High dimensional statistics for genomic data

Laurent Jacob

February 28, 2017

Summary of the previous class

- Relationship to ML estimation.
- Validation.

Outline

1. Non-parametric methods:
   - Kernel methods.
   - Deep learning.
2. Unsupervised learning.

Part IV

Non-parametric methods

Parametric vs non parametric

- Methods presented so far are linear, parametric (but ridge penalized techniques can be kernelized).
- Some techniques (nonparametric) intend to make fewer assumptions: often means that the estimators we consider are not expressed as a parametric function of the descriptors.
- Sometimes used to say that we are not relying on a parameterized family of probability distribution. But SVM would fall in this category.
- Actually hard to draw a clear line between parametric and nonparametric:

> *It is difficult to give a precise definition of nonparametric inference, and if I did venture to give one, no doubt I would be barraged with dissenting opinion.*
>
> L. Wasserman, *All of Nonparametric Statistics*

Parametric vs non parametric

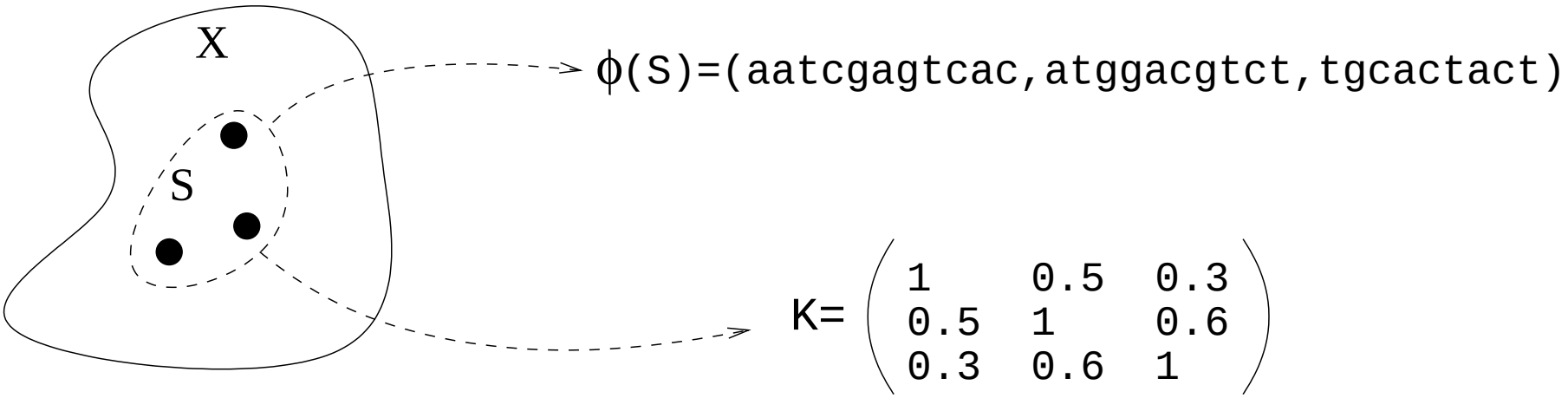
- Risk of confusion between "fewer assumptions" and "assumptions which are not clearly formulated".
- What matters is how we control the complexity of the family of functions we consider.
- Nonparametric techniques correspond to particular types regularity, it is useful to try and understand what this type is.
- Often corresponds to functions which can be written as a parametric function of the training set, or some nonlinear dictionary.

Kernel methods (adapted from JP Vert)

Representation by pairwise comparisons

- All supervised learning algorithms we have seen rely on some vector **representation** of the samples as vectors.
- Finding such a description is not always simple:
  - Which descriptors (think of the protein or molecule examples in the introduction).
  - Too many descriptors make the algorithms untractable.
  - A choice of descriptor does not necessarily make the classes linearly separable.
- Positive definite kernels address these issues in some cases.

Representation by pairwise comparisons



Idea

- Define a "comparison function": $K : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$.
- Represent a set of n data points $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ by the $n \times n$ **matrix**:

$$[\mathbf{K}]_{ij} := K(\mathbf{x}_i, \mathbf{x}_j).$$

Positive Definite (p.d.) Kernels

We will restrict ourselves to a **particular class** of pairwise comparison functions:

Definition

A **positive definite (p.d.) kernel** on a set $\mathcal{X}$ is a function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that is **symmetric**:

$$\forall \left(\mathbf{x}, \mathbf{x}'\right) \in \mathcal{X}^2, \quad \mathbf{K}\left(\mathbf{x}, \mathbf{x}'\right) = \mathbf{K}\left(\mathbf{x}', \mathbf{x}\right),$$

and which satisfies, for all $N \in \mathbb{N}$, $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) \in \mathcal{X}^N$ and $(a_1, a_2, \ldots, a_N) \in \mathbb{R}^N$:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j K\left(\mathbf{x}_i, \mathbf{x}_j\right) \geq 0.$$

Similarity matrices of p.d. kernels

Remarks

- Equivalently, a kernel K is p.d. if and only if, for any $N \in \mathbb{N}$ and any set of points $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) \in \mathcal{X}^N$, the **similarity matrix** $[\mathbf{K}]_{ij} := K(\mathbf{x}_i, \mathbf{x}_j)$ **is positive semidefinite**.
- **Kernel methods** are algorithms that take such matrices as input.

The simplest p.d. kernel, for vectors

Lemma

Let $\mathcal{X} = \mathbb{R}^d$. The function $K : \mathcal{X}^2 \mapsto \mathbb{R}$ defined by:

$$\forall \left(\mathbf{x}, \mathbf{x}'\right) \in \mathcal{X}^2, \quad \mathbf{K}\left(\mathbf{x}, \mathbf{x}'\right) = \left\langle \mathbf{x}, \mathbf{x}' \right\rangle_{\mathbb{R}^{\mathbf{d}}}$$

is p.d. (it is often called the **linear kernel**).

Proof:

- $\langle \mathbf{x}, \mathbf{x}' \rangle_{\mathbb{R}^d} = \langle \mathbf{x}', \mathbf{x} \rangle_{\mathbb{R}^d}$
- $\sum_{i=1}^N \sum_{j=1}^N a_i a_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle_{\mathbb{R}^d} = \| \sum_{i=1}^N a_i \mathbf{x}_i \|_{\mathbb{R}^d}^2 \geq 0$ □

The simplest p.d. kernel, for vectors

Lemma

Let $\mathcal{X} = \mathbb{R}^d$. The function $K : \mathcal{X}^2 \mapsto \mathbb{R}$ defined by:

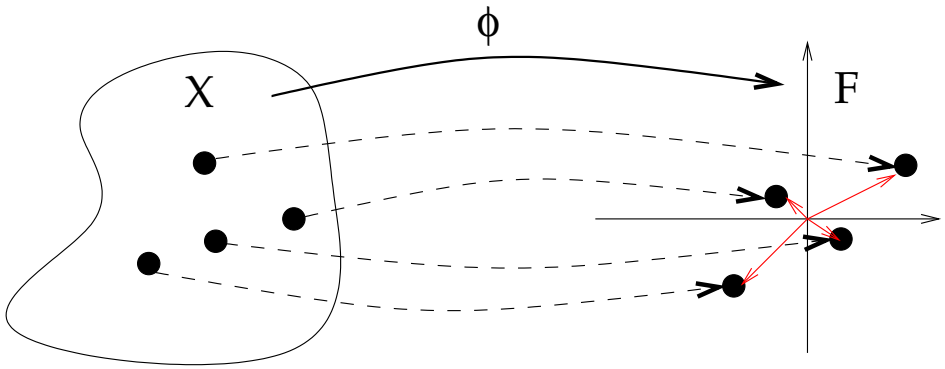
$$\forall \left(\mathbf{x}, \mathbf{x}'\right) \in \mathcal{X}^2, \quad \mathbf{K}\left(\mathbf{x}, \mathbf{x}'\right) = \left\langle \mathbf{x}, \mathbf{x}' \right\rangle_{\mathbb{R}^\mathbf{d}}$$

is p.d. (it is often called the **linear kernel**).

Proof:

- $\langle \mathbf{x}, \mathbf{x}' \rangle_{\mathbb{R}^d} = \langle \mathbf{x}', \mathbf{x} \rangle_{\mathbb{R}^d}$
- $\sum_{i=1}^N \sum_{j=1}^N a_i a_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle_{\mathbb{R}^d} = \| \sum_{i=1}^N a_i \mathbf{x}_i \|_{\mathbb{R}^d}^2 \geq 0$ □

A more ambitious p.d. kernel



Lemma

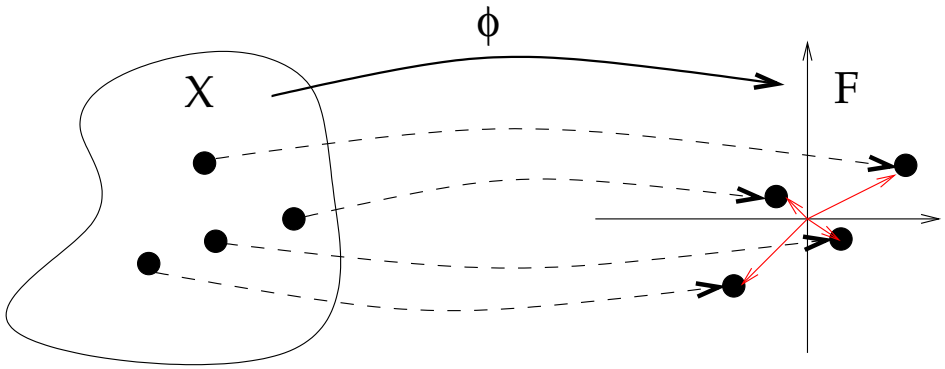
Let $\mathcal{X}$ **be any set**, and $\Phi : \mathcal{X} \mapsto \mathbb{R}^d$. Then, the function $K : \mathcal{X}^2 \mapsto \mathbb{R}$ defined as follows is p.d.:

$$\forall \left(\mathbf{x}, \mathbf{x}'\right) \in \mathcal{X}^2, \quad K\left(\mathbf{x}, \mathbf{x}'\right) = \left\langle \Phi\left(\mathbf{x}\right), \Phi\left(\mathbf{x}'\right) \right\rangle_{\mathbb{R}^d} .$$

Proof:

- $\left\langle \Phi\left(\mathbf{x}\right), \Phi\left(\mathbf{x}'\right) \right\rangle_{\mathbb{R}^d} = \left\langle \Phi\left(\mathbf{x}'\right), \Phi\left(\mathbf{x}\right) \right\rangle_{\mathbb{R}^d}$
- $\sum_{i=1}^N \sum_{j=1}^N a_i a_j \left\langle \Phi\left(\mathbf{x}_i\right), \Phi\left(\mathbf{x}_j\right) \right\rangle_{\mathbb{R}^d} = \| \sum_{i=1}^N a_i \Phi\left(\mathbf{x}_i\right) \|_{\mathbb{R}^d}^2 \geq 0$ □

A more ambitious p.d. kernel



Lemma

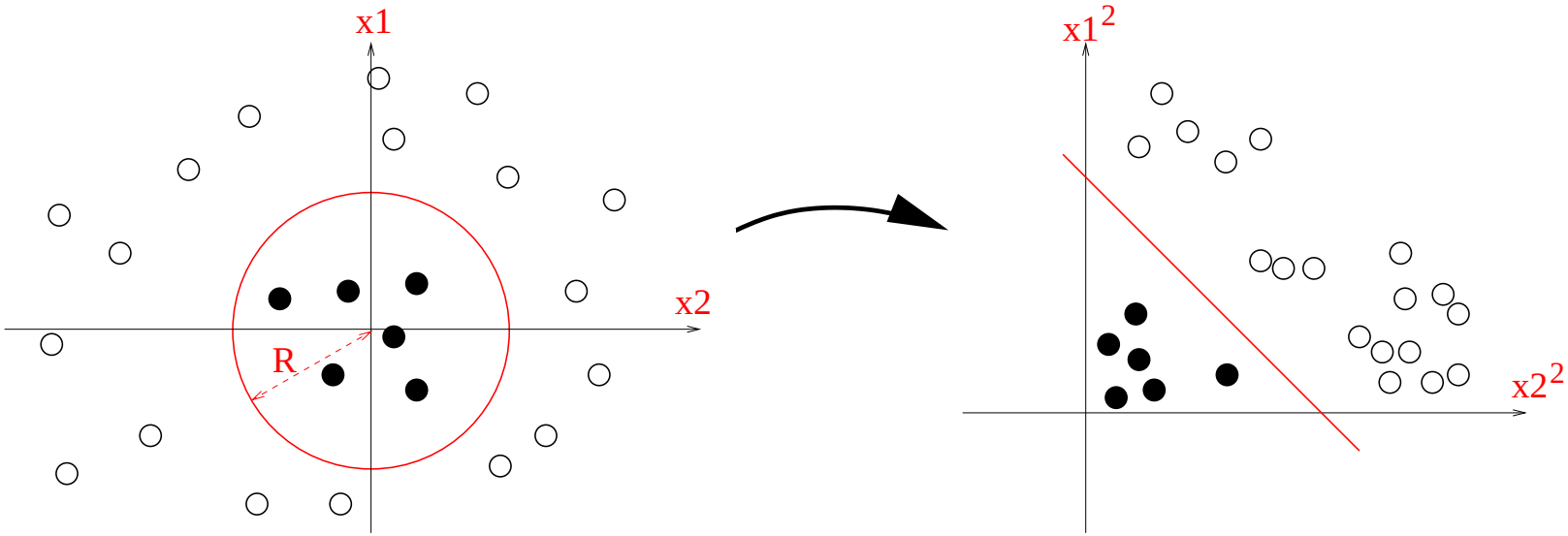
Let $\mathcal{X}$ **be any set**, and $\Phi : \mathcal{X} \mapsto \mathbb{R}^d$. Then, the function $K : \mathcal{X}^2 \mapsto \mathbb{R}$ defined as follows is p.d.:

$$\forall \left(\mathbf{x}, \mathbf{x}'\right) \in \mathcal{X}^2, \quad K\left(\mathbf{x}, \mathbf{x}'\right) = \left\langle \Phi\left(\mathbf{x}\right), \Phi\left(\mathbf{x}'\right) \right\rangle_{\mathbb{R}^d} .$$

Proof:

- $\left\langle \Phi\left(\mathbf{x}\right), \Phi\left(\mathbf{x}'\right) \right\rangle_{\mathbb{R}^d} = \left\langle \Phi\left(\mathbf{x}'\right), \Phi\left(\mathbf{x}\right) \right\rangle_{\mathbb{R}^d}$
- $\sum_{i=1}^N \sum_{j=1}^N a_i a_j \left\langle \Phi\left(\mathbf{x}_i\right), \Phi\left(\mathbf{x}_j\right) \right\rangle_{\mathbb{R}^d} = \| \sum_{i=1}^N a_i \Phi\left(\mathbf{x}_i\right) \|_{\mathbb{R}^d}^2 \geq 0$ □

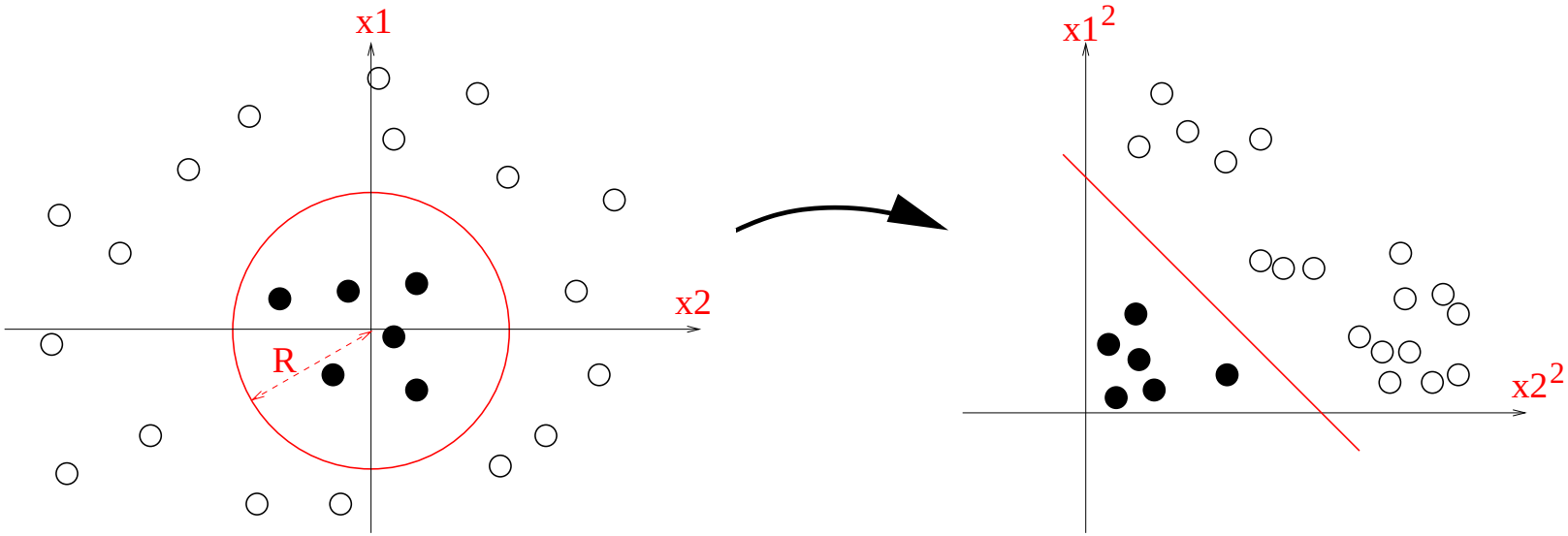
Example: polynomial kernel



For $\mathbf{x} = (x_1, x_2)^\top \in \mathbb{R}^2$, let $\Phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \in \mathbb{R}^3$:

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= x_1^2 x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2 x_2'^2 \\ &= \left(x_1x_1' + x_2x_2'\right)^2 \\ &= \left\langle \mathbf{x}, \mathbf{x}' \right\rangle_{\mathbb{R}^2}^2 \ . \end{aligned}$$

Example: polynomial kernel



For $\mathbf{x} = (x_1, x_2)^\top \in \mathbb{R}^2$, let $\Phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \in \mathbb{R}^3$:

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= x_1^2 x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2 x_2'^2 \\ &= \left(x_1x_1' + x_2x_2'\right)^2 \\ &= \left\langle \mathbf{x}, \mathbf{x}' \right\rangle_{\mathbb{R}^2}^2 \ . \end{aligned}$$

Conversely: Kernels as inner products

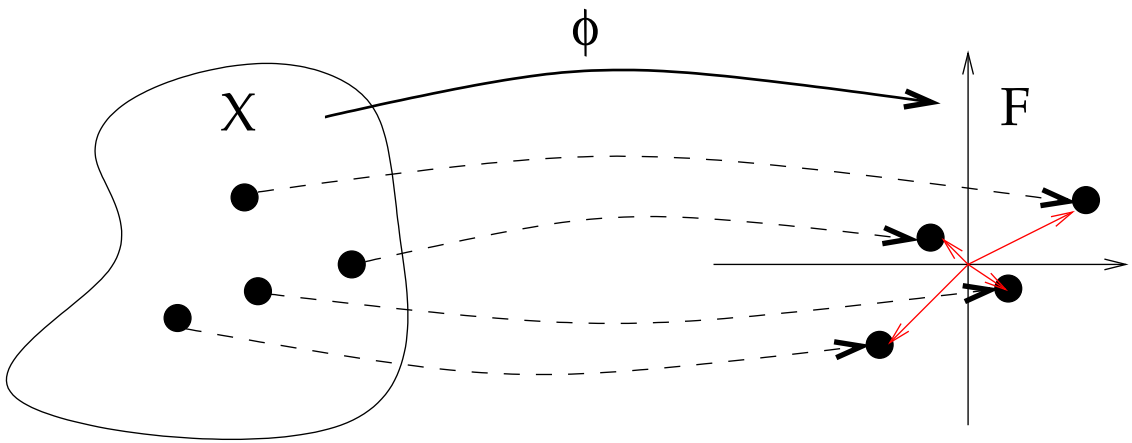
Theorem (Aronszajn, 1950)

K is a p.d. kernel on the set $\mathcal{X}$ **if and only if** there exists a Hilbert space $\mathcal{H}$ and a mapping

$$\Phi : \mathcal{X} \mapsto \mathcal{H}$$

such that, for any $\mathbf{x}, \mathbf{x}'$ in $\mathcal{X}$:

$$K\left(\mathbf{x}, \mathbf{x}'\right) = \left\langle \Phi\left(\mathbf{x}\right), \Phi\left(\mathbf{x}'\right) \right\rangle_{\mathcal{H}} .$$



Kernels on strings: substring indexation

How can we build kernel on non-vectorial objects like strings?

An approach

Index the feature space by fixed-length strings, i.e.,

$$\Phi(\mathbf{x}) = (\Phi_u(\mathbf{x}))_{u \in \mathcal{A}^k}$$

where $\Phi_u(\mathbf{x})$ can be:

- the number of occurrences of u in $\mathbf{x}$ (without gaps) : **spectrum kernel** (Leslie et al., 2002)
- the number of occurrences of u in $\mathbf{x}$ up to m mismatches (without gaps) : **mismatch kernel** (Leslie et al., 2004)
- the number of occurrences of u in $\mathbf{x}$ allowing gaps, with a weight decaying exponentially with the number of gaps : **substring kernel** (Lohdi et al., 2002)

Example: spectrum kernel (1/2)

Kernel definition

- The 3-spectrum of

$$\mathbf{x} = \texttt{CGGSLIAMMWFGV}$$

is:

$$(\texttt{CGG},\texttt{GGS},\texttt{GSL},\texttt{SLI},\texttt{LIA},\texttt{IAM},\texttt{AMM},\texttt{MMW},\texttt{MWF},\texttt{WFG},\texttt{FGV}) \ .$$

- Let $\Phi_u(\mathbf{x})$ denote the number of occurrences of u in $\mathbf{x}$. The k-spectrum kernel is:

$$K(\mathbf{x}, \mathbf{x}') := \sum_{u \in \mathcal{A}^k} \Phi_u(\mathbf{x}) \Phi_u(\mathbf{x}') \ .$$

Example: spectrum kernel (2/2)

Implementation

- The computation of the kernel is formally a sum over $|\mathcal{A}|^k$ terms, but at most $|\mathbf{x}| - k + 1$ terms are non-zero in $\Phi(\mathbf{x}) \implies$ **Computation in** $O(|\mathbf{x}| + |\mathbf{x}'|)$ with pre-indexation of the strings.
- Fast classification of a sequence $\mathbf{x}$ in $O(|\mathbf{x}|)$:

$$f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) = \sum_u w_u \Phi_u(\mathbf{x}) = \sum_{i=1}^{|\mathbf{x}|-k+1} w_{x_i \ldots x_{i+k-1}}.$$

Remarks

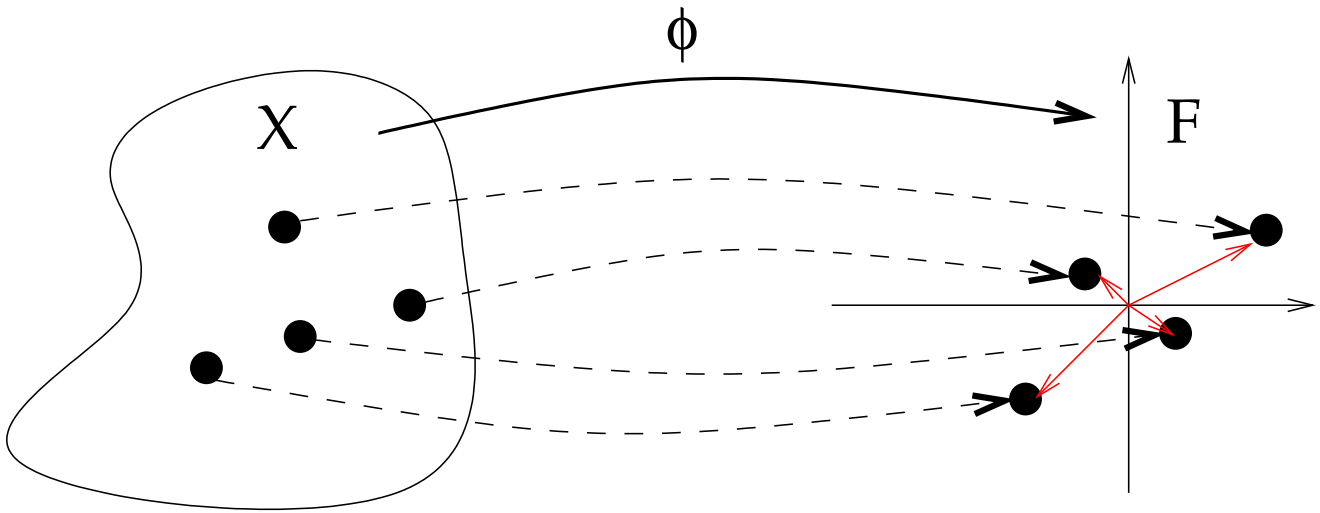
- Work with any string (natural language, time series...)
- **Fast and scalable**, a good default method for string classification.
- Variants allow matching of k-mers up to m **mismatches**.

The kernel trick: motivation

- Choosing a p.d. kernel K on a set $\mathcal{X}$ amounts to **embedding the data in a Hilbert space**: there exists a Hilbert space $\mathcal{H}$ and a mapping $\Phi : \mathcal{X} \mapsto \mathcal{H}$ such that, for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$,

$$\forall \left(\mathbf{x}, \mathbf{x}'\right) \in \mathcal{X}^2, \quad K\left(\mathbf{x}, \mathbf{x}'\right) = \left\langle \Phi\left(\mathbf{x}\right), \Phi\left(\mathbf{x}'\right) \right\rangle_{\mathcal{H}} .$$

- However this mapping might **not be explicitly given**, nor convenient to work with in practice (e.g., large or even infinite dimensions).
- A solution is to work implicitly in the feature space!



The kernel trick

Proposition

Any algorithm to process finite-dimensional vectors that can be expressed **only in terms of pairwise inner products** can be applied to potentially infinite-dimensional vectors in the feature space of a p.d. kernel by **replacing each inner product evaluation by a kernel evaluation**.

Remarks:

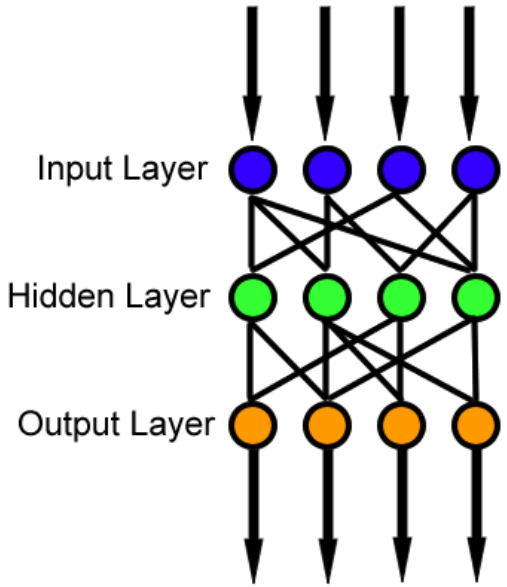
- The proof of this proposition is trivial, because the kernel is exactly the inner product in the feature space.
- This trick has **huge practical applications**.
- Vectors in the feature space are only manipulated **implicitly**, through pairwise inner products.
- Applies to ridge regression, SVM, k-means, PCA and many more classical methods.

Representation by pairwise comparisons

- Which descriptors (think of the protein or molecule examples in the introduction).
  In some cases, it is easier to build a kernel than an individual description.
- Too many descriptors make the algorithms untractable.
  N^2 **scaling whatever the underlying RKHS dimension. Can be further reduced using approximations.**
- A choice of descriptor does not necessarily make the classes linearly separable.
  Introduce non-linearity by just changing the kernel.

Deep learning (adapted from J. Mairal)

A quick zoom on feed forward neural networks



- Each neuron computes a linear combination of its inputs, then applies a non-linearity.
- Weights of the linear combination are optimized by **backpropagation**.

A quick zoom on feed forward neural networks

The goal is to learn a **prediction function** $f : \mathbb{R}^p \to \mathbb{R}$ given labeled training data $(x_i, y_i)_{i=1,\dots,n}$ with x_i in $\mathbb{R}^p$, and y_i in $\mathbb{R}$:

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(f)}_{\text{regularization}} .$$

What is specific to multilayer neural networks?

- The "neural network" space $\mathcal{F}$ is explicitly parametrized by:

$$f(x) = \sigma_k(A_k \sigma_{k-1}(A_{k-1} \dots \sigma_2(A_2 \sigma_1(A_1 x)) \dots)).$$

- Finding the optimal $A_1, A_2, \dots, A_k$ yields a **non-convex** optimization problem in **huge dimension.**

A quick zoom on convolutional neural networks

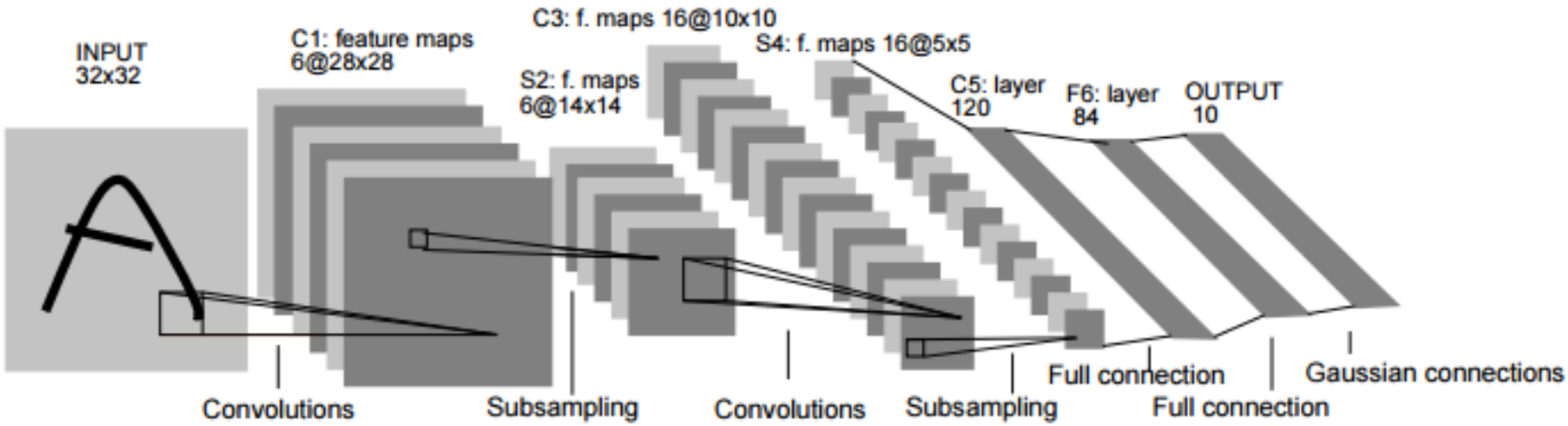


Figure: Picture from Le Cun *et al.*, 1998

- CNNs perform "simple" operations such as convolutions, pointwise non-linearities and subsampling.
- for most successful applications of CNNs, **training is supervised**.

A quick zoom on convolutional neural networks

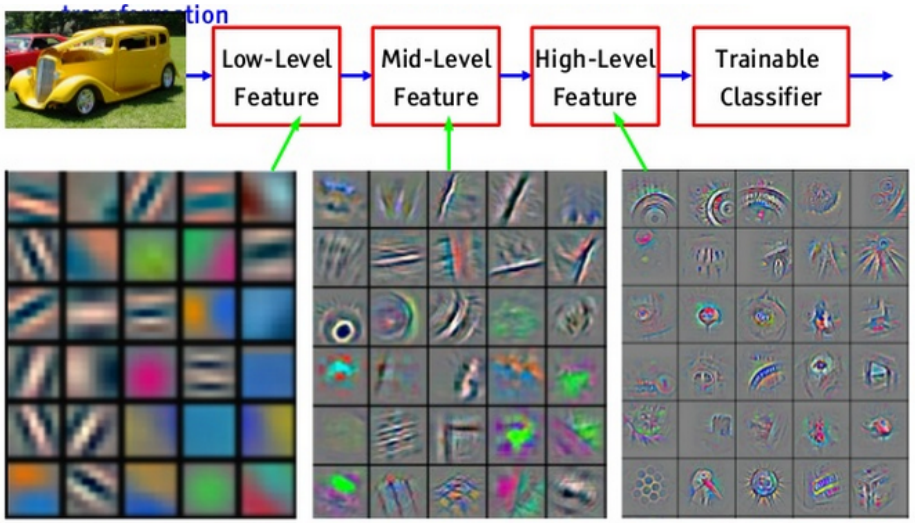


Figure: Picture from Yann LeCun's tutorial, based on Zeiler and Fergus, 2014.

- Technically learning weight of linear functions.
- In practice, leads to learning sets of relevant features from the input.
- In particular, captures compositional structures in images and provides some invariance.

A quick zoom on convolutional neural networks

- Recent success in computer vision because of large amount of available data and computing power (neither was true 10 years ago).
- Large amount of data calls for complex sets of function (remember the first class). CNNs provide such sets $\mathcal{F}$.
- Computing power makes it possible to solve

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(f)}_{\text{regularization}} .$$

CNNs in computational biology

Two key points for CNNs to perform well:

- Large amount of training data. Not true for many problems in biology.
- Relevant features for problem at hand. Makes sense for computer vision, translation to biology is not straighforward.

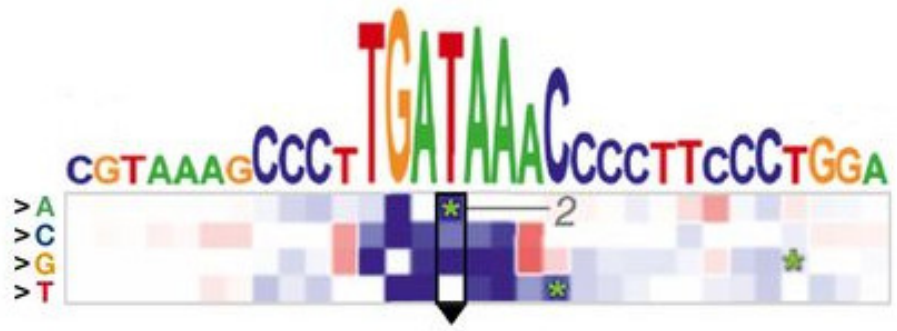


Figure: DeepBind filter, adapted from Angermueller *et al.*, 2016.

K nearest neighbors

k nearest neighbors

- Old (50's), simple to understand and to implement technique: assign the label given by majority vote of the nearest neighbors.
- More formally:

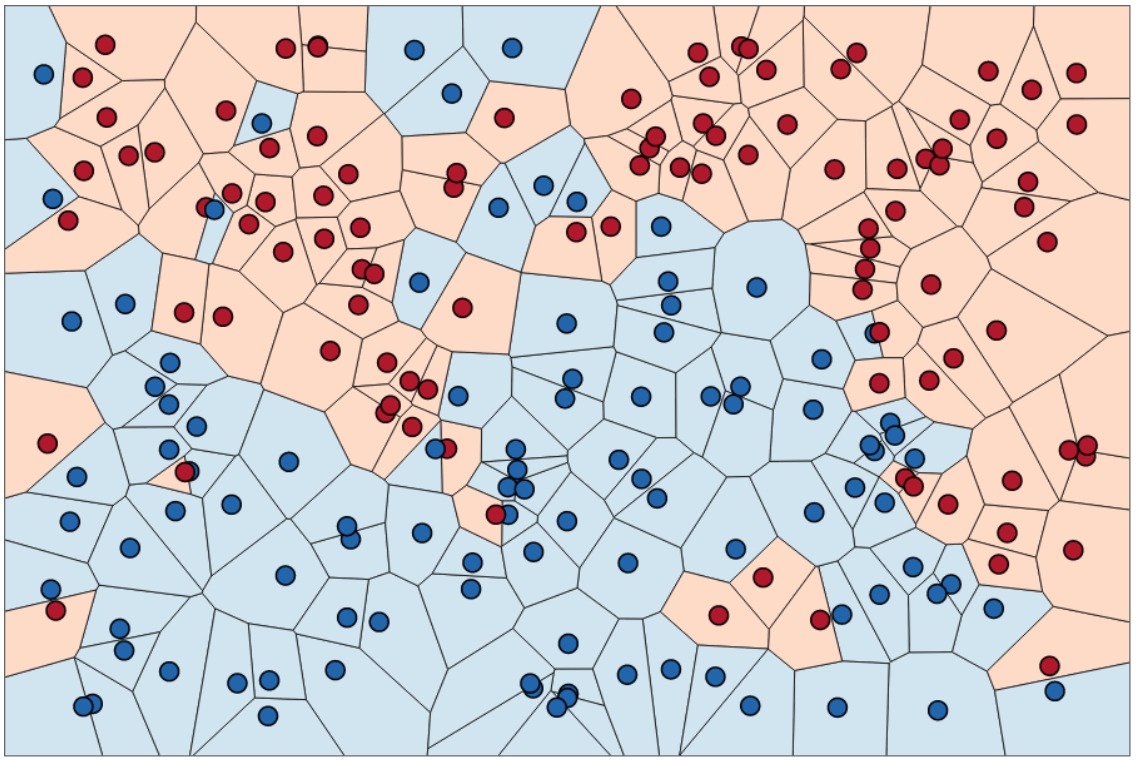
$$f(x) = \text{sign}\left(\sum_{i=1}^{n} y_i \mathbf{1}_{x_i \in \mathcal{V}^k(x)}/k\right),$$

  where $\mathcal{V}^k(x)$ is the set of k closest training points to x and binary classes are encoded by -1 and 1.
- Straightforward generalization to regression.
- Often a good baseline in practice.

k nearest neighbors

k controls the regularity of the resulting classification function:

- Small k lead to very adjusted functions
- Larger k lead to smoother functions (lower variance, higher bias).

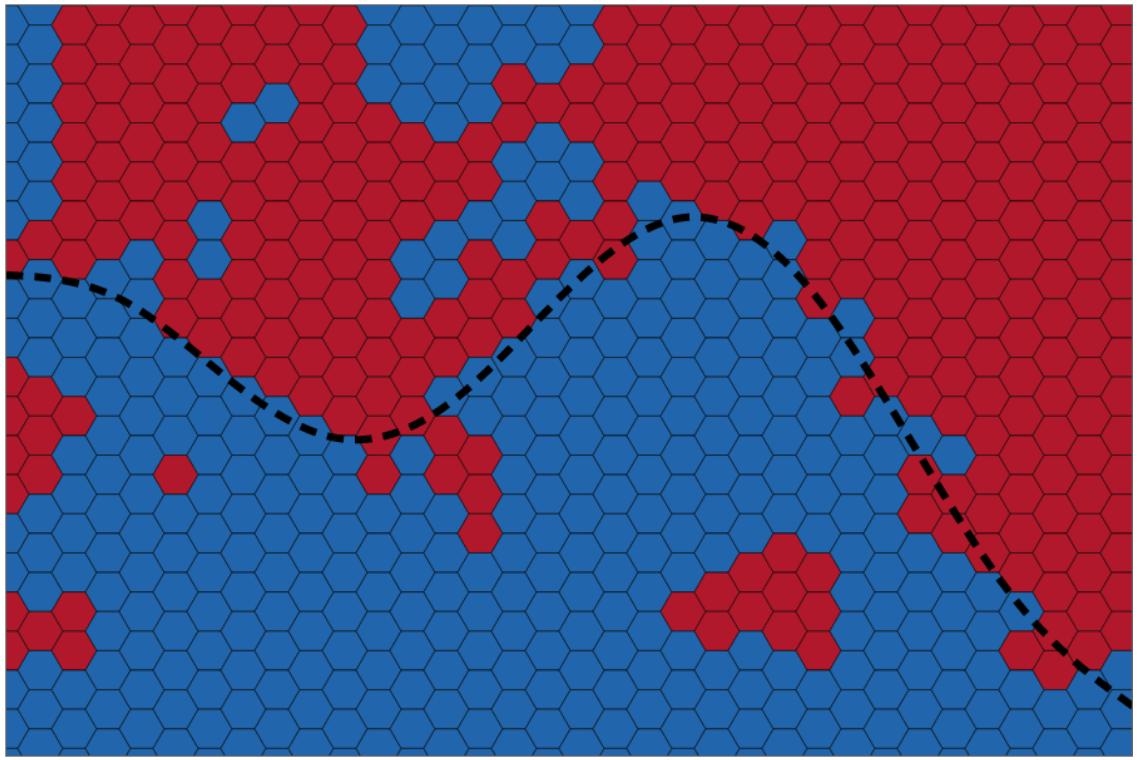


(from S. Fortmann-Roe's webpage)

k nearest neighbors

k controls the regularity of the resulting classification function:

- Small k lead to very adjusted functions
- Larger k lead to smoother functions (lower variance, higher bias).

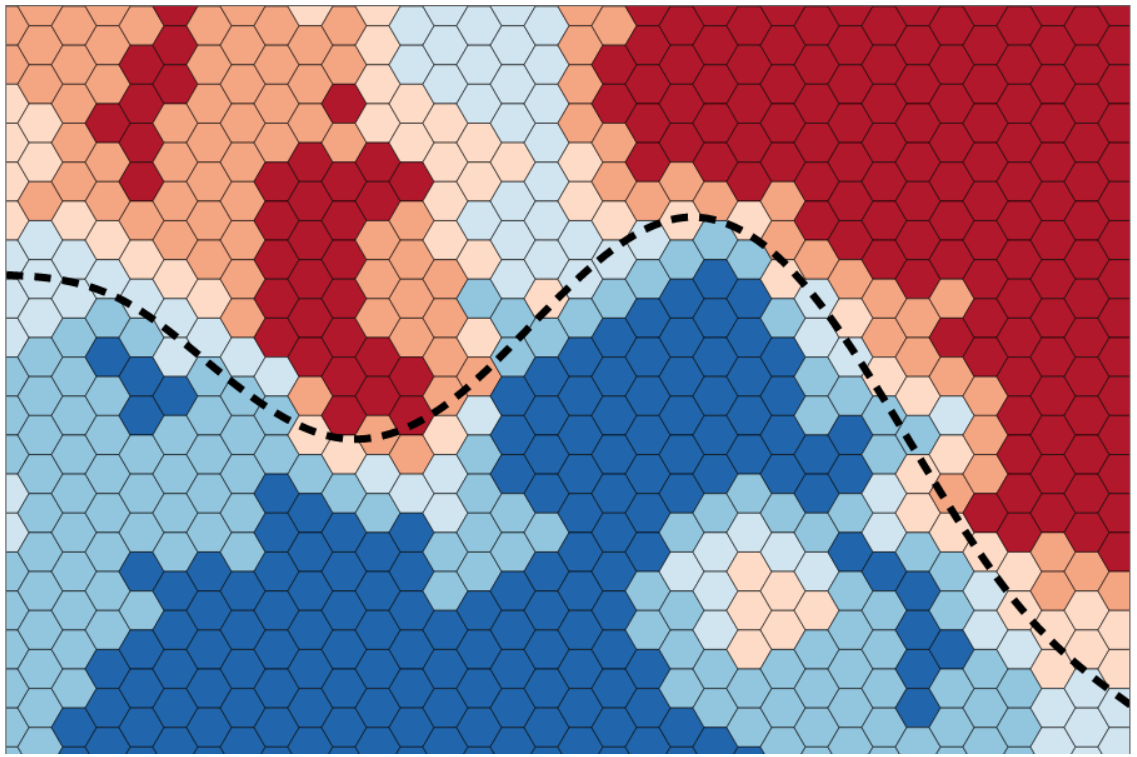


1-nn decision

k nearest neighbors

k controls the regularity of the resulting classification function:

- Small k lead to very adjusted functions
- Larger k lead to smoother functions (lower variance, higher bias).

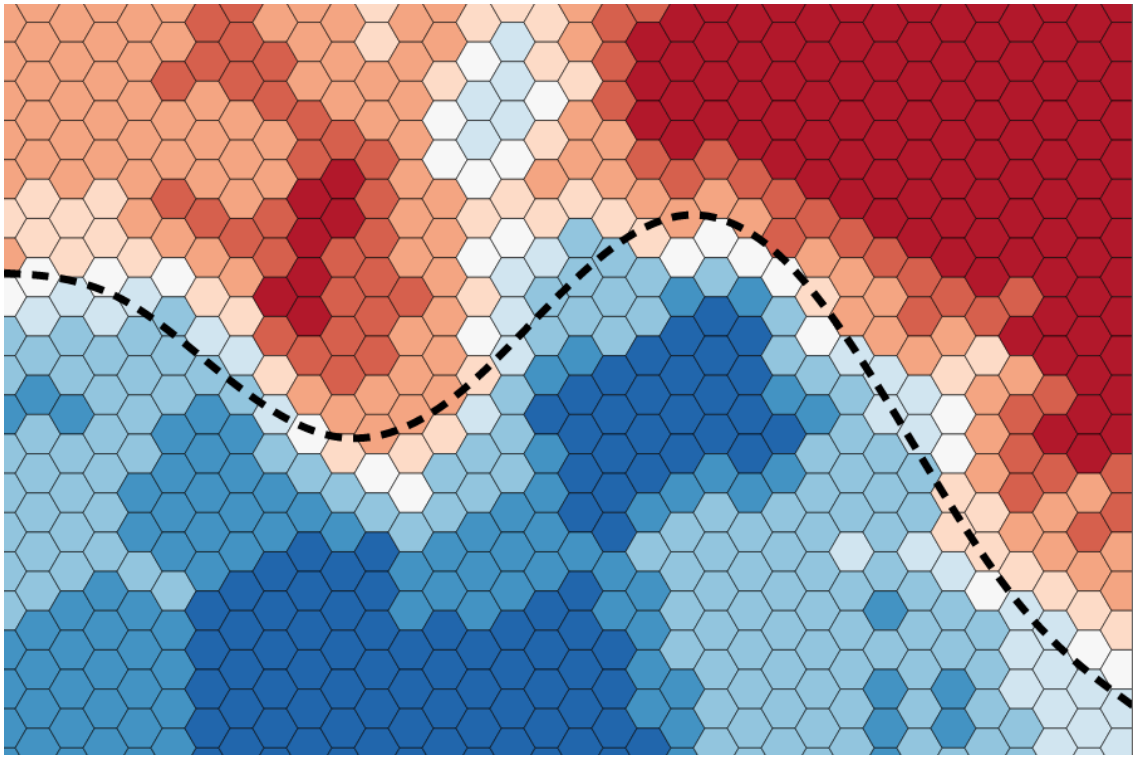


5-nn decision

k nearest neighbors

k controls the regularity of the resulting classification function:

- Small k lead to very adjusted functions
- Larger k lead to smoother functions (lower variance, higher bias).

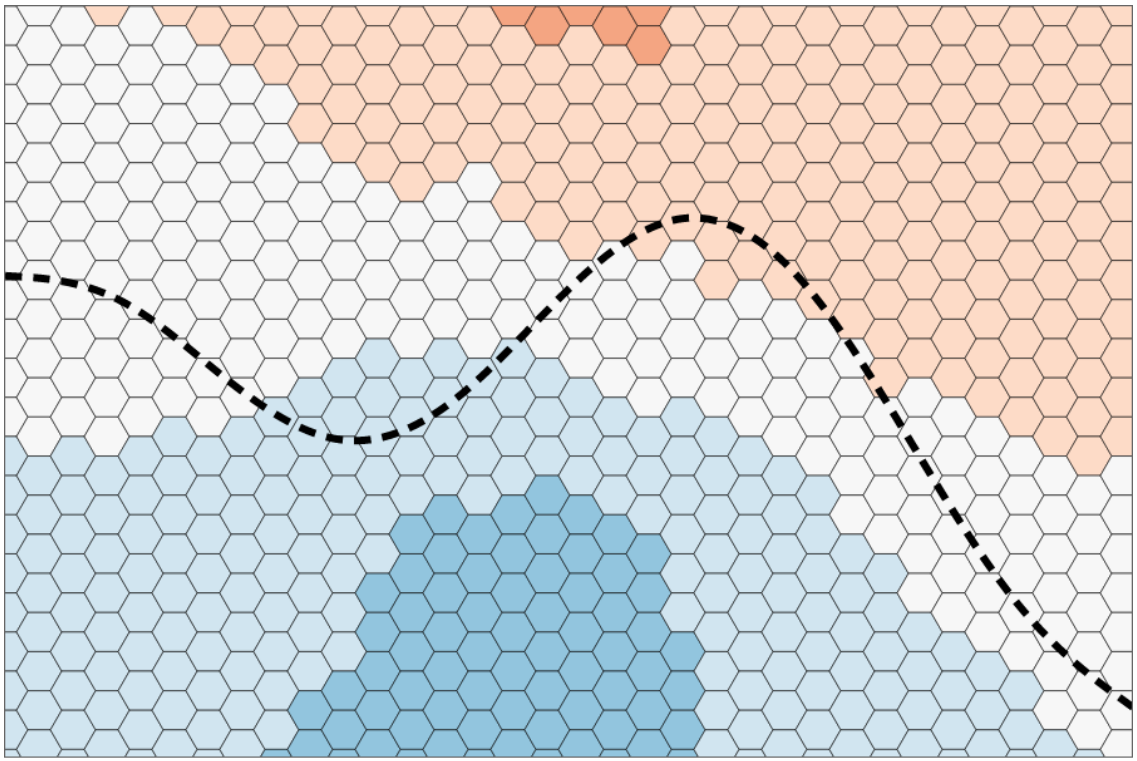


10-nn decision

k nearest neighbors

k controls the regularity of the resulting classification function:

- Small k lead to very adjusted functions
- Larger k lead to smoother functions (lower variance, higher bias).



100-nn decision

Computational aspect

- "Lazy" learning: no training phase required.
- Classifying a new point can be expensive when n grows.
- Three classes of solutions: fast exact algorithm, approximate algorithm, probably approximate algorithms.

Link to previous (parametric) techniques

- Representer theorem (simplified): given a training sample $(x_i, y_i)_{i=1,\dots,n} \in \mathbb{R}^p \times \mathbb{R}$, an arbitrary empirical loss function R and a strictly monotonically increasing real valued function g, any β^* verifying

$$\beta^* = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left(R((x_i, y_i, \beta^\top x_i)) + g(\|\beta\|) \right)$$

  admits a representation of the form $\beta^* = \sum_{i=1}^n \alpha_i x_i$, $\alpha_i \in \mathbb{R}$.
- Consequence: for ridge linear regression, logistic regression and SVM, decision function is $f(x) = \sum_{i=1}^n \alpha_i x_i^\top x$.
- Can also be thought of like a vote, weighted by Euclidean distances of training points with tested point.

Universal consistency

- More general definition of consistency: an estimator is **consistent** if the expectation of its risk converges to the Bayes risk:

$$\mathbf{E}R_n \to R^* \text{ as } n \to \infty,$$

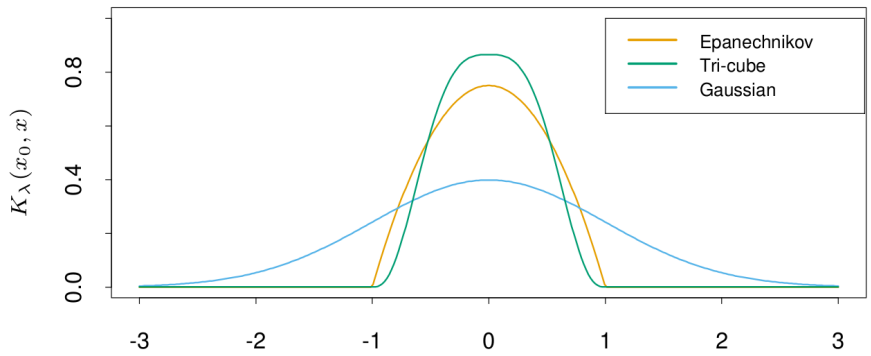
  where R_n is the risk of an estimator based on n i.i.d samples.
- An estimator is **universally consistent** if it is consistent for any distribution of (X, Y).
- **Stone's theorem**: gives sufficient conditions for rules of the form $\sum_{i=1}^{n} y_i w_i(x)$ where the $w_i(x)$ are non-negative weights and sum to one to be universally consistent.
- knn satisfies these conditions for $k \to \infty$ and $k/n \to 0$.

Nadaraya-Watson method

- More generally, other weighting schemes for the vote (rather than discontinuous membership to k nearest neighbors) are possible.
- Nadaraya-Watson:

$$f(x) = \frac{\sum_{i=1}^n y_i K_\lambda(x, x_i)}{\sum_{i=1}^n K_\lambda(x, x_i)},$$

  where $K_\lambda(x, x_i) = K\left(\frac{|x - x_i|}{\lambda}\right)$ is a **kernel** function measuring how close x and x_i are.
- Examples of kernels:



(from The Elements of Statistical Learning)

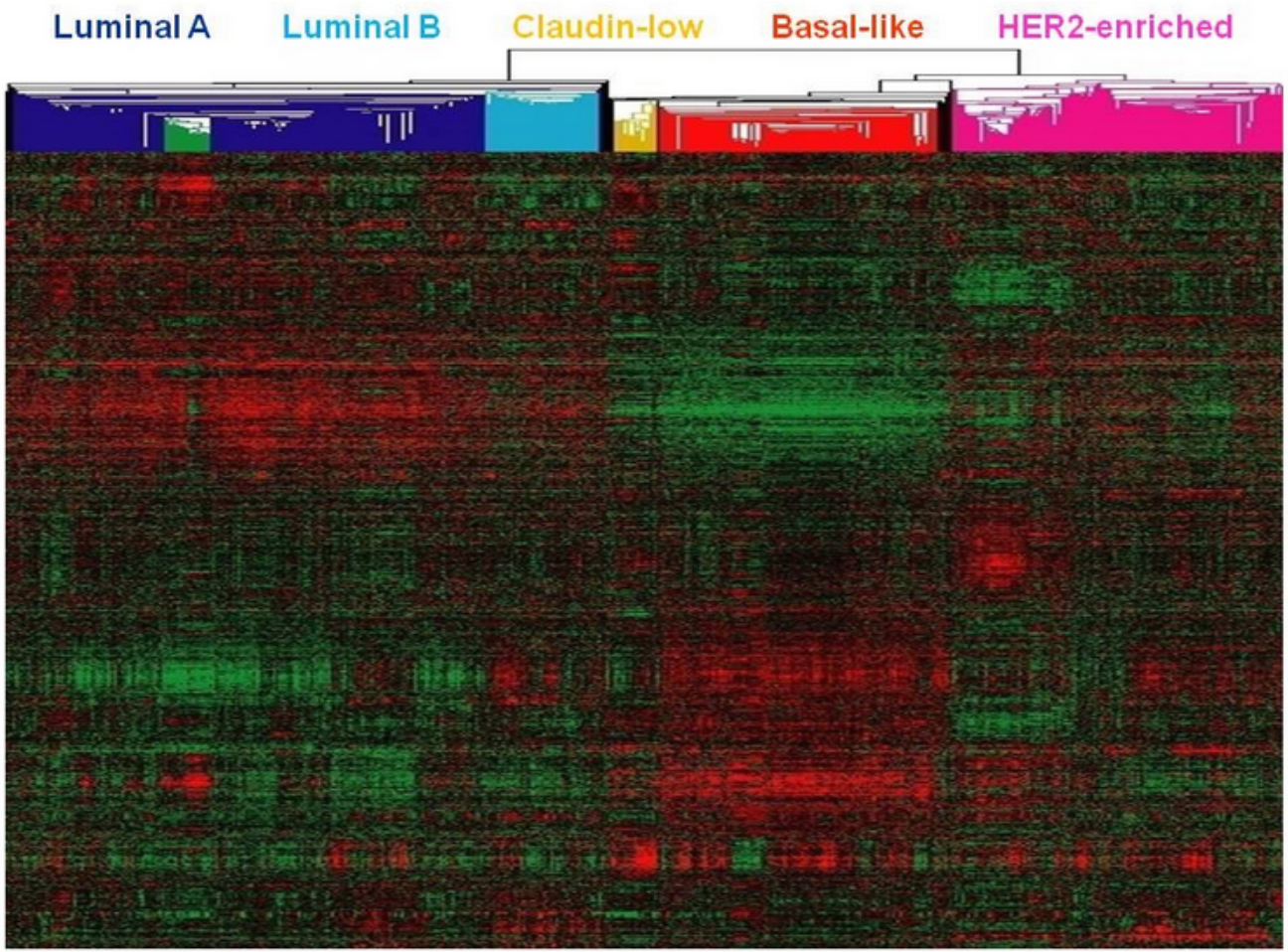
- Also universally consistent (using Stone's theorem).

Part V

Unsupervised estimation and matrix decomposition

Back to one of our introductive examples

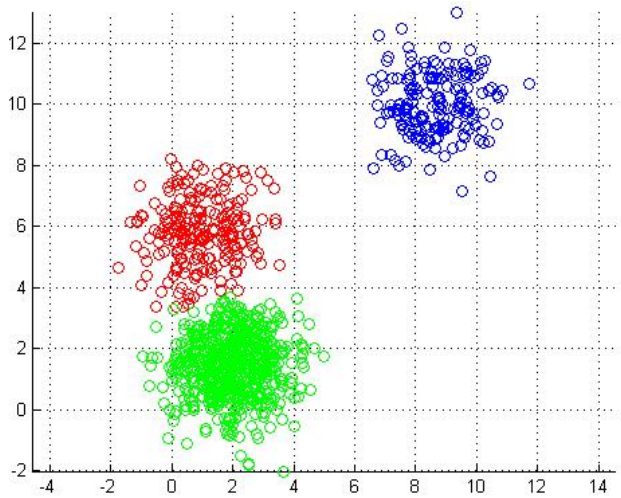
Gene expression clustering



(from C. Perou's website)

Are there groups of breast tumors with similar gene expression profile?

K-means



(From the Matlab website)

- The informal objective of clustering is to identify groups of samples such that samples within each group are close and samples across groups are far away.
- The k-means algorithm aims at minimizing $\sum_{i=1}^{n} \|x_i - v_{c(i)}\|^2$, where $c(i)$ is the cluster (group) to which x_i is assigned and the v_j are the cluster centers.

K-means

Algorithm :

- Choose k points x_i as cluster centers.
- Iterate :
  1. Given the cluster centers, assign each x_i to the cluster c whose center v_c is the closest.
  2. Given the assignments for x_i, each cluster center is the mean of its points: $v_c = \sum_{i:c(i)=c} x_i$.

K-means

- The minimized objective $\sum_{i=1}^{n} \|x_i - v_{c(i)}\|^2$ also writes:

$$\min_{U \in \{0,1\}^{n \times k}, V \in \mathbb{R}^{p \times k}} \|X - UV^\top\|_F^2, \sum_{j=1}^{k} u_{ij} = 1 \forall i = 1, \ldots, n,$$

  where $X \in \mathbb{R}^{n \times p}$ is the matrix whose rows are the x_i.
- This algorithm can be thought of as a constrained likelihood maximization for some model. **[Exercise:]** which one?

- This point of view highlights some implicit hypotheses made when using this algorithm for clustering. **[Exercise:]** which ones?

K-means

- The minimized objective $\sum_{i=1}^{n} \|x_i - v_{c(i)}\|^2$ also writes:

$$\min_{U \in \{0,1\}^{n \times k}, V \in \mathbb{R}^{p \times k}} \|X - UV^\top\|_F^2, \sum_{j=1}^{k} u_{ij} = 1 \forall i = 1, \ldots, n,$$

  where $X \in \mathbb{R}^{n \times p}$ is the matrix whose rows are the x_i.
- This algorithm can be thought of as a constrained likelihood maximization for some model. **[Exercise:]** which one?

$$x_i \sim \mathcal{N}(U_{i,.} V^\top, \sigma^2 I).$$

- This point of view highlights some implicit hypotheses made when using this algorithm for clustering. **[Exercise:]** which ones?

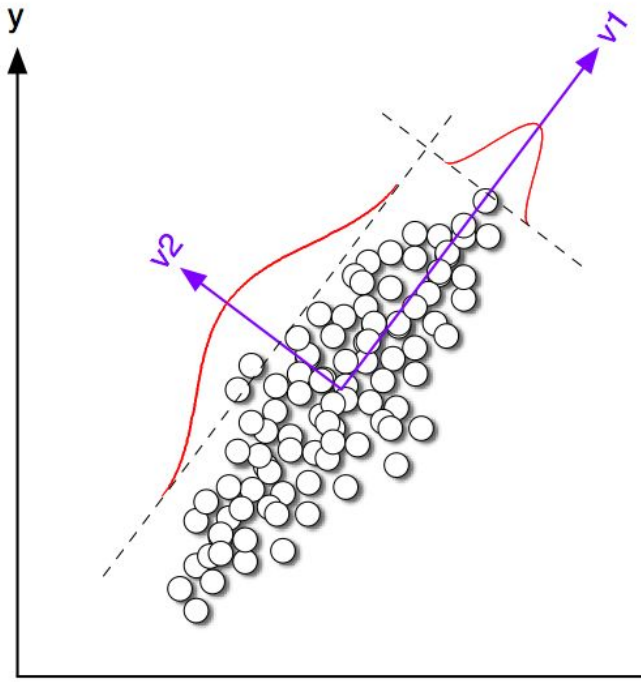
K-means

- The minimized objective $\sum_{i=1}^{n} \|x_i - v_{c(i)}\|^2$ also writes:

$$\min_{U \in \{0,1\}^{n \times k}, V \in \mathbb{R}^{p \times k}} \|X - UV^\top\|_F^2, \sum_{j=1}^{k} u_{ij} = 1 \forall i = 1, \ldots, n, \qquad (1)$$

  where $X \in \mathbb{R}^{n \times p}$ is the matrix whose rows are the x_i.
- **[Exercise :]** Is (1) convex? What are the consequences on the result of the k-means algorithm?
- There exist plenty of other clustering algorithms, notably hierarchical ones.

Principal components analysis



(figure by T. Jehan)

- We want to build a small number of axes retaining the largest amount of variance.
- Other way of decomposing the variance than clustering (continuous vs discrete factors).

Principal components analysis

Algorithm :

- Formally, the axis yielding the highest empirical variance for the projected x_i is:

$$\max_{v \in \mathbb{R}^p, \|v\|=1} \hat{Var}(Xv) \propto \max_{v \in \mathbb{R}^p, \|v\|=1} \left(Xv - \mathbf{1}\overline{(Xv)}\right)^\top \left(Xv - \mathbf{1}\overline{(Xv)}\right) \tag{2}$$

$$= \max_{v \in \mathbb{R}^p, \|v\|=1} v^\top X^\top X v \tag{3}$$

$$= \max_{v \in \mathbb{R}^p, \|v\|=1} \sum_{l=1}^{\mathrm{rank}(X)} e_l^2 \left(v^\top q_l\right)^2 = \textbf{[Exercise]}, \tag{4}$$

where $X = PEQ^\top$ is the singular value decomposition of X and where we assumed the columns of X were centered.

Principal components analysis

Algorithm :

- Formally, the axis yielding the highest empirical variance for the projected x_i is:

$$\max_{v \in \mathbb{R}^p, \|v\|=1} \hat{Var}(Xv) \propto \max_{v \in \mathbb{R}^p, \|v\|=1} \left(Xv - \mathbf{1}\overline{(Xv)}\right)^\top \left(Xv - \mathbf{1}\overline{(Xv)}\right) \tag{2}$$

$$= \max_{v \in \mathbb{R}^p, \|v\|=1} v^\top X^\top X v \tag{3}$$

$$= \max_{v \in \mathbb{R}^p, \|v\|=1} \sum_{l=1}^{\text{rank}(X)} e_l^2 \left(v^\top q_l\right)^2 = e_1^2, \tag{4}$$

  where $X = PEQ^\top$ is the singular value decomposition of X and where we assumed the columns of X were centered.
- We can then look for other direction of high variance, orthogonal to the previous ones. Following the same reasoning, the solutions are given by the eigen vectors of $X^\top X$.

Principal component analysis

Proposition

For any matrix $X \in \mathbb{R}^{n \times p}$,

$$\underset{rank(A) \leq k}{\arg\min} \ \|X - A\|_F^2 = P_k E_k Q_k^\top,$$

where $PEQ^\top$ *is the singular value decomposition of* X, P_k *and* Q_k *the restrictions of* P *and* Q *to their* k *first columns and* E_k *the restriction of* E *to its* k *first rows and columns.*

- This is a results by Eckart and Young (1936).
- Atypical point: this is a non convex problem but for which we can characterize a global minimum.

Principal component analysis

$$\min_{rank(A) \leq k} \|X - A\|_F^2$$

also writes

$$\min_{U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{p \times k}} \|X - UV^\top\|_F^2, \; U^\top U = I, V^\top V \text{ diagonal.}$$

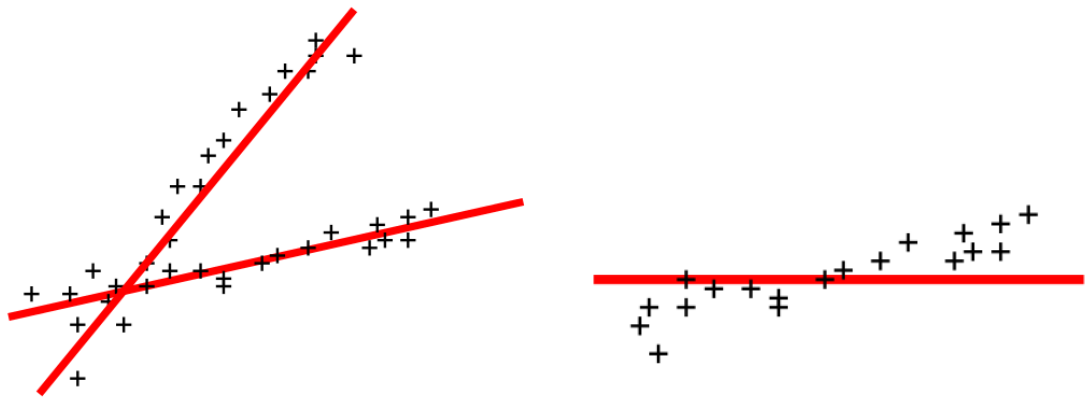
- PCA can also be thought of as a matrix decomposition problem (the solution to both problems is the same).
- The minimized objective is the same as the one minimized by k-means, but the constraints are different.
- Here again, PCA can be thought of as a constrained maximization of the likelihood of a linear Gaussian model.

Penalized matrix decomposition

$$\min_{U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{p \times k}} \|X - UV^\top\|_F^2, \; U \in \mathcal{M}_U, V \in \mathcal{M}_V.$$

Once the problem is cast in this general framework, it becomes natural to introduce penalized extensions:

- Sparse dictionary learning : sparse U. Each sample is explained by a small number of axes.
- Sparse PCA : sparse V. Each axis is a combination of a small number of original variables.



(from F. Bach's slides)

Penalized matrix decomposition

$$\min_{U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{p \times k}} \|X - UV^\top\|_F^2, \; U \in \mathcal{M}_U, V \in \mathcal{M}_V.$$

Other extensions :

- Non-negative matrix factorization (NMF).
- Structured penalties.

Algorithms :

- Convex relaxations (Bach et al., 2008).
- Iterative optimization over U and V.

Penalized matrix decomposition

Points we did not touch:

- Other methods: ICA, CCA.
- Non linear, non parametric methods. Possible to adapt linear methods using positive definite kernels.