Ensemble learning

Diane Lingrand

POLYTECH NICE SOPHIA

UNIVERSITÉ CÔTE D'AZUR

SI4

2021 - 2022

Ensemble learning

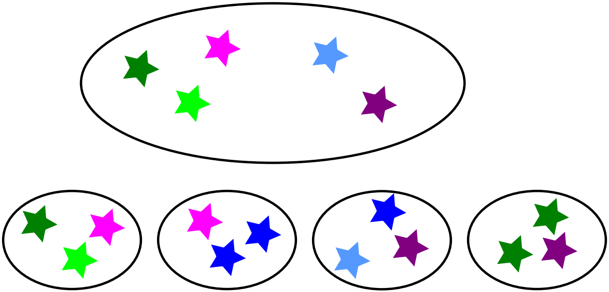
- meta-algorithm combining different learners
- different methods :
  - bagging :
    - weak learners learned independently
    - voting (classification) or averaging (regression)
    - bootstrapping - features sampling - Random Forests
  - boosting :
    - weak learners learned sequentialy
    - focus on erroneus samples
  - stacking :
    - weak learners learned independently
    - meta-model built on top of the output of the weak learners

Outline

Outline

Bagging

- bagging = combining the results of multiple models, e.g. decision trees.
  - but we need different models
  - for regression : average the different results
  - for classication : vote for the classes
- bootstrapping : repeatedly selects a subset with replacement of the training set



Random Forests

- bootstrapping on the m data
- bootstrapping on the n features
  - n or $n/3$ features for regression, $\sqrt{n}$ for classification
- scikit-learn implementation : `sklearn.ensemble.RandomForestClassifier` and `RandomForestRegressor`

Outline

Outline

Context

- Supervised learning
  - learning dataset
  - test dataset
- Binary classification

Binary classification

- Separation between data corresponding to some criterions and data not corresponding.

Main idea of boosting

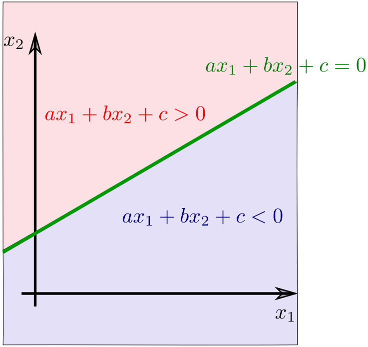
- Build a strong learner from weak learners
- Example :
  - Green corresponds to fir tree.
  - Vertical shapes corresponds to fir tree.
  - Shapes with bottom larger than top corresponds to fir tree.
  - When dominant color is red, it is not a fir tree.
  - ....
- Example of sport bet and heuristics.
  - We ask experts simple rules that are true more than 50%
  - We focus on examples for which a rule fails and we ask other rules to experts.
  - And loop the process

Origin of boosting

- Mathematician Kearns asks : "Is it possible to make as good as possible a weak learner" (that is to say better than random) ?
- Schapire, answered in 90 : " Yes ! " , and exhibited the first elementary boosting algorithm which shows that a weak binary classifier can always improve by being trained on 3 subsamples well chosen.
- The choice of the weak learner does not have any importance (a decision tree, a bayesian classifier, a SVM, a Neural Network, etc.), but one has to choose the 3 training subsamples with respect to its performance.

An example for a weak classifier : line in 2d plane

- points lie in a 2d plane
- the weak classifier is defined by a line of equation $ax_1 + bx_2 + c = 0$
  - divides the plane into 2 area :
    - $(x_1; x_2)$ for which $ax_1 + bx_2 + c > 0$
    - $(x_1; x_2)$ for which $ax_1 + bx_2 + c < 0$
  - learning means to find coefficients a, b, and c.



Outline

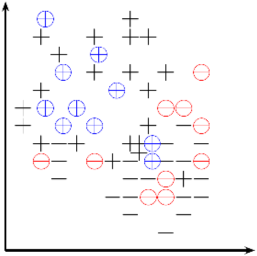
Elementary boosting algorithm

S : learning data set of m elements

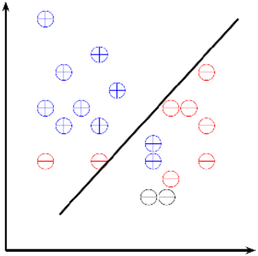
1. Learn the classifier h_1 on subset $S_1 \subset S$. Test of h_1 on $S \setminus S_1$.
2. Learn the classifier h_2 on subset $S_2 \subset S \setminus S_1$ with half of elements of S_2 wrongly classified by h_1.
3. Learn classifier h_3 on subset $S_3 \subset S \setminus S_1 \setminus S_2$: contains elements for which rules h_1 and h_2 answer differently.
4. H : Majority vote between answers of h_1, h_2 and h_3.

A toy example (1)

Classifiers are lines. Learning a classifier corresponds to find the linear separation of data.



S : set of $+$ and -
S_1 : reds and blues

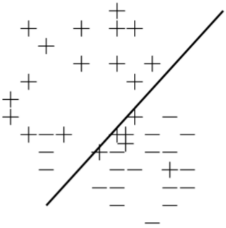


Classifier h_1 learned on S_1.

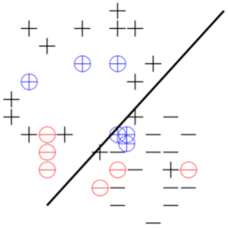
Elementary boosting algorithm

S : learning data set of m elements

1. Learn the classifier h_1 on subset $S_1 \subset S$. Test of h_1 on $S \setminus S_1$.
2. Learn the classifier h_2 on subset $S_2 \subset S \setminus S_1$ with half of elements of S_2 wrongly classified by h_1.
3. Learn classifier h_3 on subset $S_3 \subset S \setminus S_1 \setminus S_2$: contains elements for which rules h_1 and h_2 answer differently.
4. H : Majority vote between answers of h_1, h_2 and h_3.



$S \setminus S_1$ and h_1



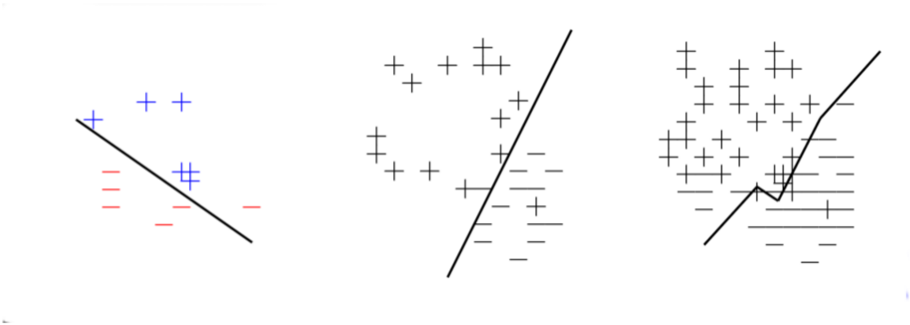
$S_2 \subset S \setminus S_1$: reds and blues

Elementary boosting algorithm

S : learning data set of m elements

1. Learn the classifier h_1 on subset $S_1 \subset S$. Test of h_1 on $S \setminus S_1$.
2. Learn the classifier h_2 on subset $S_2 \subset S \setminus S_1$ with half of elements of S_2 wrongly classified by h_1.
3. Learn classifier h_3 on subset $S_3 \subset S \setminus S_1 \setminus S_2$: contains elements for which rules h_1 and h_2 answer differently.
4. H : Majority vote between answers of h_1, h_2 and h_3.

A toy example (3)



h_2 learned on S_2 S_3 and h_3 S and H

Outline

Probabilistic booosting

3 main ideas :

1. A set of specialized experts and ask them to vote to take a decision.
2. Adaptive weighting of votes by multiplicative update.
3. Modifying example distribution to train each expert, increasing the weights iteratively of examples misclassified at previous iteration.

General algorithm

- S learning dataset
- Initialisation : all samples have same weights $(D_1(i), i \in \{1 \dots m\})$.
- Iterations : for $t \in \{1 \dots T\}$
  - **learn** h_t by minimisation of an **error**
  - **compute weight** α_t and **weights** $D_t(i)$ (or distribution)
- Compute strong classifier :

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

Goal of Adaboost

- Minimisation of exponential loss : $E_{x,y}\left[e^{-yH(x)}\right]$
- Iteration t : $H_t(x) = H_{t-1}(x) + \alpha_t(x)h_t(x)$

$$
\begin{aligned}
E_{x,y}\left[e^{-yH_t(x)}\right] &= E_x\left[E_y\left[e^{-yH_{t-1}(x)}.e^{y\alpha_t h_t(x)}|x\right]\right] && (1)\\
&= E_x\left[E_y\left[e^{-yH_{t-1}(x)}\left[e^{-\alpha_t}P(y=h_t(x)) + e^{\alpha_t}P(y\neq h_t(x))\right]|x\right]\right] && (2)
\end{aligned}
$$

Minimum when

$$-e^{-\alpha_t}P(y=h_t(x)) + e^{\alpha_t}P(y\neq h_t(x)) = 0$$

$$\Rightarrow \alpha_t = \frac{1}{2}\frac{P(y=h_t(x))}{P(y\neq h_t(x))}$$

Adaboost : **Ada**ptative **boost**ing

- $S = \{(x_1, y_1), \dots (x_m, y_m)\}$ with $x_i \in X$ and $y_i \in \{-1, +1\}$
- Initialisation : $D_1(i) = \frac{1}{m}$ with $i \in \{1 \dots m\}$
- For $t \in \{1 \dots T\}$:
  - **find** $h_t : X \to \{-1, +1\}$ minimizing error ϵ_t defined by :

$$\epsilon_t = \sum_{i=1}^{m} D_t(i)[y_i \neq h_t(x_i)]$$

  - compute **weights** :

$$\alpha_t = \frac{1}{2} \ln(\frac{1 - \epsilon_t}{\epsilon_t})$$

  - compute **distribution** :

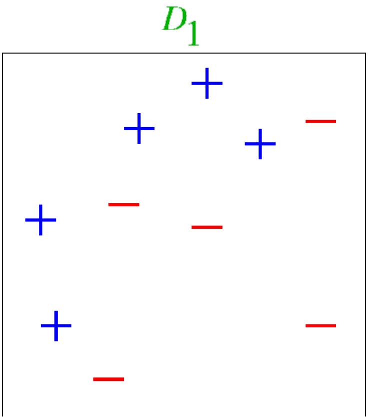
$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t \, y_i \, h_t(i))}{Z_t}$$

  where Z_t is for normalisation : $1 = \sum_{i=1}^{m} D_t(i)$
- Compute **strong classifier** :

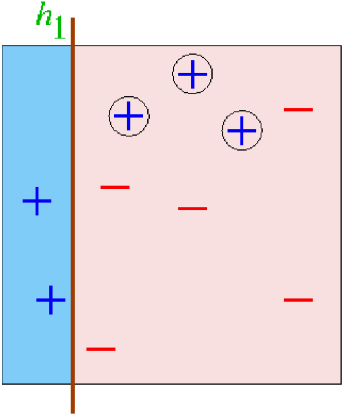
$$\text{sign}\left(H(x) = \sum_{t=1}^{T} \alpha_t \, h_t(x)\right)$$

Outline

Initialisation



D_1 :	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1



$$\epsilon_1 = 0.30$$

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-\epsilon_1}{\epsilon_1}\right) = \frac{1}{2}\ln\left(\frac{0.7}{0.3}\right) = 0.42$$

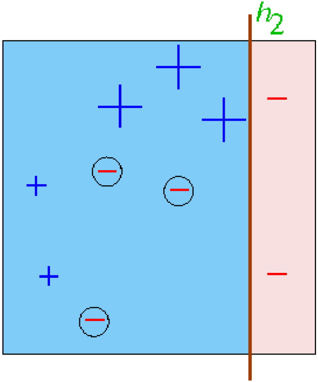
$$D_2(1) = \frac{1}{Z_1}\, D_1(1)\, e^{\alpha_1} = \frac{0.152}{Z_1}$$

$$D_2(4) = \frac{1}{Z_1}\, D_1(4)\, e^{-\alpha_1} = \frac{0.065}{Z_1}$$

$$Z_1 = 3 \times 0.152 + 7 \times 0.065 = 0.911$$

D_2 :

0.167	0.167	0.167	0.071	0.071	0.071	0.071	0.071	0.071	0.071



$$\epsilon_2 = 0.21$$

$$\alpha_2 = \frac{1}{2}\ln\left(\frac{1-\epsilon_2}{\epsilon_2}\right) = 0.65$$

$$D_3(1) = \frac{D_2(1)}{Z_2}\, e^{-\alpha_2} = 0.167\, e^{-0.65} = \frac{0.0876}{Z_2}$$

$$D_3(4) = \frac{D_2(4)}{Z_2}\, e^{-\alpha_2} = 0.071\, e^{-0.65} = \frac{0.036}{Z_2}$$

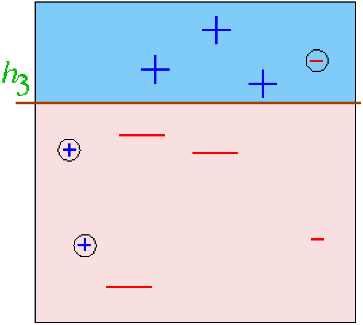
$$D_3(8) = \frac{D_2(8)}{Z_2}\, e^{+\alpha_2} = 0.071\, e^{+0.65} = \frac{0.1357}{Z_2}$$

$$Z_2 = 3\times 0.0876 + 4\times 0.036 + 3\times 0.1357 = 0.814$$

D_3 :

0.11	0.11	0.11	0.044	0.044	0.044	0.044	0.17	0.17	0.17

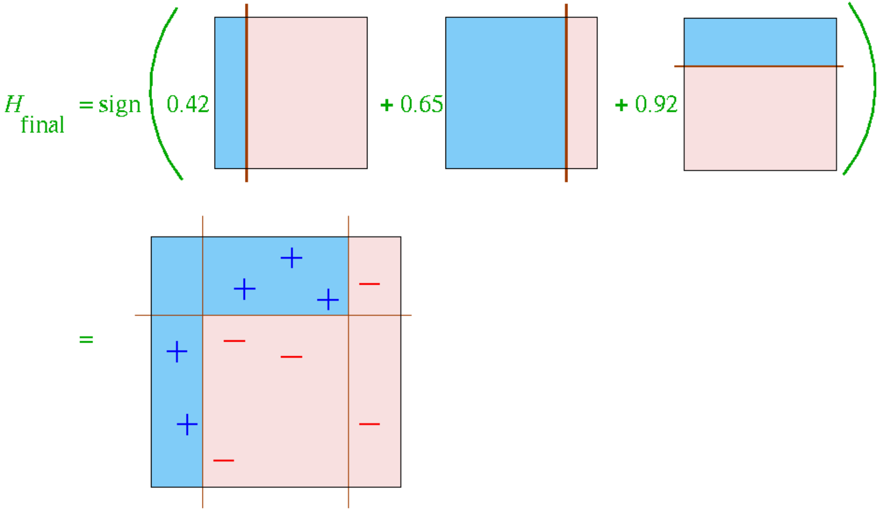
Step 3



$$\epsilon_3 = 3 \times 0.044 = 0.13$$

$$\alpha_3 = 0.92$$

Final classifier



Advantages and drawbacks

- Advantages :
  - Fast
  - Easy to implement
  - only 1 parameter : number of boosting iterations
  - extensible to multi classes classification
  - able to detect outliers
- Drawbacks :
  - depends on learning data and weak classifiers
  - could fail if :
    - the weak classifier is too complex
    - low margins $\rightarrow$ overfitting
    - the weak classifier is too weak
    - underfitting
  - noise sensitive

Multi-class boosting : AdaBoost SAMME

- $S = \{(x_1, y_1), \ldots (x_m, y_m)\}$ with $x_i \in X$ and $y_i \in \{1 \ldots k\}$
- Initialisation : $D_1(i) = \frac{1}{m}$ with $i \in \{1 \ldots m\}$
- For $t \in \{1 \ldots T\}$:
  - **find** $h_t : X \to \{1 \ldots k\}$ minimizing error ϵ_t defined by :

$$\epsilon_t = \sum_{i=1}^{m} D_t(i) g(y_i, h_t(x_i)) \text{ with } g(a, b) = 1 \text{ if } a = b \text{ else } 0$$

  - compute **weights** :

$$\alpha_t = \frac{1}{2} \ln(\frac{1 - \epsilon_t}{\epsilon_t}) + \frac{1}{2} \ln\left(k - 1\right)$$

  - compute **distribution** :

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t \, g(h_t(x_i), y_i))}{Z_t} \text{ where } 1 = \sum_{i=1}^{m} D_t(i)$$

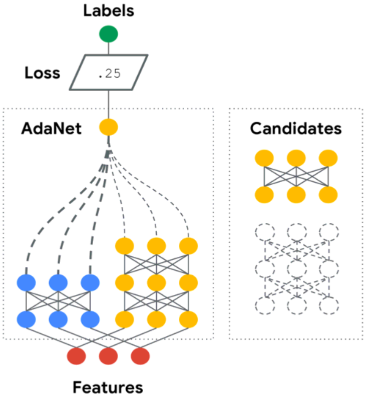
- Compute **strong classifier** :

$$\arg\max_k \left(\sum_{t=1}^{T} \alpha_t \, g(h_t(x), k) \right)$$

Example using scikit-learn

```
from sklearn.datasets import load_iris
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
boosting = AdaBoostClassifier(n_estimators=20, algorithm='SAMME.R')
boosting.fit(X_train, y_train)
y_pred = boosting.predict(X_test)
print('f1 score: ', f1_score(y_pred,y_test,average=None))
```



AdaNet : https://github.com/tensorflow/adanet

Outline

Stacking

- for each sample i
  - for each learner
    - compute the prediction of the sample by the learner
  - arrange them in a vector x^i
- learn a new machine learning algorithm
  - with the dateset $\{x^i\}$
  - could be a logistic regression or any other ml algorithm
- scikit-learn implementation : `sklearn.ensemble.StackingClassifier`[1]

1. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html#sklearn.ensemble.StackingClassifier