EE-613: Machine Learning for Engineers

Feature extraction and selection

(and a bit of PAC)

François Fleuret

Dec 16th, 2015

Introduction

Dimension reduction

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

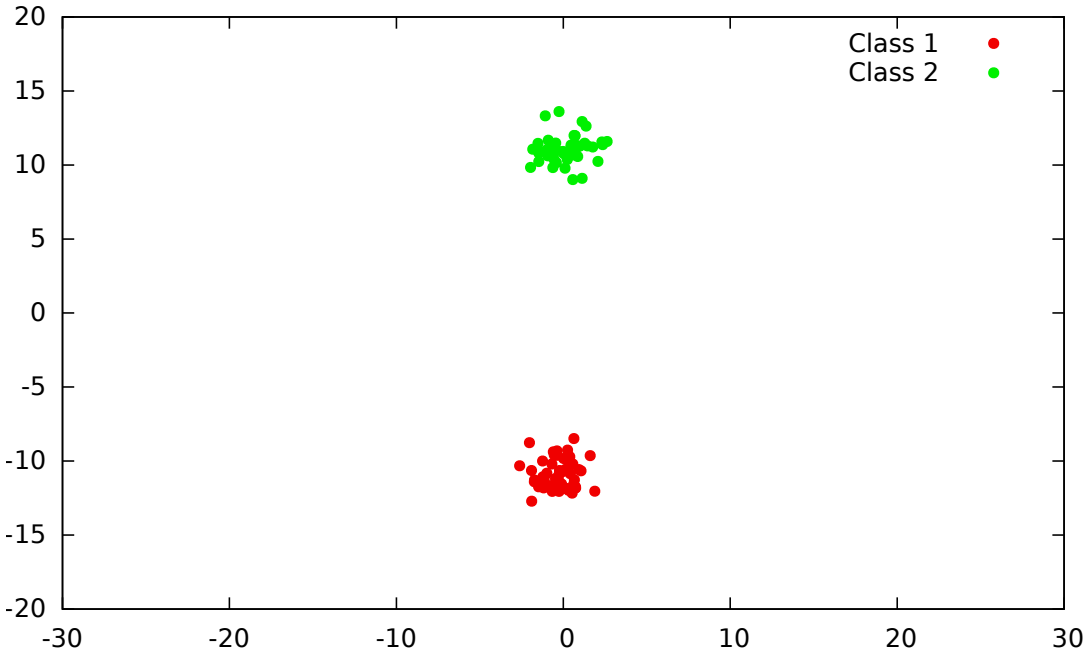
- Reduce over-fitting
- Reduce the computational cost
- Facilitate the understanding of causal relations
- Facilitate the visualization of data

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

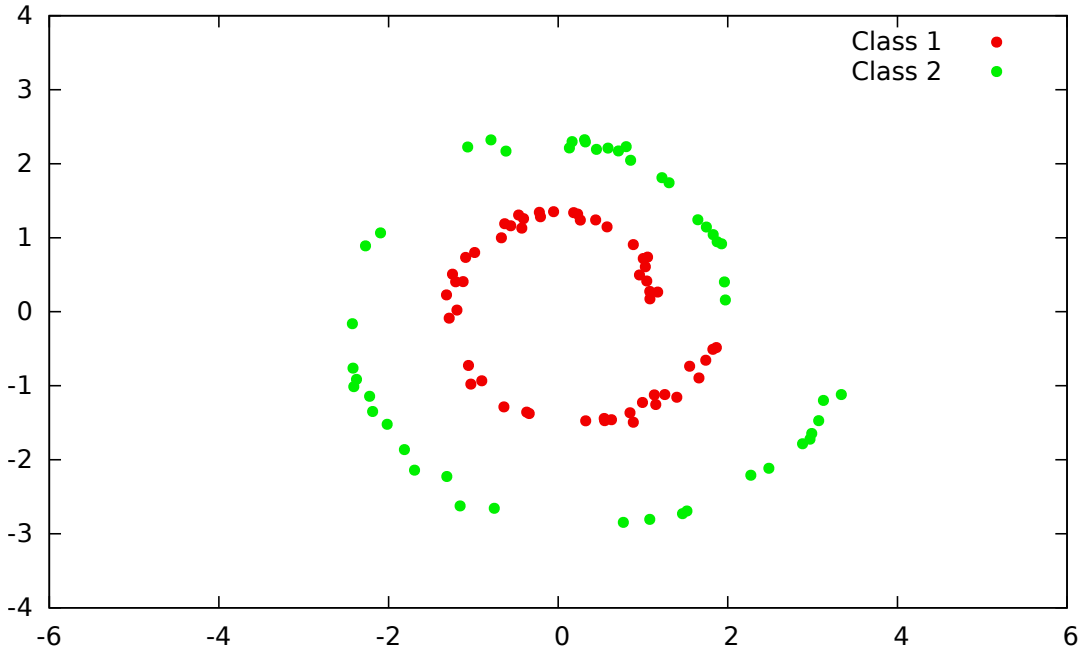
There are two main strategies to achieve dimension reduction:

- **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.

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

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

such that

$$X' = (X_{\nu(1)}, \ldots, X_{\nu(d)})$$

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.

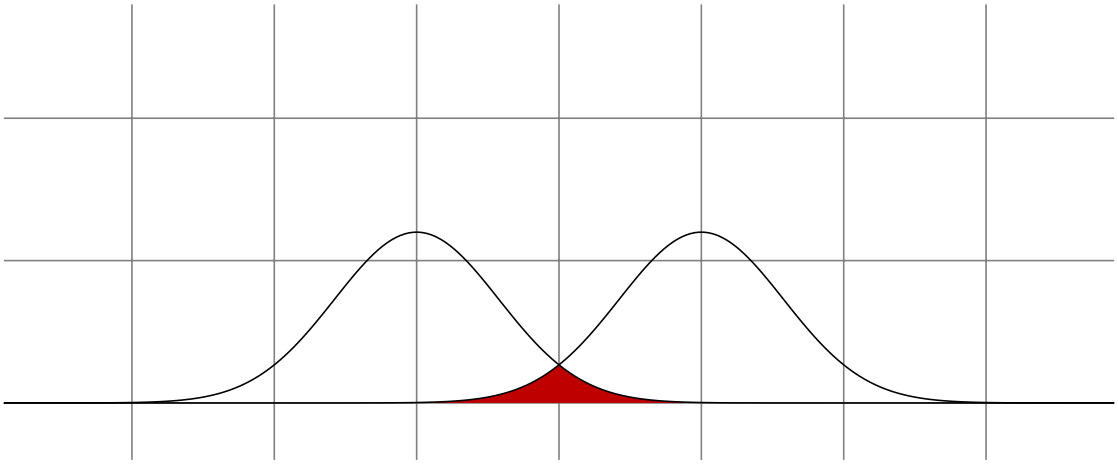
The alternative is a **filter**, which is predictor-agnostic.

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

Forward selection based on marginal information
Fisher score

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

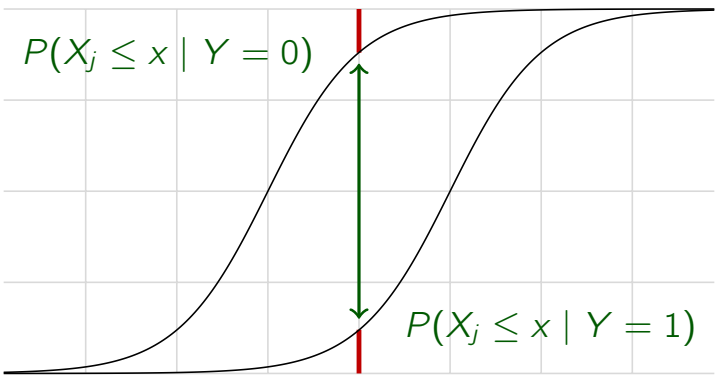
$$S(X_j) = \frac{\sum_{i=1}^{C}(E(X_j|Y=i) - E(X_j))^2 P(Y=i)}{\sum_{i=1}^{C} V(X_j|Y=i)P(Y=i)}$$



Forward selection based on marginal information
Kolmogorov-Smirnov test

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 $\{1, \ldots, C\}$ 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 $X_1, \ldots, X_D$ i.i.d. of distribution $\mathcal{B}(0.5)$, and

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

Then

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

and

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

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

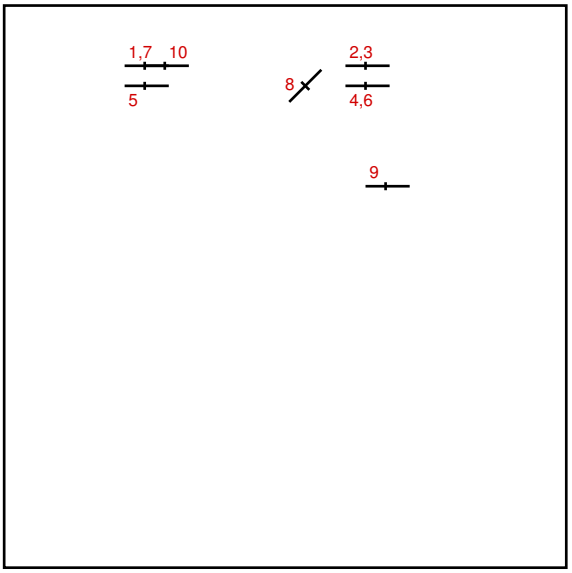
Selected features should be individually informative and weakly dependent.

CMIM selects features as follows:

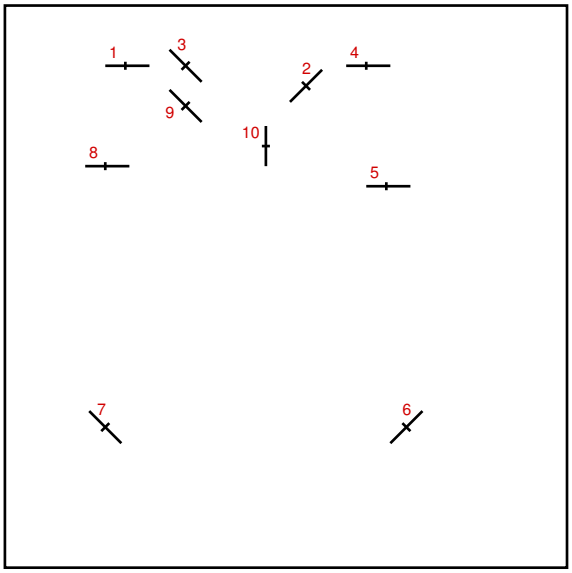
$$\begin{aligned} \nu(1) &= \arg\max_n I(Y, X_n) \\ \forall k,\ 1 \le k < K, \quad \nu(k+1) &= \arg\max_n \left\{ \min_{l \le k} I\left(Y, X_n \,|\, X_{\nu(l)}\right) \right\} \end{aligned}$$

Face vs. non-face patches.

Forward selection based on marginal information
CMIM (cont.)



Mututal information



Conditional Mutual Information

Forward selection based on marginal information
CMIM (cont.)

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

$$S_w = \sum_{i=1}^{n} (x_i - \bar{x}_{y_i})(x_i - \bar{x}_{y_i})^T$$

and the between-class scatter matrix

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

maximize the ratio between the two

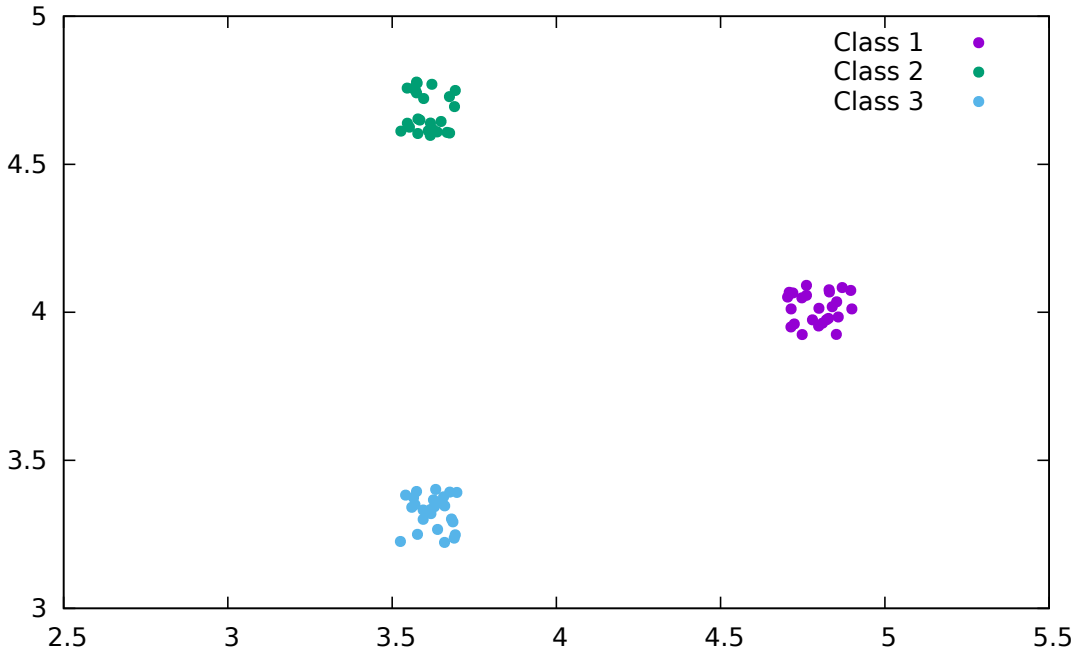
$$\hat{w} = \arg\max_{w} \frac{w^T S_b w}{w^T S_w w}$$

which amounts to solving

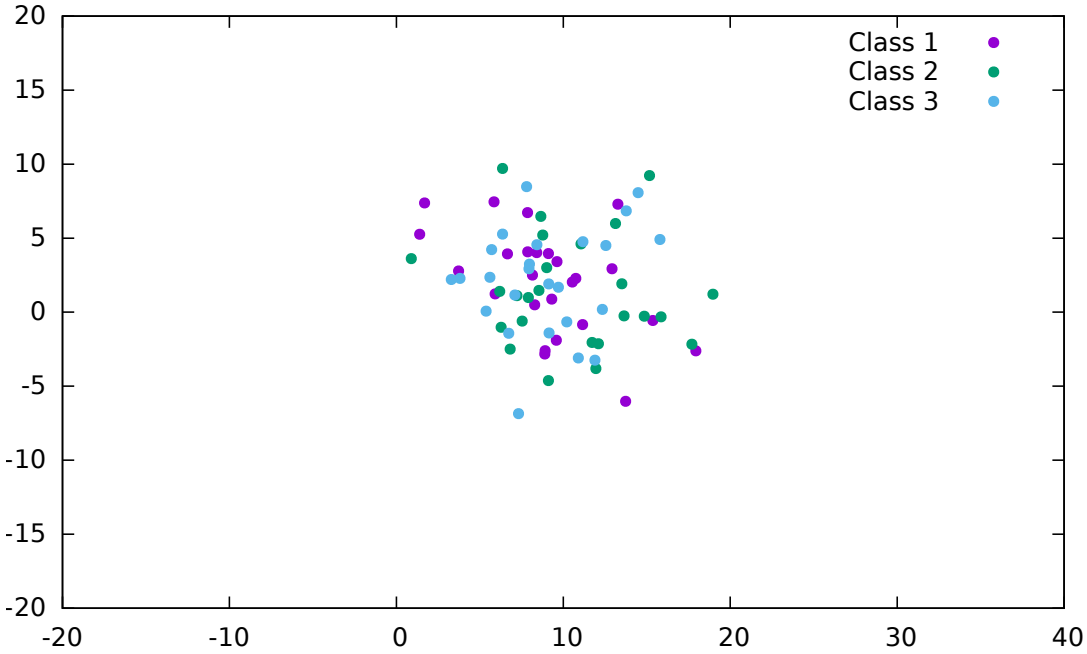
$$S_b w = \lambda S_w w.$$

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

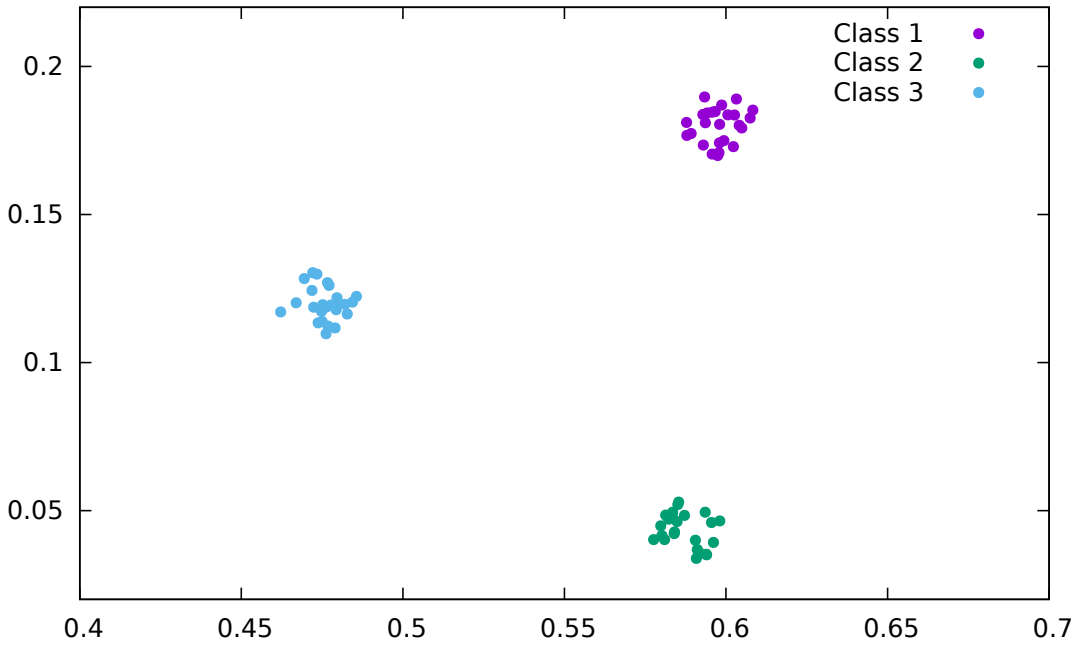
We consider the following population



We add 8 additional random dimensions $\sim \mathcal{U}[0, 10]$ 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

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

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

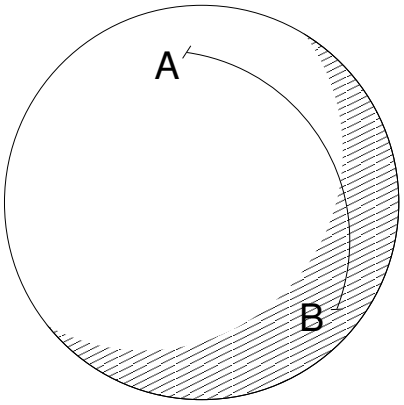
$$\sum_n \min_c \|x_n - \alpha_c\|^2$$

The extracted discrete feature is

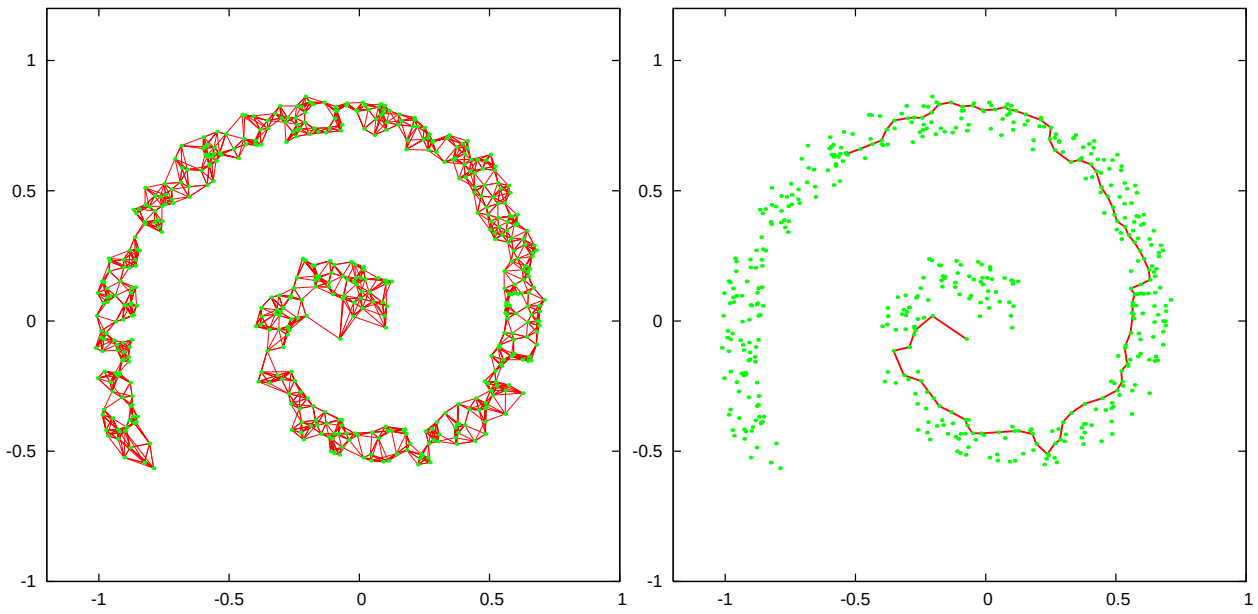
$$\arg\min_c \|x_n - \alpha_c\|^2 .$$

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 $x_1, \ldots, x_n \in \mathbb{R}^d$, 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 $y_1, \ldots, y_n$ in $\mathbb{R}^r$ such that

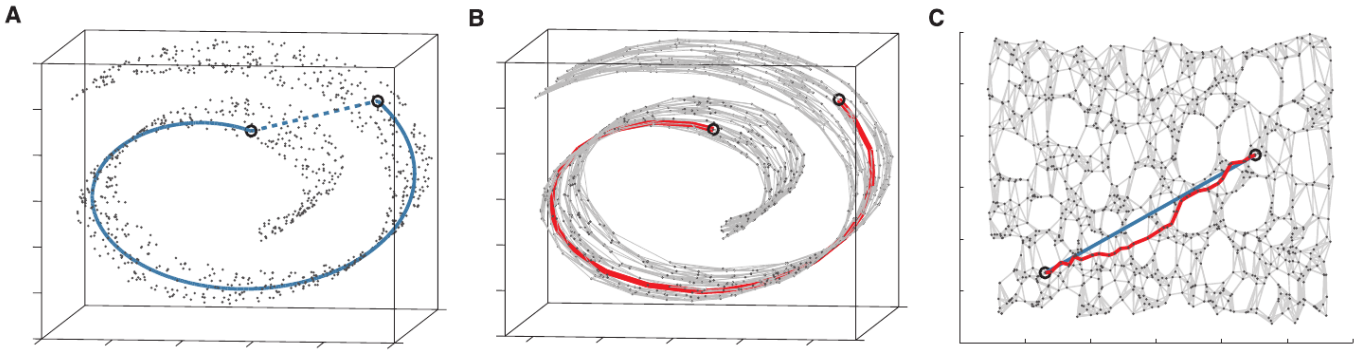
$$\forall i, j, \ \Delta(i,j) \simeq \|y_i - y_j\|$$

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

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

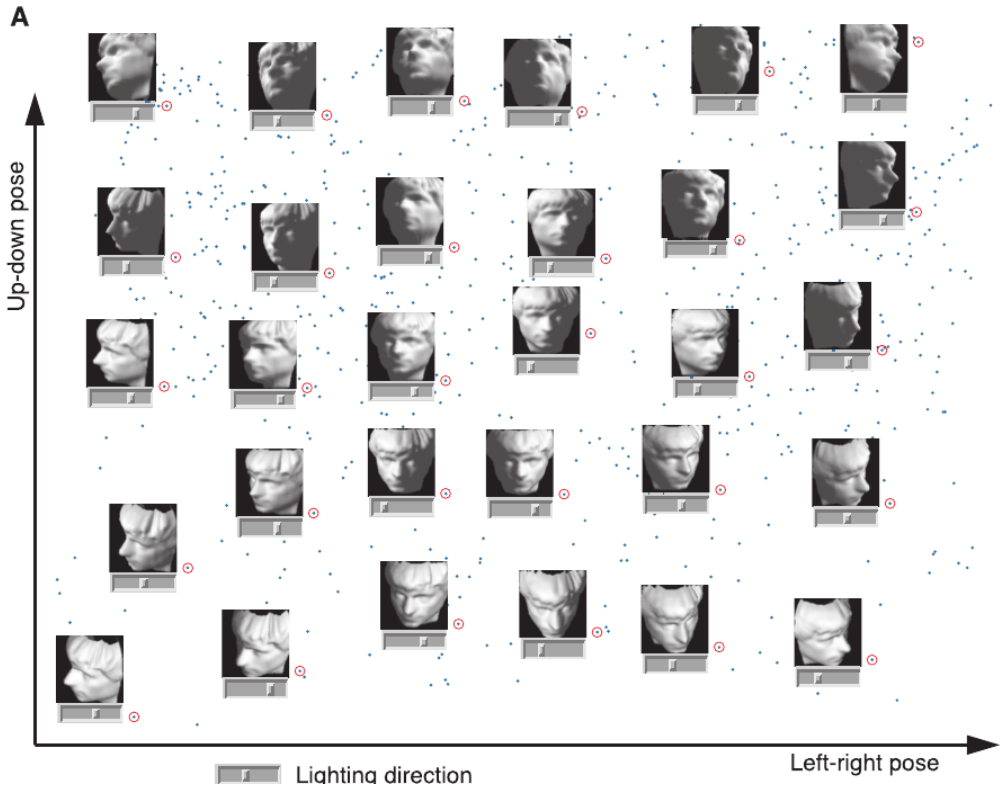
Hence we finally have associated one y_i in $\mathbb{R}^r$ to every x_i.

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



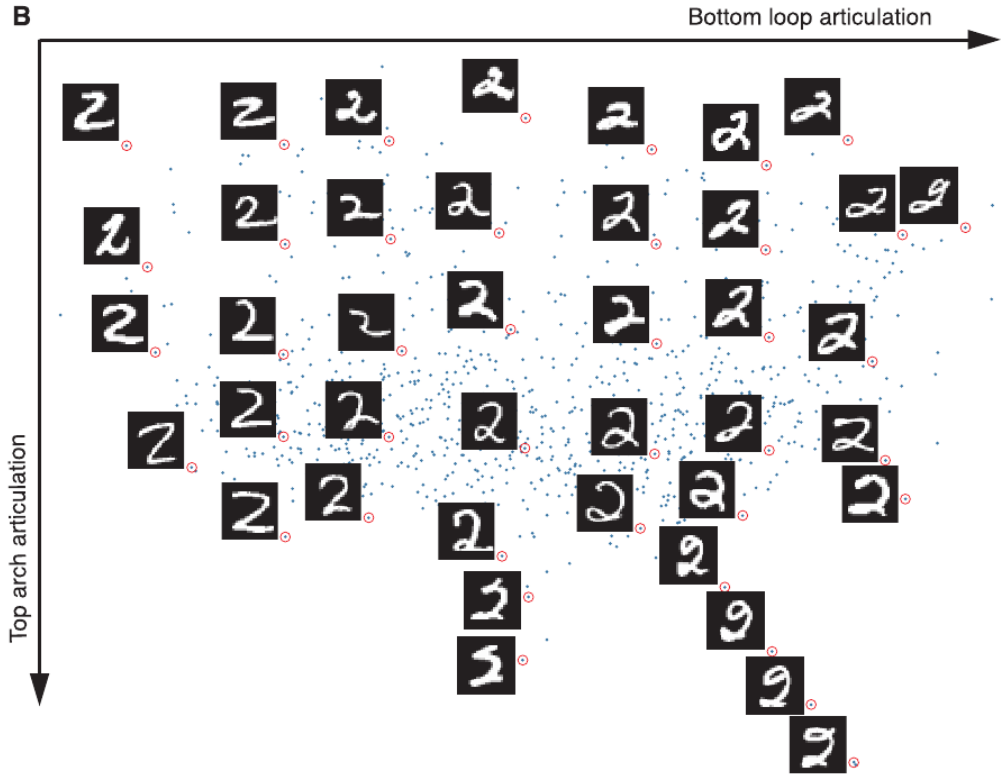
(J. B. Tenenbaum, V. de Silva and J. C. Langford, 2000)

Feature extraction
Isomap (cont.)



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

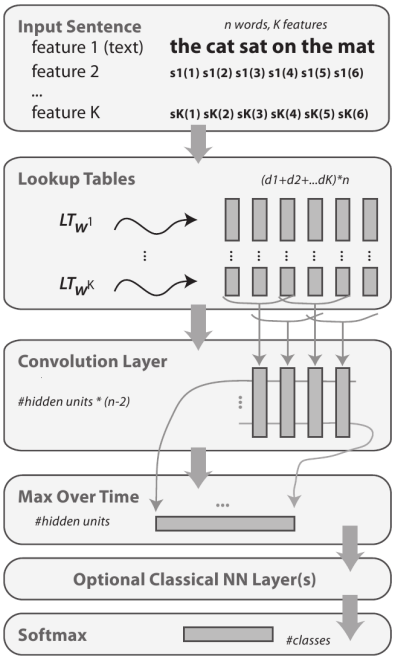
Isomap
Digits

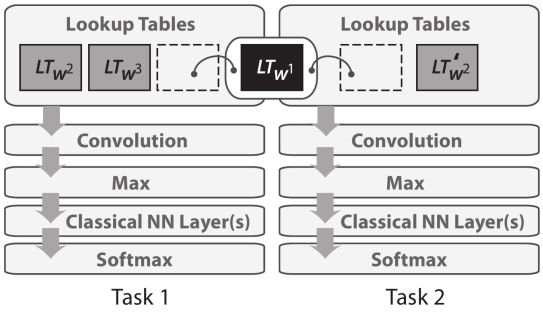


(J. B. Tenenbaum, V. de Silva and J. C. Langford, 2000)

Neural networks
Multi-task learning for NLP

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)

Neural networks
Multi-task learning for NLP

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

Regularization and sparsity

We have seen that for linear regression one minimize

$$L(\alpha) = \sum_n \left(\sum_d \alpha_d x_n^d - y_n \right)^2$$

and that we can add a L^2 penalty on the coefficients (aka "Ridge regression")

$$L(\alpha) = \sum_n \left(\sum_d \alpha_d x_n^d - y_n \right)^2 + \lambda \sum_d \alpha_d^2$$

such a penalty favor smaller and more homogeneous weights. It also makes finding the optimal parameters properly conditioned for small data sets.

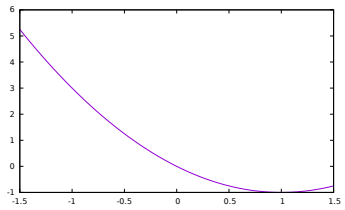
Using a L^1 penalty

$$L(\alpha) = \sum_n \left(\sum_d \alpha_d x_n^d - y_n \right)^2 + \lambda \sum_d |\alpha_d|$$

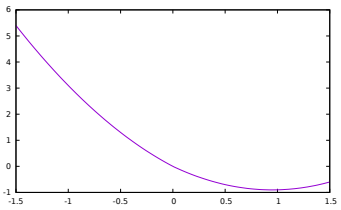
leads the least important model parameters to zero, since the improvement they bring has to be greater than λ for their optimal value to be non-zero.

This is still convex !

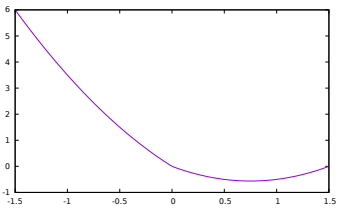
Regularization and sparsity
LASSO (cont.)



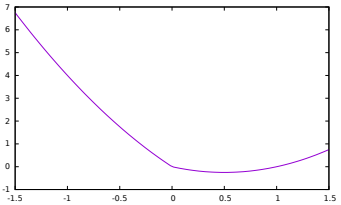
$(x-1)^2$



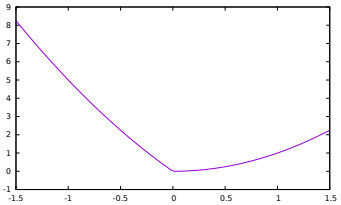
$(x-1)^2 + 0.1|x|$



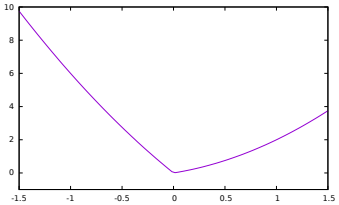
$(x-1)^2 + 0.5|x|$



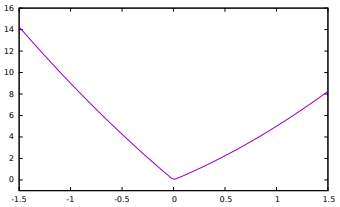
$(x-1)^2 + 1.0|x|$



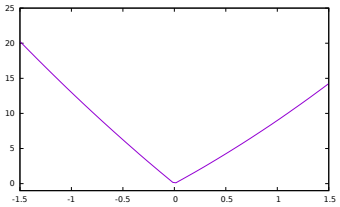
$(x-1)^2 + 2.0|x|$



$(x-1)^2 + 3.0|x|$



$(x-1)^2 + 6.0|x|$



$(x-1)^2 + 10.0|x|$

Regularization and sparsity
Toy example

We consider

$$\beta = (1, 2, 3, 4, 5, 0, 0, 0, 0, 0)$$

and $\forall n = 1, \dots, N$

$$x_n \sim \mathcal{N}(0, 1)$$

and

$$y_n = \langle \beta, x_n \rangle + \mathcal{N}(0, 1).$$

We estimate

$$\hat{\beta} = \arg\min_{\beta} \sum_n (y_n - \langle \beta, x_n \rangle)^2 + \lambda_2 \sum_d \beta_d^2 + \lambda_1 \sum_d |\beta_d|.$$

Regularization and sparsity
Toy example (cont.)

λ_1	λ_2	$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$	$\hat{\beta}_4$	$\hat{\beta}_5$	$\hat{\beta}_6$	$\hat{\beta}_7$	$\hat{\beta}_8$	$\hat{\beta}_9$	$\hat{\beta}_{10}$
-	-	1.037	2.033	2.999	3.995	5.041	0.039	-0.002	-0.055	0.005	0.084
-	1.000	1.037	2.030	2.996	3.991	5.035	0.039	-0.002	-0.054	0.005	0.084
-	2.000	1.035	2.027	2.992	3.987	5.029	0.040	-0.002	-0.054	0.005	0.084
-	4.000	1.033	2.024	2.987	3.978	5.020	0.040	-0.002	-0.054	0.004	0.084
-	8.000	1.028	2.017	2.975	3.962	5.003	0.040	-0.003	-0.054	0.003	0.084
-	16.000	1.020	2.002	2.951	3.932	4.964	0.041	-0.004	-0.053	0.001	0.084
-	32.000	1.003	1.973	2.903	3.871	4.895	0.043	-0.007	-0.052	-0.004	0.084
-	64.000	0.972	1.918	2.814	3.756	4.758	0.046	-0.011	-0.050	-0.012	0.084
-	128.000	0.913	1.818	2.651	3.542	4.500	0.050	-0.018	-0.046	-0.026	0.084
-	256.000	0.814	1.644	2.377	3.182	4.067	0.056	-0.027	-0.040	-0.046	0.083
-	512.000	0.670	1.380	1.968	2.644	3.404	0.061	-0.037	-0.032	-0.067	0.077
-	1024.000	0.493	1.045	1.463	1.975	2.571	0.058	-0.040	-0.022	-0.077	0.065
-	2048.000	0.323	0.703	0.966	1.310	1.724	0.047	-0.035	-0.014	-0.069	0.048
-	4096.000	0.191	0.424	0.573	0.781	1.037	0.032	-0.024	-0.008	-0.050	0.031
1.000	-	1.037	2.032	2.999	3.994	5.040	0.039	-0.001	-0.054	0.005	0.083
2.000	-	1.037	2.031	2.998	3.993	5.040	0.038	-0.001	-0.053	0.004	0.083
4.000	-	1.036	2.028	2.997	3.992	5.039	0.037	-	-0.052	0.003	0.081
8.000	-	1.034	2.027	2.995	3.991	5.035	0.035	-	-0.050	0.001	0.079
16.000	-	1.031	2.023	2.991	3.986	5.029	0.032	-	-0.046	-	0.075
32.000	-	1.023	2.016	2.984	3.978	5.024	0.024	-	-0.038	-	0.066
64.000	-	1.008	2.003	2.965	3.961	5.010	0.008	-	-0.022	-	0.048
128.000	-	0.978	1.975	2.930	3.929	4.983	-	-	-	-	0.014
256.000	-	0.917	1.916	2.865	3.864	4.925	-	-	-	-	-
512.000	-	0.793	1.800	2.734	3.734	4.802	-	-	-	-	-
1024.000	-	0.546	1.564	2.474	3.471	4.570	-	-	-	-	-
2048.000	-	0.051	1.093	1.952	2.941	4.092	-	-	-	-	-
4096.000	-	-	0.090	0.876	1.947	3.149	-	-	-	-	-

Regularization and sparsity
Remarks

Some remarks

- The L^1 and L^2 penalty terms can be combined (elastic net)
- It can be shown that LASSO is equivalent to SVM
- All these regularization can be interpreted in a Bayesian framework with the proper priors on the parameters
- As for feature selection, these methods can be used both as a regularizer to mitigate over-fitting and reduce the computational cost, or as a mean to interpret the data

PAC bounds

Introduction
Classification

The usual setting for learning for classification:

- A training set,
- a family of classifiers,
- a test set.

Learning means to choose a classifier according to its performances on the training set to get good performances on the test set.

We will use the following notation:

- $\mathcal{X}$ the space of the objects to classify (for instance images)
- $\mathcal{C}$ the family of classifiers
- $S = ((X_1, Y_1), \ldots, (X_{2N}, Y_{2N}))$ a random variable on $(\mathcal{X} \times \{0, 1\})^{2N}$ standing for the training and test samples.
- F a random variable on $\mathcal{C}$ standing for the learned classifier. It can be a deterministic function of S or not.

(1) The set $\mathcal{C}$ contains all the classifiers obtainable with the learning algorithm.

For an ANN for instance, there is one element of $\mathcal{C}$ for every single configuration of the synaptic weights.

(2) The variable S is not one sample, but a family of $2N$ samples with their labels. It contains both the training and the test set.

Gap between training and test error
One fixed f

For every $f \in \mathcal{C}$, let $\xi(f, S)$ denote the difference between the training and the test errors of f.

$$\xi(f, S) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} 1\{f(X_{N+i}) \neq Y_{N+i}\}}_{\text{test error}} - \underbrace{\frac{1}{N}\sum_{i=1}^{N} 1\{f(X_i) \neq Y_i\}}_{\text{training error}}$$

Where $1\{t\}$ is equal to 1 if t is true, and 0 otherwise. Since S is random, this is a random quantity.

Gap between the test and the training error
Data-dependent f

Given $\eta \geq 0$, we want to bound the probability that the test error is less than the training error plus η.

$$P\left(\xi(F, S) \leq \eta\right) \quad \geq \quad ?$$

Not that here F is not constant anymore and is a function of the samples $X_1, \ldots, X_{2N}$ and the $Y_1, \ldots, Y_N$.

Where we see that for any fixed f, the test and training errors are likely to be similar ...

Given a family of independent random variables $Z_1, \ldots, Z_N$, bounded $\forall i, Z_i \in [a_i, b_i]$, if S denotes $\sum_i Z_i$, then $\forall t \geq 0$ we have Hœffding's inequality (1963)

$$P(S - E(S) > t) \quad \leq \quad \exp\left(-\frac{2t^2}{\sum_i (b_i - a_i)^2}\right)$$

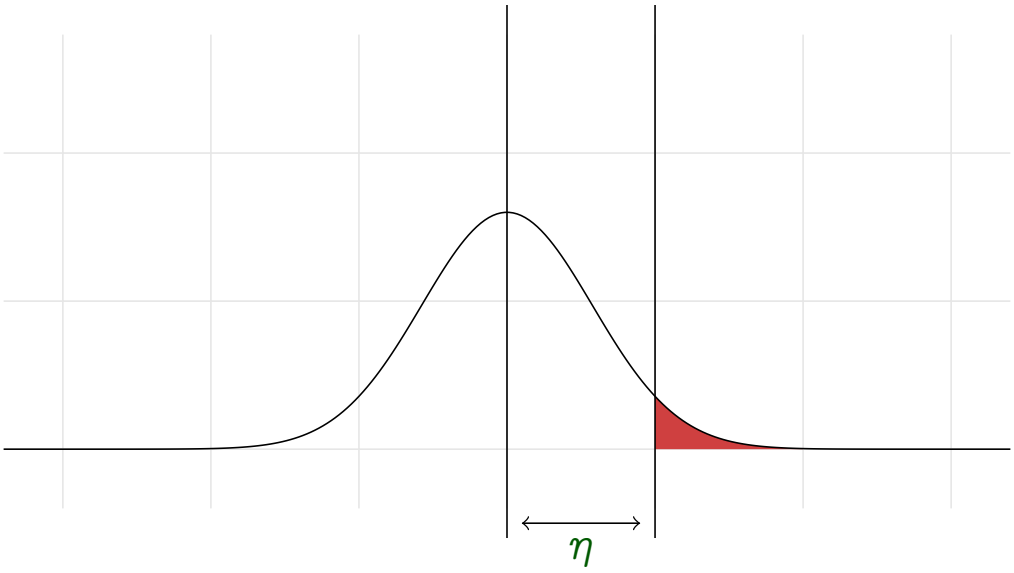
This is an concentration result: It tells how much S is concentrated around its average value.

Note that the $1\{f(X_i) \neq Y_i\}$ are i.i.d Bernoulli, and we have

$$\begin{aligned} \xi(f, S) &= \frac{1}{N}\sum_{i=1}^{N} 1\{f(X_{N+i}) \neq Y_{N+i}\} - \frac{1}{N}\sum_{i=1}^{N} 1\{f(X_i) \neq Y_i\} \\ &= \frac{1}{N}\sum_{i=1}^{N} \underbrace{1\{f(X_{N+i}) \neq Y_{N+i}\} - 1\{f(X_i) \neq Y_i\}}_{\Delta_i} \end{aligned}$$

Thus ξ is the averaged sum of the Δ_i, which are i.i.d random variables on $\{-1, 0, 1\}$ of zero mean.

Hence, when f is fixed we have (Hœffding):



$$\forall f,\ \forall \eta \geq 0,\quad P\left(\xi(f, S) > \eta\right) \leq \exp\left(-\frac{1}{2}\eta^2 N\right)$$

Where we realize that the probability the chosen F fails is lower than the probability that there exists an f that fails . . .

We have

$$\begin{aligned} P(\xi(F, S) > \eta) &= \sum_f P(F = f, \xi(F, S) > \eta) \\ &= \sum_f P(F = f, \xi(f, S) > \eta) \\ &\leq \sum_f P(\xi(f, S) > \eta) \\ &\leq \|\mathcal{C}\| \exp\left(-\frac{1}{2}\eta^2 N\right) \end{aligned}$$

This is our first generalization bound!

Union bound
We can fix the probability

We can conversely express the gap as a function of the probability we want.

If we define

$$\epsilon^\star \quad = \quad \|\mathcal{C}\| \, \exp\left(-\frac{1}{2}\,\eta^2\,N\right)$$

We have

$$\sqrt{2\frac{\log\|\mathcal{C}\| - \log\epsilon^\star}{N}} \quad = \quad \eta$$

Union bound
We can fix the probability

Hence from

$$P(\xi(F,\,S) > \eta) \quad \leq \quad \|\mathcal{C}\| \, \exp\left(-\frac{1}{2}\,\eta^2\,N\right)$$

we get

$$P\left(\xi(F,\,S) > \sqrt{2\,\frac{\log\|\mathcal{C}\| \,+\, \log\frac{1}{\epsilon^\star}}{N}}\right) \quad \leq \quad \epsilon^\star$$

Thus, with probability $1 - \epsilon^\star$, the gap between the train and test errors grows as the square root of the log of the nb. of classifiers.

Proving a bound with "real" predictors is in general a bit technical.

However, if we fix the number of weak learners to N, impose that their coefficients takes only the values ± 1, and bound the number of Boosting rounds to M, we know that we have a total of

$$(2N)^M$$

possible predictors, and we can use our bound directly.

For decision trees, we can do something similar.

If we have N possible Boolean decisions to put at the nodes, and the depth is bounded by D, then there are at most

$$N^{2^{D-1}}$$

trees.

Where we realize that we can arbitrarily distribute allowed errors on the fs before looking at the training data . . .

Until now, the only quantity we control is $||\mathcal{C}||$: If we *know* that certain mappings can be removed without hurting the train error, we can remove them and get a better bound.

Can we do something better than that?

We introduce $\eta(f)$ as the control we want between the train and test error if f is chosen. Until now, this was constant.

Now we want to bound the probability that the gap between the training and the test error for f is lower than

Prior on $\mathcal{C}$
Let make η depend on F

Let $\epsilon(f)$ denote the (bound on the) probability that the constraint is not verified for f:

$$\begin{aligned} P(\xi(F, S) > \eta(F)) &\leq P\left(\exists f \in \mathcal{C}, \xi(f, S) > \eta(f)\right) \\ &\leq \sum_f P(\xi(f, S) > \eta(f)) \\ &\leq \sum_f \epsilon(f) \end{aligned}$$

and we have

$$\forall f,\ \eta(f) = \sqrt{2 \frac{\log \frac{1}{\epsilon(f)}}{N}}$$

Prior on $\mathcal{C}$
Let make η depend on F

Let define $\epsilon^\star = \sum_f \epsilon(f)$ and $\rho(f) = \frac{\epsilon(f)}{\epsilon^\star}$. The later is a distribution on $\mathcal{C}$.

Note that both can be fixed arbitrarily, and we have

$$\forall f,\ \eta(f) = \sqrt{2 \frac{\log \frac{1}{\rho(f)} + \log \frac{1}{\epsilon^\star}}{N}}$$

Prior on $\mathcal{C}$
Let's put everything together

Our final result is that, if

- we choose a distribution ρ on $\mathcal{C}$ arbitrarily,
- we choose $0 < \epsilon^\star < 1$ arbitrarily,
- we sample a pair S training set / test set each of size N,
- we choose a F after looking at the training set.

Then, we have with probability greater than $1 - \epsilon^\star$:

$$\xi(F, S) \leq \sqrt{2 \frac{\log \frac{1}{\rho(F)} + \log \frac{1}{\epsilon^\star}}{N}}$$

where $\xi(F, S)$ is the difference between the test and train errors.

Prior on $\mathcal{C}$
This is a philosophical theorem!

If we see $-\log \rho(f)$ as the "description" length of f (think Huffman). Our result (true with probability $1 - \epsilon^\star$)

$$\xi(F, S) \leq \sqrt{2 \frac{\log \frac{1}{\rho(F)} + \log \frac{1}{\epsilon^\star}}{N}}$$

says that picking a classifier with a long description leads to a bad control on the test error.

Entities should not be multiplied unnecessarily.

Principle of parsimony of William of Occam (1280 – 1349). Also known as Occam's Razor.

Practical session

Practical session
Random vs. wrapper vs. filter

If you type

```
wget http://www.idiap.ch/~fleuret/files/EE613/EE613-pw14.tgz
tar zxvf EE613-pw14.tgz
cd EE613/pw14
wget http://www.idiap.ch/~fleuret/files/EE613/mnist.tgz
tar zxvf mnist.tgz
./do.sh demo
```

you should obtain after a few seconds a printout ending with

```
Training the decision tree ... done.
************************************************************
Using 784 questions.
Train error 0 %
Test error  0.83 %
************************************************************
```

Practical session
Random vs. wrapper vs. filter

We re-use the classes we had for decision trees, and will try to find good subsets of questions to train them.

```
class DecisionTree : public Classifier {
public:
  DecisionTree(QuestionSet *question_set);
  virtual ~DecisionTree();
  virtual void train(VignetteSet *train_set);
  virtual int predict(VignetteSet *vs, int n_vignette);
};

class QuestionSet {
public:
   virtual int nb_questions() = 0;
   virtual bool response(int n_question,
                         VignetteSet *vs, int n_vignette) = 0;
};
```


Practical session
Random vs. wrapper vs. filter

We have a new class which models a subset of an original set of questions:

```
class SubQuestionSet : public QuestionSet {
public:
  SubQuestionSet(QuestionSet *source, int *selected);
  ~SubQuestionSet();
  virtual int nb_questions();
  virtual bool response(int n_question,
                        VignetteSet *vs, int n_vignette);
};
```

So for instance, if we want to train a tree using the 10 first questions of a PixelQuestionSet, we would write:

```
PixelQuestionSet qs(train_image_set->width(),
                    train_image_set->height());

int selected[qs.nb_questions()];
for(int k = 0; k < qs.nb_questions(); k++) {
  if(k < 10) {
    selected[k] = 1;
  } else {
    selected[k] = 0;
  }
}

SubQuestionSet sqs(&qs, selected);
print_errors_for_question_set(train_image_set, test_image_set,
                              &sqs);
```


We have the three following functions to help us:

```
scalar_t mutual_information(int nb, int *x, int *y);

void tag_subset(int nb_total, int nb_to_tag,
                int *tagged);

void tag_the_top_ones(int nb_total, scalar_t *scores, int nb_to_tag,
                      int *tagged);
```

François Fleuret
IDIAP Research Institute
francois.fleuret@idiap.ch
http://www.idiap.ch/~fleuret