Algorithms for Data Analysis

Julien Tissier

December 1, 2017

Overview

1. Introduction to Data Analysis
   - Definition
   - Tools and libraries
   - Numpy
   - Pandas
   - Matplotlib
2. Machine Learning
   - Definition
   - Supervised vs. Unsupervised
   - Training, test & validation sets
   - Overfitting/Underfitting
3. Supervised Learning
   - k-Nearest Neighbors
   - Linear Models
   - Decision Trees

What is data analysis?

Data

Pieces of information (measurement, values, facts...) that can be :

- structured (matrices, tabular data, RDBMS, time series...)
- unstructured (news articles, webpages, images/video...)

Data Analysis

Process of preparing, transforming and using models to find more information from data, as well as visualizing results.

How to analyze data?

There are usually two steps in data analysis. The first one is to find and **develop models** that can extract useful information from data (with languages like R or MATLAB). The second one is to **develop programs** that can be used in production systems (with languages like Java or C++).

With a growing popularity among scientists as well as the development of efficient libraries (numpy, pandas), **Python** became a great tool for data analysis. Python has many advantages :

- great for string/data processing
- can be used for both prototyping/production
- has a lot of existing libraries
- can easily integrate C/C++/FORTRAN legacy code
- easy to read/develop

Libraries

This course will be based on Python 3.5 (or above) and the following libraries :

- IPython (6.2+) : enhanced Python shell
- numpy (1.13+) : fast/efficient arrays and operations
- pandas (0.20+) : data structures (Series/DataFrame)
- matplotlib (2.1.0+) : plots and 2D visualization
- scipy (0.19+) : scientific algorithms
- scikit-learn (0.19+) : machine learning algorithms

Jupyter Notebook will also be used to give you samples of code, as they provide a more interactive way to learn and discover how these libraries work.

Numpy

Numpy (**Num**erical **Py**thon) is a high performance scientific computing library that can be used for matrices computations, Fourier transforms, linear algebra, statistical computations...

The main type of data in Numpy is the **ndarray** :

- n-dimensional array
- fixed size
- homogeneous datatypes
- similar to C arrays (continuous block of memory)

Why is Numpy efficient?

As a high level language, Python is slow to do any heavy computations, especially if very large arrays are involved. Numpy solves this problem thanks to the ndarray datatypes :

- efficient memory management (continuous block)
- use C loops instead of Python loops for computations on array
- vectorized operations (computations are done block by block, not element by element)
- rely on low-level routines for some operations (BLAS/LAPACK)

Numpy

Example

```
import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([6, 7, 8, 9])

c = a * b

d = np.array([[1, 2], [3, 4]])
```

Pandas

Pandas is a high-performance Python library used to work with data and analyze them. It contains many pre-implemented methods to read and parse data, as well as common statistical computations (mean, variance, correlation...).

Pandas has two main datatypes:

- Series : one-dimensional container. Indexes can be integers (like an array) or other objects (string, date...)
- DataFrame : tabular data, like a spreadsheet. It contains multiple rows and multiple columns. It can be seen as a collection of Series.

Pandas

Example (Series)

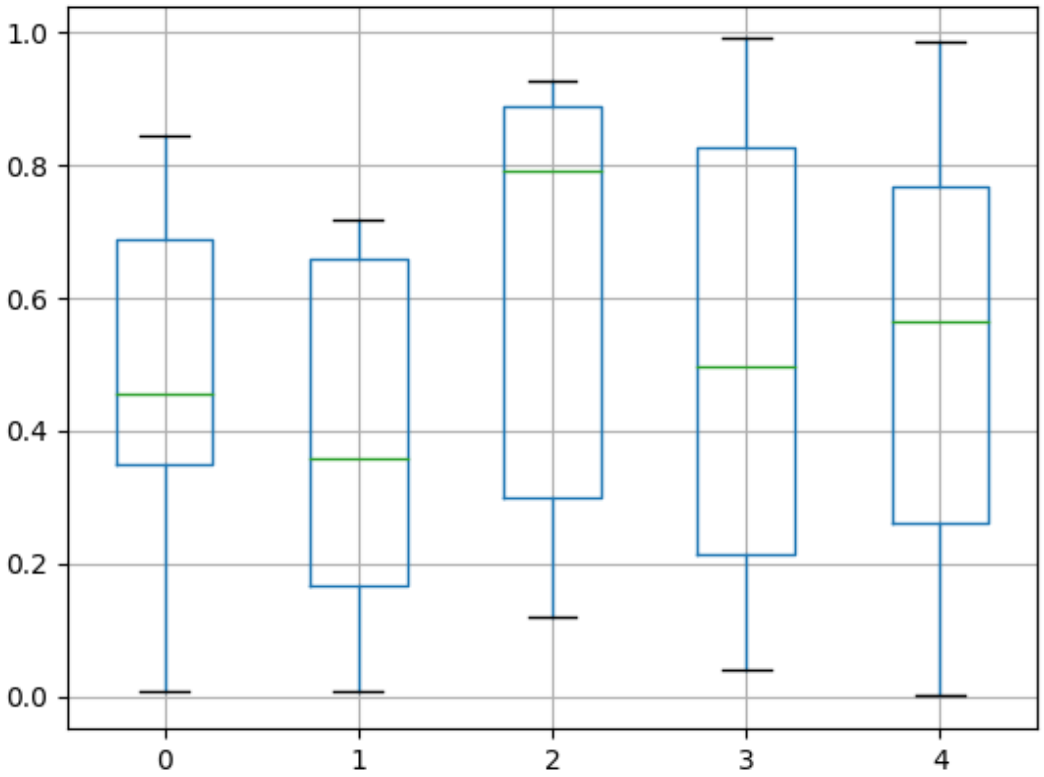
```
from pandas import Series

s1 = Series([4, 7, 9, -1])
s2 = Series([12, 3.2, "John"],
            index=["mark", "gpa", "name"])
```

Example (DataFrame)

```
from pandas import DataFrame
df = DataFrame({"key1": [1, 2, 3],
                "key2": [4, 5, 6]})
```

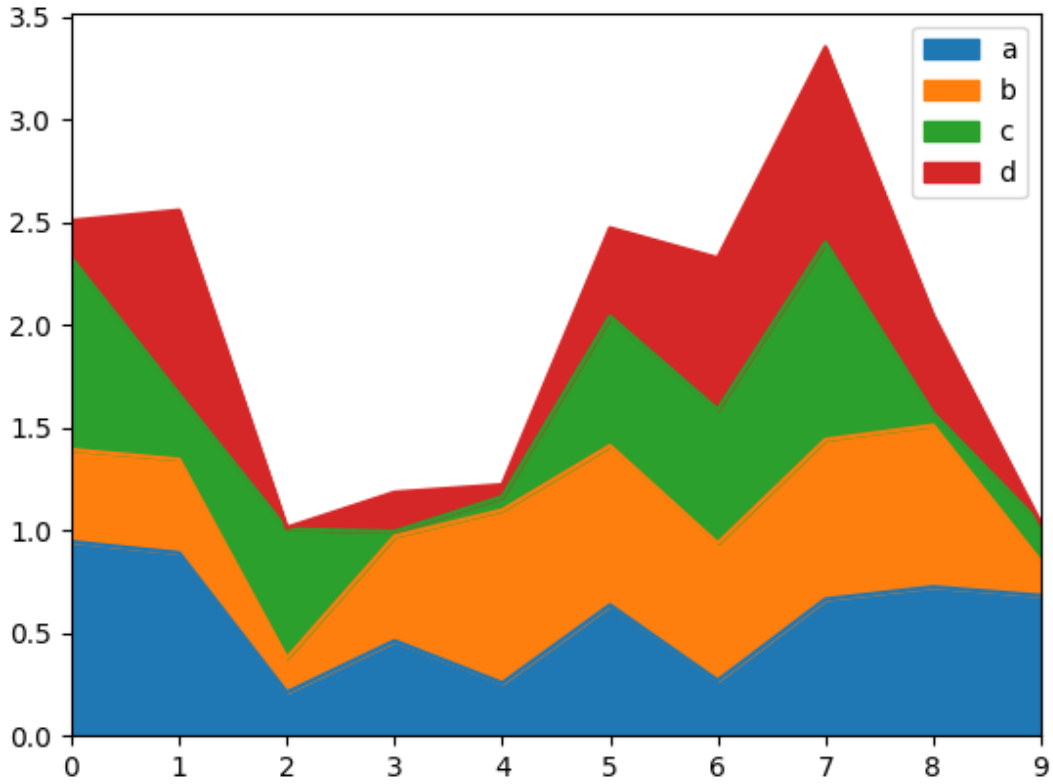
Matplotlib

Matplotlib is a plotting library used to visualize data and create graphics. Pandas directly uses matplotlib for representation.



Matplotlib

Matplotlib is a plotting library used to visualize data and create graphics. Pandas directly uses matplotlib for representation.



What is Machine Learning?

Machine Learning (ML)

ML is when a computational system can **automatically learn** and extract **knowledge from data** without being explicitly programmed. ML is at the intersection of statistics, artificial intelligence and computer science.
ML is now everywhere:

- automatic friend tagging in your Facebook photos
- music recommendation for Spotify/YouTube
- self-driven cars
- medical diagnosis in a hospital
- and many more...

ML can be **supervised** or **unsupervised**.

What is Machine Learning?

Brief history

At the beginning, many decision-making applications (like spam detection) were built with a lot of manually programmed if/else conditions. These methods have some drawbacks :

- it requires the **knowledge of an expert** to design the rules
- it is **very specific** to a task

With ML, you only need to feed a model with your data. The algorithm will find the rules by itself.

Supervised Learning

In supervised learning, you give the algorithm two kinds of data :

- **inputs** (an image, information about a person...)
- **outputs** (name of the object, the salary...)

The algorithm learns to find a way to **produce the output given the input**. The algorithm is then capable of producing the output of an input it has never seen before.

Supervised learning can be used to :

- recognize handwritten digits (input=image, output=digit)
- identify spams (input=mail, output=yes/no)
- predict a price (input=house infomation, output=price)

Supervised Learning

The inputs are in general real value vectors (sometimes it can be binary vectors). Each dimension of this vector is called a **feature**.

There are two main problems that can be solved with supervised learning :

- **classification** : the output is the class the item belongs to. It can be a binary classification problem (yes/no) or a multi-class classification problem (class 1, class 2, class 3...)
- **regression** : the output is a real value number

Unsupervised Learning

In unsupervised learning, there are no known output, so you only give the algorithm the inputs. For example, if you want to regroup similar clients according to their customer habits, you do not know in advance what are the groups.

The main problems that can be solved with unsupervised learning are :

- **clustering**
- **dimensionality reduction**

How to evaluate a ML algorithm?

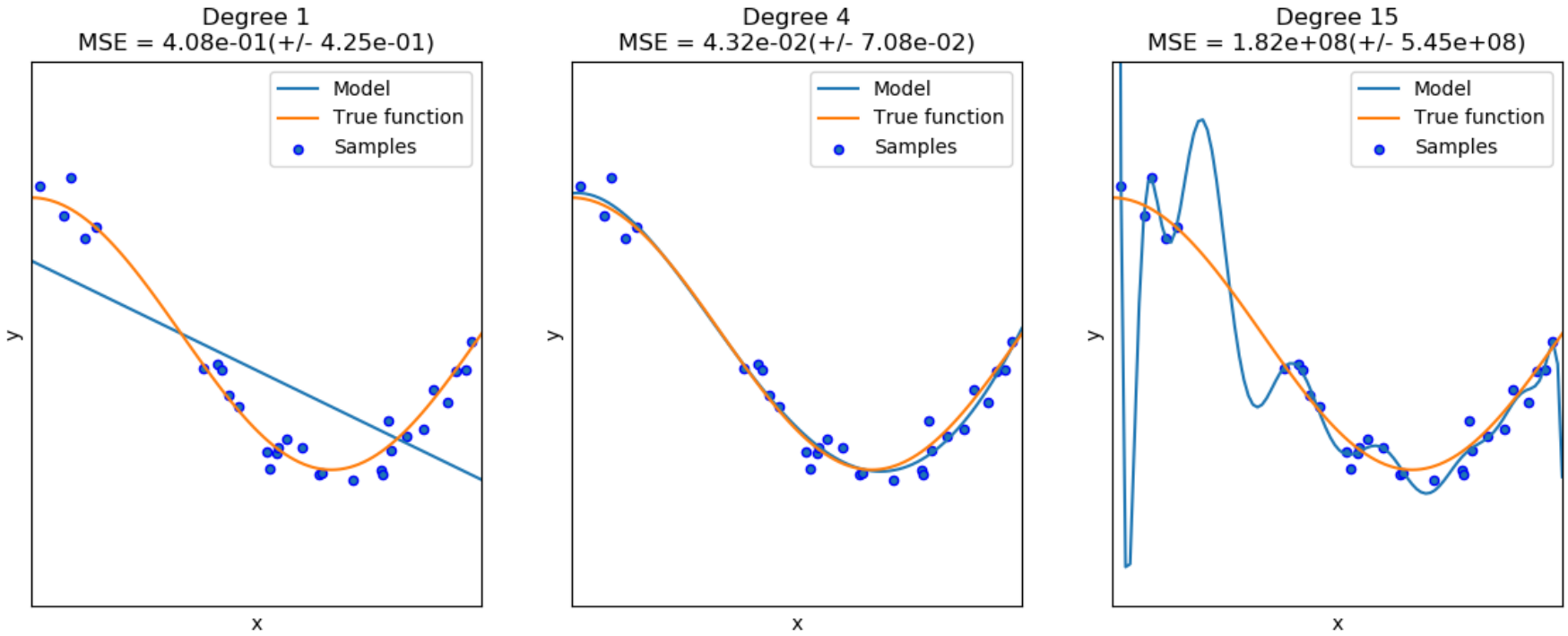
After training a ML algorithm with data, we need to evaluate its performance. One way would be to feed the algorithm with the same data, compute the output and compare it with the original output.

This method is bad because it does not tell us the capacity of the algorithm **to generalize** (is the algorithm able to perform well on unseen data ?). One way to solve this problem is to separate the data into 3 sets :

- training set : used to train the algorithm
- validation set : used to adjust the algorithm
- test set : used to measure the performance of the algorithm

Problems of ML

- **underfitting** : when the model is too simple. The training score and the test score are both bad (left)
- **overfitting** : when the model is too complex (right). The training score is good but the test score is bad. The model is not able to generalize well.

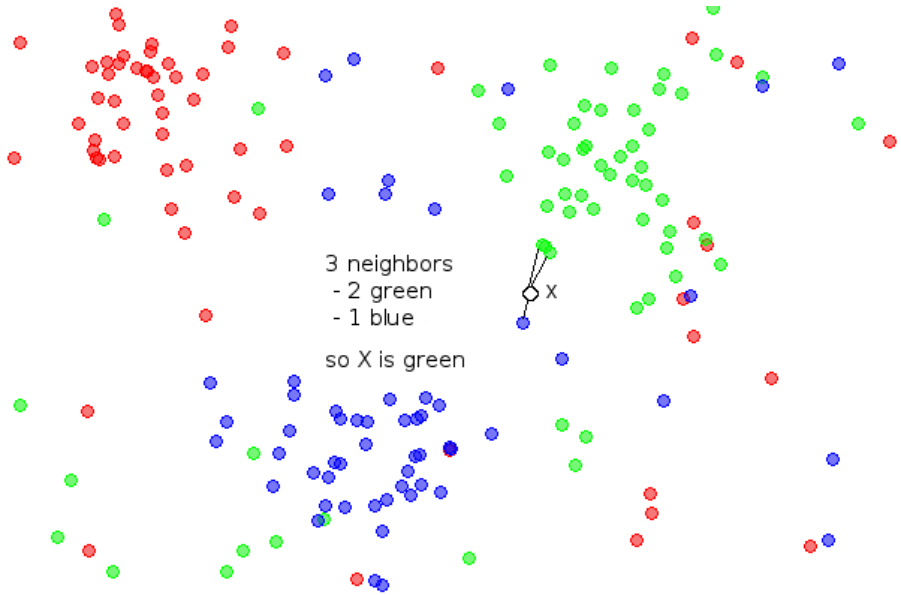


k-Nearest Neighbors - Classification

Inputs : set of training points $\mathcal{S}$, point X to classify
Procedure :

- find the k closest points (the neighbors) to X from $\mathcal{S}$
- find the most frequent class c among the neighbors
- assign c to X

The k closest points are defined as the points with minimal distance (usually the Euclidean distance) to X. If there is a tie in finding c, we can increase/decrease k until the tie is broken.



k-Nearest Neighbors - Regression

Inputs : set of training points $\mathcal{S}$, point X to predict value
Procedure :

- find the k closest points (the neighbors) to X from $\mathcal{S}$
- compute the average value v of all neighbors
- assign v to X

If k is set to 1, the value assigned to X is simply the value of its nearest neighbor.

k-Nearest Neighbors

The k-nearest neighbors is one of the simplest model in ML. It usually has two main parameters : **the number of neighbors** used and **the distance** used to find the closest points.

Pros

- model is easy to understand
- does not require extended tuning to achieve good performance
- works well as a baseline before training more complex models

Cons

- slow when many training points or many features
- not very good on sparse data

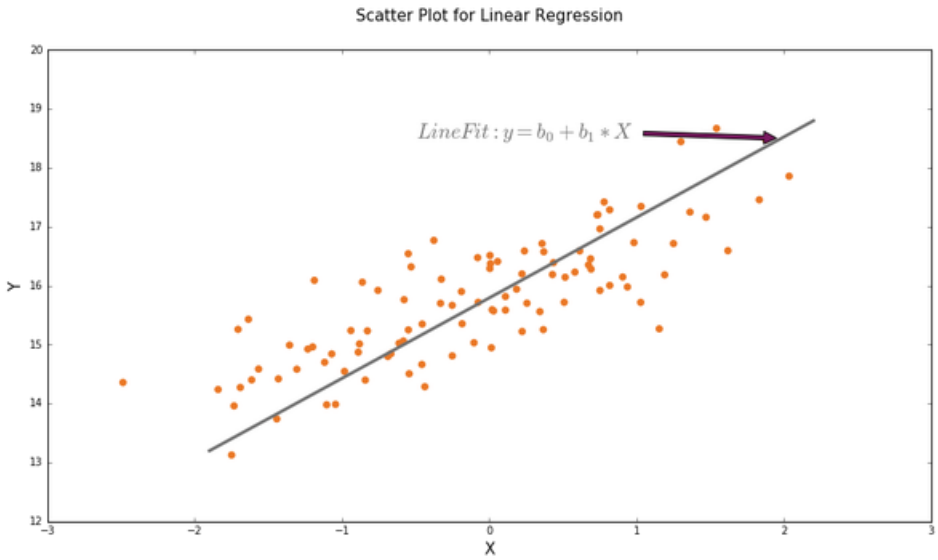
Linear Models - Regression

Linear models are usually used to make values prediction. The output $\hat{y}$ is a **linear function** of each features x_i of a given input vector X (hence the name of **linear** models).

$$\hat{y} = \sum_{i=1}^{k} w_i * x_i + b$$

The model learns the coefficients w_i and b. The model is trained to minimize the **Mean Squared Error (MSE)** on all training points X in $\mathcal{S}$ (where y is the correct value of X).

$$MSE = \frac{1}{\#\mathcal{S}} \sum_{X \in \mathcal{S}} (\hat{y} - y)^2$$



Linear Models - Regression

- Training

  Inputs : set of training points $\mathcal{S}$

  Procedure :
  - compute the MSE
  - compute the partial derivatives $\frac{\partial MSE}{\partial w_i}$ and $\frac{\partial MSE}{\partial b}$
  - solve the linear system given by the equations $\frac{\partial MSE}{\partial w_i} = 0$ and $\frac{\partial MSE}{\partial b} = 0$
  - the solution is unique and gives you the w_i and b

  Output : learned w_i and b

- Predicting

  Inputs : learned w_i and b, point X to predict value

  Procedure : compute $\hat{y}$ with the formula and w_i, b, x_i

  Output : predicted value $\hat{y}$

Linear Models - Classification

Linear models can also be used for either binary or multi-class classification problems. For binary classification (1 or 0 labels), the model evaluates the output with :

$$\hat{y} = \begin{cases} 1 & \text{if } w \cdot x + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

We can rewrite the condition to remove the bias b with $\sum_{i=0}^{k} w_i * x_i \geq 0$ and set x_0 to 1 for all vectors X. The objective of the model is to find a good w to predict the correct label (such that $\hat{y} = y$). Different methods exist to find w :

- perceptron
- logistic regression
- support vector machines (SVM)

Linear Models - Perceptron

We can define a loss ℓ (called the zero-one loss) for each point (x, y) from our training set as follows :

$$\ell(x, y, w) = 1_{[(w \cdot x)y \leq 0]}$$

The perceptron algorithm learns the w weights as follows :

- initialize w to 0
- **for** step in [1..n]
  - **for** x in training dataset

    if $(w \cdot x) \times y \leq 0$

    $\mathrm{w} = \mathrm{w} + \lambda\ \mathrm{y} \cdot \mathrm{x}$

The perceptron algorithm only updates the weights w when it encounters a misclassified example. If the dataset is **linearly separable**, the perceptron algorithm is guaranteed to find an optimal solution.

Linear Models - Logistic Regression

Instead of directly finding the output $\hat{y}$ with the sign function, we can compute the **probability of belonging to class 1** defined as follows :

$$P(y = 1|x) = \sigma(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}} \in [0, 1]$$

The idea is to find w such that this probability is high (near 1) for points labeled with 1, and low (near 0) for points labeled with 0. We can define another loss function (called negative log-likelihood) :

$$\ell(x, y, w) = -y \times \log(\sigma(w \cdot x)) - (1 - y) \times \log(1 - \sigma(w \cdot x))$$

The model is trained to minimize the loss for all training examples. Since this loss function is convex, we can use different algorithms like gradient descent, L-BFGS...

Linear Models

Linear models have two main parameters : **the weight of regularization** and **the type of regularization** used.

Pros

- models are fast to train and predict
- can be used with large datasets (scalability)
- works well on sparse data

Cons

- learned coefficients are not very interpretable
- not very good at detecting correlated features

Decision Trees

Decision trees are mostly used for classification problems. They learn a **sequence of conditions** (if/else questions) to find the class of a vector x. Let's suppose we have the following dataset and we want to predict if the game can be played (9 yes / 5 no) :

	Outlook	Humidity	Wind	Play
1	Sunny	High	Weak	No
2	Sunny	High	Strong	No
3	Overcast	High	Weak	Yes
4	Rain	High	Weak	Yes
5	Rain	Normal	Weak	Yes
6	Rain	Normal	Strong	No
7	Overcast	Normal	Strong	Yes
8	Sunny	High	Weak	No
9	Sunny	Normal	Weak	Yes
10	Rain	Normal	Weak	Yes
11	Sunny	Normal	Strong	Yes
12	Overcast	High	Strong	Yes
13	Overcast	Normal	Weak	Yes
14	Rain	High	Strong	No

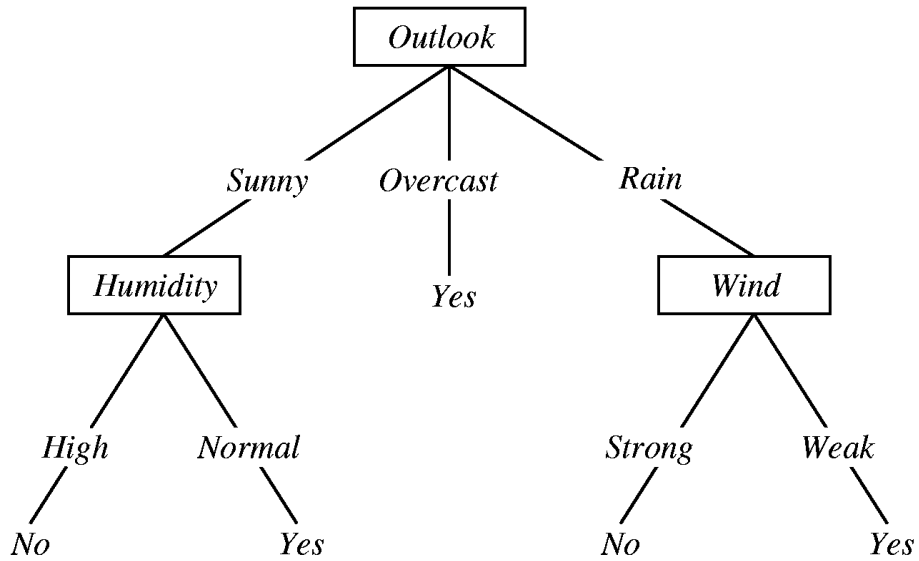
Decision Trees

Many algorithms exist to create a decision tree. One of them is ID3. At each step, it computes the entropy of each unused attributes with :

$$Entropy(S) = \sum_{x \in X} -p(x) \log_2 p(x)$$

where $p(x)$ is the ration between the number of elements in x and in S. The attribute that has the lower entropy is chosen as a node in the tree.

	Outlook	Humidity	Wind	Play
1	Sunny	High	Weak	No
2	Sunny	High	Strong	No
3	Overcast	High	Weak	Yes
4	Rain	High	Weak	Yes
5	Rain	Normal	Weak	Yes
6	Rain	Normal	Strong	No
7	Overcast	Normal	Strong	Yes
8	Sunny	High	Weak	No
9	Sunny	Normal	Weak	Yes
10	Rain	Normal	Weak	Yes
11	Sunny	Normal	Strong	Yes
12	Overcast	High	Strong	Yes
13	Overcast	Normal	Weak	Yes
14	Rain	High	Strong	No



Decision Trees

Decision trees have many parameters to control their size : maximum depth, number of leaf, acceptable ratio in a node... They provide a strong understandable tool for people not familiar with the ML field.

Pros

- can visually be understood
- can be used with large datasets
- no preprocessing needed (normalization of features)

Cons

- tend to easily overfit
- poor generalization performance