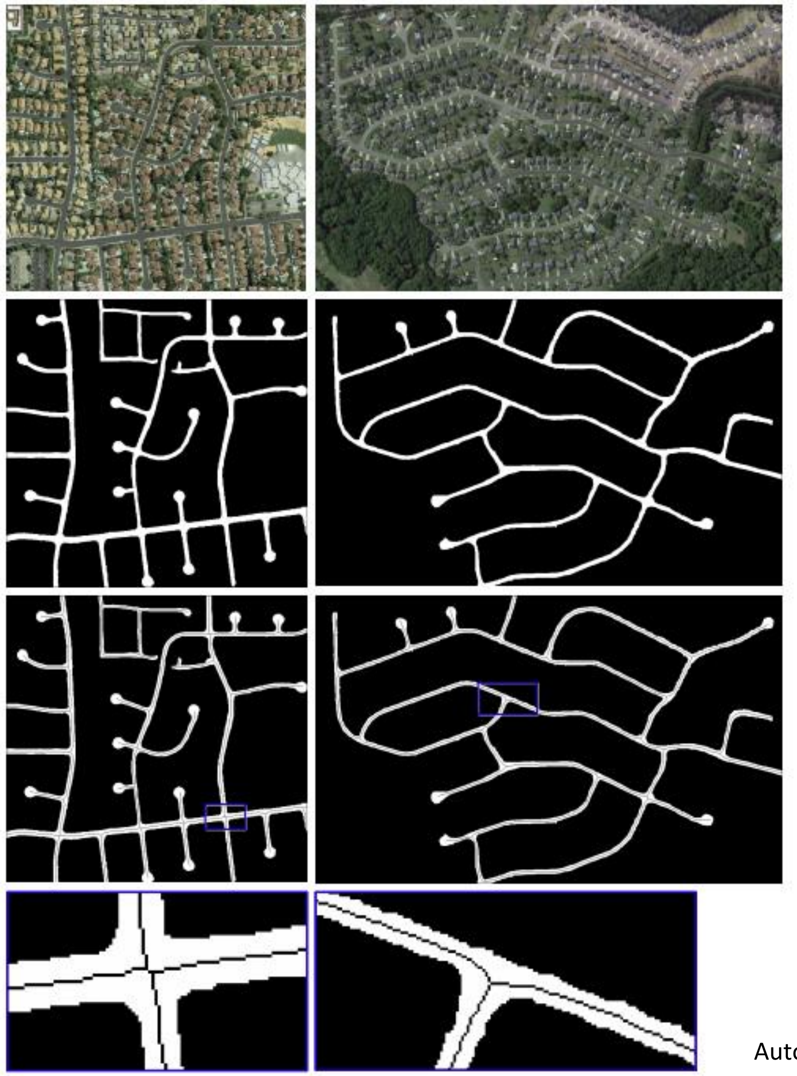
João Sacadura, 2021



Caminhos agrícolas



Edificios Industriais



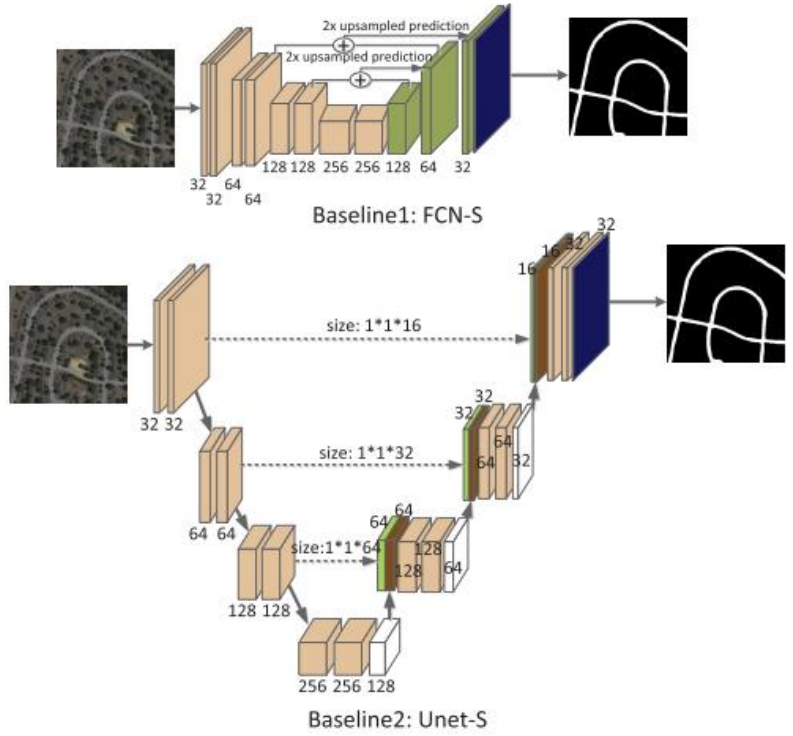


Fig. 3. Architecture of two baseline networks. In FCN-S, feature maps summation and 2× upsampled prediction are utilized. In Unet-S, concat layer is employed to concatenate two groups of feature maps. The dotted line represents convolution operation with 1 × 1 kernel.

Compared with the conventional multilayer perceptron (MLP), which only consists of fully connected layers, the convolutional network has less parameters due to its local connectivity characteristic.

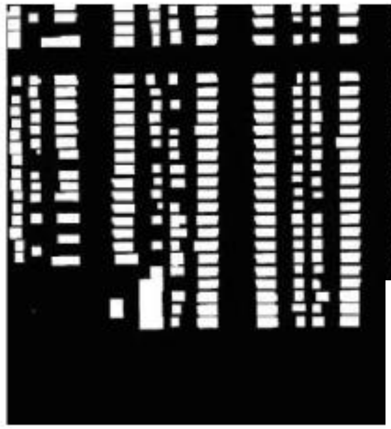
For example, for a 300 × 300 image, we assume that there are ten hidden neurons. There are 300×300×10 = 900 000 weight parameters for MLP.

In convolutional network, if we use 10×10 local connectivity pattern, the number of weight parameters is 10×10×10 = 1000.

Automatic road detection and centerline extraction via cascaded end-to-end convolutional neural network," IEEE Transactions on Geoscience and Remote Sensing, vol. 55, no. 6, pp. 3322–3337, 2017.

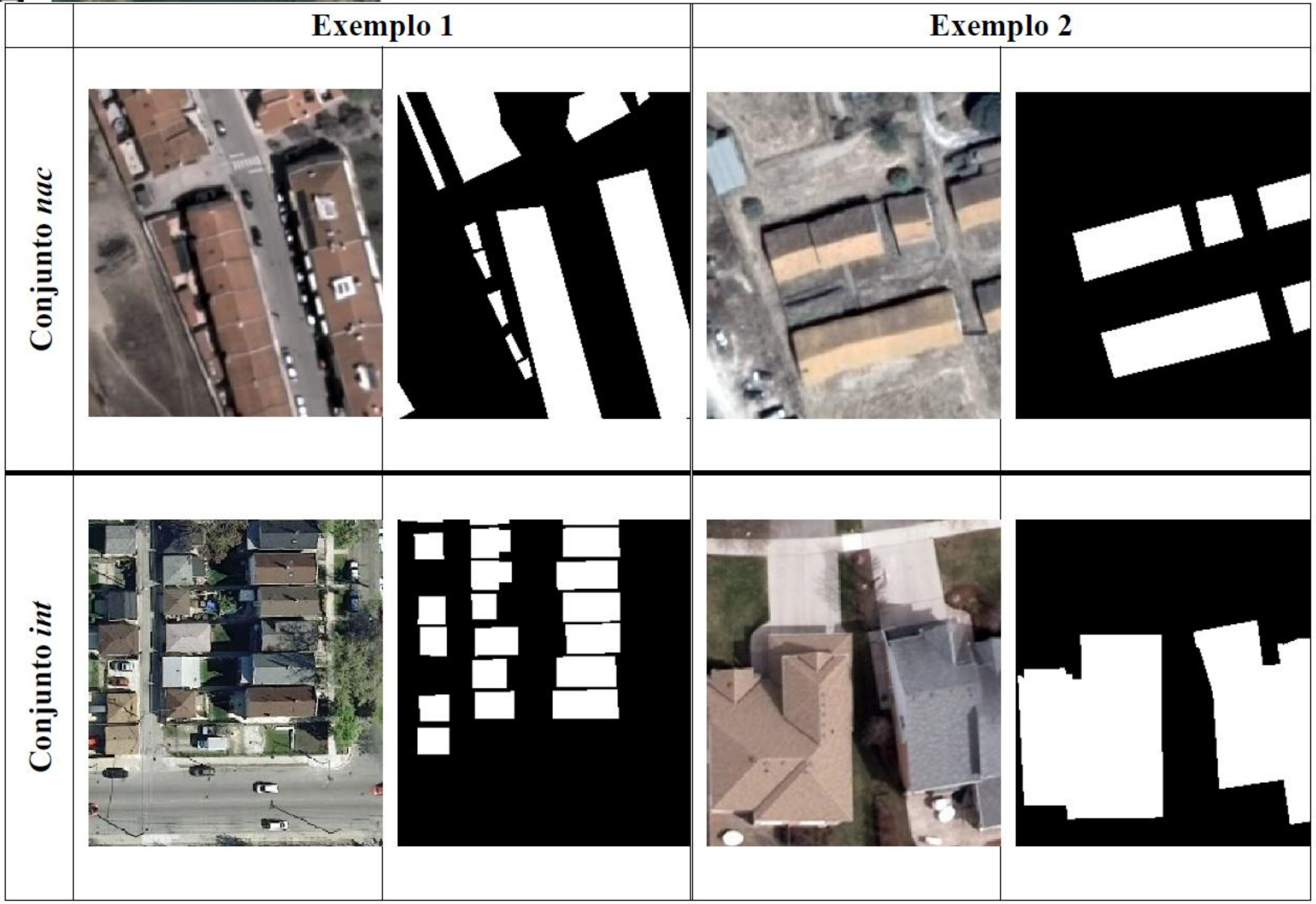
Inria Aerial Image Labeling, Maggiori et al. em 2017

Chicago

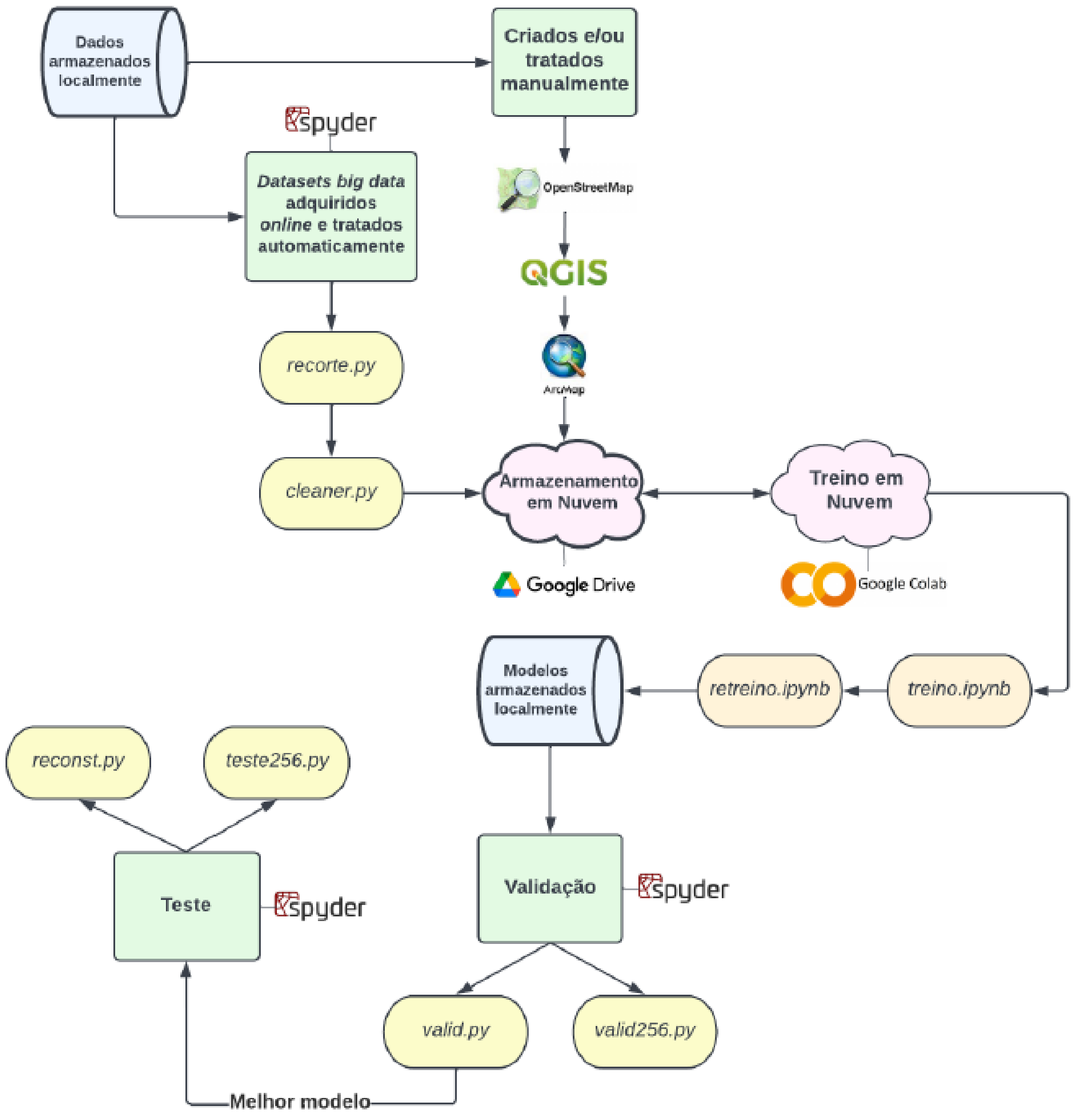

Chicago – reference

Kitsap County, WA – Reference

Vienna



Redes neurais de convolução na classificação de edifícios em imagens de alta resolução espacial, Tese Mestrado EGeoespacial, Henrique Silva, 2022



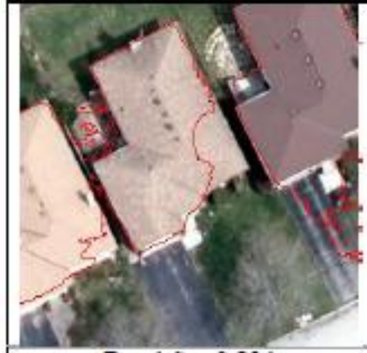
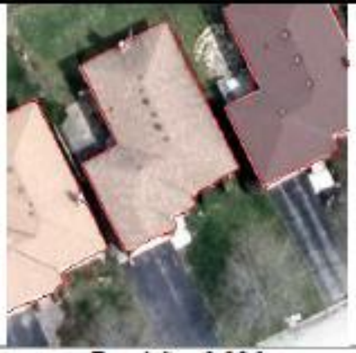
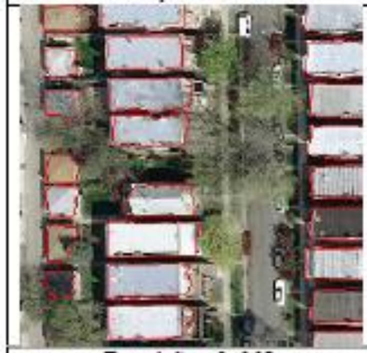
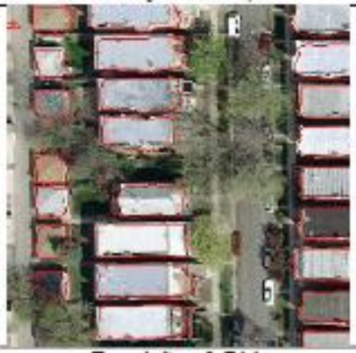
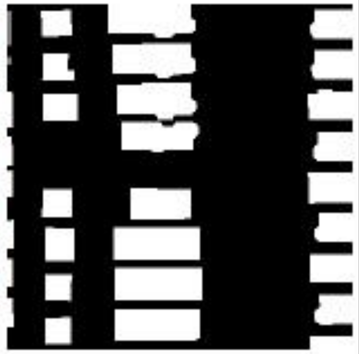
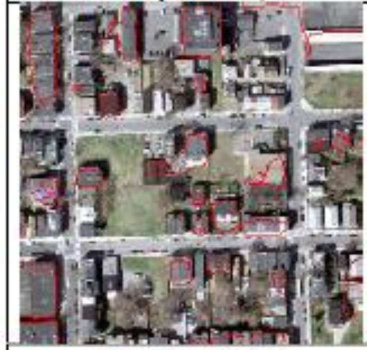
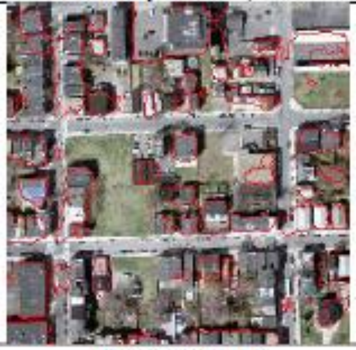
Exemplo de previsão *Inria* em *shp*	Exemplo de previsão *Full* em *shp*	Máscara verdadeira
 **Precisão:** 0.834 **Revocação:** 0.928 **Pontuação F1:** 0.878	 **Precisão:** 0,996 **Revocação:** 0,983 **Pontuação F1:** 0,989	
 **Precisão:** 0,668 **Revocação:** 0,759 **Pontuação F1:** 0,710	 **Precisão:** 0,711 **Revocação:** 0,802 **Pontuação F1:** 0,754	
 **Precisão:** 0,666 **Revocação:** 0,338 **Pontuação F1:** 0,449	 **Precisão:** 0,513 **Revocação:** 0,537 **Pontuação F1:** 0,525	

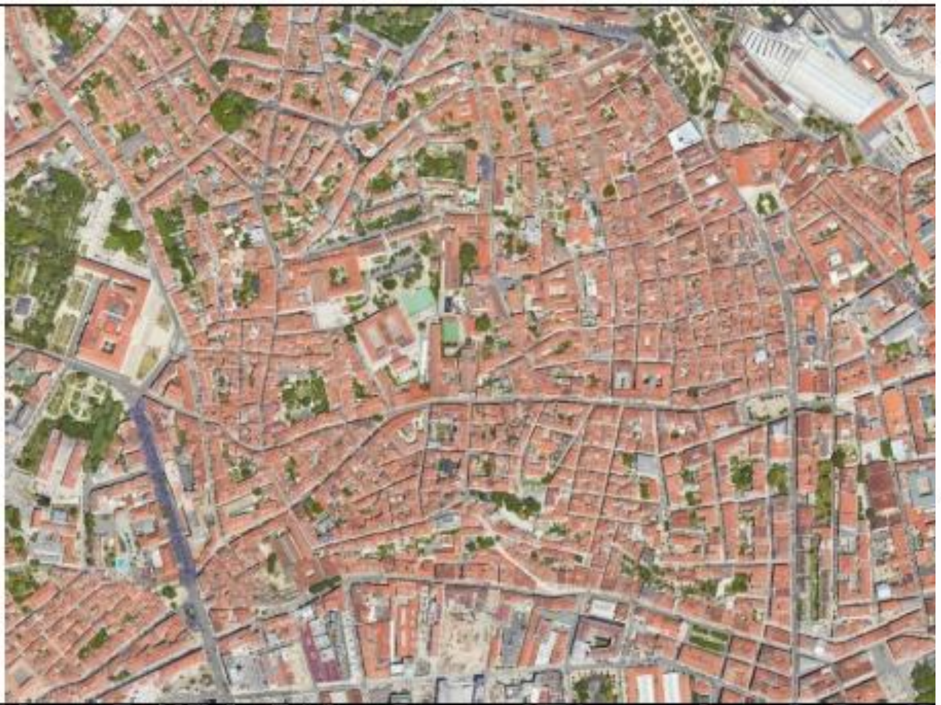
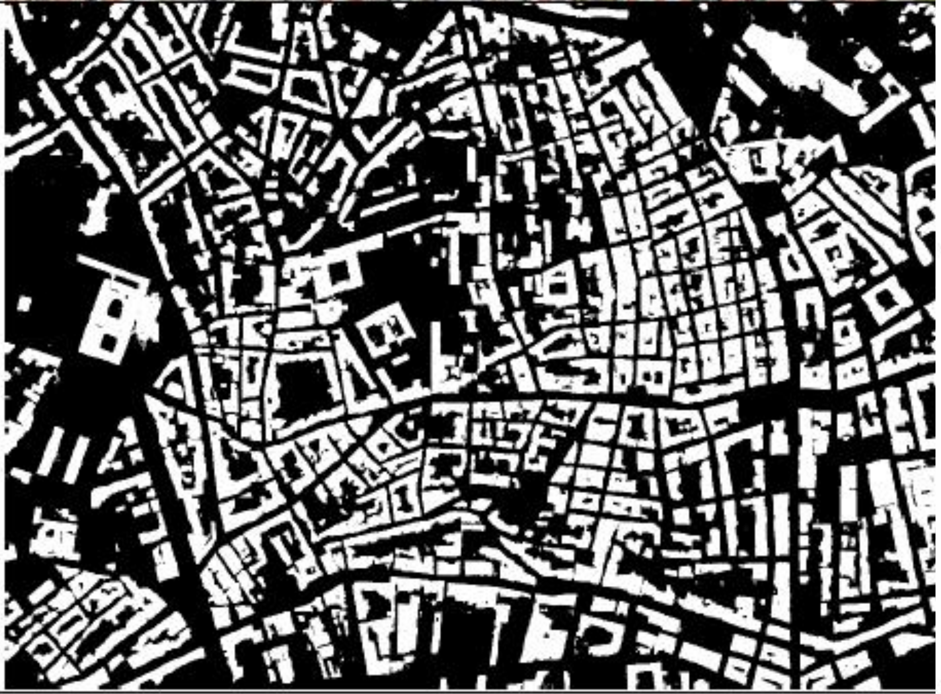
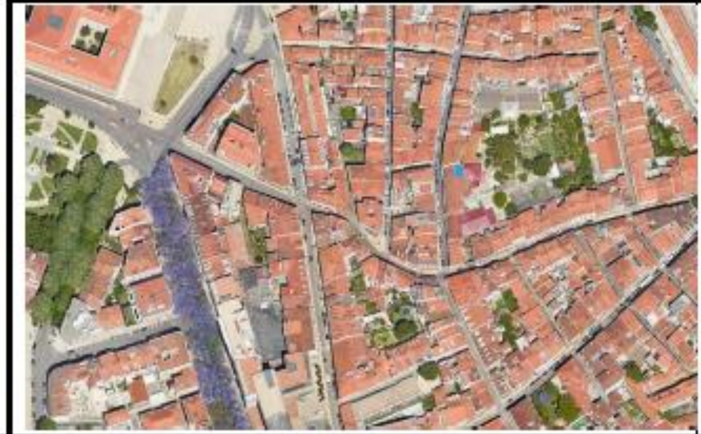
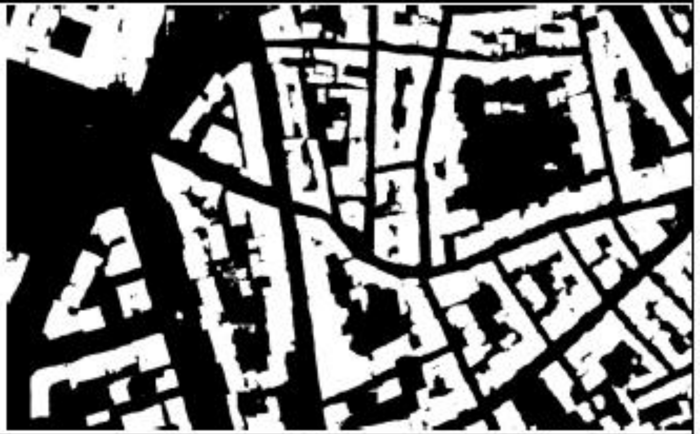
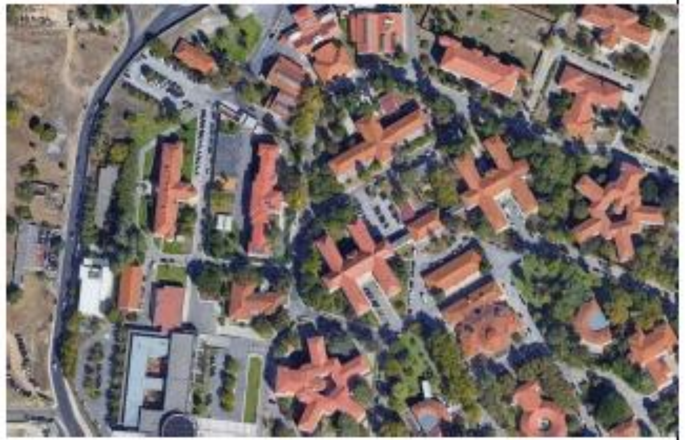
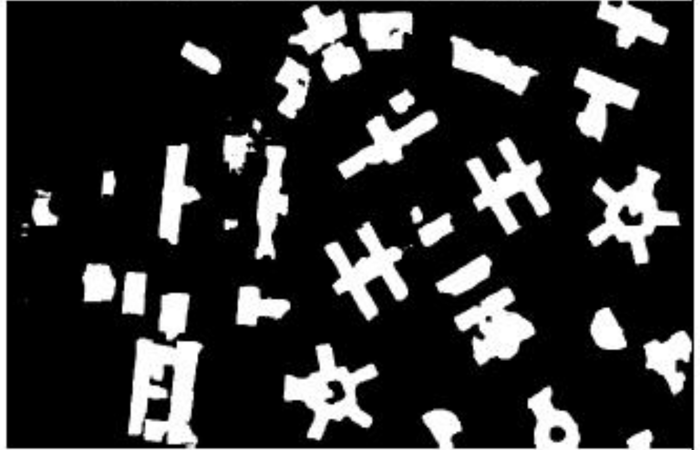
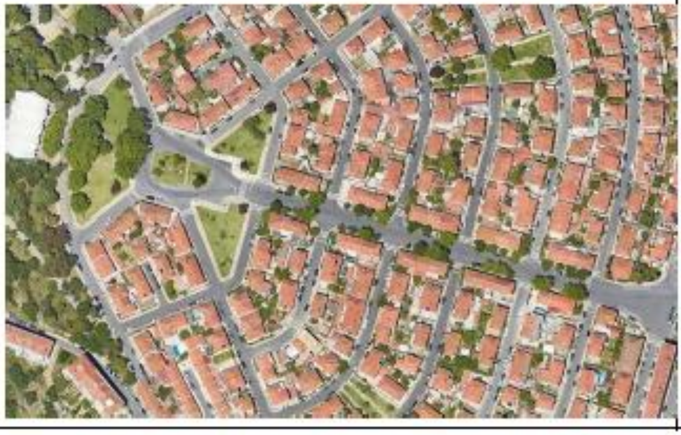
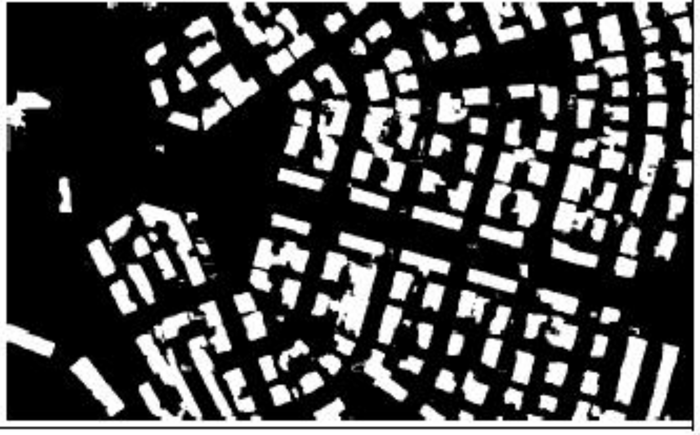
Imagem de satélite *RGB* da Baixa de Lisboa	
Máscara binária de previsão *Full+*	

	Imagem de satélite *RGB*	Máscara binária de previsão *Full+*
Jardim S. Bento, Lisboa		
Parque da Saúde, Lisboa		
Bairro Madre de Deus, Lisboa		

Identificação e co
de árvores em ort

U-Net
.Imagem RGB+IV
.Ortos 2023 (IFAP)