Introduction to optimization and machine learning

Lesson 3 : basis of machine learning

Sébastien Verel

Laboratoire d'Informatique, Signal et Image de la Côte d'opale (LISIC)
Université du Littoral Côte d'Opale, Calais, France
http://www-lisic.univ-littoral.fr/~verel/

March, 2023

General outline

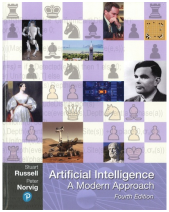
- Introduction to optimization problems
- Introduction to machine learning
- Fundamentals of optimization methods
- Fundamentals of machine learning methods
- Practice of some algorithms using python

Outline of the day

- Linear regression methods (supervised learning)
- Classification : decision tree
- Non-supervised classification
- Ensemble methods : random forest, gradient boosting

Bibliography

Data science :
fondamentaux et études de cas
Machine learning avec Python et R
Eric Biernat
Michel Lutz
EYROLLES



- Data Science : fondamentaux et études de cas, Machine Learning avec Python et R , Eric Biernat, Michel Lutz, Eyrolles, 2015.
- Artificial Intelligence : A Modern Approach, Fourth edition, 2020, Stuart Russell and Peter Norvig.
- Vincent Barra, Antoine Cornuéjols, Laurent Miclet, "Apprentissage Artificiel. Concepts et algorithmes. De Bayes et Hume au Deep learning" Eyrolles. Mars 2021. 990 pages.

Machine Learning

E : set of all possible tasks.
S : a system (a machine)

A more formal definition [T.M. Mitchell, 1997]

$T \subset E$: set of tasks called *training set*
$P : \mathcal{S} \times E \to \mathbb{R}$: performance metric of a system on tasks.

A system S **learns** from an experience Exp if
the performance of S on tasks T, measured by P, is improving.

$$P(S_{\text{before Exp}}, T) \leq P(S_{\text{after Exp}}, T)$$

Example

Task T : Classifier of emails during one day
Performance P : rejection rate of spams by S
Experience Exp : 1 weak of emails from users

Learning from L. Valliant, 1984 [Turing award, 2010]

PAC ("Probably Approximately Correct")

In model of PAC Learning under the uniform distribution on X, a learning problem is defined with a concept class $\mathcal{C}$, which is just a collection of functions $f : X \to \mathbb{R}$; "*We learn a class $\mathcal{C}$ of functions*".

A learning algorithm A for $\mathcal{C}$ is a randomized algorithm which has limited access to an unknown target function $f \in \mathcal{C}$.
The two access models are :

- random : A can draw pairs $(x, f(x))$ where $x \in X$ is uniformly random
- queries : A can request the value $f(x)$ for any $x \in X$ of its choice.

A is given as input an accuracy parameter $\epsilon \in [0, 1/2]$.
Output of A : a hypothesis function $h : X \to \mathbb{R}$.

PAC learning

A learns $\mathcal{C}$ with error ϵ if for any $f \in \mathcal{C}$, with high probability,
A outputs an h which is ϵ-close to f : $\mathrm{dist}(f, h) \leq \epsilon$.

Data type

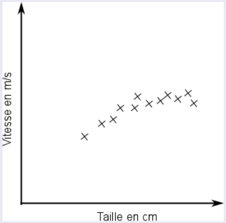
- **Quantitative** data
  mesurable quantity, answer to "how much ?"
  allow computation (mean, etc.),
  comparaisons (equality, difference, inferior/superior)
  - Numerical : $\in \mathbb{R}$
  - Discrete : number of values are limited
- **Qualitative** data
  quality or features
  answer to the "category"
  - Nominale (categorial), ex : eyes color
    comparison (equality / difference)
  - Ordinal
    Order between elements (degree to test, etc.)
    comparison : superior / inférior
- **Structured** data
  relations, etc.
  - Tree, graph, complex data, etc.

Typology according to available information

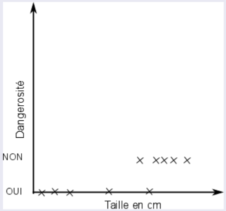
- Supervised learning :
  Learn from a set of examples :
  $\{(x_i, y_i) \mid i \in 1..n\}$
- Non-supervised learning :
  Learn from a set of example without labels (cf. clustering)
  $\{x_i \mid i \in 1..n\}$
- Semi-supervised learning :
  Learn from a set of examples with, and without labels
- Reinforcement learning :
  Learn when the actions on environment
  are rewarded by a score
- ...

Typology according to data

- Regression : (x_i, y_i) with $y_i \in \mathbb{R}$



- Classification : (x_i, y_i) with y_i discrete



Univariate linear regression

Definition of the model

$$\text{input value } x \xrightarrow{\text{model } h} \text{output value } y$$

With univariate linear regression :

$$h_\theta(X) = \theta_0 + \theta_1 X$$

Find parameter θ such that $h_\theta(X)$ is the closest to Y

Mean square error function

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

Gradient descent : $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

Multivariate linear regression

Definition of the model

$$\text{input value } x \xrightarrow{\text{model } h} \text{output value } y$$

With multivariate linear regression :

$$h_\theta(X) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \ldots + \theta_n X_n$$

Mean square error function

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

Gradient descent : $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
Normalization (scaling) of the predictors : reduced centered variable ($\mu = 0$, $\sigma^2 = 1$)

Practice with scikit-learn

Scikit-learn is a library in python with MLmodels, and related tools. Open source, BSD license, `https://scikit-learn.org/`

From example Ordinary Least Square

```
from sklearn import linear_model

reg = linear_model.LinearRegression()
reg.fit ([[0, 0], [1, 1], [2, 2]], [0, 1, 2])

print(reg.coef_)
```

see code : `linear_reg.ipynb` and url :
`https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-download-auto-examples-linear-model-plot-ols-py`

Polynomial regression

Definition of the model

With polynomial linear regression :

$$h_\theta(X) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_1^2 + \theta_4 X_2^2 + \theta_5 X_1 X_2 + \ldots$$

Mean square error function

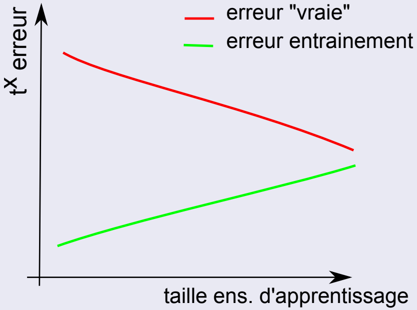
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

Linear regression is simple, and fun, but...

Errors of models

Relation between errors

- Learning error : error rate on the training set
- "True" error : error rate on the all possible examples



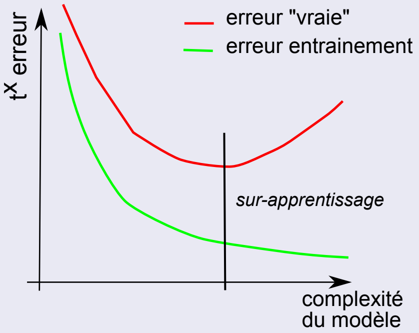
Overfitting

Too much learning on training

Over-fitting of the model on the training set

$\Rightarrow$ Loss of generalization capacity

$\approx$ Learning "by heart"



Complexity metrics such as the polynomial degree

Bias-variance relation

Suppose a function f to learn such that $y_i = f(x_i) + \epsilon_i$,
where ϵ_i is the noise a null mean, and variance σ^2.

$\{(x_i, y_i)\}$ a training set, h a learnt model, and $\bar{h}$ the "average" model learn on all possible sets.

The variance of a model can be decomposed (see proof) by :

$$E[(h(x_i) - y_i)^2] = E[(h(x_i) - f(x_i))^2] + E[(f(x_i) - y_i)^2]$$

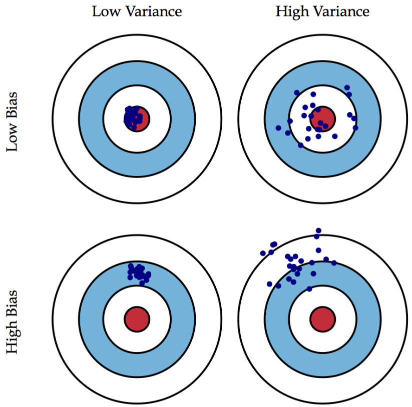
Relation biais/variance

$E[(h(x_i) - y_i)^2] =$

$$E[(\bar{h}(x_i) - f(x_i))^2] + E[(h(x_i) - \bar{h}(x_i))^2] + E[(f(x_i) - y_i)^2]$$

$$\text{Variance of model} = \text{Bias}^2 + \text{Variance} + \text{Variance of Noice}$$

Overfitting : bias-variance tradeoff



- Error from the **bias** : difference between predictions and true values
- Erreur from the **variance** : variability of the prediction for one given data x

Source Scott Fortmann-Roe :
`http://scott.fortmann-roe.com/docs/BiasVariance.html`

Goodness of fit : Coefficient of determination

Definition

$$R^2 = 1 - \frac{\text{Variance of the residus}}{\text{Variance of the data}}$$

with residus : $r_i = h_\beta(x_i) - y_i$

The R^2 is the part of variance of f explained by the model h

- $R^2 \leq 1$, but R^2 can be negative
- $R^2 = 0$ when the model h is equal to mean value, *i.e.* when $h(x) = E[y_i]$.
- $R^2 < 0$ when the model h is worst than $h(x) = E[y_i]$
- $R^2 = 1$ when the model h is perfect, $h(x) = f(x)$ on the set to estimate R^2
- In general, but depending on the context, a relevant R^2 is above 0.8.

Goodness of fit : Coefficient of determination

Definition

$$R^2 = 1 - \frac{\text{Variance of the residus}}{\text{Variance of the data}}$$

with residus : $r_i = h_\theta(x_i) - y_i$

Evaluation of a model quality

Technique

Partitionning the set into :

- **Training** set ($\approx 70\%$)
- **Test** set, independent one ($\approx 30\%$)

The error rate can be estimated without bias on the test set.

Drawback

- An initial large set is requiered
- Dilemma :
  - The larger the test set, the better the estimation is
  - The larger the training set, the better the model is

Resampling method

Allow to estimate the generalization error

K-folds cross-validation

```
Partition randomly the set into $K$ blocks
For each blok $k$,
    Design a model using the $k-1$ other training blocks
    Compute the test error $e_k$ on the block $k$
Compute the mean of errors $e_k$
```

Other techniques :

- Leave-one-out ($K = n$)
- Bootstrap, bagging, etc.

Exercice

See example in scikit-learn "Underfitting vs. Overfitting" :

Polynomial regression of $f(x) = \cos(\frac{3\pi}{2}x)$ with polynomial of degree 1, 4, and 10

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

polynomial_features = PolynomialFeatures(degree=degrees[i],
                 include_bias=False)
linear_regression = LinearRegression()
pipeline = Pipeline([
   ("polynomial_features", polynomial_features),
   ("linear_regression", linear_regression) ])

pipeline.fit(X[:, np.newaxis], y)
```

Sparse approach : regulization methods

Objective function (nota : penalty approach)

$$J(\theta) = L(\theta) + \Omega(\theta)$$

- $L(\theta)$: **training error**,
  distance between data and prediction of the model
- $\Omega(\theta)$: **regularization**, cost of the model complexity

Goal of sparse mode

Reduce the complexity of the model in order to reduce the variance of prediction, and to reduce the generalization error.

Regularization : model complexity metrics

- Norme L_2 : $\Omega(\omega) = \lambda \|\omega\|^2 = \lambda \sum_{i=1}^{n} \omega_i^2$
- Norme L_1 : $\Omega(\omega) = \lambda \|\omega\|_1 = \lambda \sum_{i=1}^{n} |\omega_i|$

Regularization methods

Ridge :

$$J(\theta) = L(\theta) + \lambda \sum_{i=1} \theta_j^2$$

Lasso :

$$J(\theta) = L(\theta) + \lambda \sum_{i=1} |\theta_j|$$

ElasticNet :

$$J(\theta) = L(\theta) + \lambda \sum_{i=1} ((1-\alpha)\theta_j^2 + \alpha|\theta_j|)$$

Warnings : parameters tuning
(use cross-validation to select relevant parameter values)

See also : forward/backward selection, LARS, dropout, etc.
Indeed, the selection of a model is also an optimization problem that combined combinatorial, and numerical optimization !...

Binary classifier

Goal

Find a function : $f(x) \in \{0, 1\}$

Approach

Transform the model h into a binary response :

$$r(x) = \begin{cases} 0 & \text{si } h(x) < 0.5 \\ 1 & \text{si } h(x) \geqslant 0.5 \end{cases}$$

The model h is interpreted as probability function :
$h(x) \in [0, 1]$, and $h(x) = Pr(y = 1|x)$.

Sigmoid fonctions

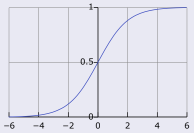
Transform a real number from $\mathbb{R}$ into $[0, 1]$ using ;

- hyperbolic tangent, inverse of normal density, logistic function

Logistic regression

Model : linear model composed with logistic function

$h_\theta(X) = g(\theta_0 + \theta_1 X_1 + \theta_2 X_2 \ \ldots + \theta_n X_n)$ with g logistic function



Loss function : cross-entropy

$j(h, y) = -\Pr(y = 1)\log\Pr(h = 1) - \Pr(y = 0)\log\Pr(h = 0)$
which gives :
$j(h(x), y) = -y\log(h(x)) - (1 - y)\log(1 - h(x))$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} j(h_\theta(x_i), y_i)$$

Multiclass case

Goal

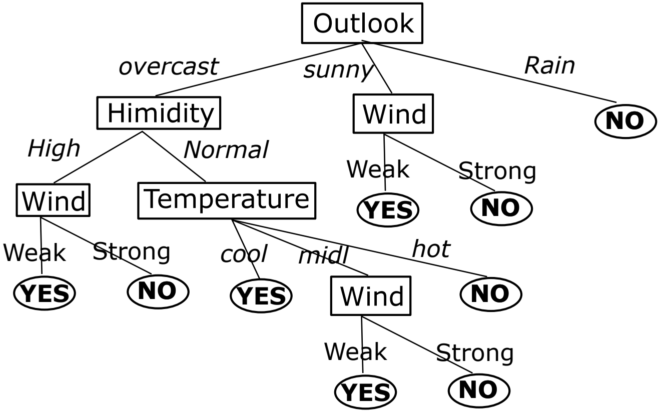
Find a function : $f(x) \in \{0, 1, \ldots, k\}$

- For each class c, we build a binary classifier $h^{(c)}$ which measures the probability $y \in c$ (and $y \notin c$)
- The predicted class is the class with the highest probability :

$$y^{\text{pred}} = \text{argmax}_{c \in C}\ h^{(c)}(x)$$

Decision tree classifier

One of the most explainable ML model



if Outlook = "overcast" and Humidity =... then playball = Yes

Compact way to represent a set of inference relations

Learning algorithm

Learning with decision tree

Design a tree :

- Nodes : **select** one attribute (decision variable X_j) as label, the edges are labelled by the value of the attributes
- Leaves : **cut** the tree with target attribute y

Targeted quality of the tree

- Low error rate
- Low generalization error
- Small tree in order to be explainable
- etc.

Learning algos : ID3, C4.5, CART, CHAID, evolutionary algo., etc.

One classical learning approach

Top-down approach (binary tree)

```
function buildTree($X$)
if all examples of $X$ are in the same class, or stop($X$) then
    Create a leaf with the label of examples
else
    Select one attribute to create a node
    Positive, and negative values in $X$ create two sets $X_0$ et $X_1$
    buildTree($X_0$)
    buildTree($X_1$)
end if
```

One classical learning approach

Greedy top-down approach

For each node,

- Select the **best** attribute according to a metric on X

Recursive call until :

- All examples are in the same class
- A new partitioning does not improve the prediction quality

- Top-down : from root node
- Greedy approach : best "local" choice without backtrack
  Beware of local optima...

Pros, and cons decision tree

Pros

- Easy to understand : explainable IA
- White-box model (neural net. is black-box)
- Data preparation is minimal : no scaling, etc.
- Numerical, and categorial data are possible
- Robust to outliers

Cons

- Learning a decision tree is NP-complet
- Greedy approach is sub-optimal
- Complex tree, overfitting
- Bias toward small tree, attribute with more values, etc.
- Interaction between is difficult to detect
- Problems are difficult to learn with tree (xor, parity, multiplexer)

ID3 (Iterative Dichotomiser 3)

Ross Quinlan, University of New South Wales, Australia, 1986

Top-down greedy algorithm
based on information gain

Principle

1. Compute entropy of all attributes X_j using training set X
2. Select the attribute with maximum information gain
3. Partitionning the set X
4. Recursive call on each subset of attribute values $X_0, X_1, \ldots$

Entropy metric

Entropy H

Quantity of information (disorder) of a set

$$H(X) = -\sum_{s \in D_X} p(s) \log_2 p(s)$$

- X : data set
- D_X : set of classes y of X
- $p(s)$: proportion of class $s \in D_X$ in X

When $H(X) = 0$, X is perfectly classified.

Information Gain

Mutual Information (cross-entropy)

Measure the joint information of 2 random variables (information one variable from the other one)

$$I(S,T) = -\sum_{s \in D_S, t \in D_T} p(s,t) \log_2 \frac{p(s,t)}{p(s)p(t)}$$

Information gain (- mutual information)

Measure the entropy difference before, and after the partitionning

$$IG(S,T) = -I(S,T) = H(S) - \sum_t p(S_t) H(S_t)$$

- $T = \{S_1, \dots\}$ partitionning of $S = \cup_t S_t$
- $p(S_t) = \sharp S_t / \sharp S$
- $H(S)$, $H(S_t)$: entropies of S, and S_t

Pseudo code

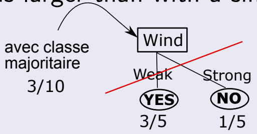
```
ID3(examples, target, attributes) :
    if attributes is empty then
        return a leaf the most frequent label
    elseif all examples are positive (resp. negatives) then
        return a leaf positive label (resp. negative)
    else
        $A \leftarrow$ attribute the highest information gain
        create a node with label $A$
        for each values $v_i$ of $A$
                add a branch $v_i$ to the node
                ID3(examples($A = v_i$), target, attributs $-A$)
```

C4.5 algorithm

Ross Quinlan, 1993

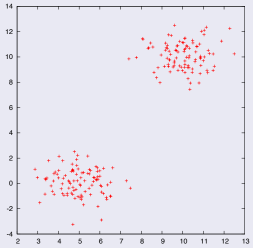
Improvement of ID3

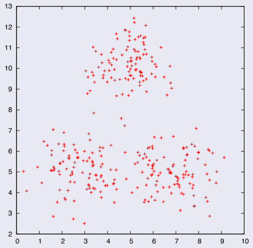
- Use the information gain ratio instead of IG :
  $IG(S, T)$ is bias toward attributes vers a larger number of values
  $ratioIG(S, T) = IG(S, T)/H(T)$
- "null" values are possible :
  Example are ignored when compute the node
- Can handle attribute with real number :
  Discretization with $P(A < a_i)$
  for all possible values of A, IG computation
- Pruning to reduce the tree size :
  Bottom-up Technique : final branches are pruned
  when the error rate is larger than with a simple leaf



Clustering : graphical examples

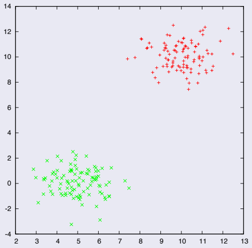
In decision / variable / attribute space :

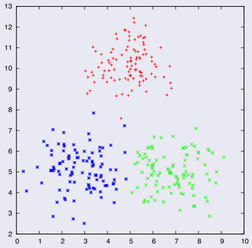




Clustering : graphical examples

In decision / variable / attribute space :





Clustering : definition

Partitionning

input :

Set of n points / examples / observations

$$E = \{e_1, e_2, \dots, e_n\}$$

output :

Partition of E

$$P = \{P_1, P_2, \dots, P_k\}$$

equivalent of coloring function $c : E \rightarrow \{1, \dots, k\}$

Clustering : definition

Partitionning

input :

Set of n points / examples / observations

$$E = \{e_1, e_2, \ldots, e_n\}$$

output :

Partition of E

$$P = \{P_1, P_2, \ldots, P_k\}$$

equivalent of coloring function $c : E \rightarrow \{1, \ldots, k\}$

How many possible partitions with k clusters ?

Clustering : definition

Partitionning

input :

Set of n points / examples / observations

$$E = \{e_1, e_2, \ldots, e_n\}$$

output :

Partition of E

$$P = \{P_1, P_2, \ldots, P_k\}$$

equivalent of coloring function $c : E \rightarrow \{1, \ldots, k\}$

How many possible partitions with k clusters ?

$$k^n / k!$$

Huge even for $n = 100$*, and* $k = 2$*, how to select one ?...*

Associated optimization problem

Optimisation problem

Criterium quality of the partition :

$U : \mathcal{P}(E) \to \mathbb{R}$

But without using labels (non-supervised learning) !

Find a good partition is equivalent to maximize a criterium quality :

$\text{argmax}_{P \in \mathcal{P}(E)} U(P)$

Use an optimization method (local, greedy, etc.)

Quality criterium

Additive expression

Usually, the criterium is additive on clusters :

$$U(P) = \sum_{i \in 1}^{k} w(P_i)$$

where w is quality metric of one cluster

Examples

Sum of square **distance** between points of the cluster :

$$w(P_i) = \sum_{x \in P_i} \sum_{y \in P_i} d^2(x, y)$$

Probability of observation of points in the cluster :

$$w(P_i) = \prod_{x \in P_i} Pr(x|\theta_i)$$

Clustering algorithms

Differents approaches :

- Hierarchical partitionning :
  Agglomeration (or division) of clusters according to criterium (distance, etc.)
  Dendrogramm
- Centroid partitionning :
  Use the center of cluster
  k-means (cf. after)
- Partitionning based on probability distribution
  A cluster is encoded by probability (parametric) distribution
  Algorithm E-M, Gaussian mixture models
- Partitionning based on density :
  According to the local density of points, cluster growth

The famous k-means method : Pseudo-code

k-means

```
Select (randomly) $k$ centers $\mu_1^{(1)}, \ldots, \mu_k^{(1)}$
repeat
    Set the examples to each center according to distance :
        $P_i^{(t)} = \{e_j \ : \ d(e_j, \mu_i^{(t)}) \leq d(e_j, \mu_a^{(t)}) \ \forall a = 1..k\}$
    Update the center of each cluster (mean of clusters) :
        $\mu_i^{(t+1)} = \frac{1}{\sharp P_i^{(t)}} \sum_{e_j \in P_i^{(t)}} e_j$
until no more modification (convergence)
```

Comments :

The algorithm is based on parameter k, and a distance metric

k-means is a local algorithm such that $U(P^{(t+1)}) < U(P^{(t+1)})$

Advantageous / drawbacks

Advantageous

- Easy to interpret
- Easy to code
- Polynomial complexity

Drawbacks

- Parameter k of the number of the clusters
- Shape of cluster are supposed to be "spherical"
- Clusters are supposed separable

Ensemble methods

Principles

Instead of one learner, use a team of learner !

Unity is strength

... if the team learns well together....

example

Majority voting :

A binary "stupid" predictor with error rate 0.4

What is the error rate of n independent "stupid" predictors with a majority voting mechanism ?

Random forest

- Efficient algorithm
- "easy" to train, parallelisation
- Less easy interpretable than decision tree,
  but better prediction performance
- Ensemble method

Breiman, Leo (2001). "Random Forests". Machine Learning 45 (1) : 5-32.

Basis idea

- Ensemble method :
  use a team of "simple" predictors
- Simple predictor :
  decision tree
- Prediction :
  - Majority voting for clustering
  - Mean of predictions for regression
- Key of the learning mechanism :
  design a set of heterogeneous trees

Principle

random forest = tree bagging + features sampling

Decision tree are too much dependent of the training set.

To build each tree, use a fragmented view of the problem :

- Draw (with replacement) a sub-set of examples
  $\Rightarrow$ bagging
- Draw (with replacement) a sub-set of features/attributes
  $\Rightarrow$ features sampling

Reduce the variance using a forest

$$V_{forest} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

where :

- σ^2 : variance of one decision tree
- B : number of trees
- ρ correlation between trees

The feature selection could reduce the coefficient ρ
($\sqrt{n}$ features are randomly selected in feature selection step)

Others versions

- Extremely randomized trees :
  For each selected variables, the split is also random (division of attribute values)
- rotation forest :
  Principle component analysis (PCA) before build the tree

Geurts, Pierre and Ernst, Damien and Wehenkel, Louis, Extremely randomized trees, Machine Learning, vol. 6, 1, pp. 3-42, 2006.

Rodriguez JJ1, Kuncheva LI, Alonso CJ, Rotation forest : A new classifier ensemble method, IEEE Trans Pattern Anal Mach Intell. 2006 Oct ;28(10) :1619-30.

Boosting

Bibliography

- adaBoost pour adaptive boosting [Gödel price 2003] :
  Yoav Freund et Robert Schapire, A desicion-theoretic generalization of on-line learning and an application to boosting, Journal of Computer and System Sciences, vol. 55, Num. 1, 1997, p. 119-139
- Gradient boosting :
  J.H. Friedman, Greedy function approximation a gradient boosting machine, Ann. Statist., Vol., Num. 5 (2001), 1189-1232.

Boosting principle

Principle

- Iterative design of meta-algorithm based on trees (or another "weak" learner)
- The errors of the previous iteration are taken into account to build the next one :
  The next weak learner learn the residus of the error of the previous ones
  The weak leaners are "boosted"

Boosting *vs.* random forest

- Random forest :
  Majority voting
  Weak learners are built in parallel
- Boosting :
  Weighted sum of the weak learners
  Weak learners are built iteratively

$$H(x) = \mathrm{sign}(\sum_{i=1}^{B} \alpha_i h_i(x))$$

Adaboost algorithm

```
Initialize the distribution of examples $\forall i = 1, \dots, m,\ D_1(i) = \frac{1}{m}$

for $t = 1, \dots, T$ do
    $h_t = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\ \epsilon_t$
            with $\epsilon_t$ the weighted classification error (residues) :
            $\epsilon_t = \sum_{i=1}^{m} D_t(i).[y_i \neq h(x_i)]$

    Weights of classifier : $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$

    Update weights : $\forall i = 1, \dots, m,\ D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$
                    with $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$
end for

$H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x))$
```

Gradient boosting : Principle

gradient boosting = descente de gradient + boosting

Boosting :
sequential learning : $H(x) = \text{sign}(\sum_{i=1}^{B} \alpha_i h_i(x))$

Gradient descent :
Generalization the loss function :
residues are replaced by the negative gradient of previous ones

See : `http://orbi.ulg.ac.be/bitstream/2268/163521/1/slides.pdf`

Conclusion

There is many techniques, algorithms.
The principle is to use :

- a good model according to data, and available information
- an efficient optimization algorithm to find the parameters of the model according to data, and available information

Have fun with optimization, and machine learning !