EE-613 – Machine Learning for Engineers

Feature selection and Boosting

François Fleuret
`https://www.idiap.ch/~odobez/teaching-epfl-ee613-2019.php`

Thu Nov 28, 2019

Dimension reduction

In many situation it makes sense to reduce the dimension of the original feature space. The objective is to:

- Reduce over-fitting
- Reduce the computational cost
- Facilitate the understanding of causal relations
- Facilitate the visualization of data

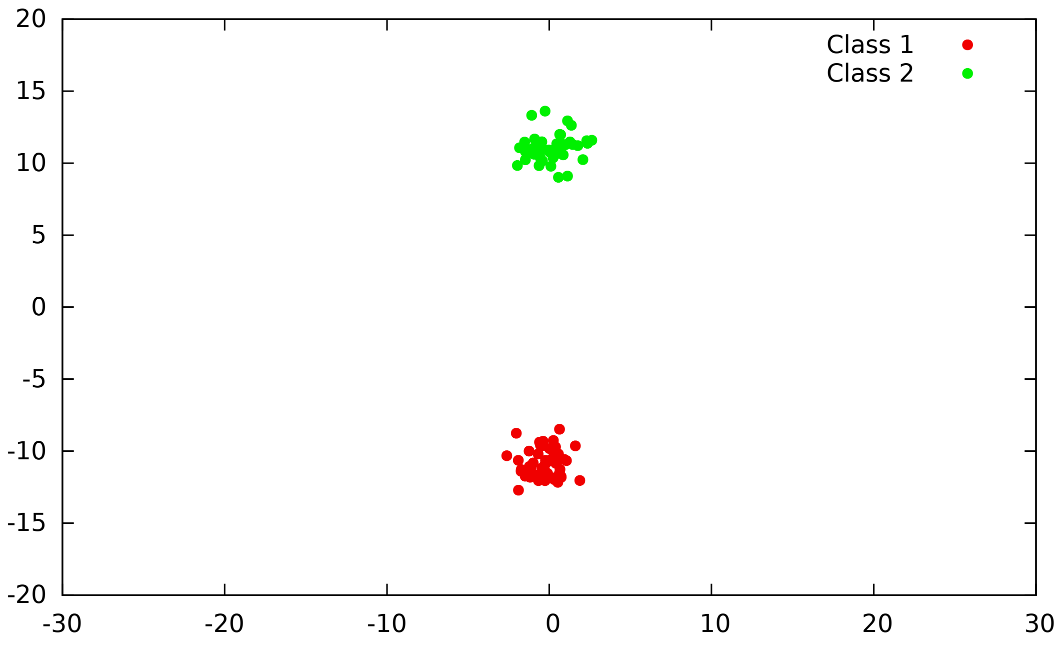
It can be combined with other procedures (supervised or not).

There are two main strategies to achieve dimension reduction:

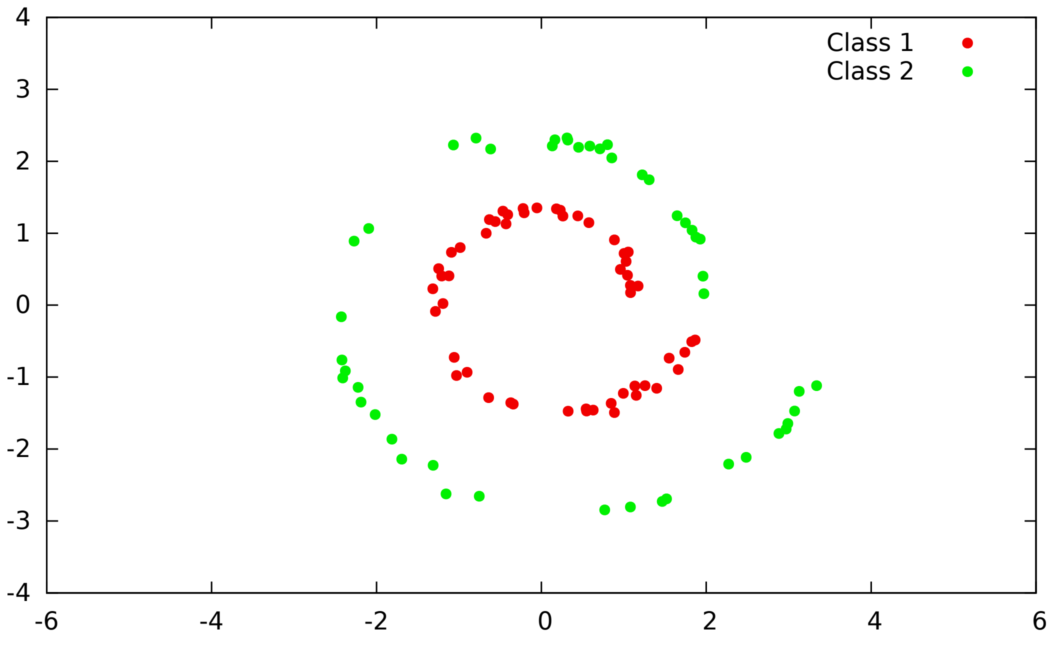
- **Feature selection** finds a projection spanned by a subset of coordinates of the original space.

- **Feature extraction** (aka “feature learning”) builds a mapping from the original space into a space of lower dimension. This is also called an **embedding**.

Feature selection is adequate if a subset of feature carries (all) the discriminating information



Feature extraction is required if none of the original feature carries the discriminating information, but it can be captured with a few quantities.



Feature selection

The dimension reduction is achieved by selecting a subset of features, without re-combining them through a mapping.

In practice, if X is our signal, r.v. on $\mathbb{R}^D$, given a target dimension d, we are looking for a list of coordinates

$$\nu(1), \dots, \nu(d)$$

such that

$$X' = \left(X_{\nu(1)}, \dots, X_{\nu(d)}\right)$$

carries as much information as possible.

It is a simple way to achieve *sparsity*.

Some classes of predictors naturally achieve feature selection during training, for instance decision trees and Boosting.

Contrary to classical linear models (perceptron or SVM) or neural networks, the proportion of features they use is specified during training, and usually very small.

As usual in machine-learning, feature selection is defined by a criterion to optimize, and an optimization procedure.

The most common optimization schemes are:

- **Forward selection**: Features are added one after another, to increase the target score optimally at every iteration.
- **Backward selection**: Starting from all the feature, they are removed one at a time, to reduce the score the least.

Such a greedy optimization is not optimal, but usually tractable.

In an unsupervised setting, the objective is to keep as much of the information of the original signal X.

In the supervised case, the goal is to keep as much information *about the value to predict Y*.

In the supervised case, features can be chosen specifically for a certain class of predictors, in which case it is called a **wrapper**.

The quality of a set of features is estimated by training a predictor and measuring empirically its performance.

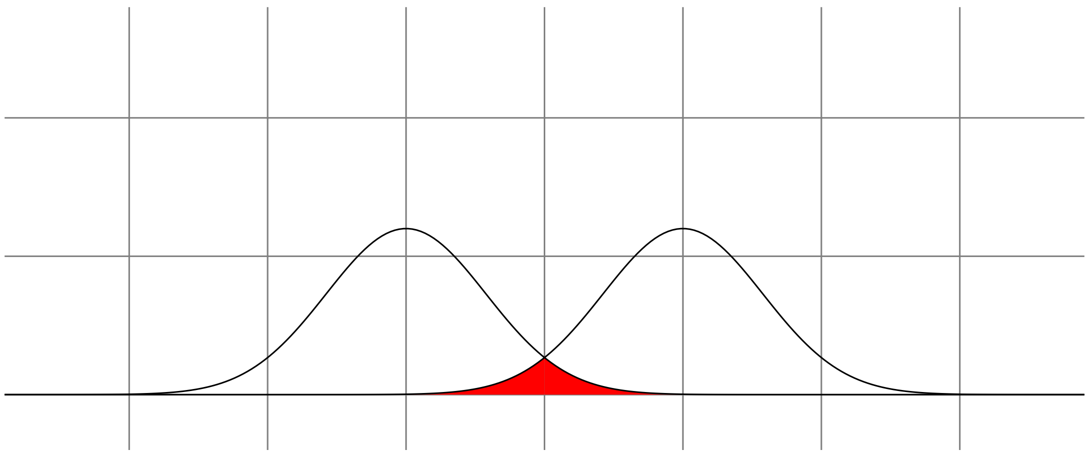
This ties the definition of "information" to the predictor, avoiding to keep useless parts of the signal, but requires heavy computation and may lead to over-fitting.

The alternative is a **filter**, which is predictor-agnostic.

The standard supervised filter strategy consists in ranking features according to their individual information content, and selecting them in order.

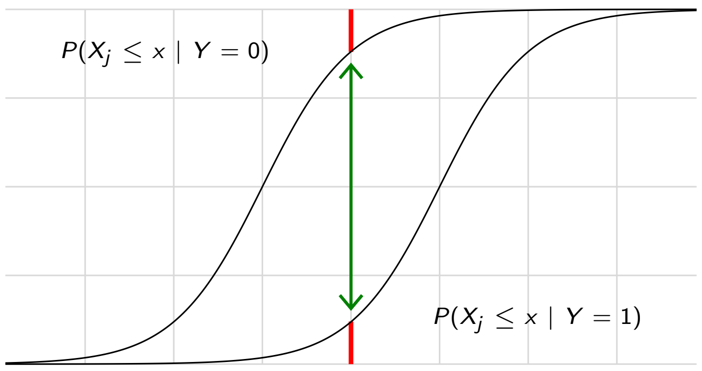
The Fisher score provides an estimate of separability of the difference classes given a single features.

$$S(X_j) = \frac{\sum_{i=1}^{C} (E(X_j|Y=i) - E(X_j))^2 P(Y=i)}{\sum_{i=1}^{C} V(X_j|Y=i) P(Y=i)}$$



Another feature-selection score is based on the non-parametric Kolmogorov-Smirnov test

$$S(X_j) = \max_x |P(X_j \leq x \mid Y = 0) - P(X_j \leq x \mid Y = 1)|$$



This score does not depend on the cond. distributions *if they are identical* and as such is consistent to estimate if the distributions differ.

A fundamental concept that can be used directly in the case of discrete features and class is the mutual information

$$\begin{aligned} I(A; B) &= H(A) + H(B) - H(A, B) \\ &= H(A) - H(A \mid B) \end{aligned}$$

It has some intuitive properties such as

$$I(A; B) = I(B; A),$$

and

$$I(A; A) = H(A).$$

If Y is a r.v. on $\{1, \dots, C\}$ standing for the label to predict, and f is a predictor of Y from X, Fano's inequality implies

$$P(f(X) \neq Y) \geq \frac{H(Y) - 1 - I(X; Y)}{\log C}$$

Hence, increasing the mutual information between X and Y lowers that lower bound, making accurate prediction feasible.

Hence, mutual information is often used as a score for feature selection

$$S(X_j) = I(X_j; Y).$$

While these scores efficiently measure the information content of individual features, they do not avoid redundancy, nor account for joint informational content.

- If a very good feature is replicated multiple times, it will be picked again and again.
- If features are individually non-informative, and jointly informative, they will not be picked.

Consider $X_1, \dots, X_D$ i.i.d. of distribution $\mathscr{B}(0.5)$, and

$$Y = \mathbf{1}_{\left\{\sum_{i=1}^{D} X_i \in 2\mathbb{Z}\right\}}.$$

Then

$$\forall i,\ I(X_i; Y) = 0$$

and

$$H(Y | X_1, \dots, X_D) = 0.$$

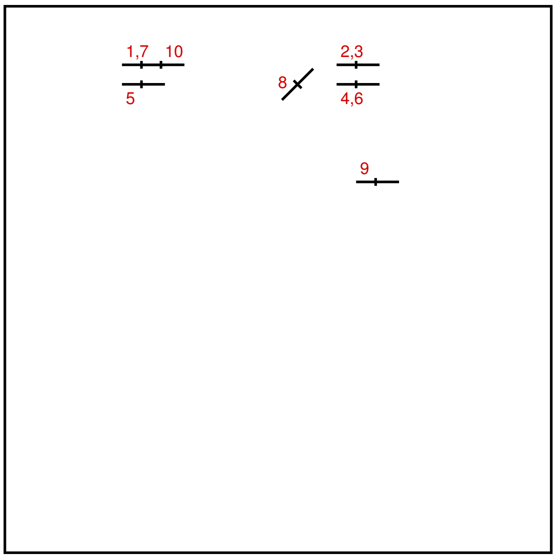
Many (many!) feature selection schemes exist to take into account joint information, often through sampling of n-uplets, and redundancy with explicit penalty terms.

Selected features should be individually informative and weakly dependent.

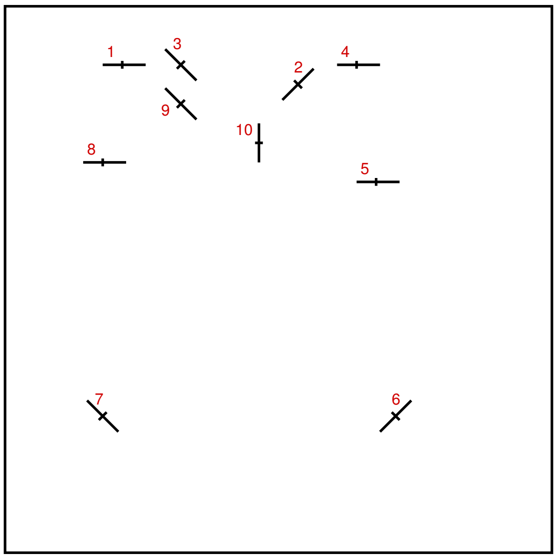
CMIM (Fleuret, 2004) selects features as follows:

$$\begin{aligned} \nu(1) &= \operatorname*{argmax}_n I\left(Y, X_n\right) \\ \forall k,\ 1 \leq k < K, \quad \nu(k+1) &= \operatorname*{argmax}_n \left\{ \min_{l \leq k} I\left(Y, X_n \,|\, X_{\nu(l)}\right) \right\} \end{aligned}$$

Face vs. non-face patches.



Mututal information



Conditional Mutual Information

We apply this algorithm to a very noisy problem of drug efficiency prediction from molecular properties.

- 1,909 molecules.
- 139,351 binary features encoding three-dimensional properties.
- Provided by DuPont Pharmaceutical.

Classifier	Error image	Error drug
CMIM feature selection + naive Bayesian	1.95%	11.72%
AdaBoost$_{reg}$ (optimized on test set)	3.06%	13.64%
MIM feature selection + naive Bayesian	8.59%	23.35%
CMIM feature selection + perceptron	9.32%	23.51%
Random feature selection + naive Bayesian	24.99%	40.13%

(Fleuret, 2004)

Feature extraction

Feature extraction is a good strategy for the visualization of data, and to control over-fitting when it is unsupervised.

However

- it may not reduce the computation, since the transformation may be expensive,
- the obtained features may be complex combinations of the original ones, hard to interpret.

PCA is the most classic unsupervised feature extraction procedure. It finds the best linear projection to minimize the L^2 norm between training points and their images.

Moreover, it ranks the extracted features according to the residual variance, hence it makes sense to take them in order until the target dimension is reached.

PCA may select directions of large variance which do not carry information about the class.

The Linear Discriminant Analysis (LDA) computes a linear transformation such that the means of the classes are pushed apart while each class is not dilated in the same directions.

It makes the assumption that the classes all follow Normal distributions of identical co-variance matrices.

Compute the with-class scatter matrix

$$\mathbf{S}_w = \sum_{i=1}^{n} (\mathbf{x}_i - \bar{x}_{y_i})(\mathbf{x}_i - \bar{x}_{y_i})^T$$

and the between-class scatter matrix

$$\mathbf{S}_b = \sum_{y=1}^{m} n_y (\bar{x}_y - \bar{x})(\bar{x}_y - \bar{x})^T$$

maximize the ratio between the two

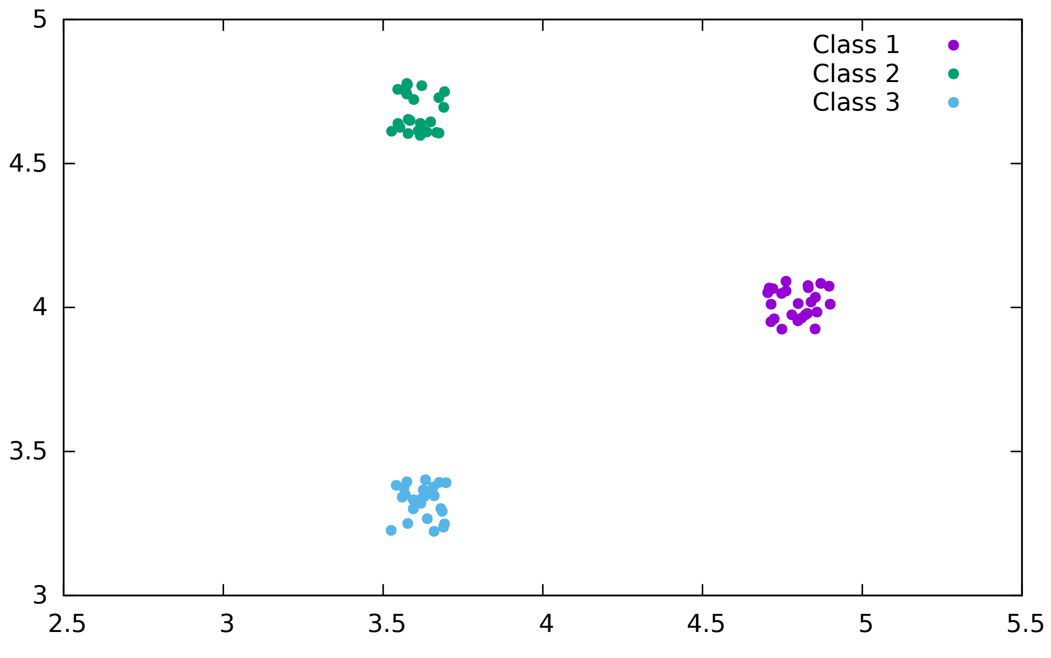
$$\hat{\mathbf{w}} = \operatorname*{argmax}_{\mathbf{w}} \frac{\mathbf{w}^T \mathbf{S}_b \mathbf{w}}{\mathbf{w}^T \mathbf{S}_w \mathbf{w}}$$

which amounts to solving

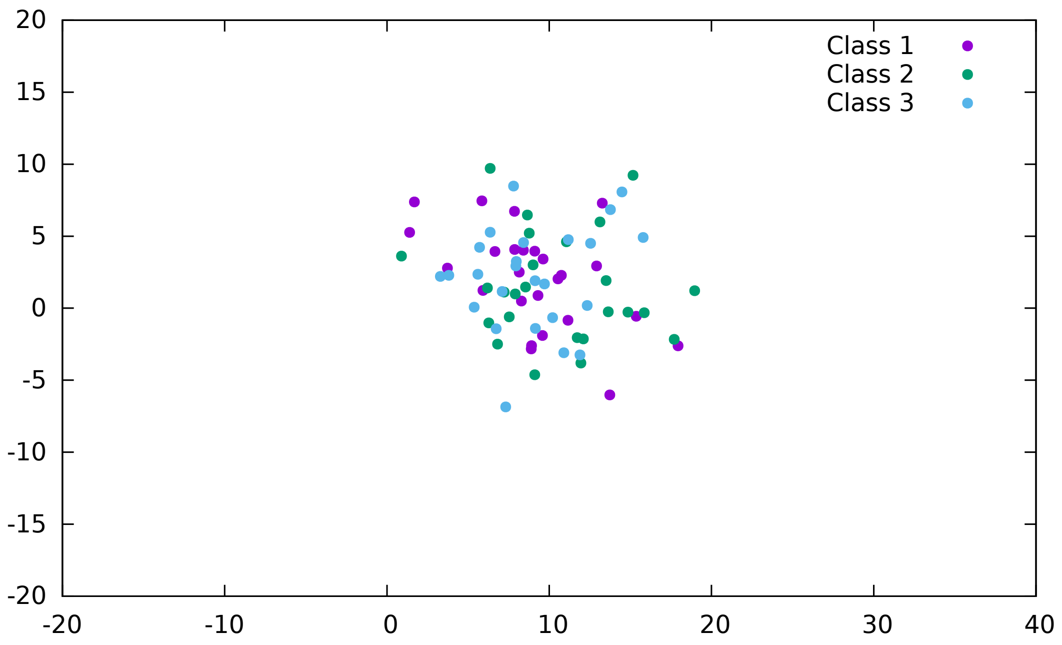
$$\mathbf{S}_b \mathbf{w} = \lambda \mathbf{S}_w \mathbf{w}.$$

for the largest possible λ. This is the generalized Eigenvalue problem.

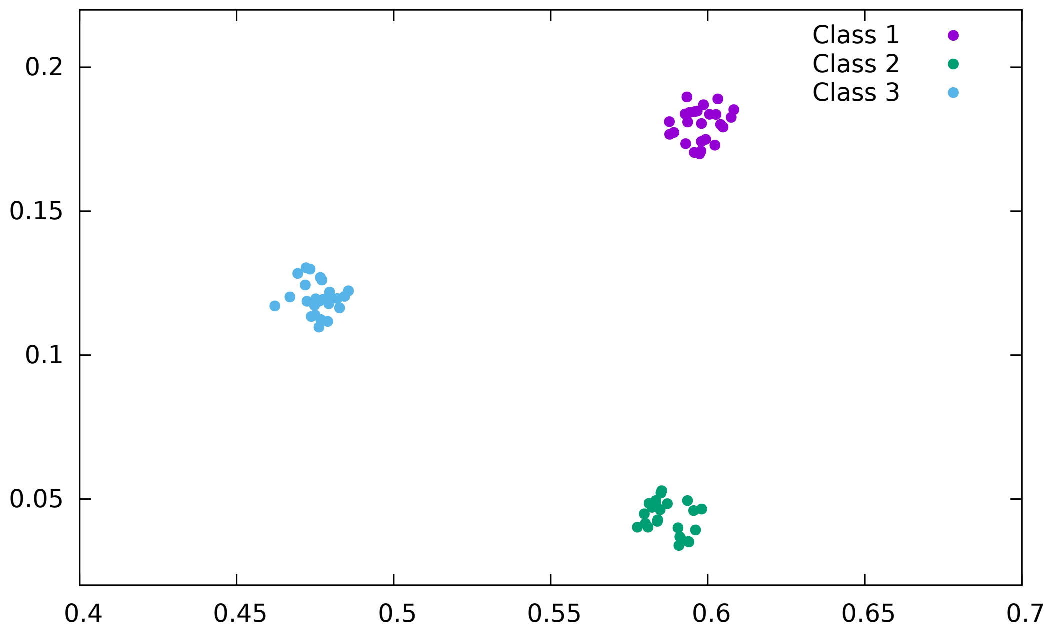
We consider the following population



We add 8 additional random dimensions $\sim \mathscr{U}[0, 10]$ and apply a random linear transformation.



After LDA, we retrieve two good directions



Unfortunately, the structure is often non-affine. This can be dealt with through kernelization, or with completely different methods.

The k-means procedure for instance can be seen as a crude discrete feature extraction method.

Given

$$x_n \in \mathbb{R}^D,\ n = 1, \dots, N$$

and a target cardinality C, it finds an set of centroids $\alpha_1, \dots, \alpha_C$ minimizing

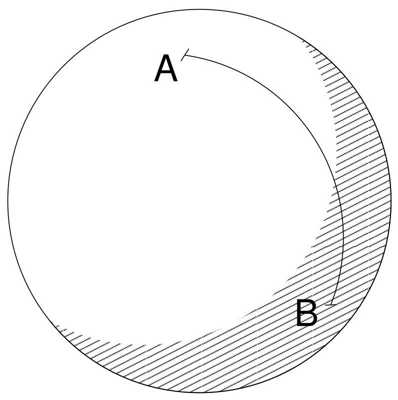
$$\sum_n \min_c \|x_n - \alpha_c\|^2$$

The extracted discrete feature is

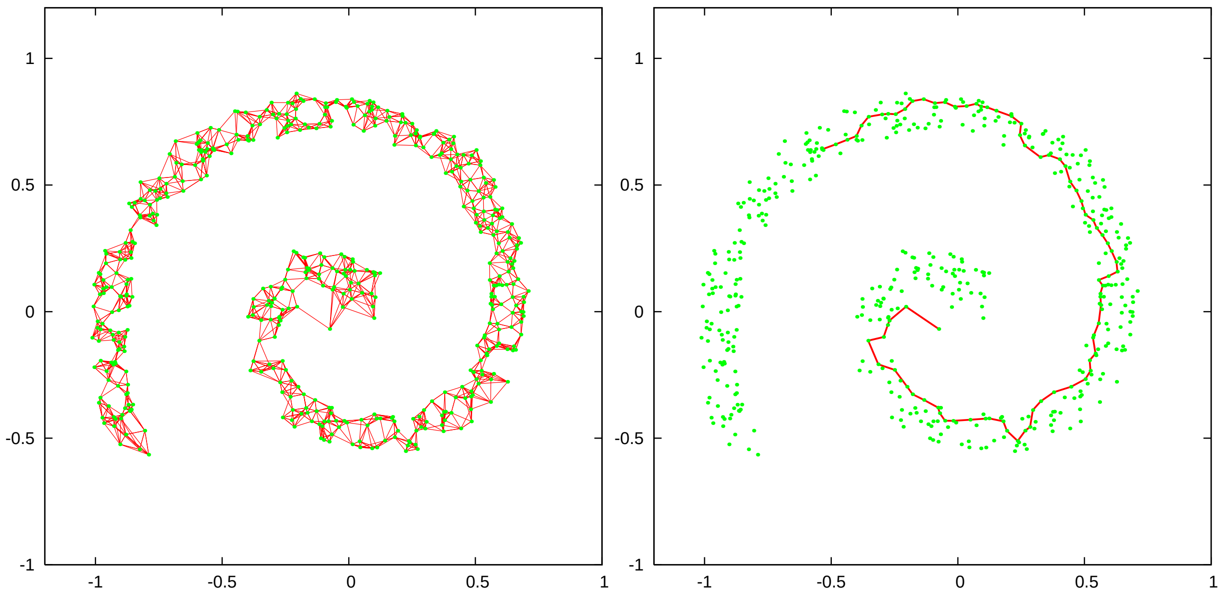
$$\operatorname*{argmin}_c \|x_n - \alpha_c\|^2 .$$

A more sophisticated algorithm to deal with non-affine structures is isomap.

It consists of approximating the geodesic distance with the euclidean distance in a space of small dimension.



Given $x_1, \dots, x_n \in \mathbb{R}^d$, Isomap first built a graph by connecting all points to its k nearest neighbors.



Then, it approximates the geodesic distance between points with the length Δ of the shortest path in that graph.

The second step consists of finding $y_1, \dots, y_n$ in $\mathbb{R}^r$ such that

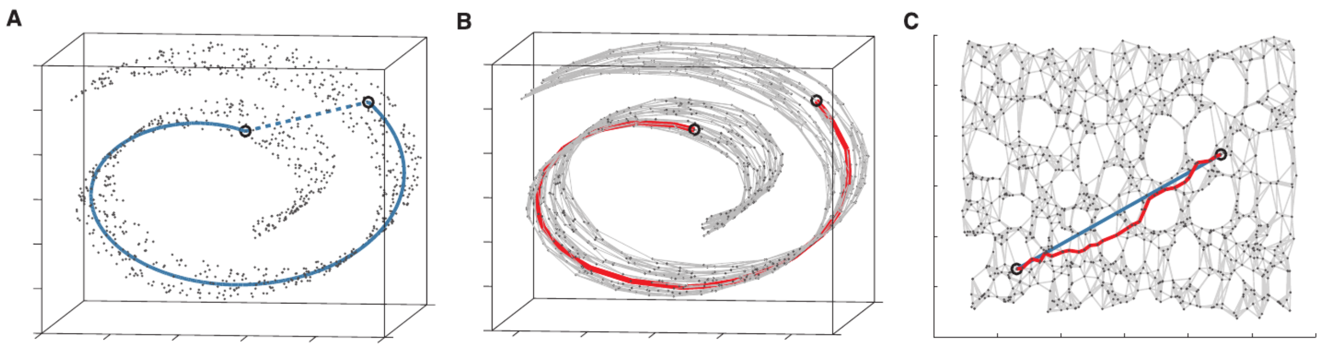
$$\forall i, j, \; \Delta(i, j) \simeq \|y_i - y_j\|$$

This is done using the multi-dimensional scaling (MDS), which minimizes

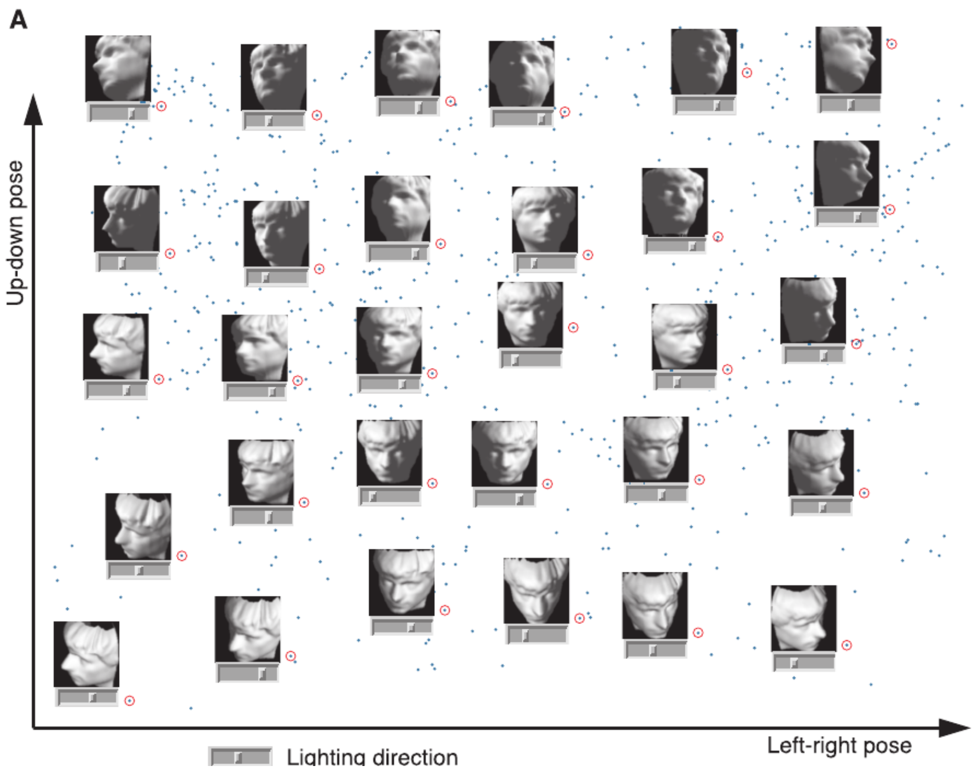
$$\sum_{i \neq j} \left(\|y_i - y_j\| - \Delta(i, j) \right)^2$$

Hence we finally have associated one y_i in $\mathbb{R}^r$ to every x_i.

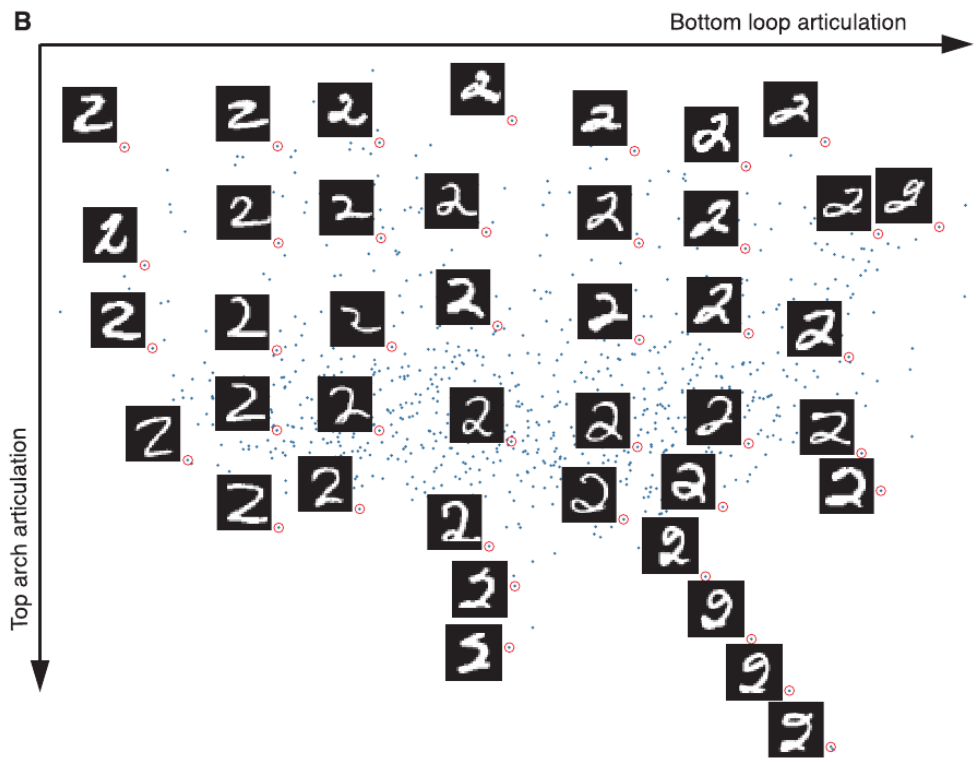
Note that we have not provided a way to generalize to other x.



(J. B. Tenenbaum, V. de Silva and J. C. Langford, 2000)

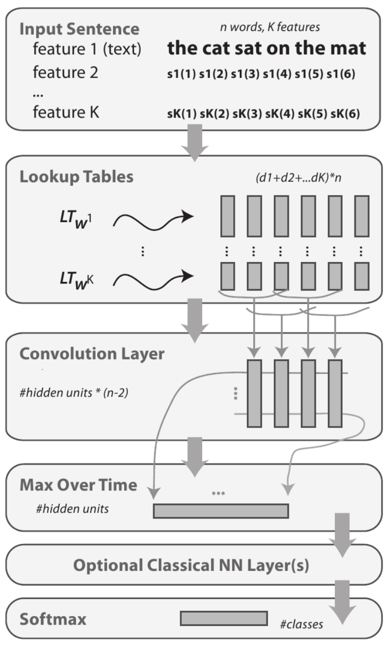


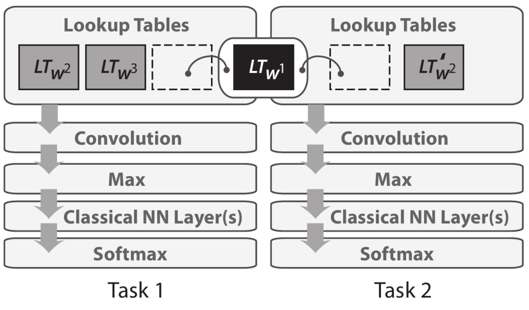
(J. B. Tenenbaum, V. de Silva and J. C. Langford, 2000)



(J. B. Tenenbaum, V. de Silva and J. C. Langford, 2000)

There are not many supervised feature extraction procedures. The most popular are ANNs, whose early layers can be trained on one task, and re-used on another.





(Collobert & Weston, 2008)

Without language model:

france	**jesus**	**xbox**	**reddish**	**scratched**	**megabits**
454	1973	6909	11724	29869	87025
persuade	thickets	decadent	widescreen	odd	ppa
faw	savary	divo	antica	anchieta	uddin
blackstock	sympathetic	verus	shabby	emigration	biologically
giorgi	jfk	oxide	awe	marking	kayak
shaheed	khwarazm	urbina	thud	heuer	mclarens
rumelia	stationery	epos	occupant	sambhaji	gladwin
planum	ilias	eglinton	revised	worshippers	centrally
goa'uld	gsNUMBER	edging	leavened	ritsuko	indonesia
collation	operator	frg	pandionidae	lifeless	moneo
bacha	w.j.	namsos	shirt	mahan	nilgiris

With language model:

france	**jesus**	**xbox**	**reddish**	**scratched**	**megabits**
454	1973	6909	11724	29869	87025
austria	god	amiga	greenish	nailed	octets
belgium	sati	playstation	bluish	smashed	mb/s
germany	christ	msx	pinkish	punched	bit/s
italy	satan	ipod	purplish	popped	baud
greece	kali	sega	brownish	crimped	carats
sweden	indra	psNUMBER	greyish	scraped	kbit/s
norway	vishnu	hd	grayish	screwed	megahertz
europe	ananda	dreamcast	whitish	sectioned	megapixels
hungary	parvati	geforce	silvery	slashed	gbit/s
switzerland	grace	capcom	yellowish	ripped	amperes

Boosting

We saw that combining decision trees with a voting procedure is very efficient. This can be generalized to other type of predictors.

However, it makes more sense to build each predictor knowing the mistakes made by the ones built so far. Boosting is based on that idea.

Let $\mathscr{W}$ denote a set of functionals from $\mathbb{R}^D$ into $\{-1, 1\}$. They are weak learners in the sense that a single one will not do the job.

For simplicity we will consider a two-class problem. Let

$$(X_t, Y_t) \in \mathbb{R}^D \times \{-1, 1\}, \quad t = 1, \dots, T$$

denote a training set.

Our goal is to combine several classifiers through a weighted voting

Hence we want to build a strong classifier of the form

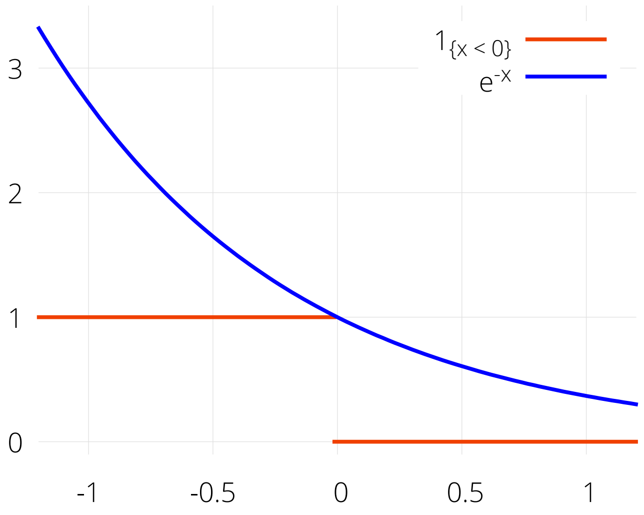
$$f(x) = \sum_k \omega_k h_k(x)$$

that has a low error as defined by

$$err(f) = \sum_t \mathbf{1}_{\{f(X_t) Y_t \leq 0\}}$$

Minimizing the error we just defined is intractable. However we have

$$\forall x \in \mathbb{R}, \mathbf{1}_{\{x<0\}} \leq \exp(-x)$$



Since

$$\lim_{x \to +\infty} \exp(-x) = 0,$$

it makes sense to minimize the exponential loss

$$L(f) = \sum_t \exp(-Y_t f(X_t))$$

to minimize $err(f)$.

We build the function f iteratively, adding one weak learner at each step.

If we have already built

$$f_k = \sum_{m=1}^{k} \omega_m h_m$$

what are ω_{k+1} and h_{k+1} ?

We can use a greedy strategy, similar to the gradient descent:

Add at every step a weak learner so that the local loss reduction is optimal.

Given a weak learner $h \in \mathscr{W}$, the local loss reduction is

$$R(h) = \left. \frac{\partial L(f_k + \omega h)}{\partial \omega} \right|_{\omega=0}.$$

We can add or substract h_{k+1}, so we want to maximize $|R(h_{k+1})|$.

$$
\begin{aligned}
R(h) &= \left.\frac{\partial L(f_k + \omega h)}{\partial \omega}\right|_{\omega=0} \\
&= \left.\frac{\partial \sum_t \exp(-Y_t(f_k(X_t) + \omega h(X_t)))}{\partial \omega}\right|_{\omega=0} \\
&= \sum_t \left.\frac{\partial \exp(-Y_t(f_k(X_t) + \omega h(X_t)))}{\partial \omega}\right|_{\omega=0} \\
&= \sum_t \exp(-Y_t f_k(X_t)) \left.\frac{\partial \exp(-\omega Y_t\, h(X_t))}{\partial \omega}\right|_{\omega=0} \\
&= \sum_t \exp(-Y_t f_k(X_t))\,(-Y_t\, h(X_t))
\end{aligned}
$$

Since $\sum_t \exp(-Y_t f_k(X_t)) = L(f_k)$, we define $\nu_t = \frac{\exp(-Y_t f_k(X_t))}{L(f_k)}$.

We have

$$
\begin{aligned}
R(h) &= \sum_t \exp(-Y_t f_k(X_t))\,(-Y_t\, h(X_t)) \\
&= \sum_t \exp(-Y_t f_k(X_t)) \left(2\,\mathbf{1}_{\{Y_t \neq h(X_t)\}} - 1\right) \\
&= L(f_k) \sum_t \nu_t \left(2\,\mathbf{1}_{\{Y_t \neq h(X_t)\}} - 1\right) \\
&= L(f_k) \left(2 \sum_t \nu_t \mathbf{1}_{\{Y_t \neq h(X_t)\}} - \sum_t \nu_t\right) \\
&= L(f_k) \left(2 \sum_{t:Y_t \neq h(X_t)} \nu_t - 1\right)
\end{aligned}
$$

Finally, with $\nu_t = \frac{\exp(-Y_t f_k(X_t))}{L(f_k)}$, we are looking for

$$\underset{h \in \mathscr{W}}{\operatorname{argmax}} \left| \sum_{t: Y_t \neq h(X_t)} \nu_t - 1/2 \right|$$

We have to find a weak learner with a weighted error as far as possible from 1/2.

As in gradient descent, we pick the weak-learner weight with a line-search.

Given h_{k+1}, we have to find ω_{k+1} to minimize the loss

$$\omega_{k+1} = \underset{\omega}{\operatorname{argmin}}\, L(f_k + \omega h_{k+1}).$$

Since $L(f_k + \omega h_{k+1})$ is differentiable in ω, ω_{k+1} is a solution of

$$\left. \frac{\partial L(f_k + \omega h_{k+1})}{\partial \omega} \right|_{\omega = \omega_{k+1}} = 0$$

$$
\begin{aligned}
&\frac{\partial L(f_k + \omega h_{k+1})}{\partial \omega} \\
&\qquad = \frac{\partial \sum_t \exp(-Y_t(f_k(X_t) + \omega h_{k+1}(X_t)))}{\partial \omega} \\
&\qquad = \sum_t \frac{\partial \exp(-Y_t(f_k(X_t) + \omega h_{k+1}(X_t)))}{\partial \omega} \\
&\qquad = -\sum_t Y_t\, h_{k+1}(X_t)\, \exp(-Y_t(f_k(X_t) + \omega h_{k+1}(X_t))) \\
&\qquad = -\sum_{t: Y_t = h_{k+1}(X_t)} \exp(-Y_t f_k(X_t) - \omega) + \sum_{t: Y_t \neq h_{k+1}(X_t)} \exp(-Y_t f_k(X_t) + \omega) \\
&\qquad = \exp(-\omega) \left(-\sum_{t: Y_t = h_{k+1}(X_t)} \exp(-Y_t f_k(X_t)) + \exp 2\omega \sum_{t: Y_t \neq h_{k+1}(X_t)} \exp(-Y_t f_k(X_t)) \right)
\end{aligned}
$$

Hence

$$
\left. \frac{\partial L(f_k + \omega h_{k+1})}{\partial \omega} \right|_{\omega = \omega_{k+1}} = 0
$$

$$
\Rightarrow \quad \omega_{k+1} = \frac{1}{2} \log \frac{\sum_{t: Y_t = h_{k+1}(X_t)} \exp(-Y_t f_k(X_t))}{\sum_{t: Y_t \neq h_{k+1}(X_t)} \exp(-Y_t f_k(X_t))}
$$

So, with $e_k(h_{k+1}) = \sum_{t: Y_t \neq h_{k+1}(X_t)} \nu_t$, we obtain

$$
\omega_{k+1} = \frac{1}{2} \log \left(\frac{1 - e_k(h_{k+1})}{e_k(h_{k+1})} \right)
$$

The weight of a weak learner is large when the weighted error is small. It is negative when the error is greater than $1/2$.

If we define the samples' weights as $\nu_t = \frac{\exp(-Y_t f_k(X_t))}{\sum_s \exp(-Y_s f_k(X_s))}$, we get

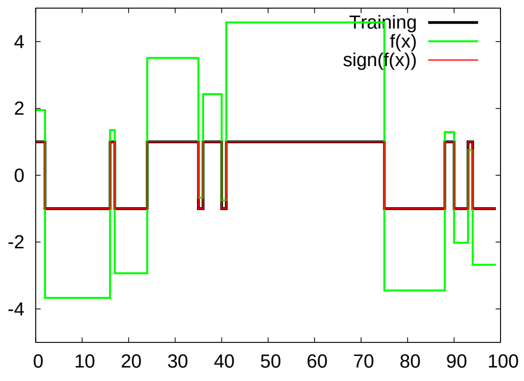
$$h_{k+1} = \operatorname*{argmax}_{h \in \mathscr{W}} \left| \sum_{t:Y_t \neq h(X_t)} \nu_t - 1/2 \right|$$

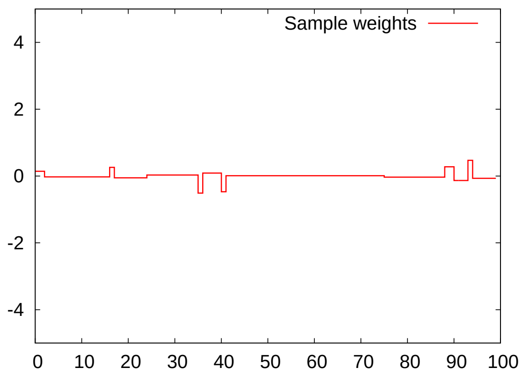
and with the weighted error $e_k(h_{k+1}) = \sum_{t:Y_t \neq h_{k+1}(X_t)} \nu_t$

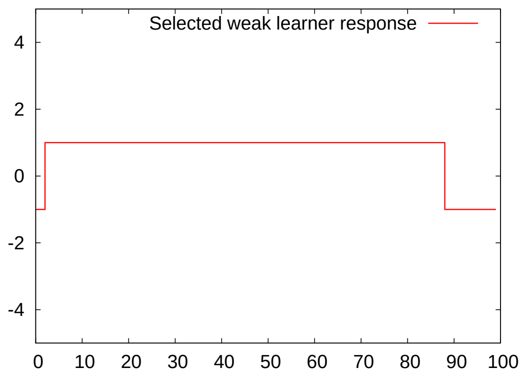
$$\omega_{k+1} = \frac{1}{2} \log \left(\frac{1 - e_k(h_{k+1})}{e_k(h_{k+1})} \right)$$

AdaBoost ("Adaptive Boosting") was formulated by Yoav Freund and Robert Schapire. The standard implementation updates the ν_t instead of computing them from scratch.

49 iterations



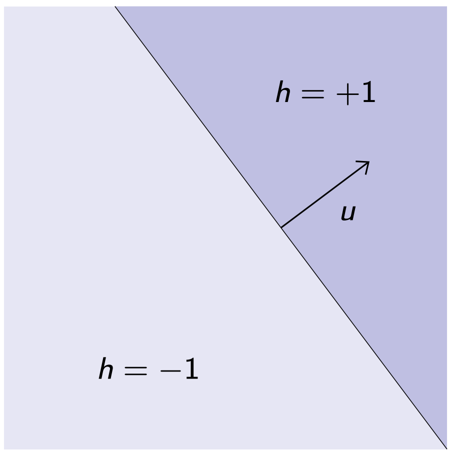




For our 2D toy problems, we use linear weak learners.

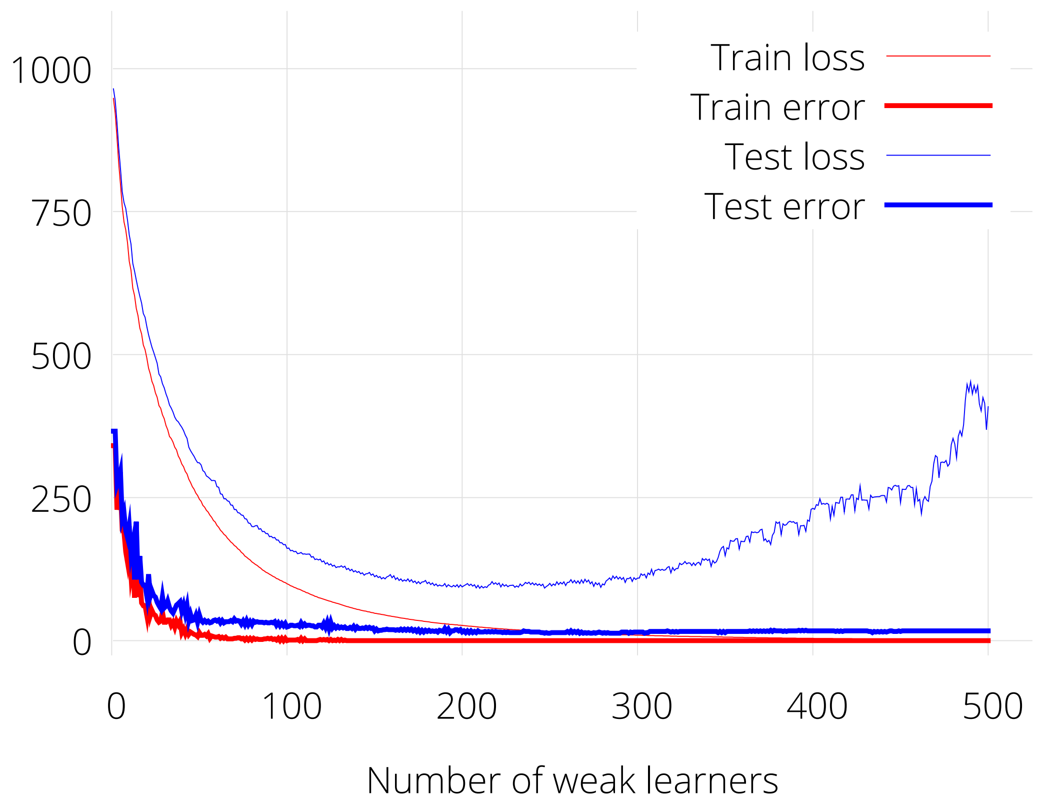
Each one is parametrized by a vector $u \in \mathbb{R}^D$ and a threshold $\rho \in \mathbb{R}$, and responds

$$h(x) = \begin{cases} +1 & \text{if } \langle x, u \rangle \geq \rho \\ -1 & \text{otherwise} \end{cases}$$

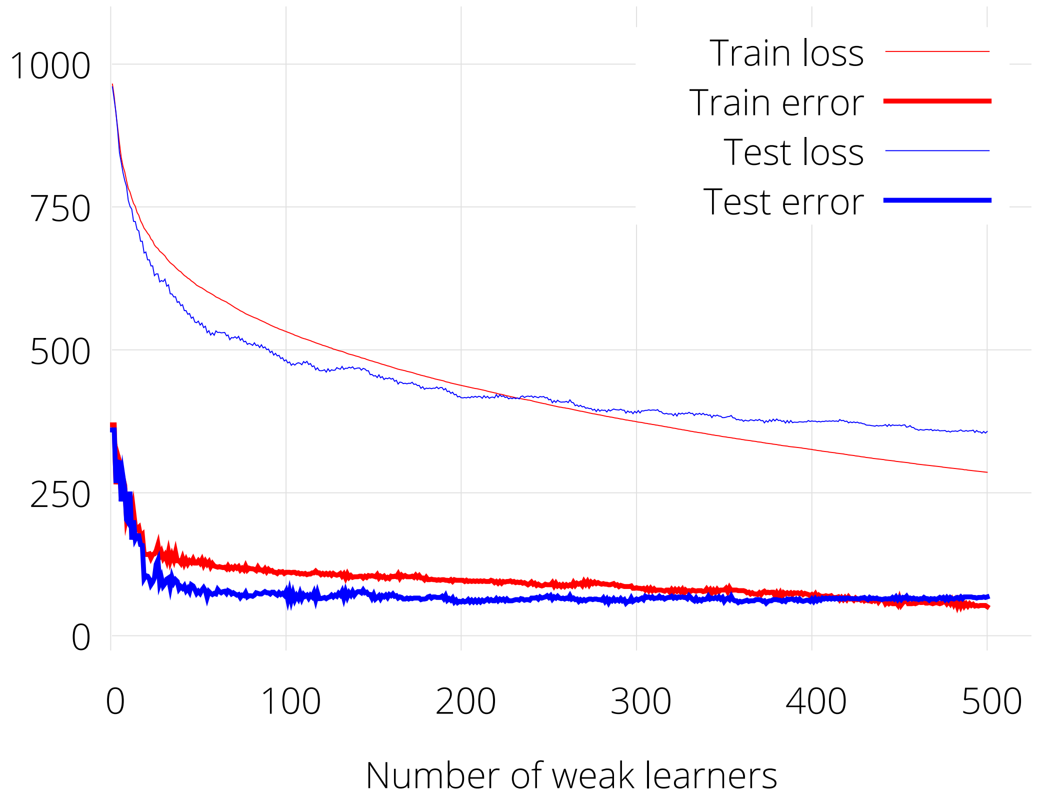


The choice of the weak learner is done by looking at a few u picked at random, with their optimal ρ.

(this slide contains videos showing the learning in progress)



(this slide contains videos showing the learning in progress)



The exponential loss gets lower when samples already well classified get an even stronger correct response.

Hence, even when the training error rate is already zero, the test error rate can still go down.

This phenomenon is related to the margin maximization: The learning process does not only put samples “on the right sides”; it pushes them as far as possible from the boundary.

Since the optimal weak learner is picked by optimizing a weighted error, Boosting can be used with any set of weak learners for which such an optimization is possible.

Moreover, by approximating the weighted error by an empirical average over a sampled subset of examples, Boosting can be used with virtually any learning method.

The most standard weak learners are decision trees, or stumps, which are decision trees of depth 1.

Making a strong predictor

We can wonder if choosing h_k and *then* ω_k is optimal.

What about picking directly

$$(h_{k+1}, \omega_{k+1}) = \operatorname*{argmin}_{h,\omega} L(f_k + \omega h)$$

or

$$h_{k+1} = \operatorname*{argmin}_{h} \left(\min_{\omega} L(f_k + \omega h) \right)$$

?

If we introduce

$$\begin{aligned} W_k^+(h) &= \sum_{t:Y_t=h(X_t)} \exp(-Y_t f_k(X_t)) \\ W_k^-(h) &= \sum_{t:Y_t\neq h(X_t)} \exp(-Y_t f_k(X_t)) \end{aligned}$$

we have

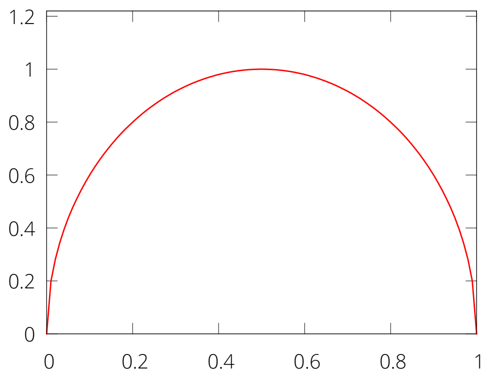
$$\begin{aligned} L(f_k + \omega h) &= \sum_t \exp\left(-Y_t \left(f_k(X_t) + \omega h(X_t)\right)\right) \\ &= \exp(-\omega) W_k^-(h) + \exp(\omega) W_k^+(h) \end{aligned}$$

and

$$\operatorname*{argmin}_{\omega} L(f_k + \omega h) = \frac{1}{2} \log \left(\frac{W_k^-(h)}{W_k^+(h)} \right).$$

Hence

$$\begin{aligned}
&\min_{\omega} L(f_k + \omega h)\\
&= L\left(f_k + \frac{1}{2}\log\frac{W_k^+(h)}{W_k^-(h)} h\right)\\
&= \sqrt{\frac{W_k^+(h)}{W_k^-(h)}} W_k^-(h) + \sqrt{\frac{W_k^-(h)}{W_k^+(h)}} W_k^+(h)\\
&= 2\sqrt{W_k^+(h) W_k^-(h)}\\
&= 2L(f_k)\sqrt{(1 - e_k(h))e_k(h)}
\end{aligned}$$



Maximizing $|e_k(h) - 1/2|$ minimizes $\sqrt{(1 - e_k(h))e_k(h)}$.

The weak-learner chosen by AdaBoost is the optimal one.

We have obtained

$$L(f_{k+1}) = \left(\min_h 2\sqrt{(1 - e_k(h))e_k(h)}\right) L(f_k)$$

Since e_k is a weighted error, we have the following results:

If

$$\exists \delta > 0,\ \forall \nu_1, \dots, \nu_T \geq 0,\ \sum_t \nu_t = 1,\ \exists h \in \mathscr{W},$$
$$\left| \sum_{t: Y_t \neq h(X_t)} \nu_t - \frac{1}{2} \right| \geq \delta$$

then

$$\exists \delta > 0, \forall k, \exists h \in \mathscr{W},\quad s.t.\ \ 2\sqrt{(1 - e_k(h))e_k(h)} \leq 1 - \delta$$

Hence, we have the following result about the convergence of the training loss to zero:

If

$$\exists \delta > 0,\ \forall \nu_1, \dots, \nu_T \geq 0,\ \sum_t \nu_t = 1,\ \exists h \in \mathscr{W},\ \left| \sum_{t: Y_t \neq h(X_t)} \nu_t - \frac{1}{2} \right| \geq \delta$$

then

$$\forall k,\ L(f_{k+1}) \leq (1 - \delta) L(f_k)$$

and

$$\lim_{k \to \infty} L(f_k) = 0.$$

The training error can be reduced arbitrarily.

Various remarks

Many practical algorithms uses "stumps" for weak learners, parametrized by a feature index d and a threshold ρ

$$h(x) = \begin{cases} -1 & \text{if } x^d < \rho \\ 1 & \text{otherwise.} \end{cases}$$

Optimizing the weak learner is usually done by sampling d and optimizing exactly ρ to maximize

$$R(d, \rho) = \Big| \underbrace{\sum_t -Y_t \exp(-Y_t f_k(X_t)) h(X_t)}_{S(d,\rho)} \Big|$$

If we sort the X_t

$$X^d_{t(1)} \leq X^d_{t(2)} \leq \dots \leq X^d_{t(T)}$$

and define

$$\rho(q) = \frac{X^d_{t(q)} + X^d_{t(q+1)}}{2}$$

we get

$$\begin{aligned} S(d, \rho(q)) &= -\sum_{r \leq q} -Y_{t(r)} \exp(-Y_{t(r)} f_k(X_{t(r)})) \\ &\quad + \sum_{r > q} -Y_{t(r)} \exp(-Y_{t(r)} f_k(X_{t(r)})) \\ &= S(d, \rho(q-1)) - 2 \exp(-Y_{t(q)} f_k(X_{t(q)})) \end{aligned}$$

So we find the best ρ by sorting the Xs according to x^d, and going through them once. Cost is $O(N \log N)$.

Given a functional f and a partitioning $D : \mathbb{R}^D \rightarrow \{1, \dots, C\}$, we want to find $\alpha_1, \dots, \alpha_C$ to minimize

$$L(f + \alpha_D) = \sum_t \exp\left(-y_t \left(f(x_t) + \alpha_{D(x_t)}\right)\right)$$

where α_D stands for a functional equal to α_d where D equals d.

$$\begin{aligned}
&\frac{\partial L(f + \alpha_D)}{\partial \alpha_d} \\
&\quad = \sum_{t:D(x_t)=d} -y_t \exp\left(-y_t \left(f(x_t) + \alpha_d\right)\right) \\
&\quad = \sum_{t:D(x_t)=d, y_t=1} - \exp\left(-f(x_t) - \alpha_d\right) + \sum_{t:D(x_t)=d, y_t=-1} \exp\left(f(x_t) + \alpha_d\right) \\
&\quad = \exp(-\alpha_d) \left(\sum_{t:D(x_t)=d, y_t=1} - \exp\left(-f(x_t)\right) + \sum_{t:D(x_t)=d, y_t=-1} \exp\left(f(x_t) + 2\alpha_d\right) \right)
\end{aligned}$$

Hence, solving

$$\frac{\partial L(f + \alpha_D)}{\partial \alpha_d} = 0$$

gives

$$\alpha_d = \frac{1}{2} \log \left(\frac{\displaystyle\sum_{t:D(x_t)=d, y_t=1} \exp\left(-f(x_t)\right)}{\displaystyle\sum_{t:D(x_t)=d, y_t=-1} \exp\left(f(x_t)\right)} \right)$$

The exponential loss can be interpreted as follows

$$L(F) = \sum_t \exp(-Y_t f(X_t)) = \hat{E}\left(\exp(-Yf(X))\right).$$

Since

$$\frac{\partial E\left(\exp(-Yf(X))\,|X = x\right)}{\partial f(x)} =$$
$$-P(Y = 1|X = x)\exp(-f(x)) + P(Y = -1|X = x)\exp(f(x)),$$

minimizing the exponential loss leads to

$$f(x) = \frac{1}{2}\log\frac{P(Y = 1|X = x)}{P(Y = -1|X = x)}$$

We can directly optimize $f(x)$ by setting

$$P(Y = 1\,|\,X = x) \quad = \quad \frac{1}{1 + \exp(-f(x))}$$

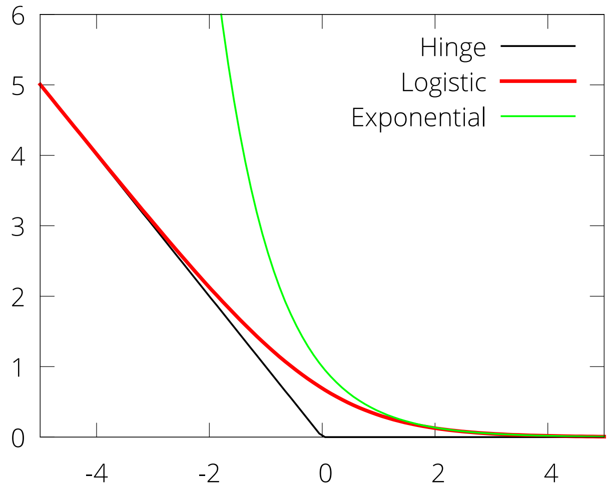
and then it makes sense to maximize

$$\log P(Y_1, \dots, Y_T\,|\,X_1, \dots, X_T) = -\sum_t \log\left(1 + \exp(-Y_t f(X_t))\right)$$

Logitboost, proposed by Friedman, Hastie and Tibshirani in 1998, minimizes the logistic loss

$$L(f) = \sum_t \log\left(1 + \exp(-Y_t f(X_t))\right)$$

with a Newton procedure.



Rankboost is designed to build a functional H such that the values over training samples are well-ordered.

We set a distribution D on $\mathscr{X} \times \mathscr{X}$ and we want to minimize

$$rloss(H) = \sum_{a,\,b} D(a,b)\mathbf{1}_{\{H(a) \leq H(b)\}}$$

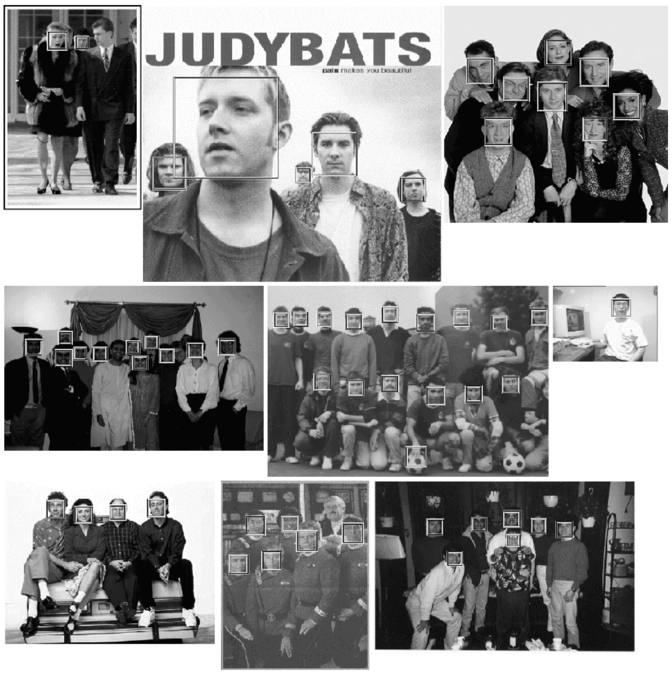
Let set $D_1 = D$. For $t = 1, \dots, T$

- Pick h_t to minimize $\sum_{a,\,b} D_t(a,b)\mathbf{1}_{\{H(a) \leq H(b)\}}$
- Choose ω_t
- Update $D_{t+1}(a,b) = \dfrac{D_t(a,b)\exp(\omega_t(h_t(a) - h_t(b))}{Z}$

We like Boosting for several reasons:

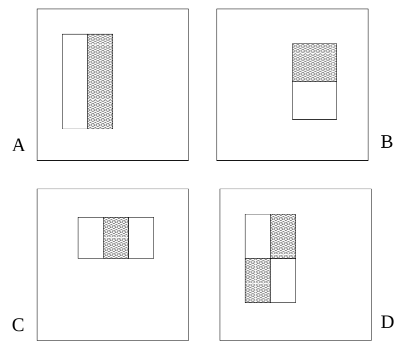
- It is simple.
- It is intuitive.
- It derives from a meaningful loss minimization.
- It allows to plug in whatever learning algorithm of choice.
- It works (limited/no over-fitting in practice).

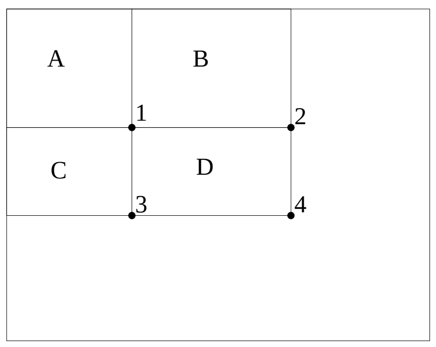
Face detection



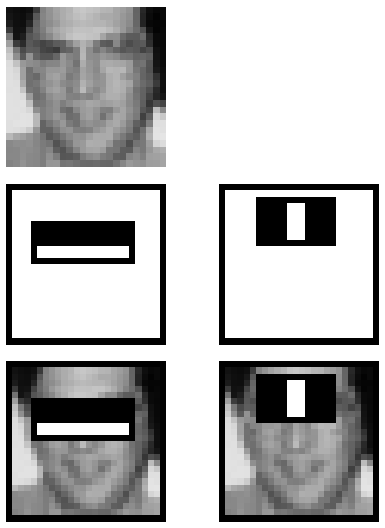
(Viola & Jones 2001)

(Viola & Jones 2001)

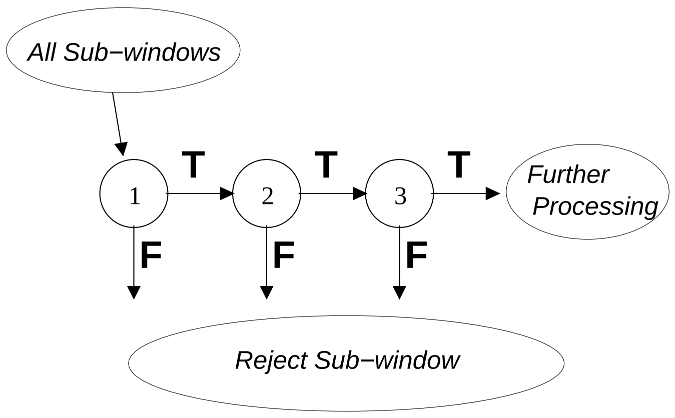




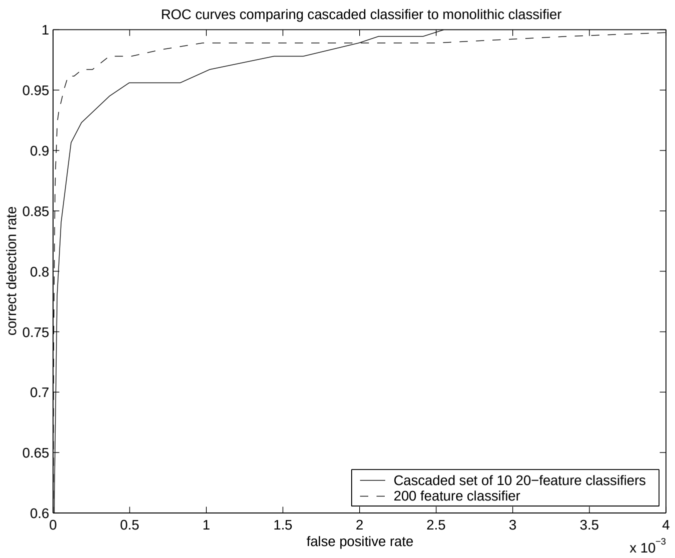
(Viola & Jones 2001)



(Viola & Jones 2001)



(Viola & Jones 2001)



(Viola & Jones 2001)

References

F. Fleuret. Fast binary feature selection with conditional mutual information. Journal of Machine Learning Research (JMLR), 5:1531–1555, 2004.