Machine Learning: from Theory to Practice

Lecture 5: Trees and ensemble methods

F. d'Alché-Buc and E. Le Pennec
florence.dalche@telecom-paristech.fr

Fall 2016

Outline

Statistical view of machine learning

Reminder

Supervised statistical learning

- $\mathcal{S}_n = \{(x_1, y_1), ..., (x_n, y_n)\}$ is an i.i.d sample of size n, drawn from the joint probability law P(X,Y) fixed but unknown.
- $\mathcal{H}$ a class of functions and a local loss function $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

Solve the following optimization problem:

$$\min_{h \in \mathcal{H}} R(h)$$

$$\text{with } R(h) = \mathbb{E}[\ell(h(x), y)]$$

Example: Prediction loss, $\ell(h(x), y) = 0$ if $h(x) = y$ and 1 otherwise

In practise : minimize $R_n(h)$, the empirical loss, while controlling the complexity of h

Machine Learning

Methodology

- Define
  - a representation space for data
  - a class of functions (a class of hypotheses) where to find the solution
  - a loss function to be minimized
  - an optimization algorithm
  - a model selection method for hyperparameters

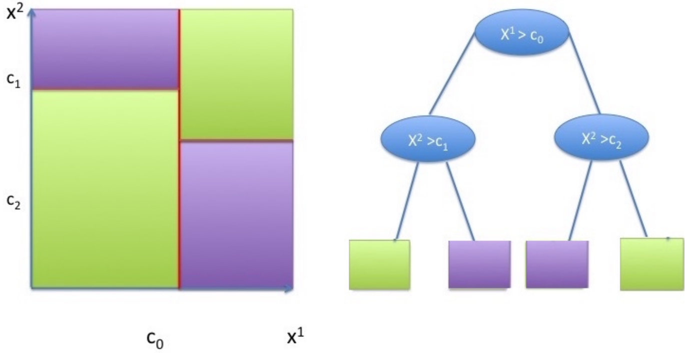
Today's agenda

- Decision and Regression Trees
- Ensemble methods

Outline

Decision trees

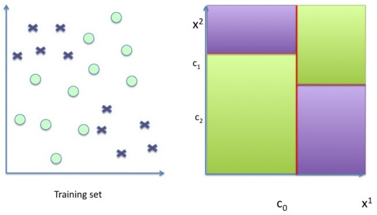
Invented between 1979 and 1983 simultaneously by L. Breiman (CART, applied stats) et col. et R. Quinlan (ID3, a new discipline called Machine Learning)



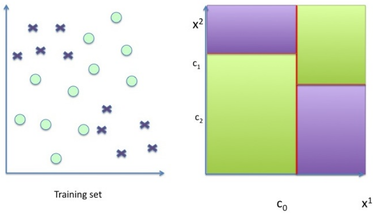
Reference:

Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees. Monterey, CA: Wadsworth Brooks/Cole Advanced Books Software.

Decision trees 1

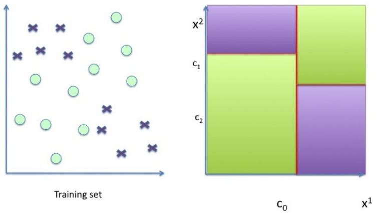


Decision trees 1



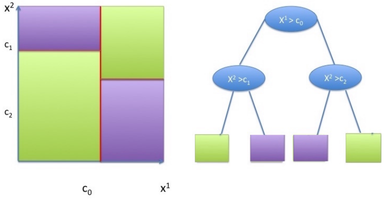
First idea:
Use not one but many linear separators to build non linear decision frontiers
Second idea

Decision trees 1



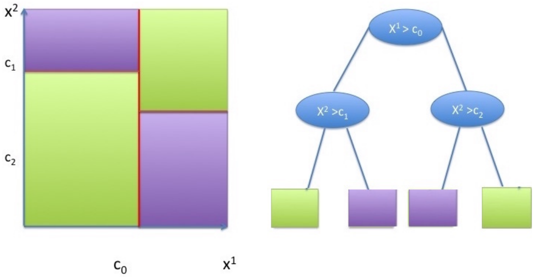
First idea:
Use not one but many linear separators to build non linear decision frontiers
Second idea
Use hyperplane of the form $x^j = c$ to keep an interpretation of the learned function

Decision tree 2



Third idea
The learned function can be represented by a tree structure whose each node is associated to a hyperplane $x^j = \theta_j$ and each leaf to a constant function i.e. a class index.

Decision tree 3



At the end of the learning phase you know every feature involved in the decision function

The tree encodes rules of logic 0^+

if $(x^{j_1} > c_{j_1})$ and $(x^{j_2} \leq c_{j_2})$ and $\ldots$ then x belongs to class k

Linear separator orthogonal to one vector fo the canonical basis

x^j : continuous variable

$$t_{j,c}(\mathbf{x}) = \mathrm{sign}(x^j - c) \tag{1}$$

x^j : categorical variable here with 2 values $\{v_1^j, \ldots, v_2^j\}$:

$$t(x; v_j) = \mathbb{I}(x^j = v_j) \tag{2}$$

A new class of constant piece-wise functions

We consider a new class of functions :

Definition

$$\forall x \in \mathbb{R}^p, h^{tree}(x) = \sum_{j=1}^{M} \alpha_m \mathbb{I}_{\mathcal{C}_m}(x),$$

where $\mathcal{C}_1 \cup \ldots \mathcal{C}_M$ is a partition of $\mathbb{R}^p$ and each $\mathcal{C}_m$ is defined by a subset of hyperplanes orthogonal to the canonical basis.

Decision and regression trees

Let $x \in \mathbb{R}^p$. Denote $m(x) \in \{1, \dots, M\}$ such that $x \in \mathcal{C}_{m(x)}$.

- Classification, by majority vote in $\mathcal{C}_m(x)$:
$$h_n(x) = \alpha_{m(x)} = \arg \max_{k=1,\dots,K} \sum_{x_i \in \mathcal{C}_m(x)} \mathbb{I}(y_i = k)$$
- Regression, by empirical mean in $\mathcal{C}_m(x)$:
$$f_n(x) = \alpha_{m(x)} = \frac{1}{|\mathcal{C}_m(x)|} \sum_{x_i \in \mathcal{C}_m(x)} y_i$$

Goal: learn $\mathcal{C}_1 \cup \dots \mathcal{C}_M$.

A local and greedy learning algorithm

(Binary tree)

1. Let $\mathcal{S}$ be the current training set
2. Let us build a root node
3. Find the best split $t(\mathbf{x})$ to apply to the current training set $\mathcal{S}$ such that the local loss $L(t, \mathcal{S})$ be minimal
4. Attach the chosen split to the current node and separate the current training dataset into two subsets $\mathcal{S}_d$ et $\mathcal{S}_g$ with the help of the split
5. Build a right node and a left node.
6. Measure the stopping criterion on the right side and if checked, then the right node becomes a leaf , otherwise go to 3 with $\mathcal{S}_d$ as the current training subset
7. Measure the stopping criterion on the left side and if checked, then the left node becomes a leaf , otherwise go to 3 with $\mathcal{S}_g$ as the current training subset

Local loss

Let $\mathcal{S}$ be the current training set and $t_{j,\tau}$ a binary split.
Note

- $\mathcal{D}(\mathcal{S}, j, \tau) = \{(\mathbf{x}, y) \in \mathcal{S}, t_{j,\tau}(\mathbf{x}) > 0\}$
- $\mathcal{G}(\mathcal{S}, j, \tau) = \{(\mathbf{x}, y) \in \mathcal{S}, t_{j,\tau}(\mathbf{x}) \leq 0\}$.

$\tau_1, \ldots, \tau_C$ can be chosen regularly spaced or using an histogram.

Local loss

Among all the parameters $(j, \tau) \in \{1, \dots, p\} \times \{\tau_1, \dots, \tau_C\}$, we search for $\hat{j}$ and $\hat{\tau}$ that minimize:

Local loss function

$$\begin{aligned} L(t_{j,\tau}, \mathcal{S}) &= \frac{n_d}{n} H(\mathcal{D}(\mathcal{S}, j, \tau)) + \frac{n_g}{n} H(\mathcal{G}(\mathcal{S}, j, \tau)) \\ n_d &= |\mathcal{D}(\mathcal{S}, j, \tau)| \\ n_g &= |\mathcal{G}(\mathcal{S}, j, \tau)| \end{aligned}$$

where $H(\mathcal{S})$ measures the impurity of a subset $\mathcal{S}$

Impurity criteria for supervised classification 1/2

For a given dataset $\mathcal{S}$ of n labeled data and for each class k:

$$p_k(\mathcal{S}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(y_i = k)$$

here are the main criteria H that can be used:

Cross entropy

$$H(\mathcal{S}) = -\sum_{k=1}^{K} p_k(\mathcal{S}) \log p_k(\mathcal{S})$$

Impurity criteria for supervised classification 2/2

Cross entropy

$$H(\mathcal{S}) = -\sum_{k=1}^{K} p_k(\mathcal{S}) \log p_k(\mathcal{S})$$

Gini index

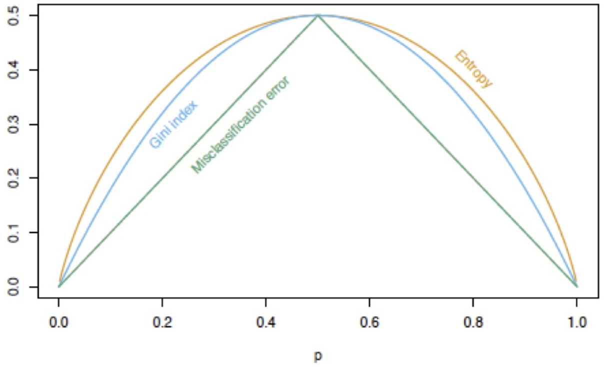
$$H(\mathcal{S}) = \sum_{k=1}^{K} p_k(\mathcal{S})(1 - p_k(\mathcal{S}))$$

Classification error

$$H(\mathcal{S}) = 1 - p_{C(\mathcal{S})},$$

with $C(\mathcal{S})$: majority class in $\mathcal{S}$.

Visualization of impurity criteria



Stopping criteria

Stop the construction if one of the following value is reached

- Maximal Depth
- Maximal number of leaves
- Minimal number of training data in a leaf

NB : If the minimal number of training data in a leaf is equal to 1, it is possible that the tree grows until a perfect classification of the training examples is achieved: **overfitting** !

Some remarks: categorical variables, multi-class classification

- To get a binary tree, if a categorical variable can take K values, it can be transformed into K binary variables.
- The learning algorithm is appropriate to solve binary classification problem as well as multi-class classification problems

Regression trees

We maximize the variance reduction !

$$L(t_{j,\tau}, \mathcal{S}) = VAR_{emp}(\mathcal{S}) - \frac{n_d}{n} VAR_{emp}(\mathcal{D}(j, \tau, \mathcal{S})) - \frac{n_g}{n} VAR_{emp}(\mathcal{G}(j, \tau, \mathcal{S}))$$

$$VAR_{emp}(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{(x_i, y_i) \in \mathcal{S}} (y_i - \bar{y})^2$$

We attempt to get homogeneous outputs !

Model selection 1/2

(1) We focus on the selection of on of the following hyperparameters:

- Maximal Depth
- Maximal number of leaves
- Minimal number of training data in a leaf

typically by $\rightarrow$ **cross-validation**.

Model selection 2/2

(2) **by pruning**
Another dataset, called validation set, is used to re-visit a tree that was grown until covering all the training examples. One only keeps branches that bring a performance improvement on validation set.

Advantages and drawbacks

Advantages

- Nonlinear interpretable decision frontier $\rightarrow$ explainable AI
- Consistency (see Scott and Nowak, IEEE Trans. Inf. Theory, 2006)
- Very efficient computation of images of h_n^{tree} or f_n^{tree} : complexity in time in $(depth)$.
- Works for continuous and categorical features
- Works directly for multi-class classification
- Very flexible

Drawbacks

- Large variance Estimator : a change in the root implies a change in the whole tree
- Learning a tree is a NP-complete problem (Hyafil and Rivest, Information Proc. Letters,1976)
- No global learning algorithm

Extension of tree approaches to other problem

Like k-nearest neighbors, SVM/SVR, trees can be adapted to other machine learning tasks:

- classification with imbalanced datasets
- multiple output prediction
- quantile regression
- ranking
- anomaly detection
- time series modeling

How ?

- Change the local loss (impurity criteria)

Outline

Ensemble methods for classification and regression

- Machine Learning not so "automatic": too many hyperparameters to tune

1. **Committee learning** or **wisdom of the crowd**: better results are obtained by combining the predictions of a set of **diverse** classifiers/regressors
2. **Meta-learning**: a procedure to automatically use a base classifier/regressor even weak to produce a performant classifier/regressor
3. **Ensemble learning**: is a kind of meta-learning, improves upon a single predictor by building an ensemble of predictors (with no hyperparameter)

Ensemble methods at a glance

- 1995: Boosting, Freund and Schapire
- 1996: Bagging, Breiman
- 1999: GradientBoosting, Friedman et al.
- 2001: Random forests, Breiman
- 2006: Extra-trees, Geurts, Ernst, Wehenkel

Ensemble methods and meta-learning

- Improve upon a single predictor by building an ensemble of predictors (with no hyperparameter)
- $\rightarrow$ meta-learning: the parameter of the *meta-learning* algorithm is those of the base learner and the size of the ensemble

Ensemble methods for regression

Let $f_t, t = 1, \ldots, T$ be T different regressors.
Notations:

$$\begin{aligned}
\epsilon_t(x) &= y - f_t(x) \\
MSE(f_t) &= \mathbb{E}[\epsilon_t(x)^2] \\
f_{ens}(x) &= \frac{1}{T}\sum_t f_t(x) \\
&= y - \frac{1}{T}\sum_t \epsilon_t(x).
\end{aligned}$$

Encourage the diversity of base predictors

$$MSE(f_{ens}) = \mathbb{E}[(y - f_{ens}(x))^2] = \frac{1}{T^2}\mathbb{E}[(\sum_t \epsilon_t(x))^2]$$

Now, if ϵ_t are mutually independent with zero mean, then we have:

$$MSE(f_{ens}) = \frac{1}{T^2}\mathbb{E}[\sum_t \epsilon_t(x)^2]$$

The more diverse are the classifiers, the more we reduce the mean square error !

Ensemble methods

- **Encourage the diversity of base predictors by:**
  - using bootstrap samples (Bagging and Random forests)
  - using randomized predictors (ex: Random forests)
  - using weighted version of the current sample (Boosting) with weights dependent on the previous predictor (adaptive sampling)

Outline

Decomposition bias/variance in regression

Given x,

$$E_S[E_{y|x}[(y - f_S(x))^2]] = noise(x) + bias^2(x) + \text{variance}(x) \quad (3)$$

noise(x): $E_{y|x}[(y - E[y])^2]$:
quantifies the error made by the Bayes model ($E[y|x]$)
$bias^2(x) = (E[y|x] - E_S[f_S(x)])^2$
measures the difference between minimal error (Bayes error) and the average model
$variance(x) = E_S[(f_S(x) - E_S[f_S(x)])^2]$
measures how much $h_S(x)$ varies from one training set to another

Introduction to bagging (regression) - 1

Assume we can generate several training samples $\mathcal{S}_1, \dots, \mathcal{S}_T$ from P(x,y).
A first algorithm:

- draw T training samples $\{\mathcal{S}_1, \dots, \mathcal{S}_T\}$
- learn a model $f_t \in \mathcal{F}$ from each training sample $\mathcal{S}_t; t = 1, \dots, T$
- compute the average model : $f_{ens}(x) = \frac{1}{T} \sum_{t=1}^{T} f_t(x)$

Introduction to bagging - 2

The bias remains the same:
$bias(x) = E_{\mathcal{S}_1,\dots,\mathcal{S}_T}[f_{ens}(x)] = \frac{1}{T}\sum_t E_{\mathcal{S}_t}[f_t(x)] = E_{\mathcal{S}}[f_t(x)]$
The variance is divided by T:
$E_{\mathcal{S}_1,\dots,\mathcal{S}_T}[(f_{ens}(x) - E_{\mathcal{S}_1,\dots,\mathcal{S}_T}[f_{ens}(x)])^2] = \frac{1}{T} E_{\mathcal{S}}[(f_S(x) - E_{\mathcal{S}}[f_{\mathcal{S}}(x))^2]$

Bagging (Breiman 1996)

In practice, we do not know P(x,y) and we have only one training sample $\mathcal{S}$.
Bagging = Bootstrap Aggregating:

- draw T bootstrap samples $\{\mathcal{B}_1 \ldots, \mathcal{B}_T\}$ from $\mathcal{S}$
- Learn a model f_t for each $\mathcal{B}_t$
- Build the average model: $f_{bag}(x) = \frac{1}{T} \sum_t f_t(x)$

Bootstrap

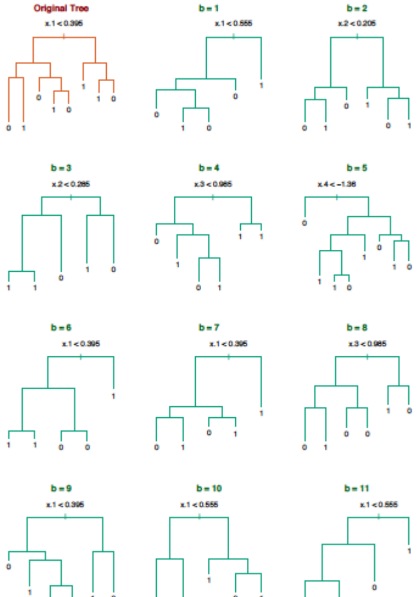
Efron 1977.

- Bootstrap is a robust estimation method
- A bootstrap sample drawn from $\mathcal{S}$ is a sample of the same size obtained by uniformly drawing examples from $\mathcal{S}$ with replacing
- If we construct B bootstrap samples and apply an estimation procedure T on$\mathcal{S}_1, \ldots, \mathcal{S}_B$ then we can compute:
  - A bootstrap average: $\bar{T} = \frac{1}{B}\sum_{b=1}^{B} T(\mathcal{S}_b)$
  - A bootstrap variance : $VAR_{boot}(T) = \frac{1}{B}\sum_{b=1}^{B}(T(\mathcal{S}_b) - \bar{T})^2$

The bootstrap variance is often used to choose among several hyperparameteres which gives the most stable estimator

Example of bagged trees

[Book: The elements of statistical learning, Hastie, Tibshirani,



Bagging in practise

- Variance is reduced but the bias can increase a bit (the effective size of a bootstrap sample is 30% smaller than the original training set $\mathcal{S}$
- The obtained model is however more complex than a single model
- Bagging works for unstable predictors (neural nets, trees)
- In supervised classification, bagging a good classifier usually makes it better but bagging a bad classifier can make it worse

Outline

Other ensemble methods

- Perturbe and combine algorithms
  - Perturbe the base predictor
  - Combine the perturbed predictors

REFS: Random forests: Breiman 2001
Geurts, Ernst, Wehenkel, Extra-trees, 2006

Random forests: Breiman 2001

Random forests algorithm

- INPUT: candidate feature F, $\mathcal{S}_{train}$
- for t=1 to T
  - $\mathcal{B}_{train}^{(t)} \leftarrow$ a bootstrap sample from $\mathcal{S}_{train}$
  - $h_{tree}^{(t)} \leftarrow$ randomized decision tree learned from $\mathcal{B}_{train}^{(t)}$
- OUTPUT: $H^T = \frac{1}{T} \sum_t h_{tree}^{(t)}$

Learning a single randomized tree

- To select a split at a node:
    - $R_f(F) \leftarrow$ randomly select (without replacement) f feature splits from F with $f << |F|$
    - Choose the best split in $R_f(F)$ (consider the different cut-points)
- Do not prune this tree

Randomized tree:

Learning a single randomized tree:

- To select a split at a node:
  - $R_K(F) \leftarrow$ randomly select (without replacement) f feature splits from F with $f << |F|$
  - Choose the best split in $R_f(F)$ (consider the different cut-points)
- Do not prune this tree

Extra-trees: Geurts et al. 2006

Extra-trees

- INPUT: candidate feature splits $F = \{1, \ldots, p\}$,S_{train}
- for t=1 to T
  - Always use $\mathcal{S}_{train}$
  - $h_{tree}^{(t)}$: extremly randomized decision tree learned from $\mathcal{S}_{train}$
- OUTPUT: $H^T = \frac{1}{T} \sum_t h_{tree}^{(t)}$

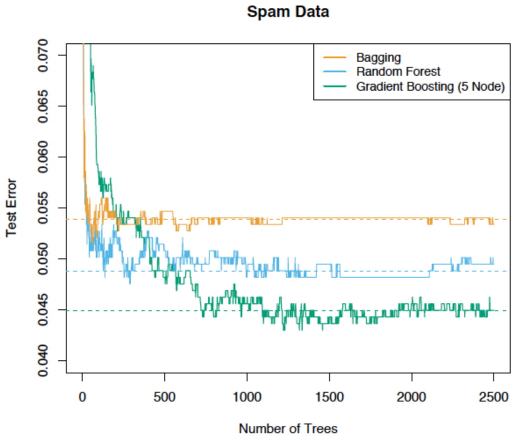
Learning a single randomized tree in extra-trees:

- To select a split at a node:
  - randomly select (without replacement) K feature splits from F with $K << |F|$
  - Draw K splits using the procedure Pick-a-random-split($\mathcal{S}$,i):
    - let a^i_{max} and a^i_{min} denote the maximal and minimal value of x_i in $\mathcal{S}$
    - Draw uniformly a cut-point a_c in $[a^i_{max}, a^i_{min}]$
- Choose the best split among the K previous splits

Do not prune this tree

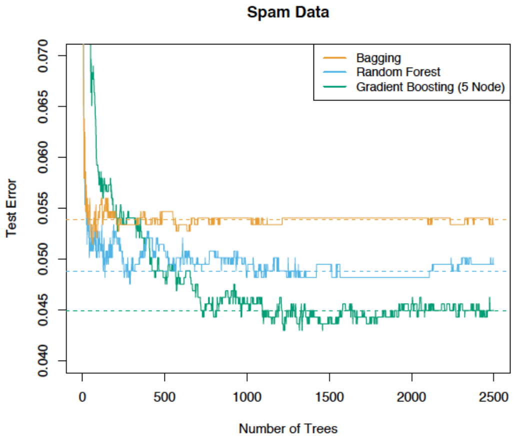
Random forest

Example of decision frontier:



Comparison (just an example)

[Book: The elements of statistical learning, Hastie, Tibshirani, Friedman, 2001]



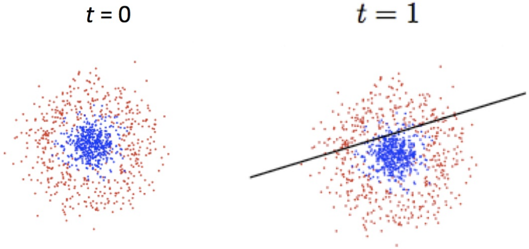
Outline

A preliminary question

- **Is it possible to "boost" a weak learner into a strong learner ?** Michael Kearns
- Yoav Freund and Rob Schapire proposed an iterative scheme, called, Adaboost to solve this problem
    - Idea: train a sequence of learners on weighted datasets with weights depending on the loss obtained so far.
    - Freund and Schapire received the Godel prize in 2003 for their work on AdaBoost.

Boosting a linear classifier

$H_1(x) = h_1(x)$
Binary Classifier: $F_1(x) = \text{sign}(H_1(x))$
Here: h_1: linear classifier
Training error$=R_n$

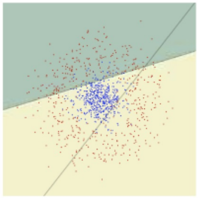


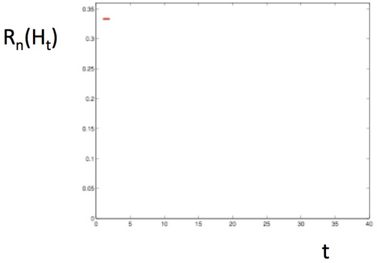
Source Jiri Matas (Oxford U.)

Boosting a linear classifier

$H_2(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x)$
Binary Classifier: $F_2(x) = \text{sign}(H_2(x))$

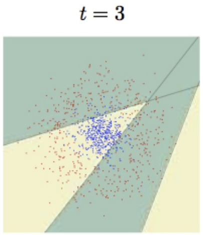
$t = 2$

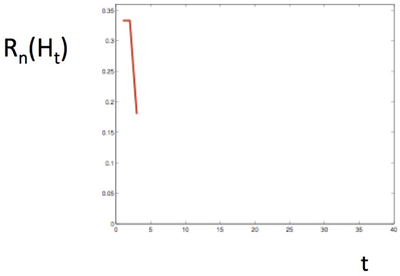




Source Jiri Matas (Oxford U.)

Boosting a linear classifier



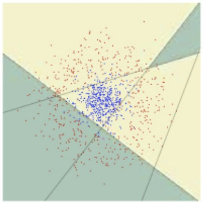


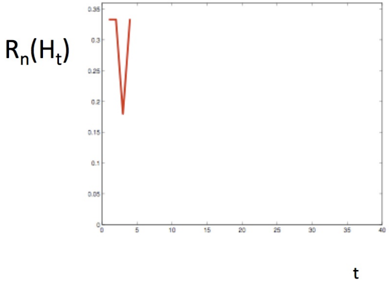
Source Jiri

Matas (Oxford U.)

Boosting a linear classifier

$t = 4$

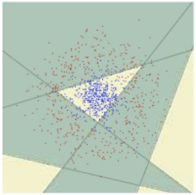


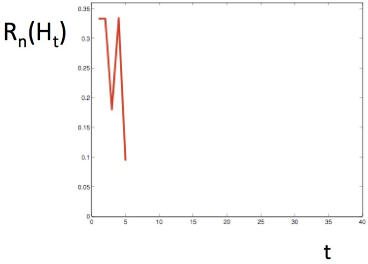


Source Jiri Matas (Oxford U.)

Boosting a linear classifier

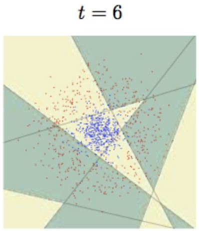
$t = 5$

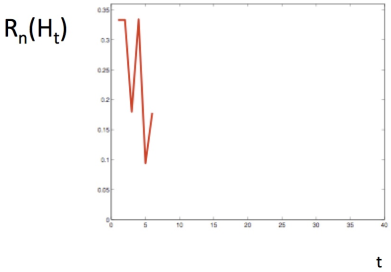




Source Jiri Matas (Oxford U.)

Boosting a linear classifier

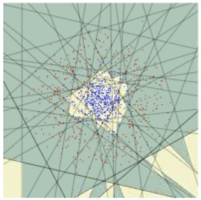


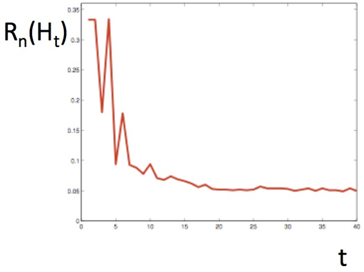


Source Jiri Matas (Oxford U.)

Boosting a linear classifier

$t = 40$





Source Jiri Matas (Oxford U.)

Weak classifier

Definition: weak classifier

A classifier whose average training error is no more than 0.5

NB : it means that we do not need to have a deep architecture as the base classifier (a "short" tree will fit for instance, a linear classifier will be perfect and so on...)

Adaboost idea

1. $\mathcal{H}$: a chosen class of "weak" binary classifiers, $\mathcal{A}$: a learning algorithm for $\mathcal{H}$

```
- Set $w_1(i) = 1/n$; $H_0 = 0$
- For $t = 1$ to $T$
    - $h_t = \arg\min_{h \in \mathcal{H}} \epsilon_t(h)$
    - with $\epsilon_t(h) = \mathbb{P}_{i \sim \mathbf{w}_t}[h(x_i) \neq y_i]$
    - Choose $\alpha_t$
    - Choose $w_{t+1}$
    - $H_t = H_{t-1} + \alpha_t h_t$
- Output $F_T = \text{sign}(H_t)$
```

$\mathcal{H}$: a chosen class of "weak" binary classifiers

```
• Set $w_1(i) = 1/n$;  $H_0 = 0$
• For $t = 1$ to $T$
    • $h_t = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \epsilon_t(h)$
    • With $\epsilon_t(h) = \mathbb{P}_{i \sim \mathbf{w}_t}[h(x_i) \neq y_i]$
    • $\epsilon_t = \epsilon_t(h_t)$
    • $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
    • let $w_{t+1,i} = \frac{w_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_{t+1}}$ where $Z_{t+1}$ is a
      renormalization constant such that $\sum_{i=1}^{n} w_{t+1,i} = 1$
• $H_t = H_{t-1} + \alpha_t h_t$
Output $F_T = \mathrm{sign}(H_t)$
```

What weight to choose ?

With the chosen definition, we have:

$$\begin{aligned} w_{t+1,i} &= \frac{w_{t,i} e^{-\alpha_t y_i h_t(x_i)}}{Z_t} \\ &= \frac{w_{t-1,i} e^{-\alpha_{t-1} y_i h_{t-1}(x_i)} e^{-\alpha_t y_i h_t(x_i)}}{Z_{t-1} Z_t} \\ &= \frac{e^{-y_i \sum_{s=1}^{t} \alpha_s h_s(x_i)}}{n \prod_{s=1}^{t} Z_s} \\ &= \frac{e^{-y_i H_t(x_i)}}{n \prod_{s=1}^{t} Z_s} \end{aligned}$$

You see the weights encourage to correct examples badly classified by the whole combination H_t

First of all let us study Z_t

$$\begin{aligned}
Z_t &= \sum_{i=1}^{n} w_t(i) e^{-\alpha_t y_i h_t(x_i)} \\
&= \sum_{i=1}^{n} w_t(i) e^{-\alpha_t y_i h_t(x_i)} \\
&= \sum_{i: y_i h_t(x_i) = +1} w_t(i) e^{-\alpha_t} + \sum_{i: y_i h_t(x_i) = -1} w_t(i) e^{\alpha_t} \\
&= (1-\epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t} \\
&= (1-\epsilon_t)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon_t \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \\
&= \ldots \\
&= 2\sqrt{\epsilon_t(1-\epsilon_t)}
\end{aligned}$$

Bound on the empirical error

Theorem

The empirical error of the classifier returned by Adaboost at time T verifies:

$$R_n(F_T) \leq e^{-2\sum_{t=1}^{T}(\frac{1}{2}-\epsilon_t)^2}.$$

Furthermore, if for all $t \in [1, T]$, $\gamma \leq (\frac{1}{2} - \epsilon_t)$, then

$$R_n(F_T) \leq e^{-2\gamma^2 T}.$$

Bound on the empirical error: proof

For all $u \in \mathbb{R}$, we have $1_{u \leq 0} \leq \exp(-u)$.
Then

$$\begin{aligned}, R_n(F_T) &= \frac{1}{n}\sum_{i=1}^{n} 1_{y_i F_T(x_i) \leq 0} \\ &\leq \frac{1}{n}\sum_{i=1}^{n} \exp(-y_i F_T(x_i)) = \frac{1}{n}\sum_{i=1}^{n} [n \prod_{t=1}^{T} Z_t] w_{t+1,i} = \prod_{t=1}^{T} Z_t\end{aligned}$$

Bound on the empirical error: proof ctd'

We can now express $\prod Z_t$ in terms of ϵ_t:

$$\begin{aligned}
\prod_{t=1}^{T} Z_t &= \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \\
&= \text{by remarkable identity} \\
&= \prod_{t=1}^{T} \sqrt{1-4(1/2-\epsilon_t)^2} \\
&\leq \prod_{t} e^{-2(1/2-\epsilon_t)^2} = e^{-2\sum_{t=1}^{T}(1/2-\epsilon_t)^2}
\end{aligned}$$

using the identity $1-u \leq \exp(-u)$.

Choice of α_t

The proof reveals serval interesting properties:

1. α_t is chosen to minimize $\prod_t Z_t = g(\alpha)$ with $g(\alpha) = (1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha}$
   - $g'(\alpha) = -(1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha}$
   - $g'(\alpha) = 0$ iff $(1-\epsilon_t)e^{-\alpha} = \epsilon_t e^{\alpha}$ iff $\alpha = 1/2 \log \frac{1-\epsilon_t}{\epsilon_t}$
2. The equality $(1-\epsilon_t)e^{-\alpha} = \epsilon_t e^{\alpha}$ means that Adaboost assigns at each time t the same distribution mass to correctly classified examples and incorrectly classified ones. However there is no contradiction because the number of incorrectly examples decreases.

Adaboost with scikitlearn

```
http://scikit-learn.org/stable/modules/ensemble.htmladaboost
>>> from sklearn.crossvalidation import crossvalscore
>>> from sklearn.datasets import loadiris
>>> from sklearn.ensemble import AdaBoostClassifier
>>> iris = loadiris()
>>> clf = AdaBoostClassifier(nestimators=100)
>>> scores = crossvalscore(clf, iris.data, iris.target)
>>> scores.mean() 0.9...
```

Outline

Boosting as a coordinate descent

At the same time, different groups proved that Adaboost writes as a coordinate descent in the convex hull of $\mathcal{H}$.

- Greedy function approximation, Friedman, 1999.
- MarginBoost and AnyBoost : Mason et al. 1999.

Gradient Boosting

- At each boosting step, one need to solve
$$(h_t, \alpha_t) = \arg\min_{h,\alpha} \sum_{i=1}^{n} \ell(y_i, H_{t-1}(x_i) + \alpha h(x_i)) = L(y, H_{t-1} + \alpha h)$$
- Gradient approximation
$L(y, H_{t-1} + \alpha h) \sim L(y, H_{t-1}) + \alpha \langle \nabla L(H_{t-1}), h \rangle.$
- Gradient boosting: replace the minimization step by a *gradient descent* type step:
    - Choose h_t as the best possible descent direction in $\mathcal{H}$
    - Choose α_t that minimizes $L(y, H + \alpha h_t)$
- Easy if finding the best descent direction is easy!

Gradient boosting and adaboost

Those two algorithms are equivalent!

- Denoting $H_t = \sum_{t'=1}^{t} \alpha_{t'} h_{t'}$,

$$\begin{aligned}\sum_{i=1}^{n} e^{-y_i(H_{t-1}(x_i)+\alpha h(x_i))} &= \sum_{i=1}^{n} e^{-y_i H_{t-1}(x_i)} e^{-\alpha y_i h(x_i)} \\ &= \sum_{i=1}^{n} w_i'(t) e^{-\alpha y_i h(x_i)} \\ &= (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{n} w_i'(t) \ell^{0/1}(y_i, h(x_i)) \\ &\quad + e^{-\alpha} \sum_{i=1}^{n} w_i'(t)\end{aligned}$$

- The minimizer h_t in h is independent of α and is also the minimizer of

$$\sum_{i=1}^{n} w_i'(t) \ell^{0/1}(y_i, h(x_i))$$

Gradient boosting and adaboost

- The optimal α_t is then given by
$$\alpha_t = \frac{1}{2}\log\frac{1-\epsilon'_t}{\epsilon'_t}$$
with $\epsilon'_t = (\sum_{i=1}^n w'_i(t)\ell^{0/1}(y_i, h_t(x_i)))/(\sum_{i=1}^n w'_i(t))$
- One verify then by recursion that
$$w_i(t) = w'_i(t)/(\sum_{i=1}^n w'_i(t))$$
and thus the two procedures are equivalent!

AnyBoost or Forward Stagewise Additive model

- General greedy optimization strategy to obtain a linear combination of *weak* predictor
    - Set $t = 0$ and $H_0 = 0$.
    - For $t = 1$ to T,
        - $(h_t, \alpha_t) = \arg\min_{h,\alpha} \sum_{i=1}^{n} \ell(y_i, H_{t-1}(x_i) + \alpha h(x_i))$
        - $H_t = H_{t-1} + \alpha_t h_t$
    - Output $H_T = \sum_{t=1}^{T} \alpha_t h_t$

Losses in Forward Stagewise Additive Modeling

- AdaBoost with $\ell(y, h) = e^{-yh}$
- LogitBoost with $\ell(y, h) = \log(1 + e^{-yh})$
- L_2Boost with $\ell(y, h) = (y - h)^2$ (Matching pursuit)
- L_1Boost with $\ell(y, h) = |y - h|$
- HuberBoost with
  $\ell(y, h) = |y - h|^2 \mathbf{1}_{|y-h|<\epsilon} + (2\epsilon|y - h| - \epsilon^2)\mathbf{1}_{|y-h|\geq\epsilon}$

  Simple principle but no easy numerical scheme except for AdaBoost and L_2Boost...

Continuous loss functions and gradient boosting

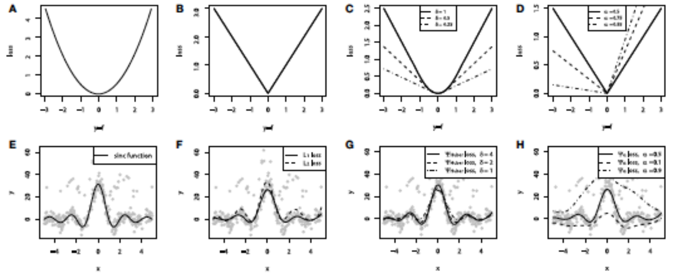


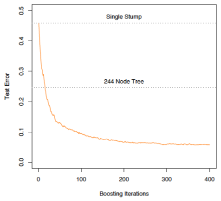
FIGURE 1 | Continuous loss functions: (A) L_2 squared loss function; (B) L_1 absolute loss function; (C) Huber loss function; (D) Quantile loss function. Demonstration of fitting a smooth GBM to a noisy sinc(x) data: **(E)** original sinc(x) function; **(F)** smooth GBM fitted with L_2 and L_1 loss; **(G)** smooth GBM fitted with Huber loss with $\delta = \{4, 2, 1\}$; **(H)** smooth GBM fitted with Quantile loss with $\alpha = \{0.5, 0.1, 0.9\}$.

L_2 Boosting

- Loss function for regression: $\ell(y, h) = (y - h)^2$
- $(h_t, \alpha_t) = \arg\min_{h,\alpha} \sum_{i=1}^{n} (y_i - H_t(x_i) + \alpha h(x_i))^2$

  Fitting the residuals.

Boosting and regularization



- You have to wait a long time to see Boosting overfit. However contrary to first assertions, Adaboost does overfit
- The surrogate loss which is empirically minimized is not exactly the prediction loss.
- Early stopping may be a first answer
- More interestingly : shrinkage

Reference:
Boosting Algorithms: learning, regularization, prediction, Buhlman, Hothorn, Statistical science, 2007.

Outline

References

- Perrone, Cooper, When classifiers disagree, 1992
- Tumer and Gosh, 1996
- Breiman, Bagging predictors, 1996
- Buhlman and Yu, Analyzing bagging, Annals of stats., 2002
- Breiman, Random Forests, Machine Learning, 2001.
- Geurts, Ernst, Wehenekl, Extra-trees, JMLR, 2006
- Boosting:
  - Freund and Schapire, 1996
  - Greedy function approximation, Friedman, 1999.
  - MarginBoost and AnyBoost : Mason et al. 1999.
  - Tutorial Boosting de Buehlman
  - Boosting, Schapire and Freund, 2012