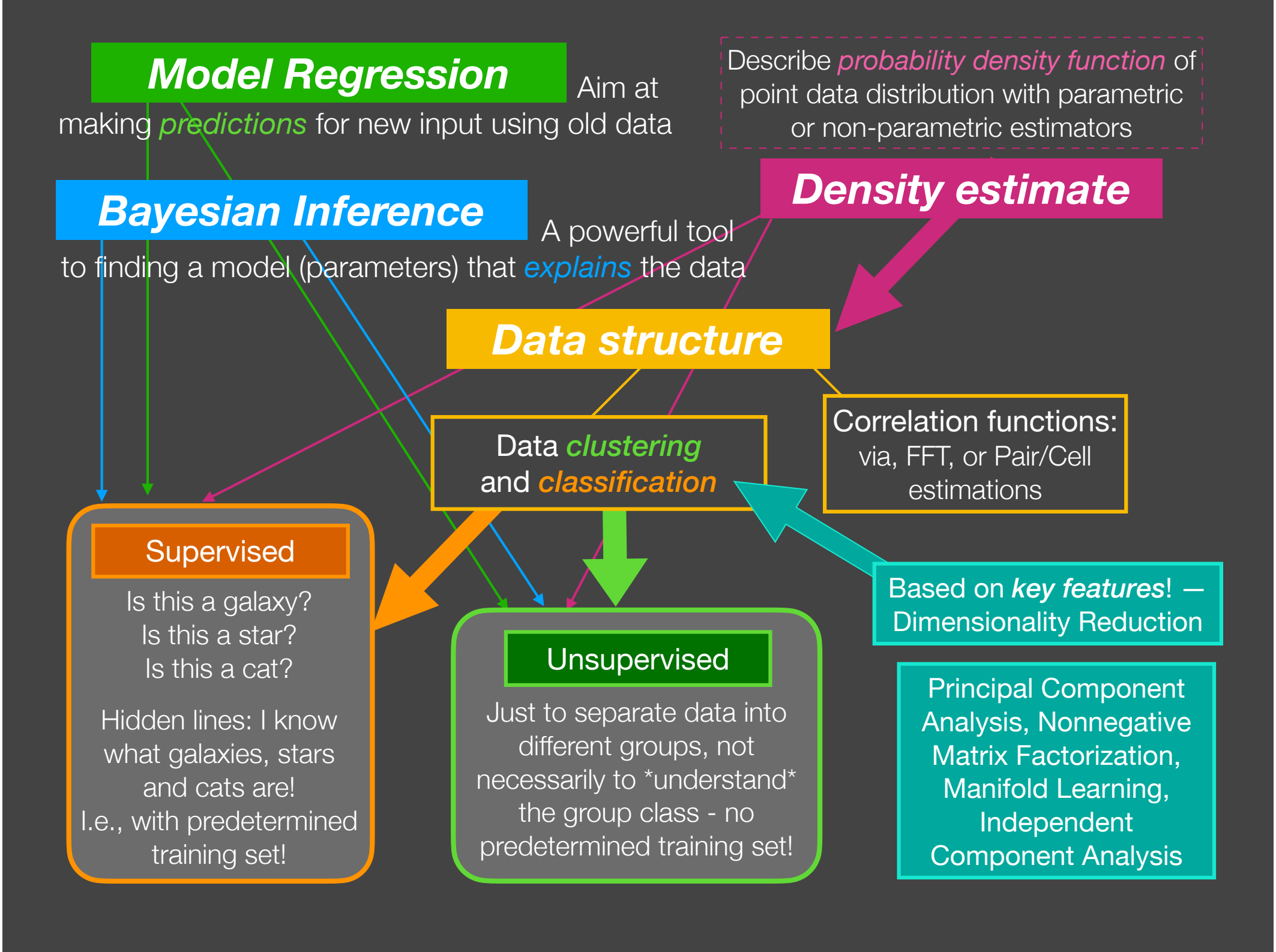
Structures in Point Data, Dimensionality Reduction, Classification

A Tuesday in 2024 Winter

9:50-12:15


清華大學天文系

Department of Astronomy, Tsinghua University


Model Regression
Aim at making predictions for new input using old data
Describe probability density function of point data distribution with parametric or non-parametric estimators
Density estimate
Bayesian Inference
A powerful tool to finding a model (parameters) that explains the data
Data structure
Data clustering and classification
Correlation functions: via, FFT, or Pair/Cell estimations
Supervised
Is this a galaxy?
Is this a star?
Is this a cat?
Hidden lines: I know what galaxies, stars and cats are!
I.e., with predetermined training set!
Unsupervised
Just to separate data into different groups, not necessarily to *understand* the group class - no predetermined training set!
Based on key features! — Dimensionality Reduction
Principal Component Analysis, Nonnegative Matrix Factorization, Manifold Learning, Independent Component Analysis

Structures in Point Data, Dimensionality Reduction, and Classification

- ***Correlation function***
  - Fourier transformation
  - Pair estimation
  - Cell estimation
- ***Density estimation***
  - Kernel density estimate
  - Nearest-neighbor density estimate
  - Gaussian Mixture model
- ***Clustering in data***
  - K-means algorithm
  - Mean-shift algorithm
  - Hierarchical clustering algorithm
- ***Classification***
  - Generative Classification
  - K-Nearest Neighbor Classification
  - Discriminative Classification

- ***Correlation function***
- ***Density estimation***
- ***Clustering in data***
- ***Classification***

Correlation function

The level of differences at different scales,
between a certain distribution of points and
that of a random distribution!

Correlation function $\xi(\mathbf{x})$

The average probability of finding a point in a volume element $\mathrm{d}V$ is proportional to the *mean* density of the background assuming random homogeneous distribution, i.e., $\mathrm{d}P \propto \bar{\rho}$, $\bar{\rho} \equiv \langle \rho(\mathbf{x}) \rangle$, where operation $\langle\ \rangle$ is taking average over the entire data region.
How about the probability of finding a pair of points?

The average probability of finding a pair of points in two volume elements $\mathrm{d}V_1$ and $\mathrm{d}V_2$, separated by a distance vector $\mathbf{x} = \mathbf{x_2} - \mathbf{x_1}$, is proportional to the *mean* density pair at the two positions, i.e., $\mathrm{d}P_{12}(\mathbf{x}) \propto \langle \rho_1 \rho_2 \rangle \mathrm{d}V_1 \mathrm{d}V_2$.

In order to make comparisons of clustering at different scales within the same field, we can normalize it in the following way:
we seek for *the average relative probability, w.r.t random mean field*, of finding a pair of points in two volume elements $\mathrm{d}V_1$ and $\mathrm{d}V_2$, separated by distance vector $\mathbf{x}$.
It is given by: $\mathrm{d}P_{12}(\mathbf{x}) = \langle \rho_1 \rho_2 \rangle / \bar{\rho}^2 \mathrm{d}V_1 \mathrm{d}V_2 = \langle (\delta_1 + 1)(\delta_2 + 1) \rangle \mathrm{d}V_1 \mathrm{d}V_2$,
where $\delta_i(\mathbf{x}) \equiv \rho_i / \bar{\rho} - 1$ is the local over-density evaluated at a given position $\mathbf{x_i}$,
Specifically, $\langle \delta \rangle = 0$.

Correlation function $\xi(\mathbf{x})$

$$\mathrm{d}P_{12}((\mathbf{x})) = \langle(\delta_1+1)(\delta_2+1)\rangle \mathrm{d}V_1\mathrm{d}V_2$$
$$= (1+\langle\delta_1\delta_2\rangle + \langle\delta_1\rangle + \langle\delta_2\rangle)\,\mathrm{d}V_1\mathrm{d}V_2 = (1+\langle\delta_1\delta_2\rangle)\,\mathrm{d}V_1\mathrm{d}V_2$$

$$\xi(\mathbf{x}) \equiv \langle\delta_1\delta_2\rangle = \langle\delta(\mathbf{x}')\delta(\mathbf{x}'+\mathbf{x})\rangle = \frac{1}{V}\int \delta(\mathbf{x}')\delta(\mathbf{x}'+\mathbf{x})\,\mathrm{d}^3x'$$

is the volume-averaged two-point auto-correlation function of the *real* over-density distribution $\delta(\mathbf{x})$, describing *the strength of clustering as a function of scale.*

In particular, $\xi(0) = \langle\,|\,\delta(\mathbf{x})\,|^2\,\rangle$ gives the variance of the density perturbation.

As a result, $\mathrm{d}P_{12}(\mathbf{x}) = <\rho_1\rho_2>/\bar{\rho}^2\mathrm{d}V_1\mathrm{d}V_2 = \big[1+\xi(\mathbf{x})\big]\mathrm{d}V_1\mathrm{d}V_2$, indicating that the auto-correlation function $\xi(\mathbf{x})$ describes *the excess or deficit probability of finding a pair (with a given separation) compared to uniform/random distribution.*

- If $\xi(\mathbf{x}) > 0$, then the clustering strength at this scale is higher than that of the random field.
- If $\xi(\mathbf{x}) = 0$, then the clustering strength at this scale is the same as that of the random field.
- If $\xi(\mathbf{x}) < 0$, then the clustering strength at this scale is lower than that of the random field.

Let's consider the Fourier space correspondence of these quantities.

Correlation function $\xi(\mathbf{x})$ and Power spectrum $P(\mathbf{k})$

Note: $\delta(\mathbf{k}) = \int_V \delta(\mathbf{x})\ \exp(-i\mathbf{k}\cdot\mathbf{x})\,\mathrm{d}\mathbf{x}$, and $\delta_k = \frac{1}{V}\int_V \delta(\mathbf{x})\ \exp(-i\mathbf{k}\cdot\mathbf{x})\,\mathrm{d}\mathbf{x}$,

where $\delta(\mathbf{x})$ is the over-density, not delta function.

Recall the correlation theorem: $H(k) = F^*(k)\cdot G(k)$ for $h(x) = \int_{\infty}^{\infty} f^*(x')g(x+x')\,\mathrm{d}x'$.

In the case of auto-correlation where $g(x) = f(x)$, then $H(k) = |F(k)|^2$.

Now we make $f(\mathbf{x}) = \delta(\mathbf{x})$, and $h(\mathbf{x}) = \int \delta(\mathbf{x}')\delta(\mathbf{x}'+\mathbf{x})\,\mathrm{d}^3x' = V\langle\delta(\mathbf{x}')\delta(\mathbf{x}'+\mathbf{x})\rangle = V\xi(\mathbf{x})$

then the Fourier transform $F(\mathbf{k}) = \delta(\mathbf{k}) = V\delta_{\mathbf{k}}$, $H(\mathbf{k}) = V\ \mathrm{FT}[\xi(\mathbf{x})] = |\delta(\mathbf{k})|^2 = V^2|\delta_{\mathbf{k}}|^2$.

Let us define power spectrum $P(\mathbf{k}) \equiv |\delta(\mathbf{k})|^2/V = |\delta_{\mathbf{k}}|^2 V$, i.e.,

$$\mathrm{FT}[\xi(\mathbf{x})] = P(\mathbf{k}) = \int \xi(\mathbf{x})\ \exp(-i\mathbf{k}\cdot\mathbf{x})\,\mathrm{d}^3x$$

$$\xi(\mathbf{x}) = \frac{1}{(2\pi)^3}\int P(\mathbf{k})\ \exp(i\mathbf{k}\cdot\mathbf{x})\,\mathrm{d}^3k$$

Power spectrum ↔ Autocorrelation
(in one domain) (in the other domain)

With the help of FFT, two-point correlation functions and the power spectrum can be connected can be calculated through Fourier transform!
Theoretically *the two provide equal amount of information.*

Correlation function $\xi(\mathbf{x})$ and Power spectrum $P(\mathbf{k})$

The Fourier transform of $V\xi(\vec{x}) = \int \delta(\vec{x'})\delta(\vec{x'}+\vec{x})\,\mathrm{d}\vec{x'}$ is given by:

$$V\int \xi(\mathbf{x})\,\exp(-i\mathbf{k}\cdot\mathbf{x})\mathrm{d}\mathbf{x} = \iint \delta(\mathbf{x}+\mathbf{x}')\delta(\mathbf{x}')\,\exp(-i\mathbf{k}\cdot\mathbf{x})\mathrm{d}\mathbf{x}'\mathrm{d}\mathbf{x}$$

$$= \int \delta(\mathbf{x}+\mathbf{x}')\,\exp[-i\mathbf{k}\cdot(\mathbf{x}+\mathbf{x}')]\,\mathrm{d}(\mathbf{x}+\mathbf{x}') \int \delta(\mathbf{x}')\,\exp(i\mathbf{k}\cdot\mathbf{x}')\mathrm{d}\mathbf{x}'$$

$$= \tilde{\delta}(\mathbf{k})\cdot\delta^*(\mathbf{k}) = |\,\tilde{\delta}(\mathbf{k})\,|^2 \equiv VP(\mathbf{k})$$ — proof, work this out yourself!

In the case of isotropy (i.e., independent of direction), $\xi(\mathbf{x})$ becomes 1d correlation function $\xi(r)$, where r is the separation of any two positions. And $P(\mathbf{k})$ becomes 1d power spectrum $P(k)$, where $k = |\mathbf{k}| = \sqrt{k_\mathrm{x}^2 + k_\mathrm{y}^2 + k_\mathrm{z}^2}$.

$$P(k) = \int_0^\infty \xi(r)\frac{\sin kr}{kr}4\pi r^2\,\mathrm{d}r, \qquad \xi(r) = \frac{1}{(2\pi)^3}\int_0^\infty P(k)\frac{\sin kr}{kr}4\pi k^2\,\mathrm{d}k$$

In particular, $\xi(0) = \dfrac{1}{(2\pi)^3}\displaystyle\int_0^\infty P(k)4\pi k^2\,\mathrm{d}k = \frac{1}{2\pi^2}\int_{-\infty}^\infty k^3P(k)\,\mathrm{d}\ln k = \int_{-\infty}^\infty \mathscr{P}(k)\,\mathrm{d}\ln k,$

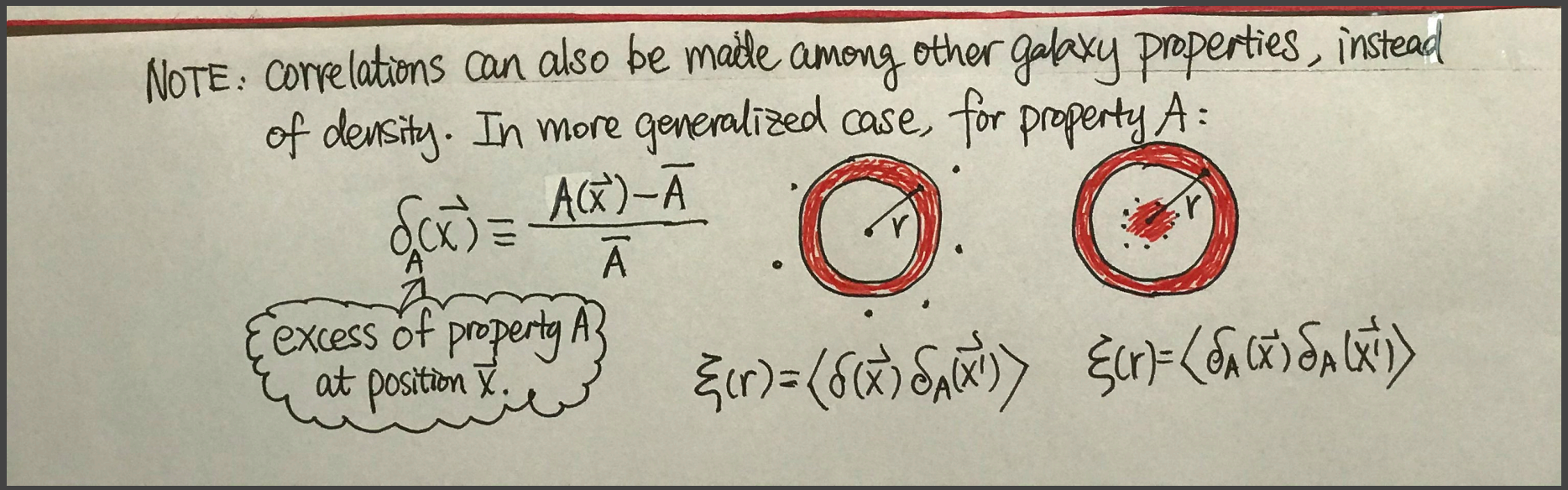
where $\mathscr{P}(k) \equiv \dfrac{k^3}{2\pi^2}P(k)$ is the (dimensionless) power spectrum per $\ln k$ interval, while $P(k)$ is the power per d^3k internal and with dimension of V.

Correlation function

Two-point correlation function of the over-density field and the power spectrum introduced here, are important concepts in the field of cosmological density perturbation and structure evolution.

Two-point auto-correlation function does not have to be only of the over-density field of a given point distribution. It can be for any properties, for example, galaxy mass, color, even formation time etc.

Two-point correlation functions do not have to be in form of auto-correlation.



The calculation of two-point correlation function does not have to be necessarily through inverse Fourier transform, we can also calculate this statistical quantity using pair and cell estimation.

Correlation function - at higher order

Just like two-point correlation function, we can also define three-point correlation function as:

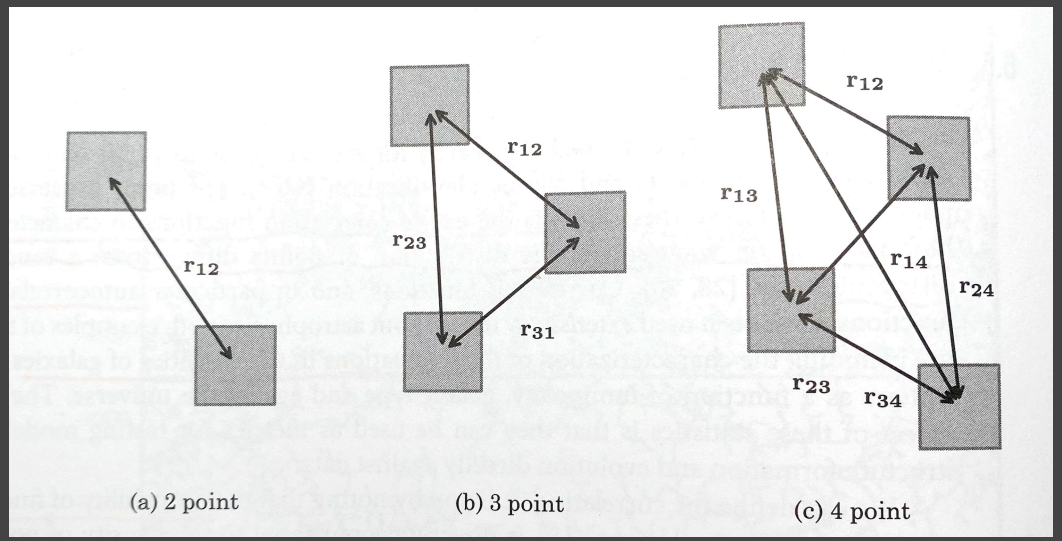
$$\mathrm{d}P_{123}(r) = \langle \rho_1 \rho_2 \rho_3 \rangle / \bar{\rho}^3 \mathrm{d}V_1 \mathrm{d}V_2 \mathrm{d}V_3 = \left[1 + \xi(r_{12}) + \xi(r_{23}) + \xi(r_{13}) + \zeta(r_{12}, r_{23}, r_{13})\right] \mathrm{d}V_1 \mathrm{d}V_2 \mathrm{d}V_3$$

where ζ is the connected three-point correlation function.

The corresponding spectrum in the Fourier domain is called the "bispectrum" $B(k_1, k_2, k_3)$.

Also, four-point correlation function corresponds to trispectrum in the Fourier domain.

For a field generated through a Gaussian random process, two-point correlation function or the power spectrum provides a full statistical description to the field. Any non-Gaussianity and/or asymmetry in the underlying mechanism that generates the field can only be reflected by higher-order statistics, e.g., three- or four-point correlation functions (or equivalently, bispectrum and trispectrum in the Fourier domain).



Note: higher-order correlation functions depends on the configuration of the triplets (3-point) and quadruplets (4-point). For example, the commonly used configuration including equilateral triangle configuration.

Correlation function - (1)

Calculation via FFT (last lecture)

Correlation function - (2)

Pair estimate method

In order to evaluate the 1D two point correlation function below:

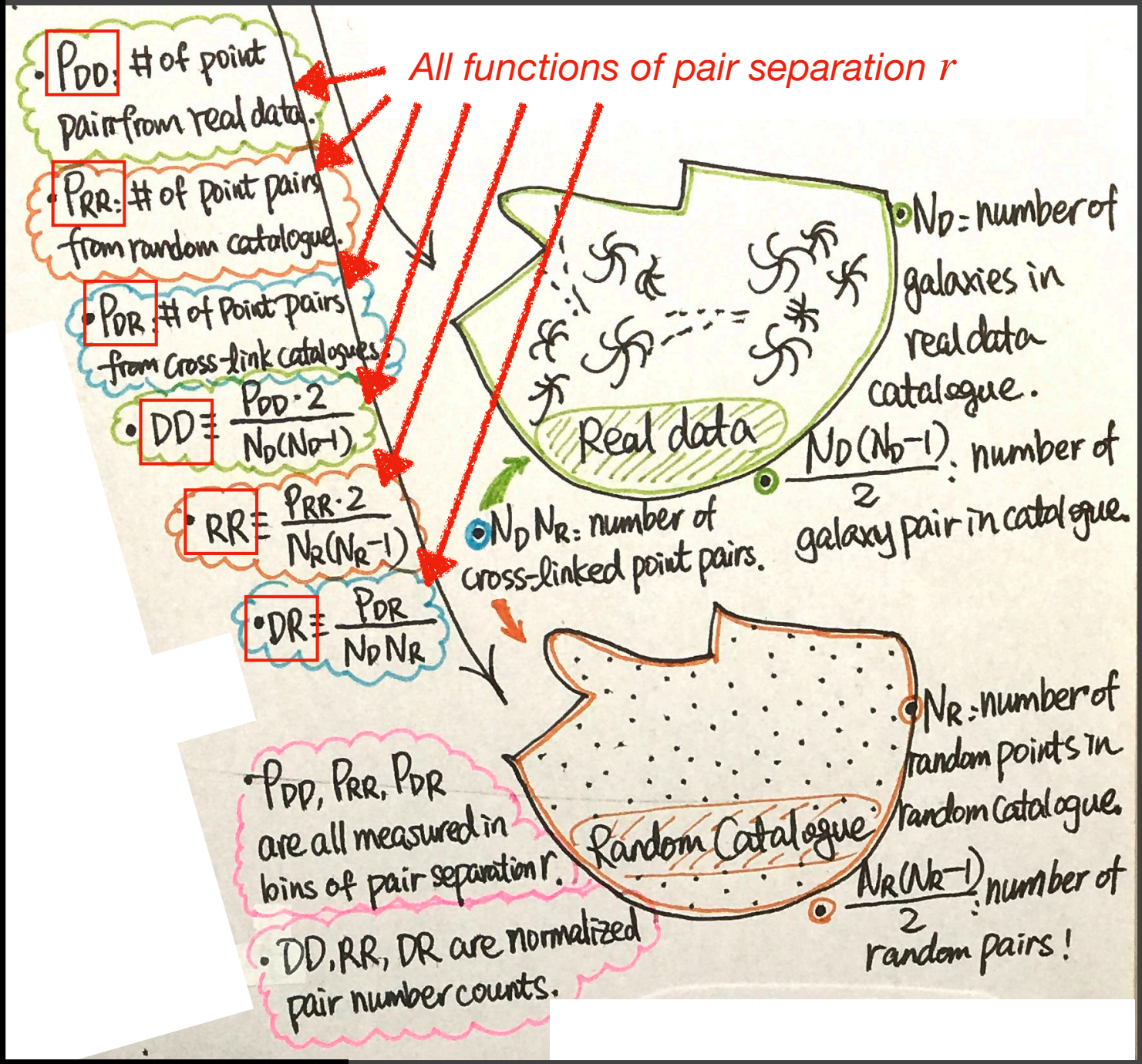
$$\xi(r \equiv x' - x) = < \delta(x)\delta(x') > = < \frac{\rho(x) - \bar{\rho}}{\bar{\rho}} \frac{\rho(x') - \bar{\rho}}{\bar{\rho}} >$$

$$= \frac{1}{\bar{\rho}^2}\left(\langle \rho(x)\rho(x') \rangle - \bar{\rho}\langle \rho(x) \rangle - \bar{\rho}\langle \rho(x') \rangle + \bar{\rho}\bar{\rho}\right)$$

Based on this formula, estimators have been designed at various levels, through the implementation of a randomly realized distribution, which has the same domain as the data distribution (but much denser) and is introduced to help to describe the uniform underlying field with a mean density of $\bar{\rho}$.

Correlation function

Pair estimate method



Note: $N_D(N_D-1)/2$, $N_R(N_R-1)/2$, $N_D N_R$ are the total pair counting statistics in the two fields.

While $P_{DD}(r)$, $P_{RR}(r)$, $P_{DR}(r)$ and the normalized $DD(r)$, $RR(r)$, $DR(r)$ are functions of pair separation r.

Note: typically we make the random field (of same shape but) 20 times denser than the real data field, so that the shot noise of this random field will not dominate the total variance of the estimator.

Correlation function

Pair estimate method

Historically used estimators:

- Peebles & Hauser (1974): $\xi_{\rm PH}(r) = \dfrac{DD(r)}{RR(r)} - 1 = \dfrac{N_{\rm R}(N_{\rm R}-1)P_{\rm DD}(r)}{N_D(N_{\rm D}-1)P_{\rm RR}(r)} - 1$
- Davis & Peebles (1983): $\xi_{\rm DP}(r) = \dfrac{DD(r)}{DR(r)} - 1 = \dfrac{2N_{\rm R}P_{\rm DD}(r)}{(N_{\rm D}-1)P_{\rm DR}(r)} - 1$
- Hamilton (1993): $\xi_{\rm H}(r) = \dfrac{DD(r)RR(r)}{DR^2(r)} - 1 = \dfrac{4N_{\rm D}N_{\rm R}}{(N_{\rm D}-1)(N_{\rm R}-1)}\dfrac{P_{\rm DD}(r)P_{\rm RR}(r)}{P^2_{\rm DR}(r)} - 1$
- Landy & Szalay (1993):

$$\xi_{\rm LS}(r) = \frac{DD(r) - 2DR(r) + RR(r)}{RR(r)} = \frac{N_{\rm R}(N_{\rm R}-1)P_{\rm DD}(r)}{N_{\rm D}(N_{\rm D}-1)P_{\rm RR}(r)} - \frac{(N_{\rm R}-1)P_{\rm DR}(r)}{N_{\rm D}P_{\rm RR}(r)} + 1$$

* the Landy-Szalay (1993) estimator is less sensitive to edge effect, which affects clustering measurements on large scales, and less sensitive to size of the random catalogue. It is thus the best estimator among all three up to now.

For implementation of correlation for other properties w, replace $DD(r)$, $RR(r)$, $DR(r)$ with $D_{\rm w}D_{\rm w}(r)$, $R_{\rm w}R_{\rm w}(r)$, $D_{\rm w}R_{\rm w}(r)$, i.e., replace number count 1 with property w.

Note: the method is straightforward, however computational expansive!
Public packages using effective tree-based algorithms are available to do so.

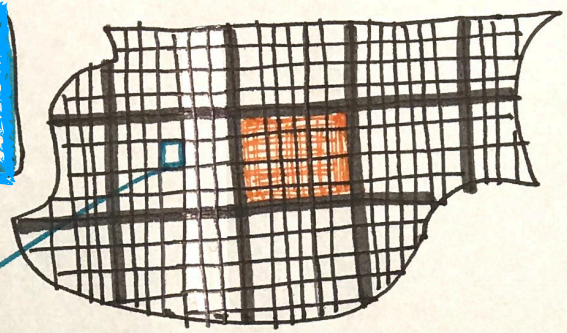
Correlation function

Cell estimate method

Pair estimation is extremely expansive at larger distances, we go for

In addition, one can use Pair-estimator method for correlations on smaller scales, and use Cell-estimator methods for correlations on larger scales.

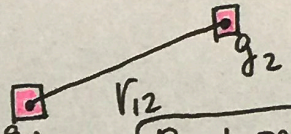
Cell-estimator



eg.
Maller et al. 2005, ApJ, 619, 147
Scranton et al. 2002, ApJ, 579, 48

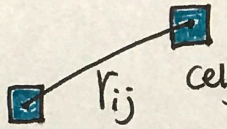
Galaxies live in cells,
cells live in tiles,
tiles live in the domain.

- Each cell has a property: $\delta_i^g = \frac{n_i^g - \bar{n}^g}{\bar{n}^g}$ average overdensity of cell i.
  n_i^g: the number of galaxies in cell i.
- Each tile has a property:
  $\bar{n}^g$: the average number of galaxies per cell, over all cells in that tile.



Point-pair counting

REPLACE



Cell-pair counting

Correlation function

Cell estimate method

over-density of cell i

Correlation function: $\xi(r) \equiv \langle \delta(x)\delta(x') \rangle = \frac{\Sigma_{i,j}\, \delta_i^{\rm g}\, \delta_j^{\rm g}\, \theta^{ij}(r_{ij})}{\Sigma_{i,j}\, \theta^{ij}(r_{ij})}$,

where $\theta^{ij}(r_{ij}) = 1$ if r_{ij} is in the r bin, and =0 otherwise.

Final estimate: $\xi(r)$ given above will be calculated using all the cells that live in all the tiles except for one tile, each time. This will be carried out for $N_{\rm tile}$ times.

The final mean is even by: $\bar{\xi}(r) = \frac{1}{N_{\rm tile}} \Sigma_k^{N_{\rm tile}}\, \xi_k(r)$

Jackknife Leave-One-Out

where $\xi_k(r)$ is the measurement using all cells in all tiles except for the k-th tile.

With jackknife estimate, the variance of the mean is given by:

$$\left[\Delta\xi(r)\right]^2 = \frac{N_{\rm tile} - 1}{N_{\rm tile}} \Sigma_k^{N_{\rm tile}} \left[\xi_k(r) - \bar{\xi}(r)\right]^2$$

The analysis has rich details, see Scranton et al. 2002, ApJ, 579, 48 (arXiv: 0107416)

For implementation of doing other auto-correlation or cross-correlation,

replace $\delta_i^{\rm g}$ with $\delta_i^w = \frac{w_i - \bar{w}}{\bar{w}}$, where w is some other property of a galaxy.

- ***Correlation function***
- ***Density estimation***
- ***Clustering in data***
- ***Classification***

Non-parametric models:

0. Histogram

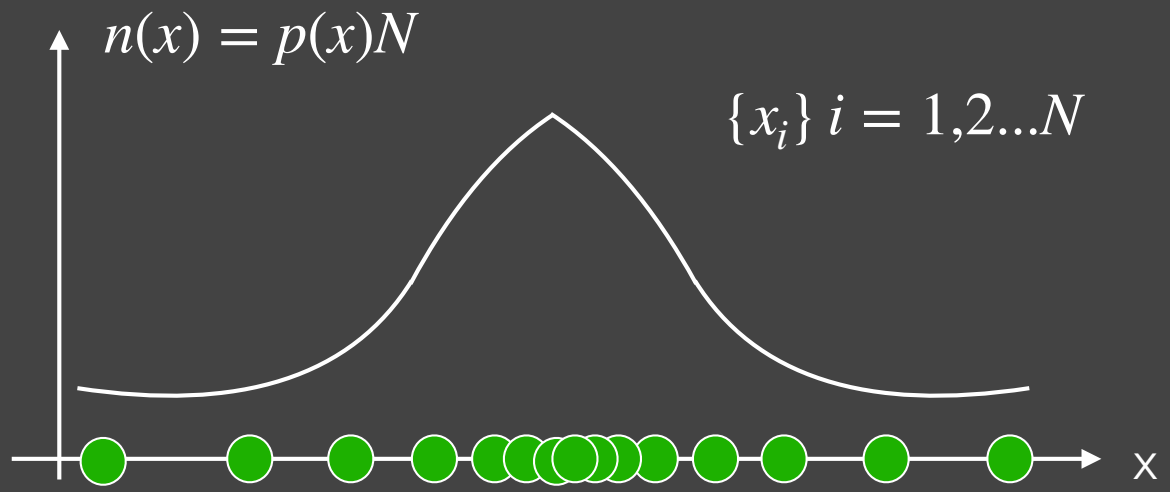
1. Kernel Density Estimator

2. Nearest-Neighbour density estimator

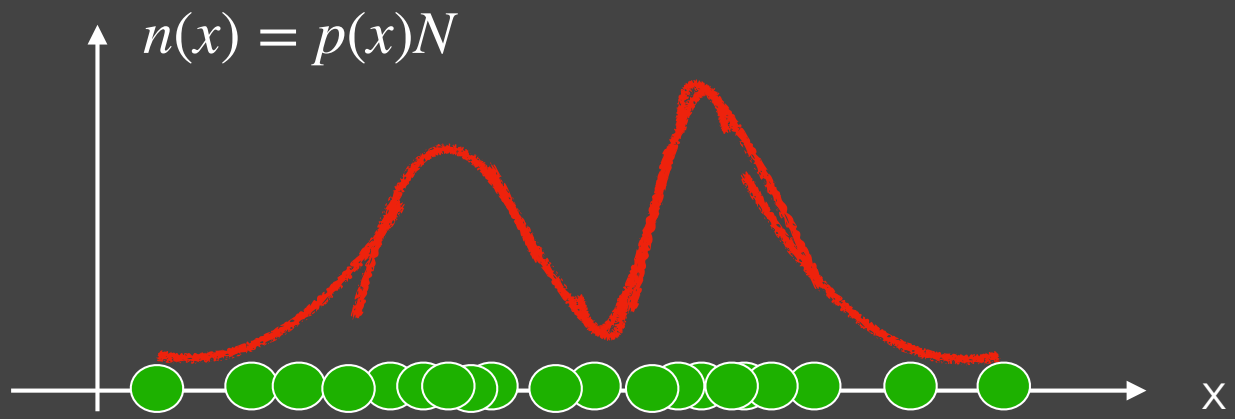
Parametric models:

3. Gaussian Mixture Model (GMM)

Density estimation of point data



Unimodal distribution



More complicated distribution

May come from e.g., measurements for histogram counting purpose, or making density predictions, or MCMC sampling for marginalized pdf estimation etc…

Density Estimation

Bayesian posterior pdf marginalization and parameter estimation, credible region

Applications of Markov Chain

Let $p(\vec{\theta}) \equiv p(\vec{\theta}\,|D,I) \propto p(D|\,\vec{\theta},I)\,p(\vec{\theta}\,|I)$

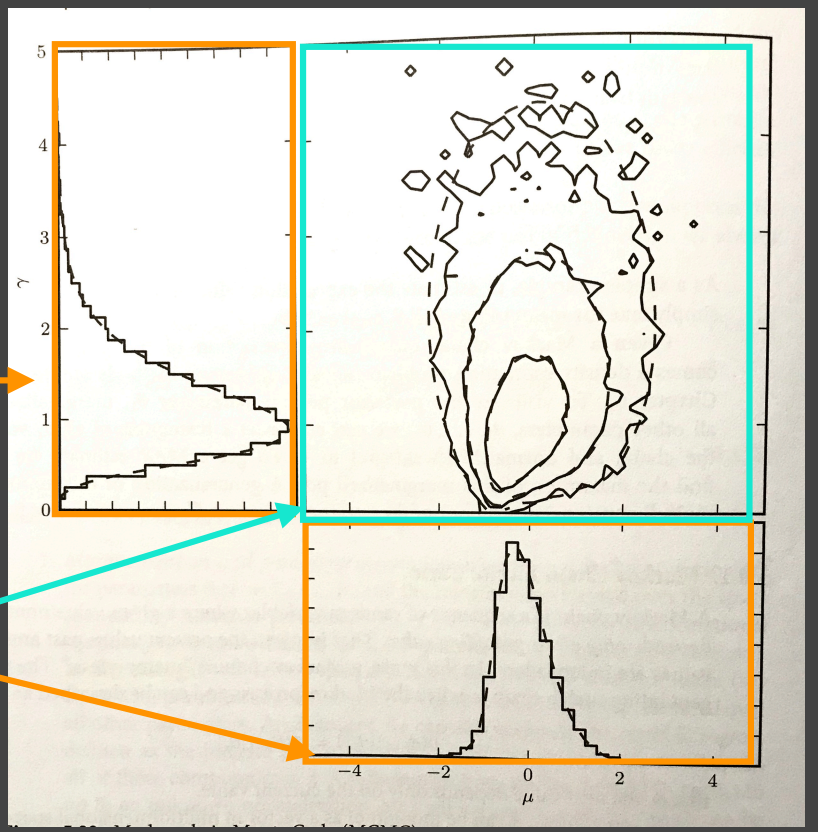
Many estimates are integrals of the following form:

$$I = \int g(\vec{\theta})\,p(\vec{\theta})\,\mathrm{d}\vec{\theta}$$

- To obtain the marginalized posterior of θ_i, one simply compute the *histogram* of θ_i using all members (after initial burn-in) in the chain, through binning the number counts or kernel estimation in the relevant parameter space.

- To obtain the covariance between any two parameters θ_m and θ_n (i.e., corner-plot), one simply takes the two relevant parameters in the pair of chain members (after initial burn-in).

$\ln[p(\mu,\gamma\,|\,\{x_i\},I)] = \mathrm{const} + (N-1)\ln\gamma - \Sigma_{i=1}^{N}\,\ln[\gamma^2 + (x_i-\mu)^2]$



Joint posterior from $N = 10$ measurements of $\{x_i\}$, which was generated to follow a Cauchy distribution with $\mu = 0, \gamma = 2$. Bayesian inference with flat priors on μ, γ.

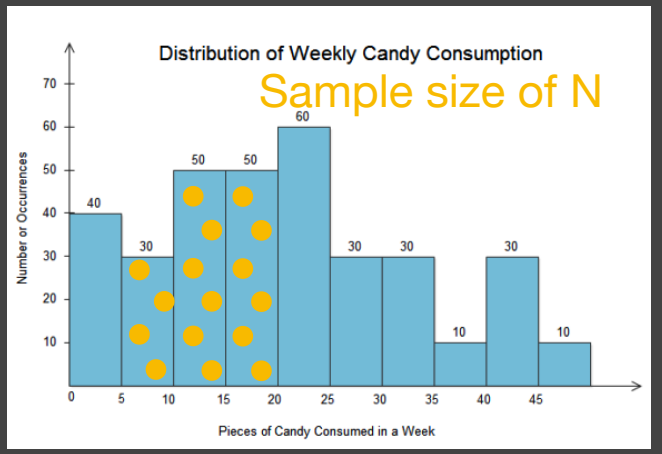
Previous lecture - MCMC, marginalized posteriors

Density Estimation

Non-parametric data description

0. Histogram: When there is no strong motivation for parametric description, go for nonparametric methods to describe your data set!

- Implicitly assuming that the estimated distribution function is piecewise constant within each bin!
- Allowing us to visualize the data and widely used in machine learning!
- Not completely parameter free!!
- *Parameter: the number of bins!*
- Bin size Δ_b ***too big***: miss fine structure in data! ***Too small***: introduce counting noise!



How to decide ideal bin number/width? The gold standard is doing cross-validation!

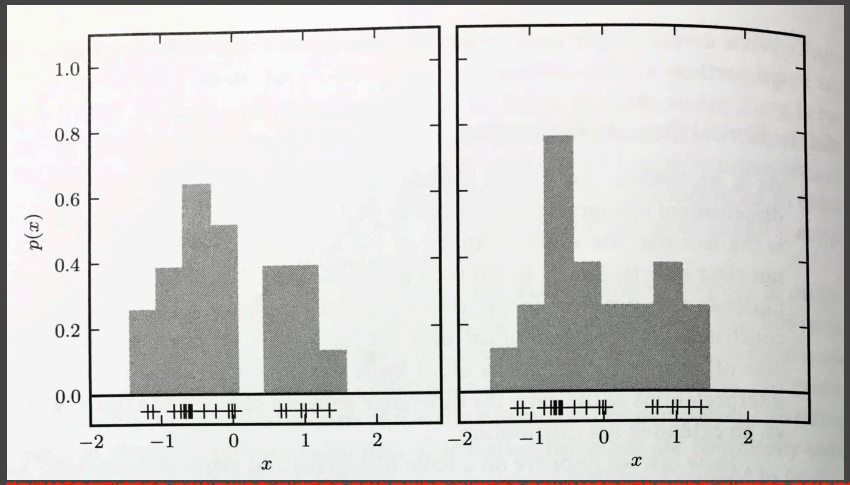
Scott's rule: $\Delta_b = \dfrac{3.5\sigma}{N^{1/3}}$ σ is the sample standard deviation, *N* is sample size. Assuming underlying distribution is *Gaussian*!

Freedman-Diaconis rule: $\Delta_b = \dfrac{2(q_{75} - q_{25})}{N^{1/3}} = \dfrac{2.7\sigma_G}{N^{1/3}}$ for *non-Gaussian & unimodal* underlying distributions

Rules of thumb above is highly simplified for quick visualization purposes.
For more complicated model, see the power of Bayesian!

Non-parametric density estimator

1. Kernel Density Estimator



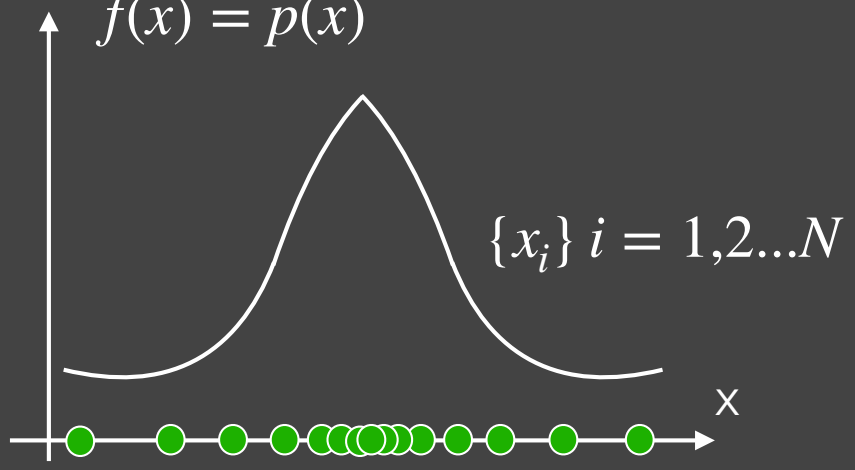
For such estimates, binning is not always necessary, but convenient for visualization purposes. However, if regular binning is involved in estimation, see following example.

Same data, using two histograms, same bin widths, offset by 0.25

- Left: might be a bi-modal distribution
- Right: might be an extended flat distribution

One possible solution:

Instead of binning data points to a regular grid, allow each point to have its own bin, and allow the bins to overlap! Then each point is replaced by a box of unit height and some predefined width, i.e., each point contributes to a "one" to the total density distribution.



Kernel density estimator (KDE)

is an estimator of the underlying *pdf* $\tilde{f}_N(\vec{x})$ at position $\vec{x}$ of dimension D:

$$\tilde{f}_N(\vec{x}) = \frac{1}{Nh^D}\Sigma_{i=1}^{N} K\left(\frac{d(\vec{x}, \vec{x}_i)}{h}\right)$$

where $K(u)$ is the kernel function and h is the bandwidth; $d(\vec{x}, \vec{x}_i)$ is a distance.

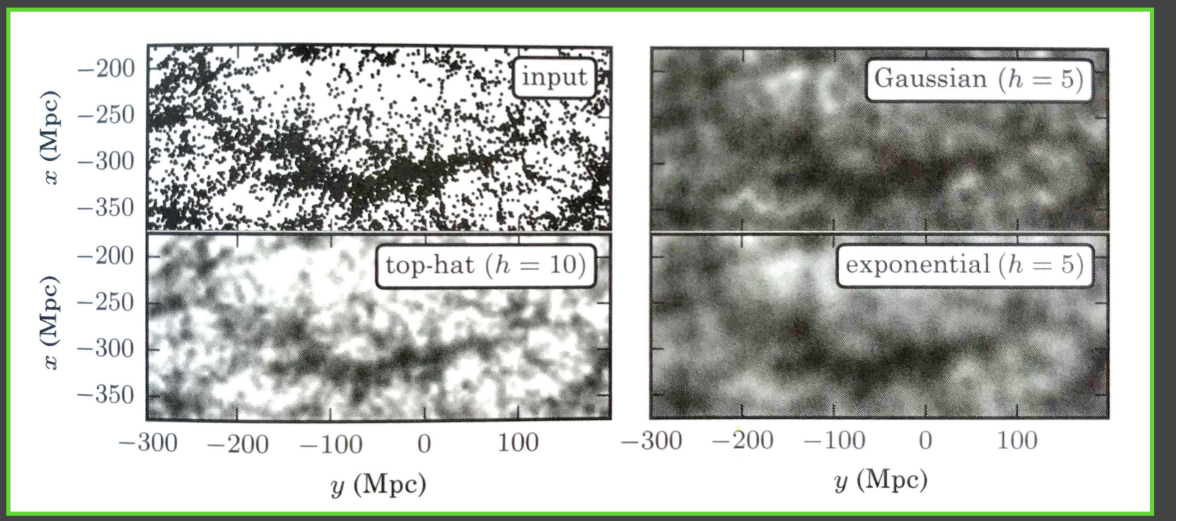
Non-parametric density estimator

Kernel density estimator

$$K(u) \geqslant 0, \quad \int K(u)\,\mathrm{d}u = 1$$

$$\mu_K = \int uK(u)\,\mathrm{d}u = 0,$$

$$\sigma_K^2 = \int u^2 K(u)\,\mathrm{d}u > 0$$



● Commonly used kernel functions:

① Gaussian kernel

$$K(u) = \frac{1}{(2\pi)^{D/2}} \exp(-u^2/2)$$

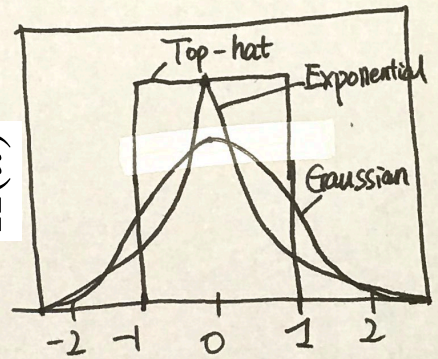
② Top-hat (Box) kernel:

$$K(u) = \begin{cases} \frac{1}{V_D(1)} & \text{if } u \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

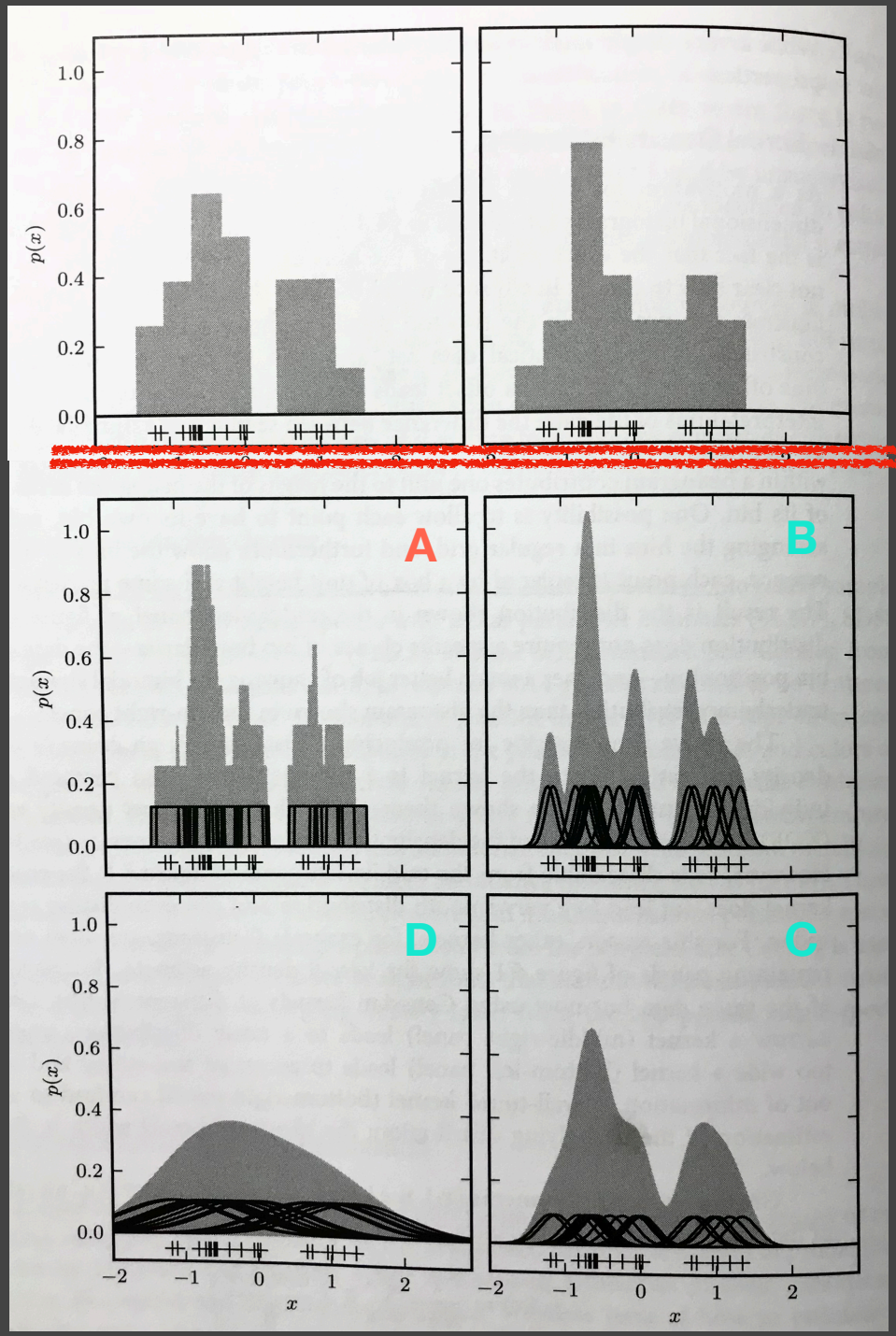
③ Exponential kernel:

$$K(u) = \frac{1}{D!\,V_D(1)} e^{-|u|}$$

- $u = \frac{d(\vec{x}, \vec{x}_i)}{h}$
- $V_D(r)$ is the Volume of a D-dimensional hypersphere of radius r.



Kernel density estimator



Top: histograms, same bin width, offset by 0.25

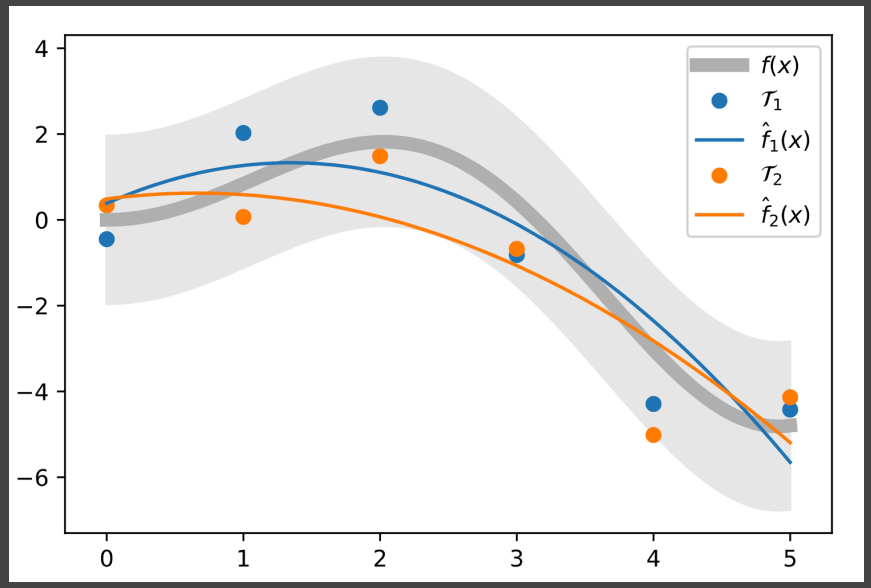
Middle and bottom: all four are using kernels

- **A:** top hat, each bin centered on individual data points
- **B-D:** Gaussian kernels with different (increasing) bandwidths.

Notice the different effects on bandwidth h !

Model bias vs. Variance

Datasets τ_1 and τ_2 … are generated by the same *true* model $t(x)$, but each time being a different realization with different noise:



The model regression even using model $t(x)$, will result $\hat{f}_1(\{x\}_{\tau_1})$ for a finite dataset τ_1, which will not be the same as $\hat{f}_2(\{x\}_{\tau_2})$ for dataset τ_2 and so on, due to two effects:

(1) due to *random noise* $\tau_1 \neq \tau_2$; and (2) discrete sampling in x.

As a result, when doing model regression using ant given model $f(x)$, we have:

***Model* Bias**: $\text{Bias}(x) = \text{E}_\tau[\hat{f}_\tau(x)] - t(x)$

— If *model bias* is large, $\text{E}_\tau[\hat{f}_\tau(x)]$ cannot converge to $t(x)$, *model* $f(x)$ *has bias!*

***Model* Variance**: $\text{Variance}(x) = \text{E}_\tau([\hat{f}_\tau(x) - \text{E}_\tau(\hat{f}_\tau(x))]^2)$

— scatter among regression model predictions using multiple realizations of measurements

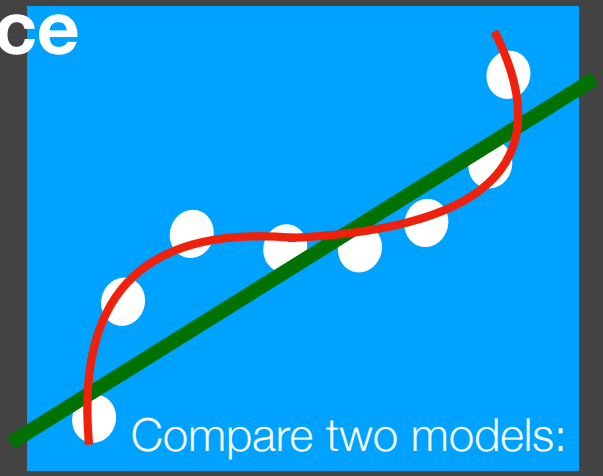
***Model* Mean square error**: $\text{MSE}(x) = \text{E}_\tau([\hat{f}_\tau(x) - t(x)]^2)$

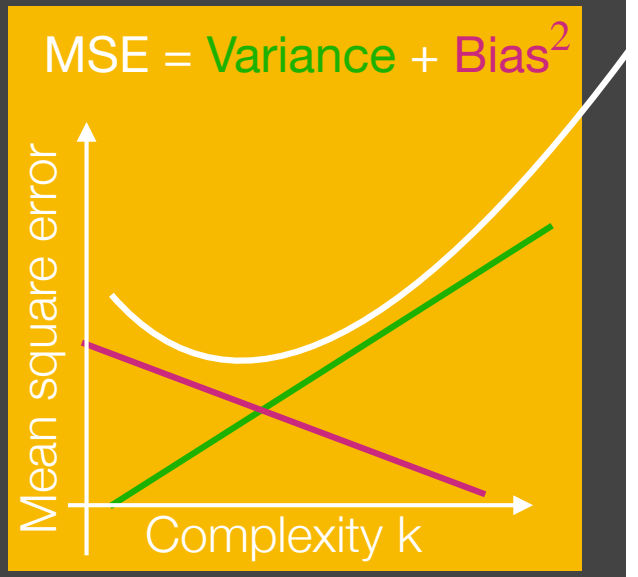
— (square) error in regression model predictions when compared to the ground-truth.

MSE = Variance + Bias^2

Best model estimate will minimize MSE!!

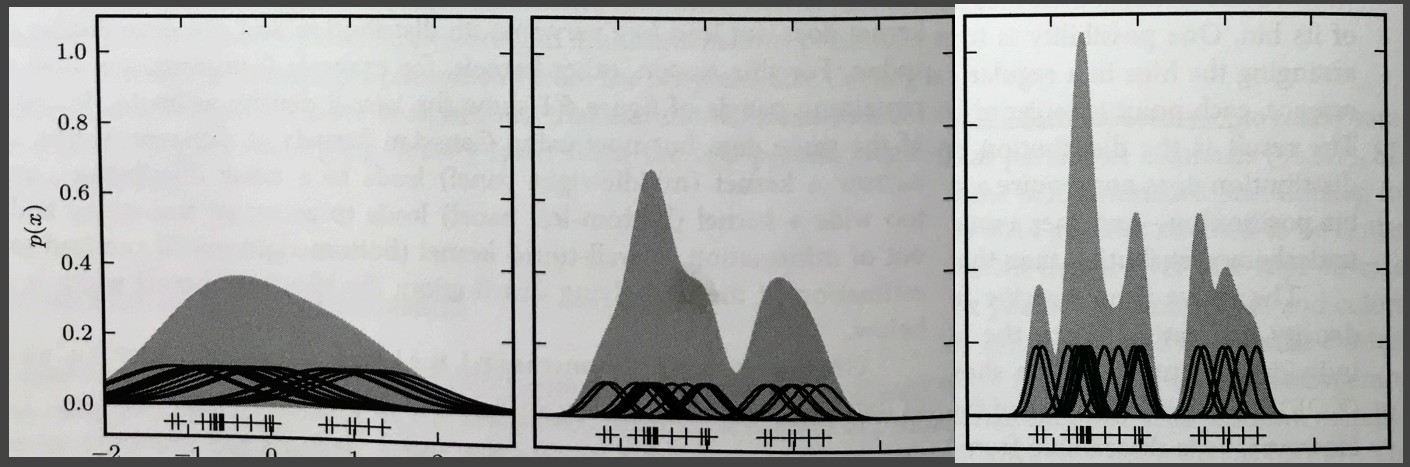
Model bias vs. Variance





Bigger bias, smaller variance

Smaller bias, bigger variance



MSE = Variance + Bias^2

if h is too large, model variance is small but we increase model bias in density estimation!

If h is too small, model bias is decreased but we increase model variance (fitting to noise);

In general how to decide bandwidth h ?

Non-parametric density estimator

Density Estimation

Cross-Validation

Previous lecture

Application example with leave-one-out CV:

Minimizing CV error to find best regularization

$$\epsilon_{\text{cv}}(\lambda) = \frac{1}{k}\Sigma_k \frac{1}{N_k}\Sigma_i^{N_k} \frac{[y_i - f(x_i \mid \vec{\theta}(\lambda)]^2}{\sigma_i^2}$$

Cross-validation

How do we choose the bandwidth h of KDE?

Best value of h would minimize the following L_2 cross-validation

$\text{CV}_{L_2}(h) \equiv \int \tilde{f}_h^2 - 2\frac{1}{N}\Sigma_{i=1}^N \tilde{f}_{h,-i}(x_i)$, where $\tilde{f}_h$ is the KDE density estimate.

This is equivalent to minimize the mean integrated square error (MISE),

$\int (\tilde{f}_h - f)^2 = \int \tilde{f}_h^2 - 2\int \tilde{f}_h f + \int f^2$. Because the theoretical density distribution f does not depend on h (thus dropped directly during minimization). Minimizing MISE is 'equivalent' to minimize $\text{CV}_{L_2}(h) \equiv \int \tilde{f}_h^2 - 2\frac{1}{N}\Sigma_{i=1}^N \tilde{f}_{h,-i}(x_i)$.

Optimal bandwidth h decreases with sample size at rate of $N^{-1/5}$, while KDE error with optimal bandwidth h converge at a rate $N^{-4/5}$ (faster than histogram ~ $N^{-2/3}$), this is the best any density estimator can do.

*Note: given bandwidth h, the *optimal* kernel function in terms of minimum variance, is given by*

$K(u) = \frac{3}{4}(1-u^2)$ *for* $|u| \leqslant 1$ *and 0 elsewhere — the Epanechnikov kernel.*

Non-parametric density estimator

2. Nearest-Neighbour density estimator

Nearest-neighbor density estimator is given by: $\hat{f}_K(\vec{x}) = \frac{K}{V_D(d_K)}$,

Assuming underlying density field is "locally" constant in the neighborhood!

where d_K is the distance from $\vec{x}$ to its K-th nearest neighbor; V_D is volume within d_K in D dimension.

The error in $\hat{f}_K(\vec{x})$ is $\sigma_f = K^{1/2}/V_D(d_K)$, the fractional error is $\sigma_f/\hat{f} = 1/K^{1/2}$.

The behavior of the inverse of the fractional error is like signal-to-noise ratio; the larger *K*, the higher S/N and the lower fractional error! However,

1. Higher fractional accuracy (with larger *K*) is achieved at the expanse of losing spatial resolution — towards the regime of larger model bias.
2. Higher spatial resolution (with smaller *K*) will cause larger variance in density estimates (governed by local noise). In practice, make *K*>5!

Can we improve this situation?

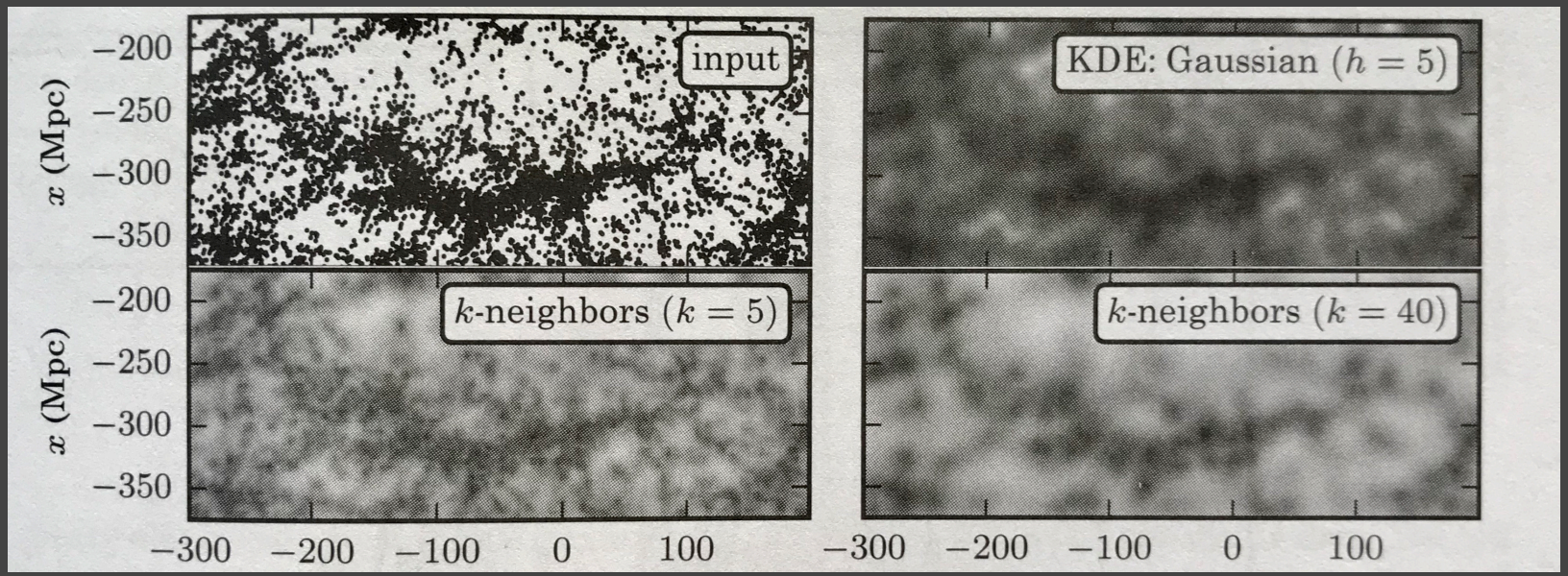
Using all distances $d_{i=1,2...K}$ to all *K* neighbors of $\vec{x}$ (derived based on Bayesian analysis):

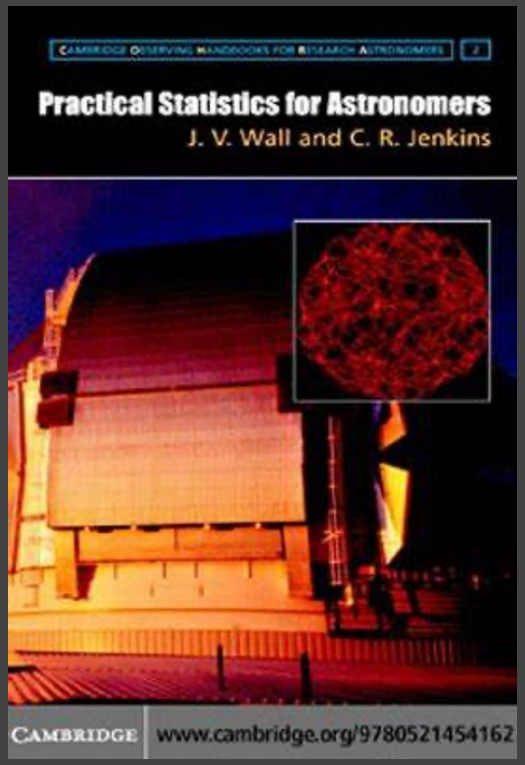
$$\hat{f}_K(\vec{x}) = \frac{C}{\Sigma_{i=1}^{K} d_i^D} \quad \text{and} \quad C = \frac{K(K+1)}{2V_D(1)}$$

Reducing error without losing spatial resolution! Powerful in the case of sparse data!

Non-parametric density estimator

2. Nearest-Neighbour density estimator





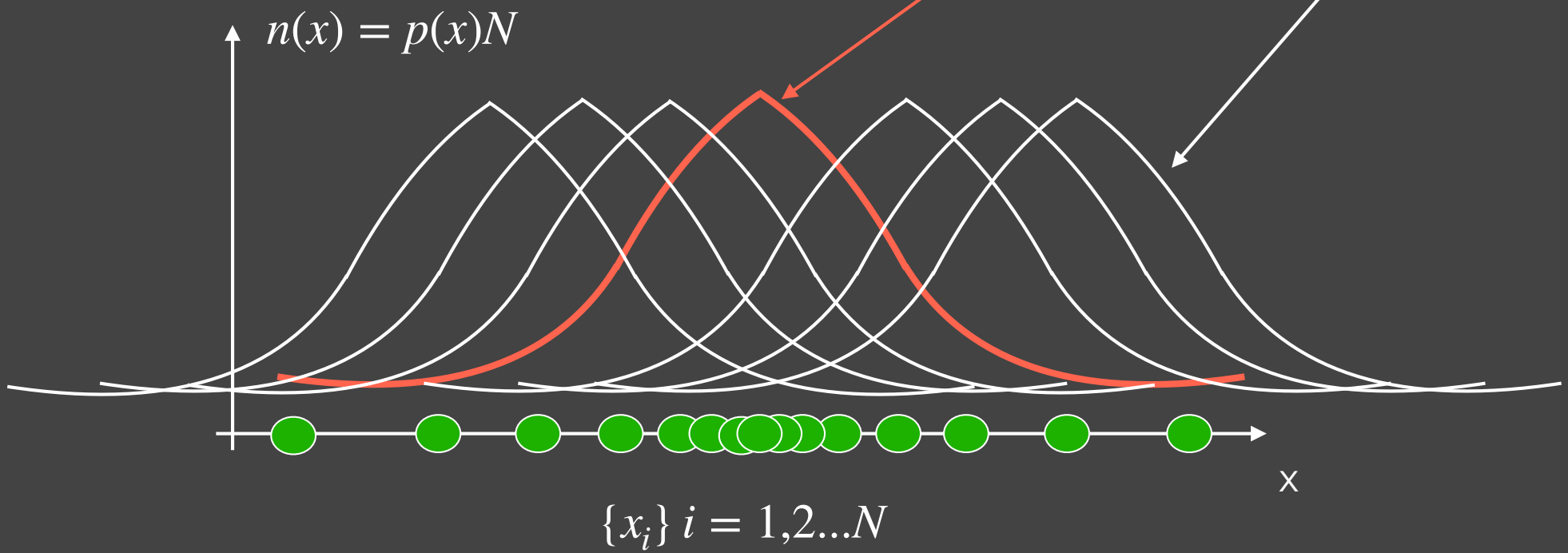
Smaller *K* allows reconstruction of finer structures (including noise) however at the cost of larger variance in density estimation; larger *K* leads to smoother distribution but at the cost of larger biased.

"The application of efficient statistical procedure has power; but the application of common sense has more!" — Wall and Jenkins

Non-parametric density estimator

3. Gaussian Mixture Model (GMM)

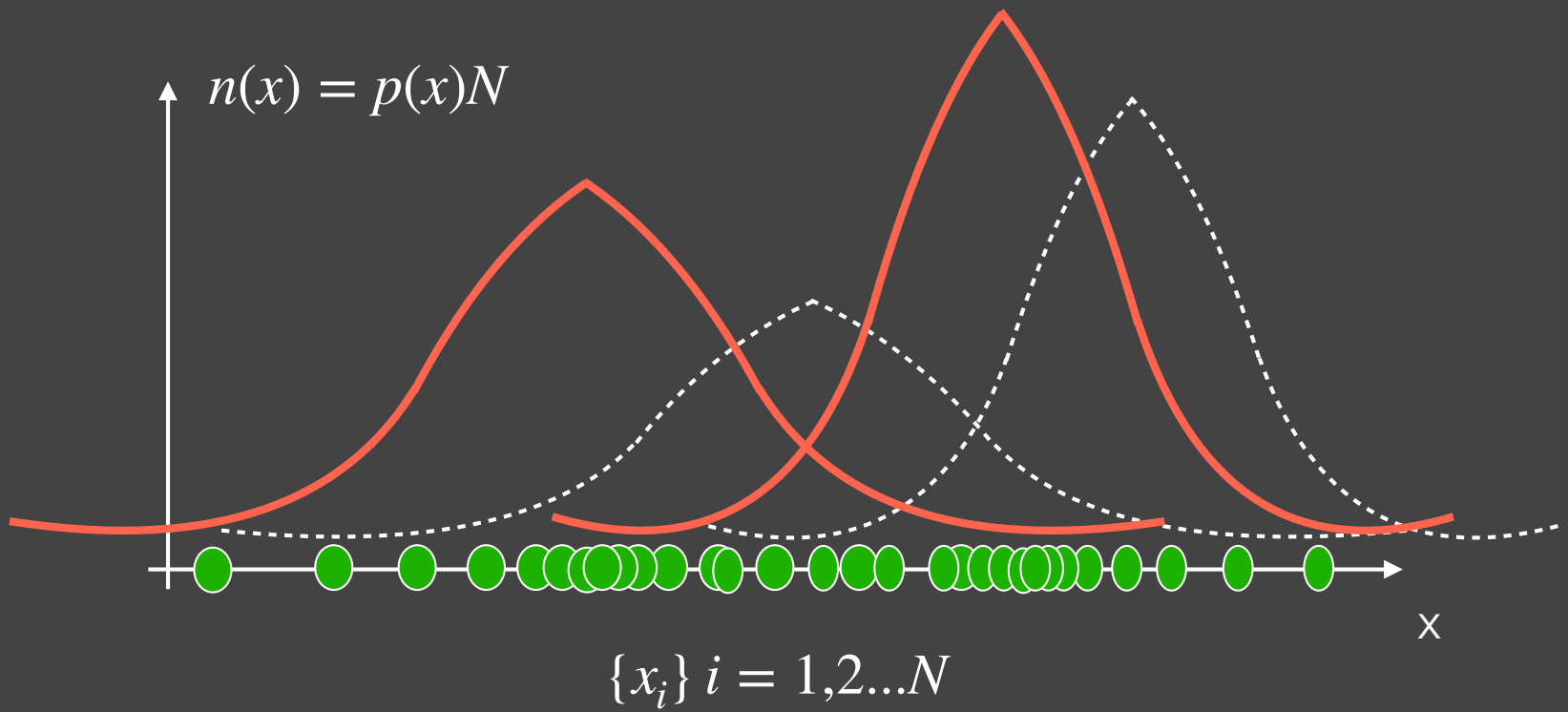
Assuming the data follow a true Gaussian ~ $\mathcal{N}(\mu_x^*, \sigma_x^{*2})$, then (μ_x^*, σ_x^*) must maximize the *joint* data likelihood $p(\{x_i\} \mid \mu_x, \sigma_x)$ or the posterior $p(\mu_x, \sigma_x \mid \{x_i\}) \propto \Pi_i \, p(x_i \mid \mu_x, \sigma_x) \, p(\mu_x, \sigma_x)$, in comparison to other sets of (μ_x, σ_x).



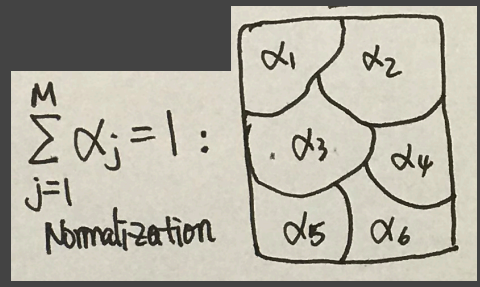
Parametric density estimator

3. Gaussian Mixture Model (GMM)

Now if the data are generated by the sum of two Gaussians, i.e., the distribution ~ $\alpha_1^*\mathcal{N}(\mu_1^*, \sigma_1^{*2}) + \alpha_2^*\mathcal{N}(\mu_2^*, \sigma_2^{*2})$, then $(\mu_1^*, \sigma_1^*, \alpha_1^*, \mu_2^*, \sigma_2^*, \alpha_2^*)$ must maximize the *joint* data likelihood or the posterior, in comparison to any other sets of $(\mu_1, \sigma_1, \alpha_1, \mu_2, \sigma_2, \alpha_2)$.



3. Gaussian Mixture Model (GMM)



Now consider a simple 1d case, assuming the underlying data density distribution is described by M Gaussian components, each $\mathcal{N}(\mu_j, \sigma_j^2)$ with a fractional amplitude of α_j, that $\Sigma_{j=1}^{M} \alpha_j = 1$. The data probability density function at x_i is thus given by: $p_{\mathrm{d}}(x_i \mid \vec{\theta}) = \Sigma_{j=1}^{M} \alpha_j \mathcal{N}(\mu_j, \sigma_j^2)\Big|_{x_i}$, where $\vec{\theta} = \{\alpha_{j=1,2...M}, \mu_j, \sigma_j\}$. This can be generalized to $p_{\mathrm{d}}(x_i \mid \vec{\theta}) = \Sigma_{j=1}^{M} \alpha_j p_*(x_i \mid \vec{\theta})$, in the case where each component's pdf $p_*(\vec{\theta})$ is not given by Gaussian.

This pdf can also be viewed as a likelihood for datum x_i. Then for the whole dataset, the joint likelihood is given by: $L = \Pi_{i=1}^{N} [\Sigma_{j=1}^{M} \alpha_j \mathcal{N}(\mu_j, \sigma_j^2)]$, the log-likelihood is then: $\ln L = \Sigma_{i=1}^{N} \ln \left[\Sigma_{j=1}^{M} \alpha_j \mathcal{N}(\mu_j, \sigma_j^2)\right]$. When the M Gaussian components with correct parameters $\{\alpha_j^*, \mu_j^*, \sigma_j^*\}$ best describe the underlying data density distribution, then this log-likelihood will be maximized.

The procedure to search for best parameters that maximize this log-likelihood is time-consuming! Assuming M=5, total of (3M-1) parameters, a grid search with 10 values for each parameter ends up with 1E14 evaluation!

Parametric density estimator

3. Gaussian Mixture Model (GMM)

The probability that a given datum x_i is generated by (or belong to) class *j,* is given by $p_*(x_i \mid \vec{\theta}_j)$, then the probability for class *j* to be responsible for point x_i given data point x_i is given by:

$w_{ij} \equiv p_j(j \mid x_i) = \dfrac{\alpha_j p_*(x_i \mid \vec{\theta}_j)}{\Sigma_{j=1}^{M} \alpha_j p_*(x_i \mid \vec{\theta}_j)}$. In case of Gaussian, $p_*(x_i \mid \vec{\theta}_j) = \mathcal{N}(\mu_j, \sigma_j^2)$.

For each point x_i, we must also have $\Sigma_{j=1}^{M} w_{ij} = 1$.

To obtain the best parameter which maximizes the log-likelihood $\ln L = \Sigma_{i=1}^{N} \ln\left[\Sigma_{j=1}^{M} \alpha_j p_*(x_i \mid \vec{\theta}_j)\right]$, we take a partial derivative with respect to parameter:

$$\frac{\partial \ln L}{\partial \theta_j} = \Sigma_{i=1}^{N} \frac{\alpha_j}{\Sigma_{j=1}^{M} \alpha_j p_*(x_i \mid \vec{\theta}_j)} \left[\frac{\partial p_*(x_i \mid \vec{\theta}_j)}{\partial \theta_j}\right]$$

, where $\vec{\theta}_j = \{\mu_j, \sigma_j\}$. Using w_{ij} above:

$$\frac{\partial \ln L}{\partial \theta_j} = \Sigma_{i=1}^{N} \left[\frac{\alpha_j p_*(x_i \mid \vec{\theta}_j)}{\Sigma_{j=1}^{M} \alpha_j p_*(x_i \mid \vec{\theta}_j)}\right]\left[\frac{1}{p_*(x_i \mid \vec{\theta}_j)} \frac{\partial p_*(x_i \mid \vec{\theta}_j)}{\partial \theta_j}\right] = \Sigma_{i=1}^{N} w_{ij} \left[\frac{\partial \ln p_*(x_i \mid \vec{\theta}_j)}{\partial \theta_j}\right].$$

zero

Parametric density estimator

3. Gaussian Mixture Model (GMM)

In the case of Gaussian $p_*(x_i \mid \vec{\theta}_j) = \mathcal{N}(\mu_j, \sigma_j^2)$, we then have

$$\frac{\partial \ln L}{\partial \theta_j} = -\Sigma_{i=1}^{N} w_{ij} \frac{\partial}{\partial \theta_j}\left[\ln \sigma_j + \frac{(x_i - \mu_j)^2}{2\sigma_j^2}\right] = 0.$$ The estimators

$$w_{ij} = \frac{\alpha_j p_*(x_i \mid \vec{\theta}_j)}{\Sigma_{j=1}^{M} \alpha_j p_*(x_i \mid \vec{\theta}_j)}$$

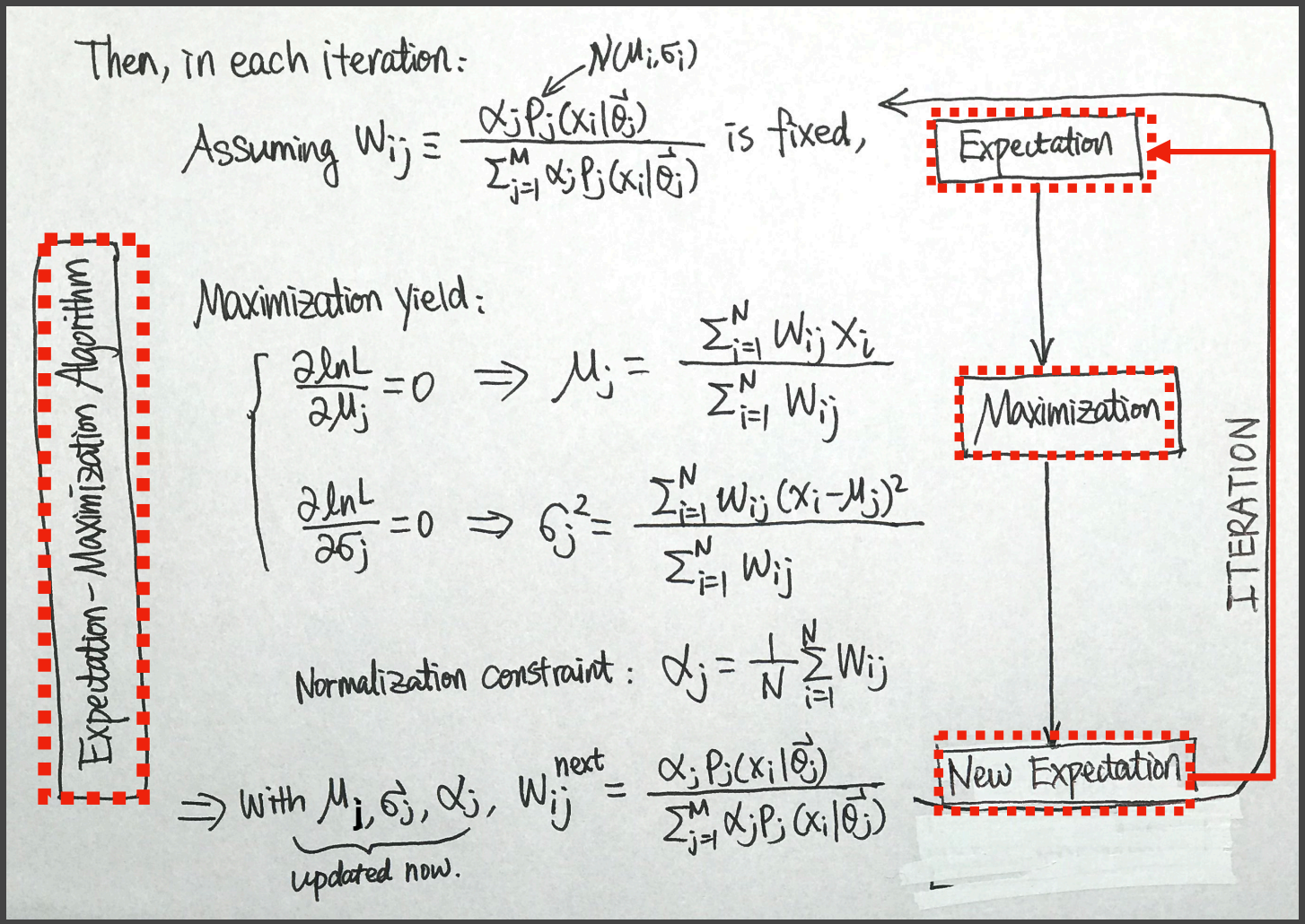
are then given by:

$$\mu_j = \frac{\Sigma_{i=1}^{N} w_{ij} x_i}{\Sigma_{i=1}^{N} w_{ij}},$$

$$\sigma_j^2 = \frac{\Sigma_{i=1}^{N} w_{ij}(x_i - \mu_j)^2}{\Sigma_{i=1}^{N} w_{ij}},$$

$$\alpha_j = \frac{1}{N}\Sigma_{i=1}^{N} w_{ij}.$$

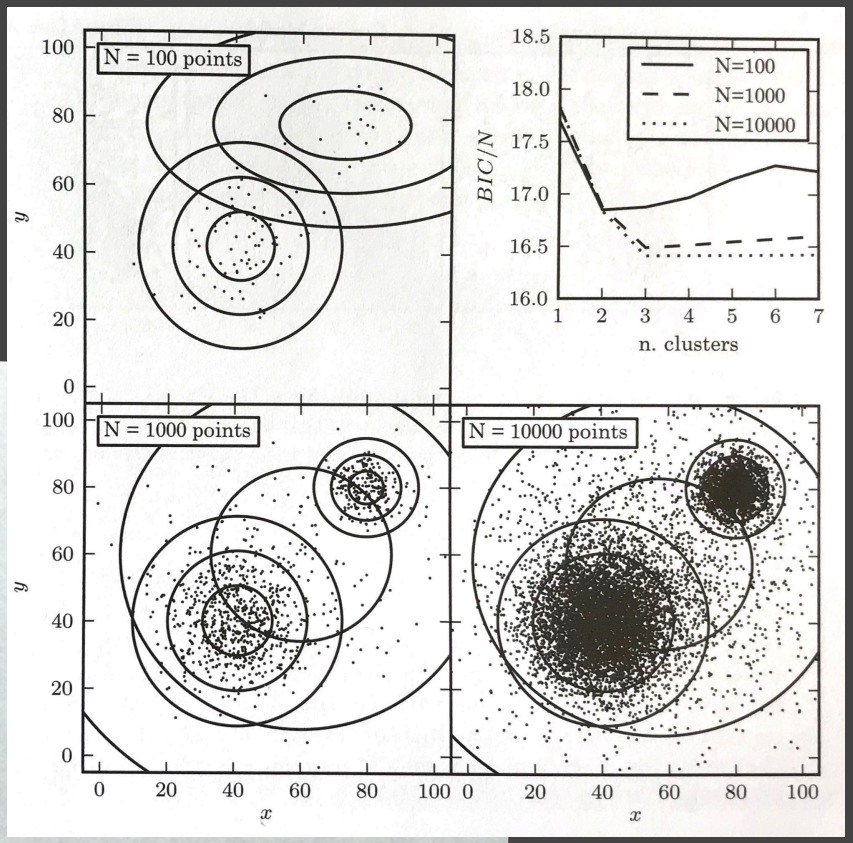
Expectation-Maximization Algorithm



Parametric density estimator

3. Gaussian Mixture Model (GMM)

The numbers of GMM components are sensitive to the number of data points!



Expectation-Maximization Algorithm

- Initialization ?
  - Setting all σ_j to sample standard deviation,
  - Setting all α_j to $1/M$,
  - Setting μ_j : randomly draw μ_j from $\{x_i\}$!

- How to decide how many GMM components ?

  BIC or cross-validation method !

NOTE: if the density estimate is the only goal, then we can use as many components as needed. In this case, GMM essentially approaches the flexibility of non-parametric estimation methods !

Parametric density estimator

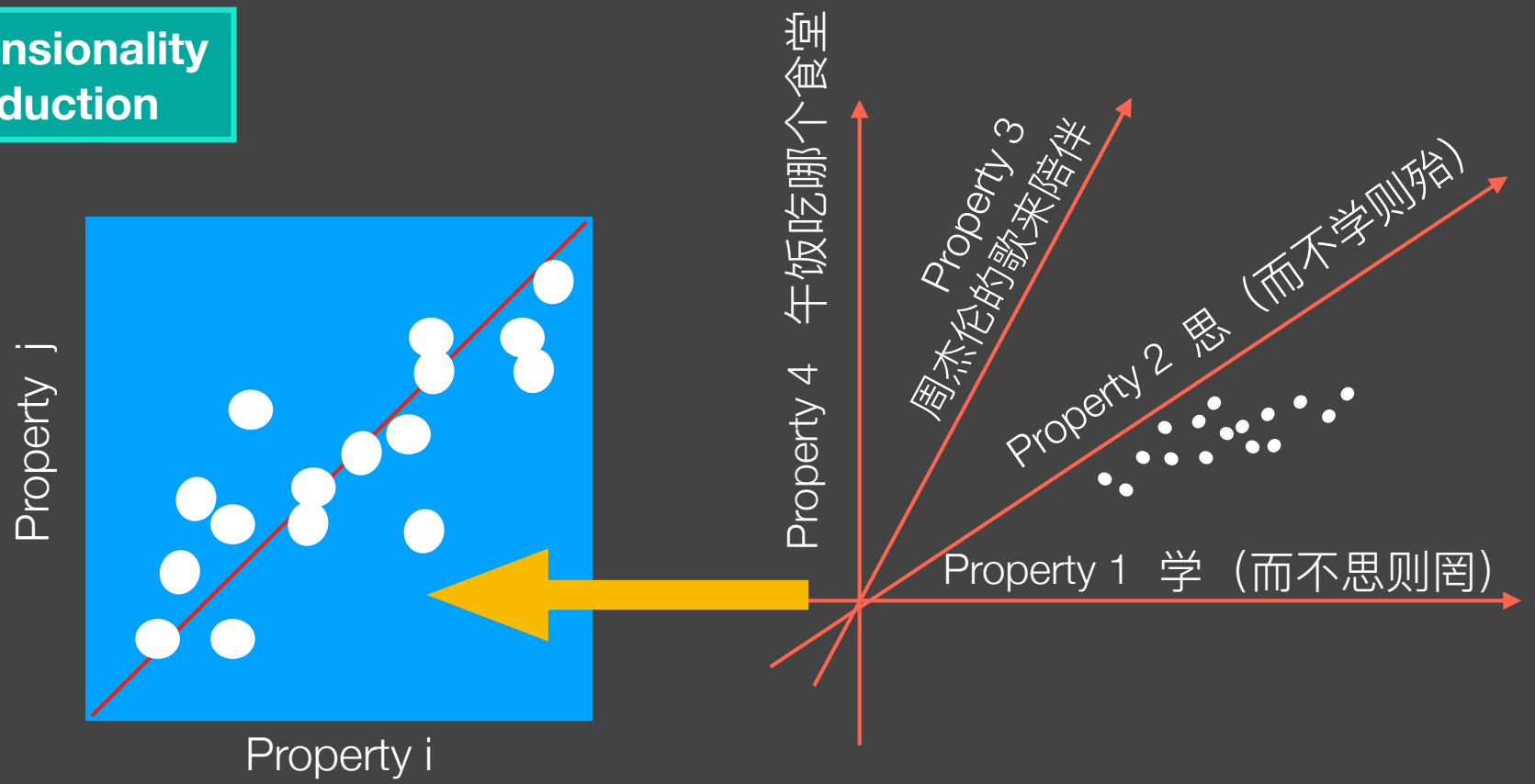
- ★ *Correlation function*
- ★ *Density estimation*
- ★ ***Clustering in data***
- ★ ***Classification***

Dimensionality Reduction

Principal Component Analysis (PCA)

Before we go into detailed algorithms of clustering and classification, we first take a look at dimensionality reduction. Because these techniques allow us to better extract key features to describe data and therefore we can achieve better performance in data clustering and classification.

Dimensionality Reduction



High dimensional data can be mind-blowing!
Correlations exist among dozens of independent variables.

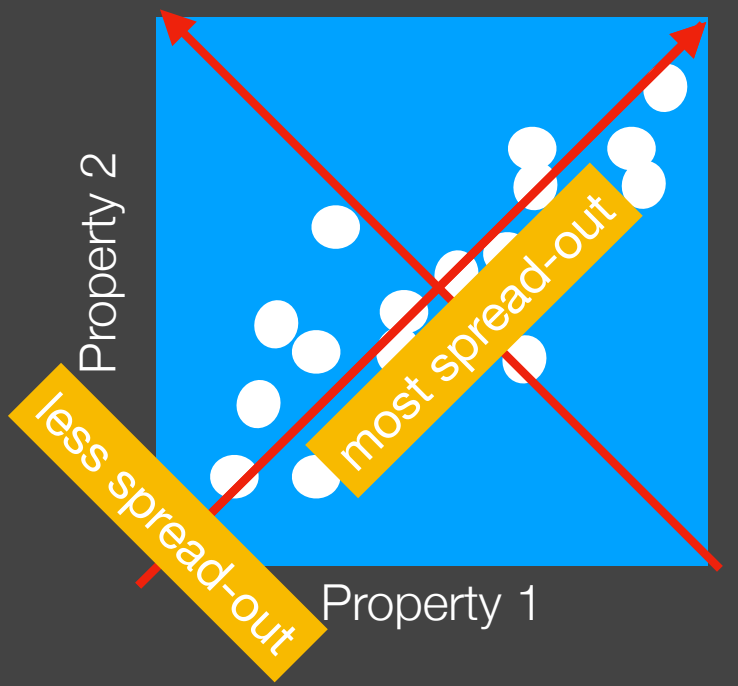
In order to exhibit key features — those most informative ones, which help us to understand the driven forces behind a problem, we can *combine* correlated properties to extract *key* features — reduce dimensionality!

Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

- most spread-out => more informative
- less spread-out => less informative

How to rotate this frame to new axis?



Covariance Matrix

$$C = \begin{bmatrix} \mathrm{Var}(x_1) & \mathrm{Cov}(x_1, x_2) & \dots & \mathrm{Cov}(x_1, x_k) \\ \mathrm{Cov}(x_2, x_1) & \mathrm{Var}(x_2) & \dots & \mathrm{Cov}(x_2, x_k) \\ \dots & \dots & \dots & \dots \\ \mathrm{Cov}(x_k, x_1) & \mathrm{Cov}(x_k, x_2) & \dots & \mathrm{Var}(x_k) \end{bmatrix}$$

N data points sitting in a *k*-dimensional feature space. Each variance of feature x_j and covariance of (x_m, x_n) in the matrix are determined by the distribution of N data points.

$$\mathrm{Var}(x_j) = \frac{1}{N-1}\Sigma_i^N (x_j^{(i)} - \overline{x_j})^2, \text{ where } j = 1,2,...k$$ Variance

$$\mathrm{Cov}(x_m, x_n) = \frac{1}{N-1}\Sigma_i^N (x_m^{(i)} - \overline{x_m})(x_n^{(i)} - \overline{x_n}), \text{ where } m, n = 1,2,...k$$ Covariance

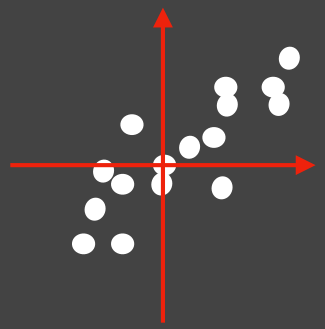
Principal Component Analysis (PCA)

What does it mean that a sample has non-zero covariance?

$\mathrm{Cov}(x_1, x_2) \neq 0$

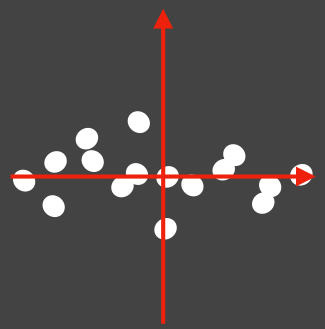
frame dependent

Correlation!



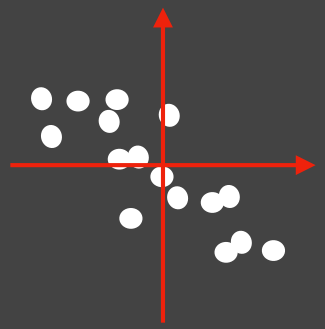
$\mathrm{Cov}(x_1, x_2) > 0$

Correlation



$\mathrm{Cov}(x_1, x_2) \approx 0$

no correlation



$\mathrm{Cov}(x_1, x_2) < 0$

anti-correlation

$$C = \begin{bmatrix} \mathrm{Var}(x_1) & \mathrm{Cov}(x_1, x_2) & \dots & \mathrm{Cov}(x_1, x_k) \\ \mathrm{Cov}(x_2, x_1) & \mathrm{Var}(x_2) & \dots & \mathrm{Cov}(x_2, x_k) \\ \dots & \dots & \dots & \dots \\ \mathrm{Cov}(x_k, x_1) & \mathrm{Cov}(x_k, x_2) & \dots & \mathrm{Var}(x_k) \end{bmatrix}$$

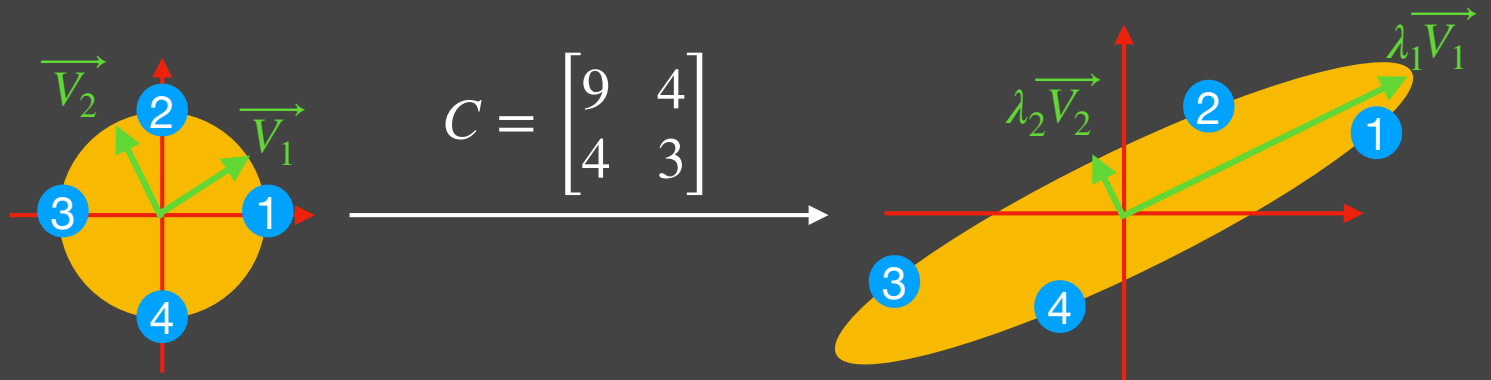
Covariance matrix: symmetric by design

What happens to a vector $\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}$*, when it is multiplied by* $C = \begin{bmatrix} \mathrm{Var}(x_1) & \mathrm{Cov}(x_1, x_2) \\ \mathrm{Cov}(x_1, x_2) & \mathrm{Var}(x_2) \end{bmatrix}$*?*

$$[C]\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} \mathrm{Var}(x_1)\,\alpha_1 + \mathrm{Cov}(x_1, x_2)\,\alpha_2 \\ \mathrm{Cov}(x_1, x_2)\,\alpha_1 + \mathrm{Var}(x_2)\,\alpha_2 \end{bmatrix}$$

Principal Component Analysis (PCA)

Two unit vectors $\overrightarrow{V_1}$ and $\overrightarrow{V_2}$ are only stretched/squashed but no rotation!



uniform circle → *elongated: stretched and rotated!*

$$C\,\overrightarrow{V_i} = \lambda_i\,\overrightarrow{V_i},\ \text{where } i = 1, 2$$

Eigenvectors of the *Covariance matrix*

Eigenvalues of *Covariance matrix*

A covariance matrix contains the "shape" information about the "distorted" sample distribution!

Principal Component Analysis (PCA)

X *is a rescaled and re-centred data matrix of* $N \times k$ *dimension*

	property 1	Property 2	…	Property k
data 1				
data 2				
…				
data N				

$$C = \frac{X^T X}{N-1} = \begin{bmatrix} \mathrm{Var}(x_1) & \mathrm{Cov}(x_1, x_2) & \dots & \mathrm{Cov}(x_1, x_k) \\ \mathrm{Cov}(x_2, x_1) & \mathrm{Var}(x_2) & \dots & \mathrm{Cov}(x_2, x_k) \\ \dots & \dots & \dots & \dots \\ \mathrm{Cov}(x_k, x_1) & \mathrm{Cov}(x_k, x_2) & \dots & \mathrm{Var}(x_k) \end{bmatrix}$$

through EVD, SVD

Maximum variation within data!

Second largest variation within data!

Least variation within data!

$\vec{V_1}$ — λ_1

$\vec{V_2}$ — λ_2

…

New PA!

$\vec{V_k}$ — λ_k

Largest

Smallest

Sort

1. for ***each*** property, rescale (dividing data variance) and re-centre (subtracting data mean);
2. Calculate the covariance matrix C, and its eigenvalues $\{\lambda\}$ and eigenvectors $\{\vec{V}\}$;
3. Rank the eigenvectors $\{\vec{V}\}$ by eigenvalues $\{\lambda\}$, sort;
4. Pick up $\vec{V}$s according to the rank to form the new principal axes!

Principal Component Analysis (PCA)

Original data matrix: $X_{[N,k]}$

Covariance matrix: $C = \frac{X^T X}{N-1} = R\, C_{\mathrm{PCA}}\, R^T$

(with possible nonzero off-diagonal components, i.e., feature correlations)

Contains rotation information

$$R_{[k,k]} = \left[\overrightarrow{V_1}\right]\left[\overrightarrow{V_2}\right][\cdots]\left[\overrightarrow{V_k}\right]$$

$R_{[k,k]}$ matrix can be obtained from Covariance matrix C through EVD, SVD

Each column of X_{PCA} is a new principal axis

New data matrix: $X_{\mathrm{PCA}} = X\,R$

PCA

New covariance matrix C_{PCA} of X_{PCA}:

$$C_{\mathrm{PCA}} = R^T C\, R = \begin{bmatrix} \lambda_1 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ \dots & & & & \\ 0 & 0 & 0 & \dots & \lambda_k \end{bmatrix}$$

Each λ is the variance of the key features in the PCA frame.

Reformation of the k-dimension parameter space. No new information is generated, but the data are better "organized" to deliver key information!

With PCA components identified, we can play clustering and classification algorithms (e.g., K-means) in PCA parameter space to achieve better performance!

Principal Component Analysis (PCA)

Application

In this game, we treat various types of galaxy spectra as features and treat fluxes at different wavelengths as data! Then each column (feature) is a galaxy spectrum. Re-centre each spectrum to the total flux such that the renormalized spectrum has a mean flux of zero. Then,

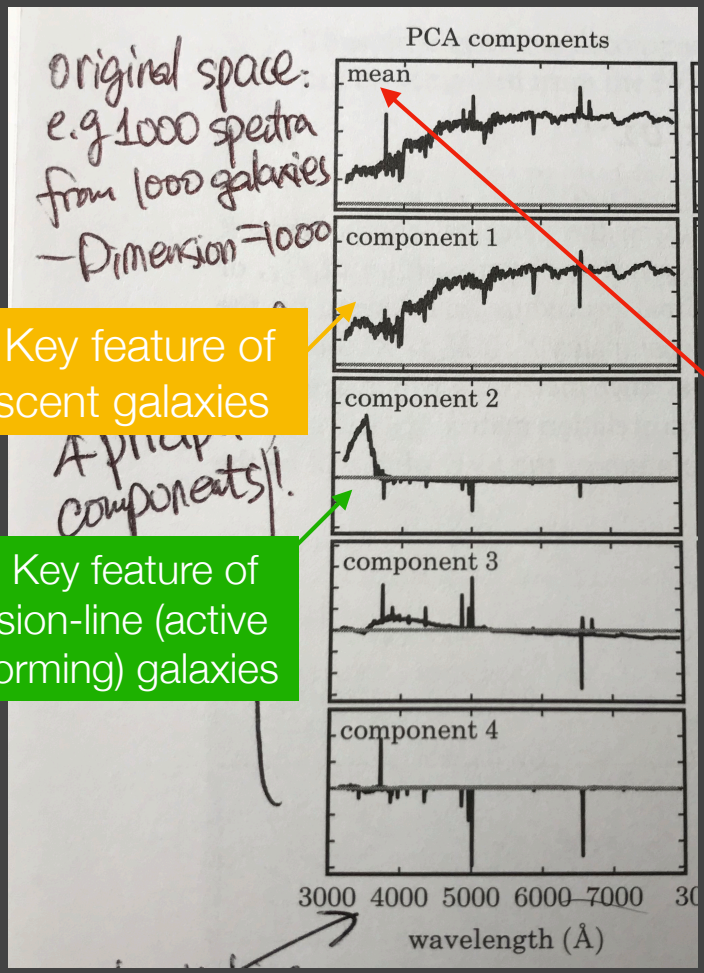
To goal is to reduce dimensions and extract key spectrum features to form a set of eigen-spectra

	Galaxy 1	Galaxy 2	...	Galaxy k
3000Å				
3001Å				
...				
9000Å				

1. In each column, the scatter (non-zero) is caused by emission or absorption features in the continuum spectrum. Some galaxies (labelled by $j = 1,2,...k$) have larger scatter/*variance* in their spectra $F_j(\lambda_i)$ across all wavelengths, and some small.

2. Same emission or absorption lines can exist among different galaxies at same wavelengths, such correlation contributes to larger *covariance* (off-diagonal terms, correlation among two galaxies) of the covariance matrix.

Principal Component Analysis (PCA)



e.g., Key feature of quiescent galaxies

e.g., Key feature of emission-line (active star-forming) galaxies

N=2000 fluxes at wavelengths λ;
k=1000 galaxy spectra

How many components should we include?

Application

Original data matrix: $X_{[N,k]}$,
each column *j* is a galaxy spectrum
$F_j(\lambda_i)$, where $i = 1, 2, ...N, j = 1, 2, ...k$.

$$X = [F_1(\lambda)][F_2(\lambda)][\cdots][F_k(\lambda)]$$

PCA

New data matrix: $X_{\rm PCA} = X\,R$,
each column *j* is a PCA eigenspectrum $e_j(\lambda_i)$.

$$X_{\rm PCA} = [e_1(\lambda)][e_2(\lambda)][\cdots][e_k(\lambda)]$$

Only first k_{PA} contribute to most variance

Now reconstruct the input spectrum $F(\lambda_i)$
of a given galaxy using first $k_{\rm PA}$ eigenspectra $e(\lambda)$:

$$F(\lambda_i) = \mu(\lambda_i) + \Sigma_{j=1}^{k_{\rm PA}}\, \omega_j\, e_j(\lambda_i),$$

Using the fact that $F(\lambda_i) - \mu(\lambda_i) = \Sigma_{j=1}^{k} \omega_j e_j(\lambda)$,
the weight ω_j can be obtained by:

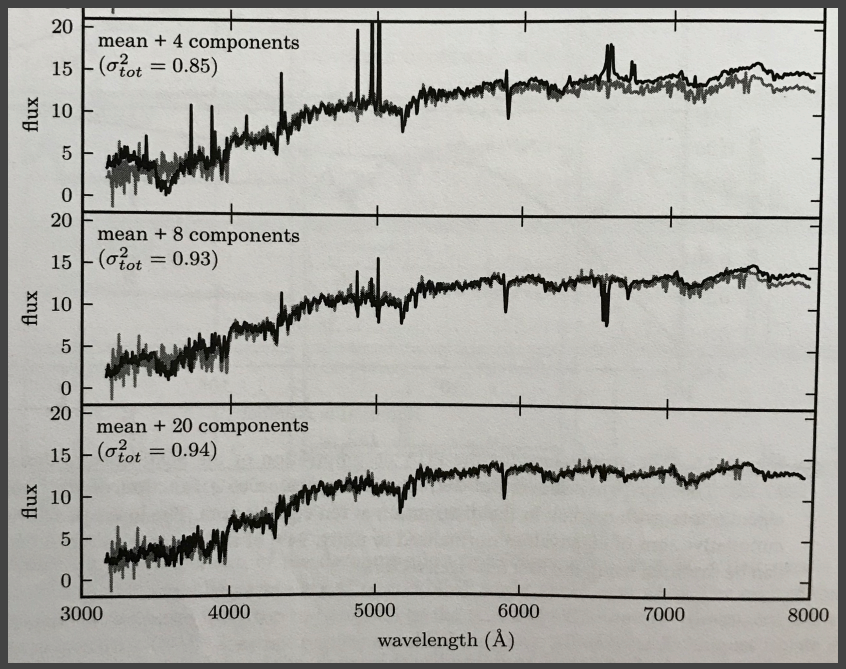
$$\omega_j = \Sigma_{i=1}^{N}\, e_j(\lambda_i)\, [F(\lambda_i) - \mu(\lambda_i)].$$

Principal Component Analysis (PCA)

N=2000 fluxes at different wavelengths per galaxy;
k=1000 galaxy spectra

PCA

k=1000 PCA eigenspectra

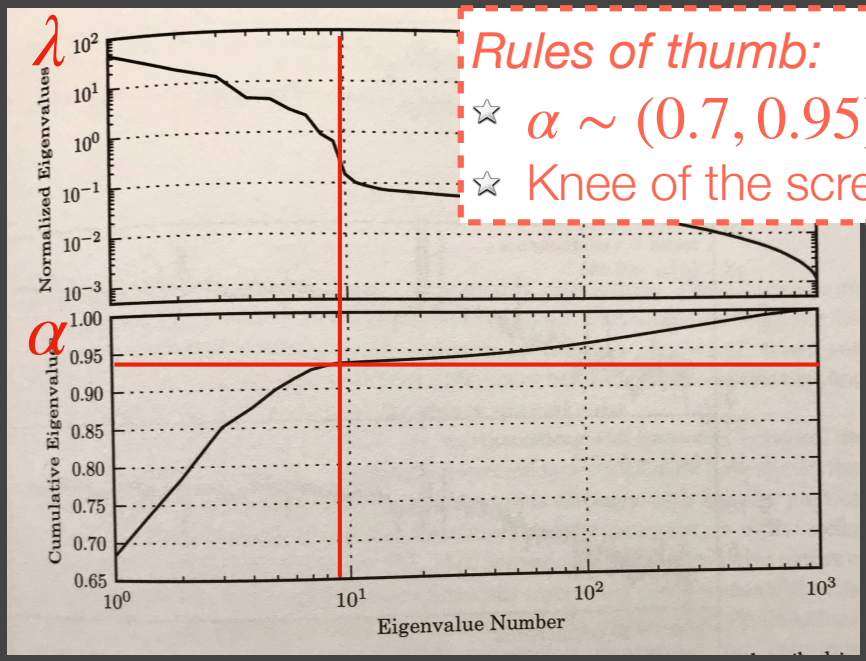


Application

How many components should we include?

Adding too many components with smaller and smaller variance essentially build up noise in reconstruction.

Define $\alpha = \frac{\Sigma_{j=1}^{k_{\rm PA}} \lambda_j}{\Sigma_{j=1}^{k} \lambda_j}$, the fraction of the total variance we wish to capture. Use ***scree plot*** to decide truncation $k_{\rm PA}$.



Rules of thumb:

- $\alpha \sim (0.7, 0.95)$
- Knee of the scree plot

After ***sorting***, only first k_{PA} eigen-components components contribute to most variance

- ★ *Correlation function*
- ★ *Density estimation*
- ★ ***Clustering in data***
- ★ *Classification*

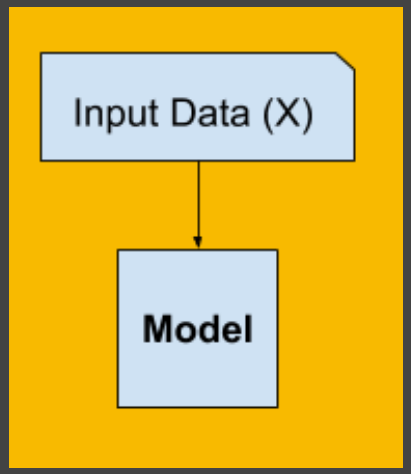
General methods

- ★ Clustering algorithms
  (1) K-means, mean-shift
  (2) Hierarchical clustering
- ★ Generative models:
  (1) *Gaussian Mixture Model*

- unsupervised learning to seek for partitioning or segmentation of data into smaller parts

Once important features are obtained, we can do data clustering!

This is a much less well-defined problem, since we are not told what kinds of patterns to look for, and there is no obvious error metric to use (unlike supervised learning, where we can compare our prediction of y_i^{mod} for a given x_i to the observed value y_i^{obs}). Decisions are made through learning/perceiving existing input. The lack of correction is generally referred to as unsupervised learning.



Unsupervised

In unsupervised learning, we are only given inputs, and the goal is to find interesting patterns/ clustering feature in the data.

General methods

★ Clustering algorithms
(1) K-means, mean-shift
(2) Hierarchical clustering

★ Generative models:
(1) *Gaussian Mixture Model*
(2) Restricted Boltzmann Machine,
(3) Variational Autoencoder,
(4) Convolution neural network,
(5) Generative Adversarial Network

Partitioning data points into *K* disjoint subsets C_k with each subset containing N_k points

Clustering by sum-of-squares Minimization: *K-Means*

The principle of clustering is to minimize: $\Sigma_{k=1}^{K} \Sigma_{x_i \in C_k} | x_i - \mu_k |^2$,

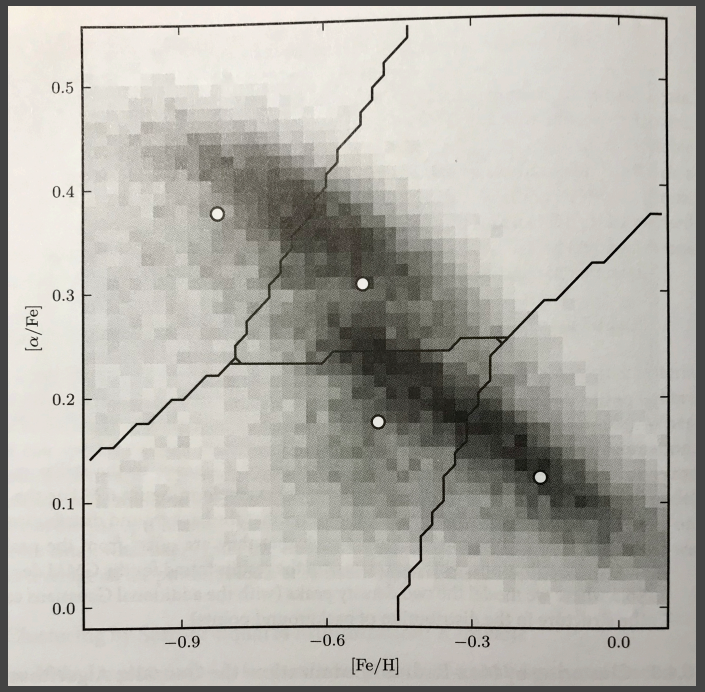
where $\mu_k = \frac{1}{N_k} \Sigma_{x_i \in C_k} x_i$ is the mean of N_k points in set C_k.

***Effective minimizing* Procedure** *for a total of K subsets:*

1. Randomly pick *K* positions as μ_k for the *K* sets at initialization.
2. Assign each point x_i to set k, whose centre μ_k is closest to this given point x_i. Do this for all points.
3. For each set k, update μ_k using the new assignments in the set in this round.
4. Repeat 2, until the set members (i.e., a point's set-identity) no longer change , i.e., μ_k stay unchanged.

Despite that global minimum is not guaranteed, Procedure never increase the sum-of-squares above.

Therefore in practices, K-means is run multiple times with different starting values for μ_k. The one with the lowest sum-of-squares error is adopted!



Clustering by non-parametric density estimation: Mean-Shift

Find local modes in a kernel density estimate of the data

"Move" the data point x_i^m in the m-th iteration in the direction of the gradient of the ***logarithmic*** *(to guarantee fast convergence)* local density field $\log \hat{f}^m(x_i^m)$, until all points eventually *converge (no longer moving)* to each other (i.e., staying at local peaks)!

$$x_i^{m+1} = x_i^m + a \nabla \log \hat{f}^m(x_i^m)$$
$$= x_i^m + \frac{a}{\hat{f}^m(x_i^m)} \nabla \hat{f}^m(x_i^m)$$

Note: $\hat{f}^m(x_i^m)$ is found by kernel density estimation (earlier in this lecture) and $\nabla \hat{f}^m(x_i^m)$ is found by kernel density estimation using gradient of the original kernel.

1. *The number of total class K will be found implicitly!*
2. *Convergence of this procedure depends on both the kernel bandwidth h and step size a!*

Clustering by non-parametric density estimation: Mean-Shift

Kernel Density Estimation

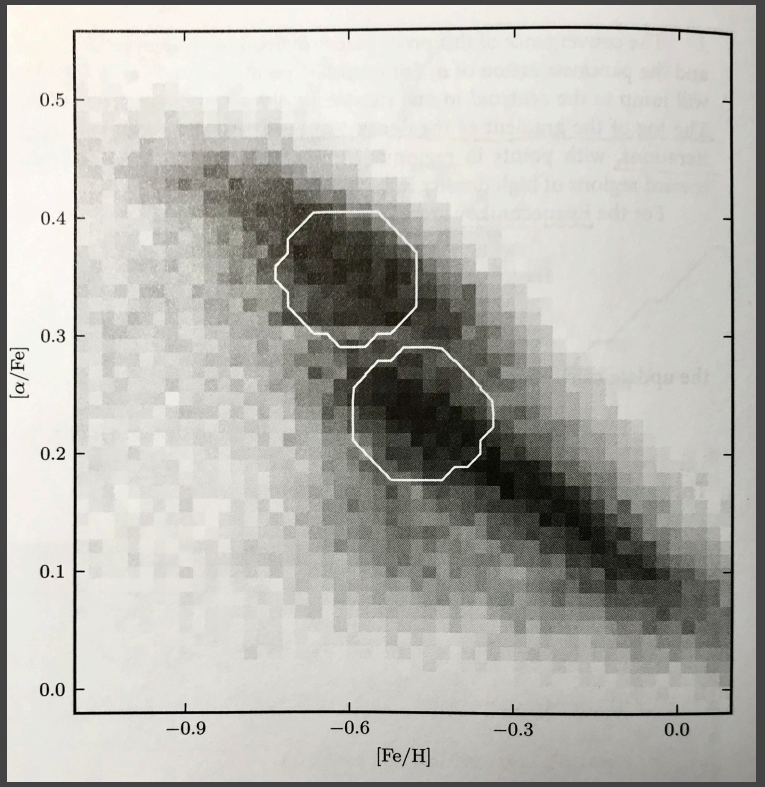
$$\hat{f}_N(\vec{x}) = \frac{1}{N h^D} \Sigma_{i=1}^{N} S\left(\frac{d(\vec{x}, \vec{x_i})}{h}\right)$$

Using Epanechnikov Kernel:

$$s(u \equiv \frac{d}{h}) = \begin{cases} \frac{3}{4}(1-u^2) & |u| \leqslant 1 \\ 0 & \text{otherwises} \end{cases}$$

For the Epanechnikov kernel, $a = \frac{h^2}{D+2}$.

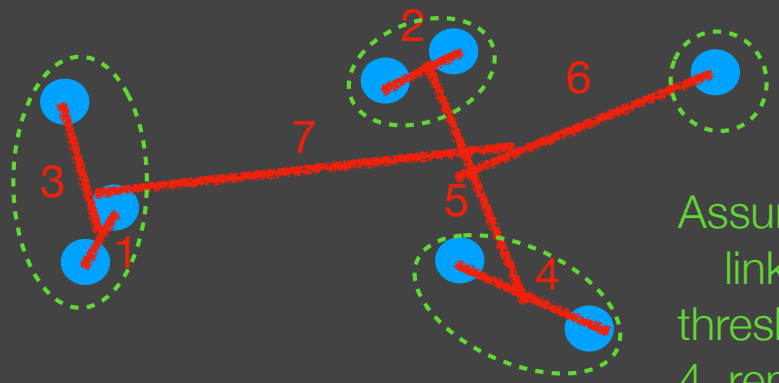
This is the *mean-shift algorithm*:
the updated position x_i^{m+1} in this case is the mean position within distance h of x_i^m.



Note: two clusters associated with local maxima are found. Points outside the circles have “decided" to stay in the background!

Hierarchical Clustering

It relaxes the need to specify the number of clusters K by finding all clusters *at all scales!*



Assuming a linkage threshold at 4, removing any longer connections.

Procedure:

1. Start by partitioning all *N* data points into *N* clusters.
2. Each time only join the "nearest" two clusters resulting in *N-1* clusters.
3. Repeat 2 until there is only one final cluster containing all *N* data points.

Define "nearest" clusters C_k and $C_{k'}$ by distance:

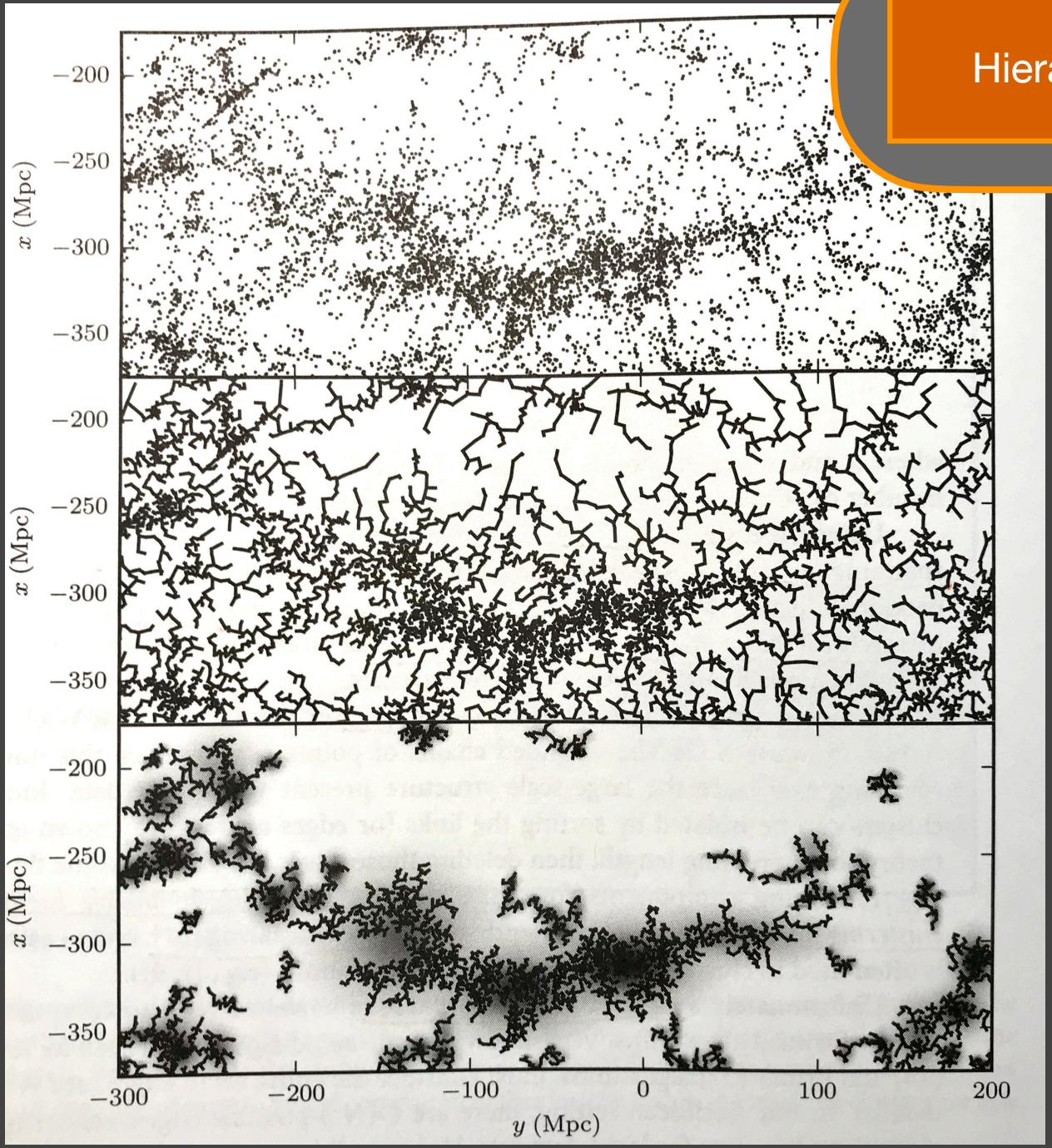
1. $d_{\min}(C_k, C_{k'}) = \min |x - x'|$, where $x \in C_k, x' \in C_{k'}$ **Loose clusters**
2. $d_{\max}(C_k, C_{k'}) = \max |x - x'|$, where $x \in C_k, x' \in C_{k'}$ **Compact clusters**
3. $d_{\text{avg}}(C_k, C_{k'}) = \frac{1}{N_k N_{k'}} \Sigma_{x \in C_k} \Sigma_{x' \in C_{k'}} |x - x'|$

Note: distances shall be calculated and stored!

4. $d_{\text{cen}}(C_k, C_{k'}) = |\mu_k - \mu_{k'}|$, where μ_k is mean of cluster C_k.

- Individual clusters can be arranged by sorting the links by increasing length, deleting links that are longer than some threshold; the remaining components form the clusters.
- Minimum spanning tree (using $d_{\min}$) with a single linkage threshold: *friends-of-friends* algorithm (computational-time of $O(N \log N)$ can be reached).

Hierarchical Clustering



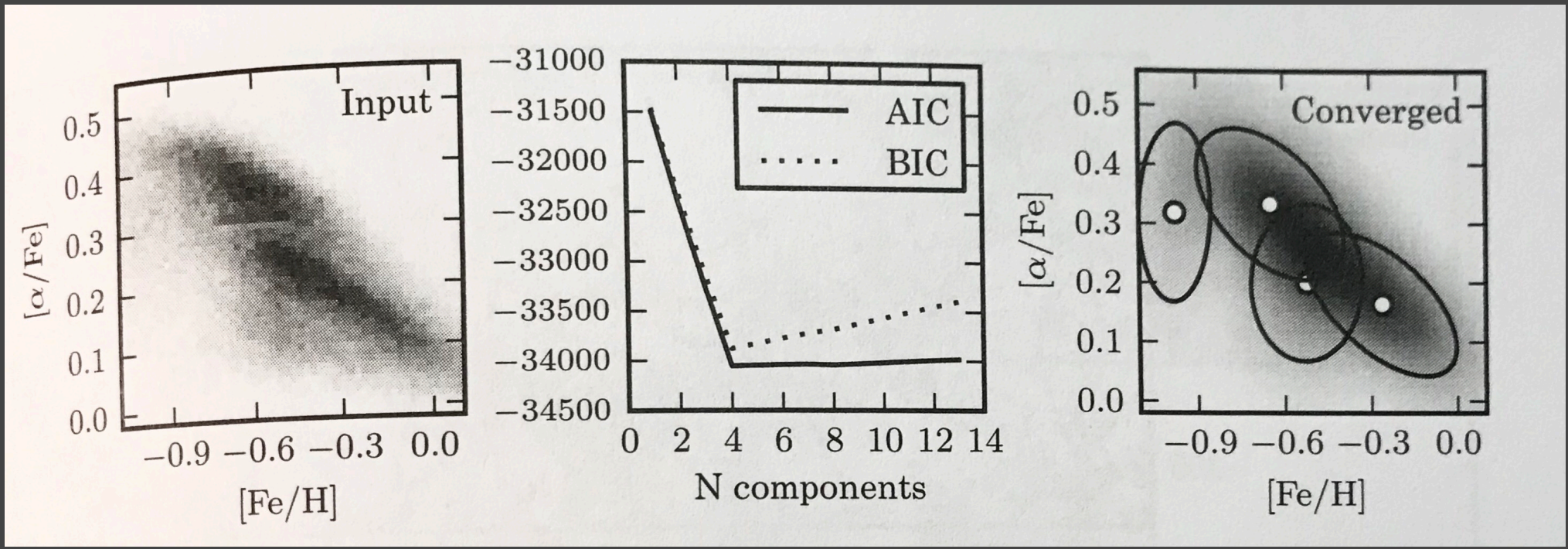
SDSS Great Wall.

Dendrogram (聚类树图) using minimum spanning tree

Removing the largest 10% of the clustering linking lengths and keeping those with more than 30 members.

Gaussian Mixture Model (GMM)

GMM is not only a parametric model for density estimation,
but also a generative model for data clustering.



Note about GMM: N-component peaks do not necessarily mean
N Gaussian components. They can represent clusters which are
not necessarily Gaussian, or can simply be strong background.

- ***Correlation function***
- ***Density estimation***
- ***Clustering in data***
- ***Classification***

- Metrics of classification
- Classification methods:

1. Generative Classification
2. K-Nearest Neighbor Classification
3. Discriminative Classification

Classification

Supervised

Is this a galaxy?
Is this a star?
Is this a cat?

Hidden lines: I know what galaxies, stars and cats are! - i.e., need a training dataset with labels

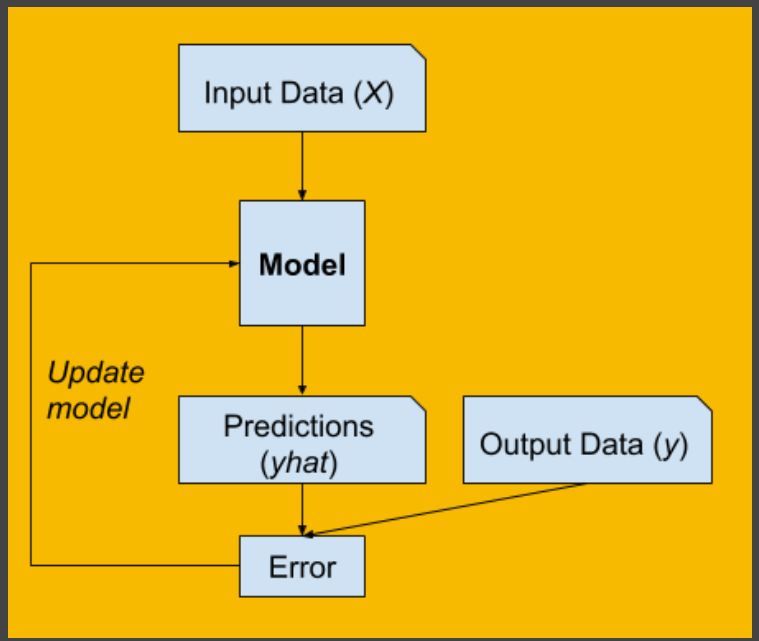
General methods

★ Generative models:
(1) Naive Bayes, Bayesian classifier
(2) Restricted Boltzmann Machine,
(3) Variational Autoencoder,
(4) Convolution neural network

★ Discriminative models:
(1) Logistic regression
(2) Support vector machine.

★ Nearest-Neighbour classier
★ Decision tree and random forest

Supervised learning requires a training dataset that is used to train a model, comprised of multiple examples, called samples, each with input variables (X) and output class labels (y). A model is trained by showing examples of inputs, having it predict outputs, and correcting the model to make the outputs more like the expected outputs.



Supervised

Is this a galaxy?
Is this a star?
Is this a cat?

Hidden lines: I know what galaxies, stars and cats are! - i.e., need a training dataset with labels

To evaluate classification:

Simple classification categories

True positives	False positives
True negatives	False negatives

$$\text{True} = \text{true positives} + \text{false negatives}$$

$$\text{Completeness (or sensitivity or recall)} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\text{Efficiency (or precision)} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$\text{Contamination (or Type-I error)} = 1 - \text{efficiency} = \frac{\text{false positives}}{\text{true positives} + \text{false positives}}$$

Generative Classification

The classifier uses specific models to describe likelihood/ probability of data being in a given class! e.g., Naive Bayes, Bayesian classifiers

K-Nearest Neighbor Classification

No models used. The classifier does no more than remembers the data!

Discriminative Classification

Pure aim at finding a boundary which best separates the data sets that belong to known classes. e.g., Logistic Regression, Support Vector Machines.

Supervised classification

Generative Classification

Training dataset: there are *K* ***pre-known*** classes for a given data set made of *N* data points $\{\vec{x}_i\}$, *where i=1,2,…N.* Each point $\vec{x}_i$ lives in a *D*-dimensional space; x_i^j denotes the *j*-th feature of the *i*-th data point, *where j=1,2,…D.* The point $\vec{x}_i$ has a known class $y_k(\vec{x}_i)$.

Working dataset (CV/test): classification algorithm will be implemented upon to predict a class that the data belong to.

For working datasets, the probability for being class *k given any* data point $\vec{x_i}$ is given by:

$$p(y_k \mid \vec{x_i}) = \frac{p(\vec{x_i} \mid y_k)\, p(y_k)}{\Sigma_m^K\, p(\vec{x_i} \mid y_m)\, p(y_m)}$$

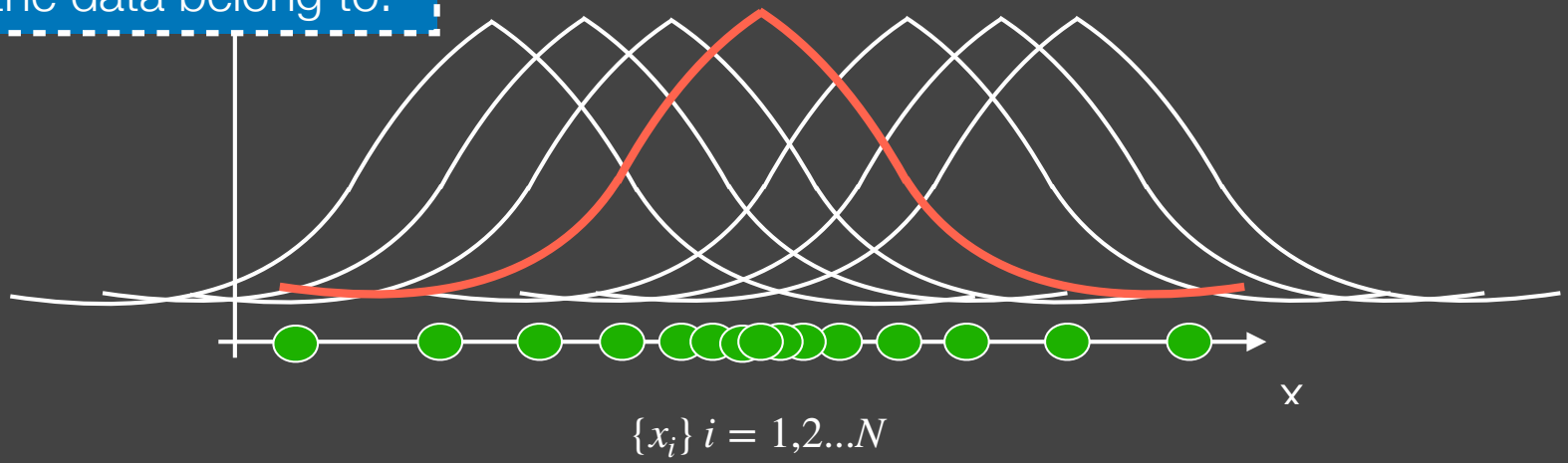
For each class, the above probabilities (e.g, $p_k(\vec{x_i}) \equiv p(\vec{x_i} \mid y_k)$ and $\pi_k \equiv p(y_k)$) can be either specified through *Bayesian inference/model regression* or readily achieved via *density estimates* from the training dataset, *where labels* $y_k(\vec{x_i})$ *are known!*.

Once the functional form of the model that calculates $p(y_k \mid \vec{x_i})$ is obtained using the training dataset, the best predicted class for a datum $\vec{x_i}$ in the working dataset will be the one with y_k which maximizes the probability $p(y_k \mid \vec{x_i})$.

Generative Classification

Training dataset: there are *K* ***pre-known*** classes for a given data set made of *N* data points $\{\vec{x}_i\}$, *where i=1,2,…N.* Each point $\vec{x}_i$ lives in a *D*-dimensional space; x_i^j denotes the *j*-th feature of the *i*-th data point, *where j=1,2,…D.* The point $\vec{x}_i$ has a known class $y_k(\vec{x}_i)$.

Working dataset (CV/test): classification algorithm will be implemented upon to predict a class that the data belong to.



Once the functional form of the model that calculates $p(y_k \mid \vec{x_i})$ is obtained using the training dataset, the best predicted class for a datum $\vec{x_i}$ in the working dataset will be the one with y_k which maximizes the probability $p(y_k \mid \vec{x_i})$.

Naive Bayes

Generative Classification

The concept of obtaining $p_k(\vec{x_i}) \equiv p(\vec{x_i} \mid y_k)$ and $\pi_k \equiv p(y_k)$ from the training set is simple, but it can be difficult to compute in many dimensions and with complicated probability distributions!

Making assumptions!

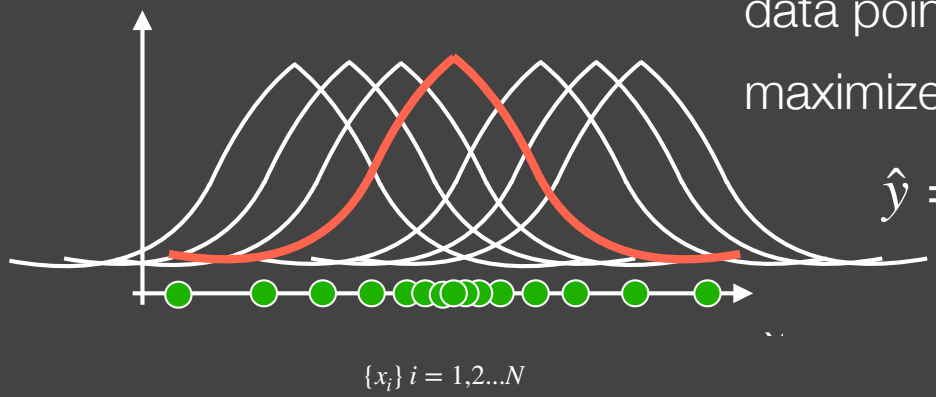
Naive Bayes: $p(x_i^1, x_i^2, \ldots x_i^D \mid y_k) = \Pi_j^D \, p(x_i^j \mid y_k)$ (for every $\vec{x_i}$)

Upon the training set

Conditionally independent — no correlations between features

$$p(y_k \mid \vec{x_i}) = x_i^1, x_i^2, \ldots x_i^D) = \frac{\Pi_j^D \, p(x_i^j \mid y_k)\, p(y_k)}{\Sigma_m^K \Pi_j^D \, p(x_i^j \mid y_m)\, p(y_m)}$$

For making prediction for a working set



The most likely estimate $\hat{y}$ for the class label of a given data point $\vec{x_i}$ ($= x_i^1, x_i^2, \ldots x_i^D$) is the one that maximizes $p(y_k \mid \vec{x_i})$, i.e.,

$$\hat{y} = \arg\max_{y_k} = \frac{\Pi_j^D \, p(x_i^j \mid y_k)\, p(y_k)}{\Sigma_m^K \Pi_j^D \, p(x_i^j \mid y_m)\, p(y_m)}$$

Generative Classification

Gaussian Naive Bayes

Gaussian distribution

Gaussian Naive Bayes: $p(x^1, x^2, \ldots x^D \mid y_k) = \Pi_j^D \, p_G(x^j \mid \mu_k^j, \sigma_k^j)$

Upon the training set Each $p_k(x^j) \equiv p(x^j \mid y_k)$ is modeled with a one-dimensional normal distribution with mean μ_k^j and width σ_k^j determined for the training set, e.g., using maximum likelihood estimate.

For making prediction for a working set Maximizing over y_k => the most likely estimate $\hat{y}$ for the class label of a given data point $\vec{x}$ $(= x^1, x^2, \ldots x^D)$ is:

$$\hat{y} = \arg\max_{y_k} = \left\{ \ln p(y_k) - \frac{1}{2} \Sigma_{j=1}^{D} \left(2\pi(\sigma_k^j)^2 + \frac{(x^j - \mu_k^j)^2}{(\sigma_k^j)^2} \right) \right\}$$

*What's behind this classifier is the assumption that the clustering features follows *uncorrelated* multivariate Gaussian distribution.*

Generative Classification

Gaussian Bayes Classifier

Further relaxing the independence assumption and allowing for correlations among different features yield the Gaussian Bayes Classifier:

$$p(\vec{x} = x^1, x^2, \ldots x^D \,|\, y_k) = \frac{1}{|\Sigma_k|^{1/2}(2\pi)^{D/2}} \exp\left\{ -\frac{1}{2}(\vec{x} - \vec{\mu_k})^T \Sigma_k^{-1} (\vec{x} - \vec{\mu_k}) \right\}$$

Multivariate Gaussian

Where Σ_k is the $D \times D$ covariance matrix with $|\Sigma_k| = \det(\Sigma_k)$, taking correlations among features into account, instead of "Naive Bayes".

Upon the training set

Model parameters $\vec{\mu_k}$ and widths $\vec{\sigma_k}$ can be determined efficiently using expectation maximization algorithm from the training set.

For making prediction for a working set

Maximizing over y_k => the most likely estimate $\hat{y}$ for the class label of a given data point $\vec{x}$ is:

$$\hat{y} = \arg\max_{y_k} = \left\{ \ln p(y_k) - \frac{1}{2} \ln |\Sigma_k| - \frac{1}{2}(\vec{x} - \vec{\mu_k})^T \Sigma_k^{-1} (\vec{x} - \vec{\mu_k}) \right\}$$

Generative Classification

Gaussian Mixture Model (GMM) Bayes Classifier

Further relaxing the multivariate Gaussian assumption, one can use GMM to describe the probability distribution of features, yielding the GMM Bayes Classifier:

Gaussian Mixture Model in D-dim feature space

GMM Bayes classifier: $p(\vec{x}\,|\,y_k) = p_{\mathrm{GMM}}(\vec{x}\,|\,y_k)$

Upon the training set

GMM parameters can be determined efficiently using *expectation maximization algorithm* upon the training set.

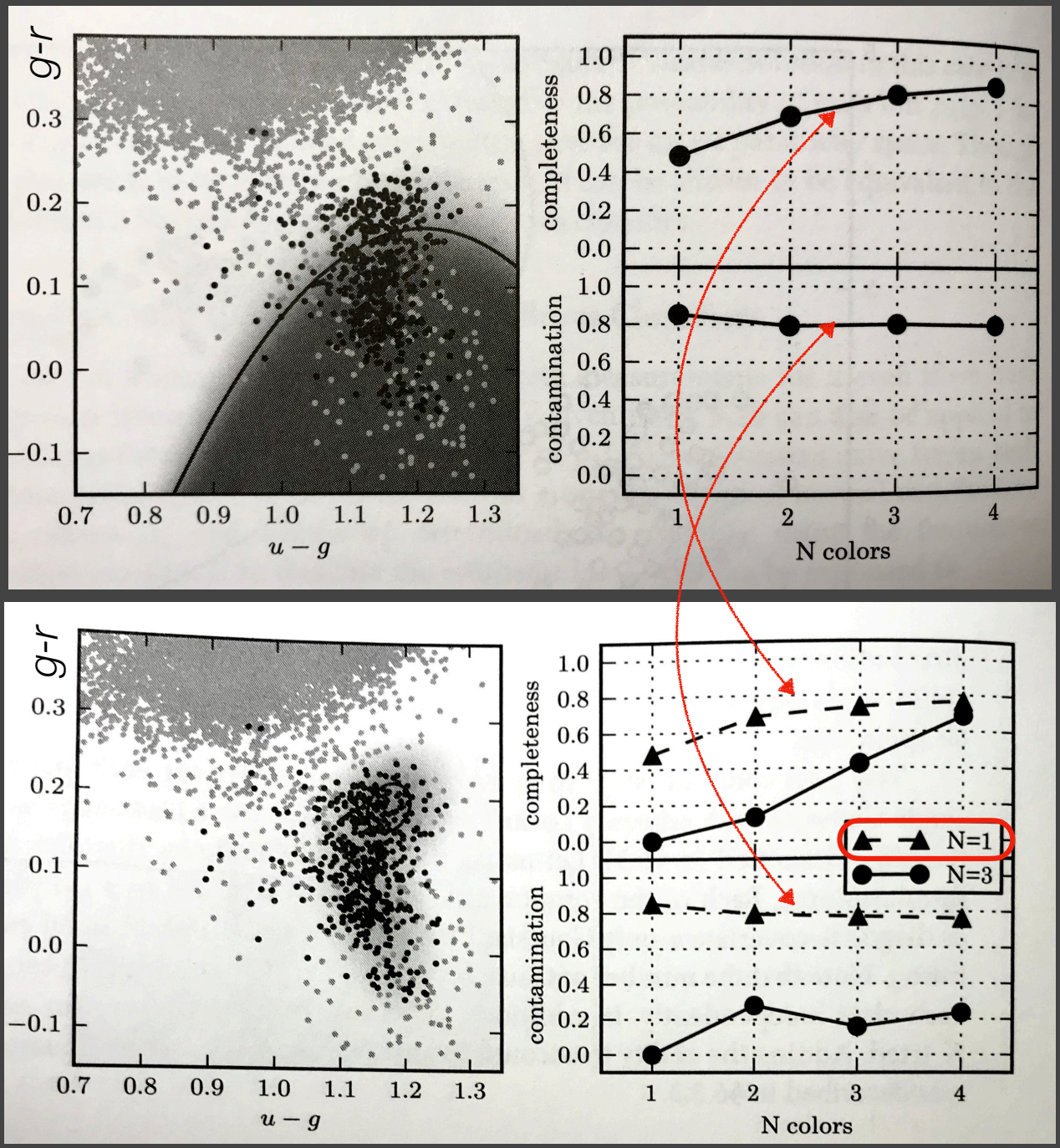
- ★ Become ultimately powerful/flexible, not necessarily physical!
- ★ Generalizing Gaussian to any desired kernel function yields Non-parametric Bayes classifier (Kernel Discriminant Analysis)

Generative Classification

Gaussian Naive Bayes Classifier

Gaussian mixture Bayes Classifier

(N indicate the # of data features - color used)



Separating variable RR Lyrae from main sequence stars

K-Nearest Neighbor Classification

No models used. The classifier does no more than remembers the data!

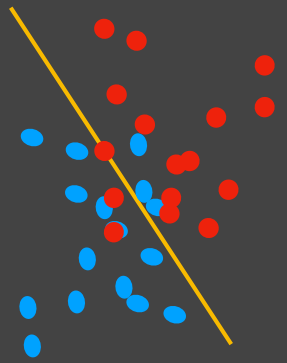
The class of a data point $\overrightarrow{x_i}$ is decided by the classes of the *k* nearest neighbors following *the majority rule*.

Justification: $p(y \mid \overrightarrow{x}) \approx p(y \mid \overrightarrow{x'})$ in close neighborhood.

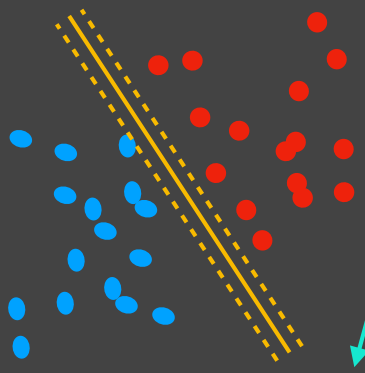
Increasing *k* decreases the variance in the classification but at the expense of an increase in the bias. Choosing *k* such that it minimizes the classification error using *cross-validation. Rule of thumb*: $k \sim \sqrt{N}$

Discriminative Classification

Purely aim at finding a boundary which best separates the data sets that belong to known classes: e.g., *Logistic Regression*, *Support Vector Machines*.



Where do I draw this line?



Where do I draw this line bundle?

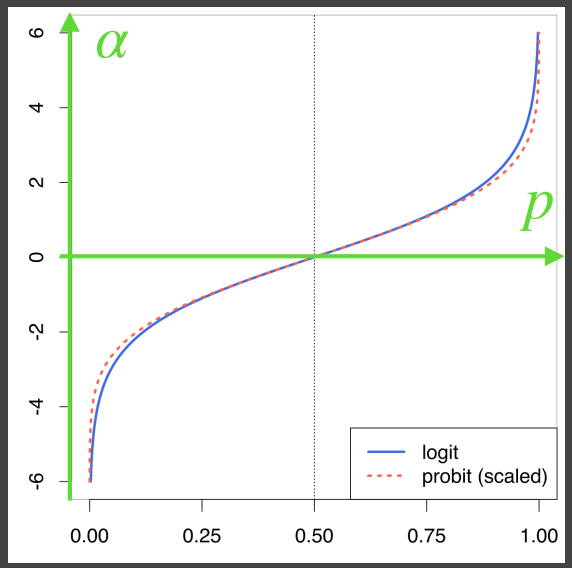
How do they work?

```
from sklearn.linear_model import LogisticRegression
logr=LogisticRegression( )
logr.fit (X, Y)
y_pred = logr.predict (X)
```

```
from sklearn.svm import LinearSVC
model=LinearSVC( )
model.fit (X, Y)
y_pred = model.predict (X)
```

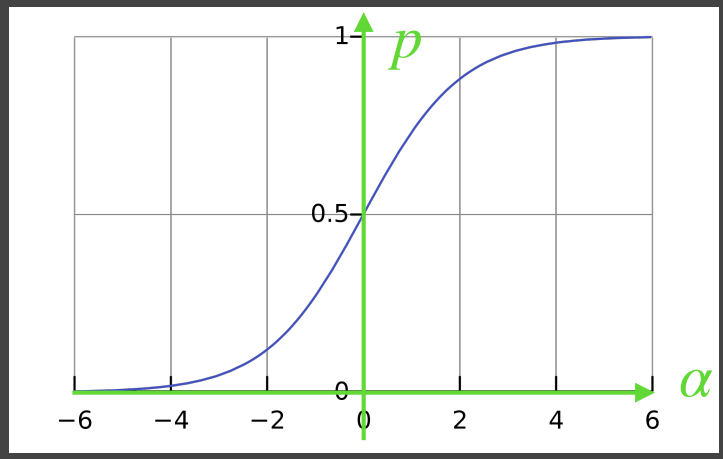
I mean how do they really work?

Logistic Regression



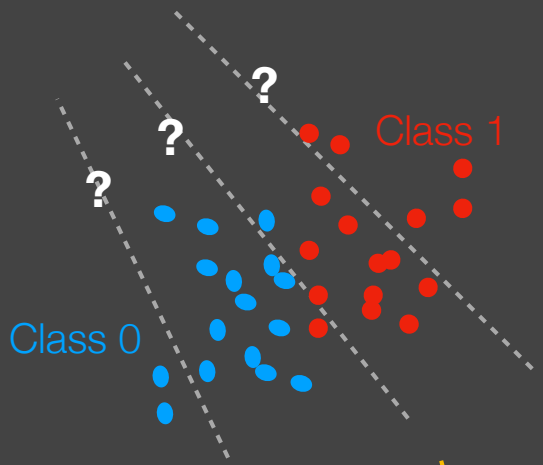
$$\text{logit}(p) = \log\frac{p}{1-p} = \alpha,$$

$$p \in [0,1),\ \alpha \in (-\infty,\ +\infty)$$

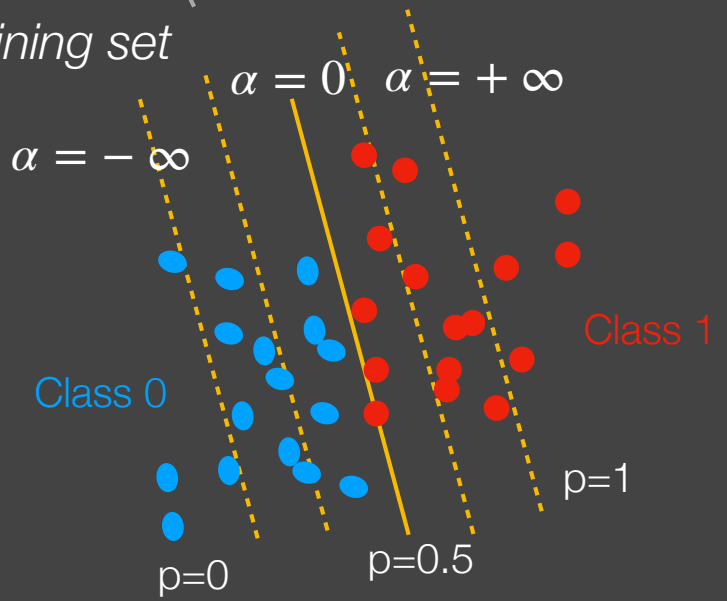


$$\text{logistic}(\alpha) = \frac{\exp(\alpha)}{1+\exp(\alpha)} = p,$$

$$\alpha \in (-\infty,\ +\infty),\ p \in [0,1)$$



training set

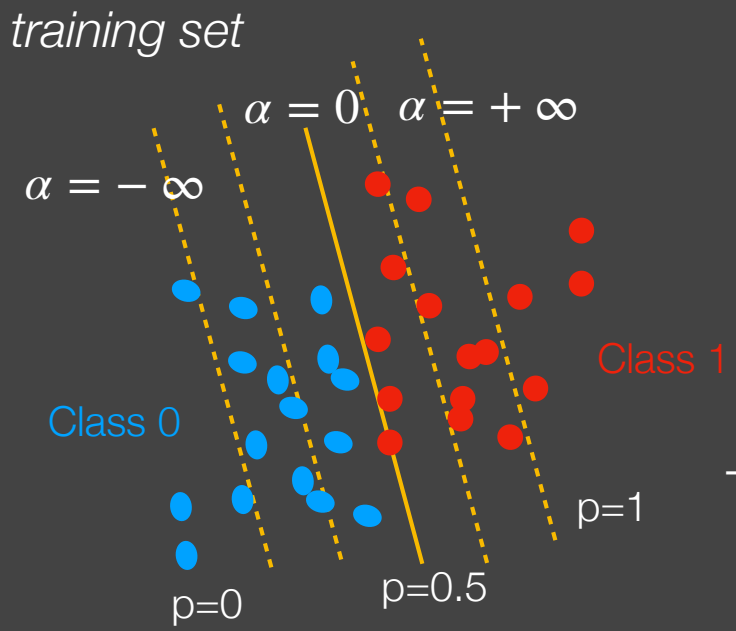


Equation $\alpha = \theta_0 + \Sigma_{j=1}^{D}\theta_j x^j$ with a given set of $\theta_{j=0,1,2...D}$ describe a set of parallel hyper-planes separating the feature space. Note α decides where this hyper-plane locates; $\alpha = 0$ (under an appropriate set of $\theta_{j=0,1,2...D}$) corresponds to the plane that separates the two classes.

α translating to $p = \text{logistic}(\alpha)$, which is used to describe a continuously varying probability for a point to belong to Class 1.

Logistic Regression

How do we train the model to "orient" the separation hyper-planes, i.e., to find the best set of $\{\theta_j\}$ which best separate the data (of training dataset)?



$p = \text{logistic}(\alpha)$ is used to describe a continues probability for a point to belong to Class 1.

Remember Bernoulli distribution:

$$P \equiv \Pi_i^N \left\{ p_i^{y_k(\vec{x_i})} (1 - p_i)^{1 - y_k(\vec{x_i})} \right\},$$

where $y_k(\vec{x_i}) = 0, 1$ is the true class label for training datum $\vec{x_i}$. p_i is the probability of $\vec{x_i}$ to be classified/predicted to belong to Class 1 according to the separation hyper-plane.

The best classification hyper-plane (given by a set of $\theta_{j=0,1,2...D}$) will maximize this joint log-likelihood for the training dataset:

$$\log P = \Sigma_i^N \left\{ y_k(\vec{x_i}) \log(p_i) + [1 - y_k(\vec{x_i})] \log(1 - p_i) \right\}$$

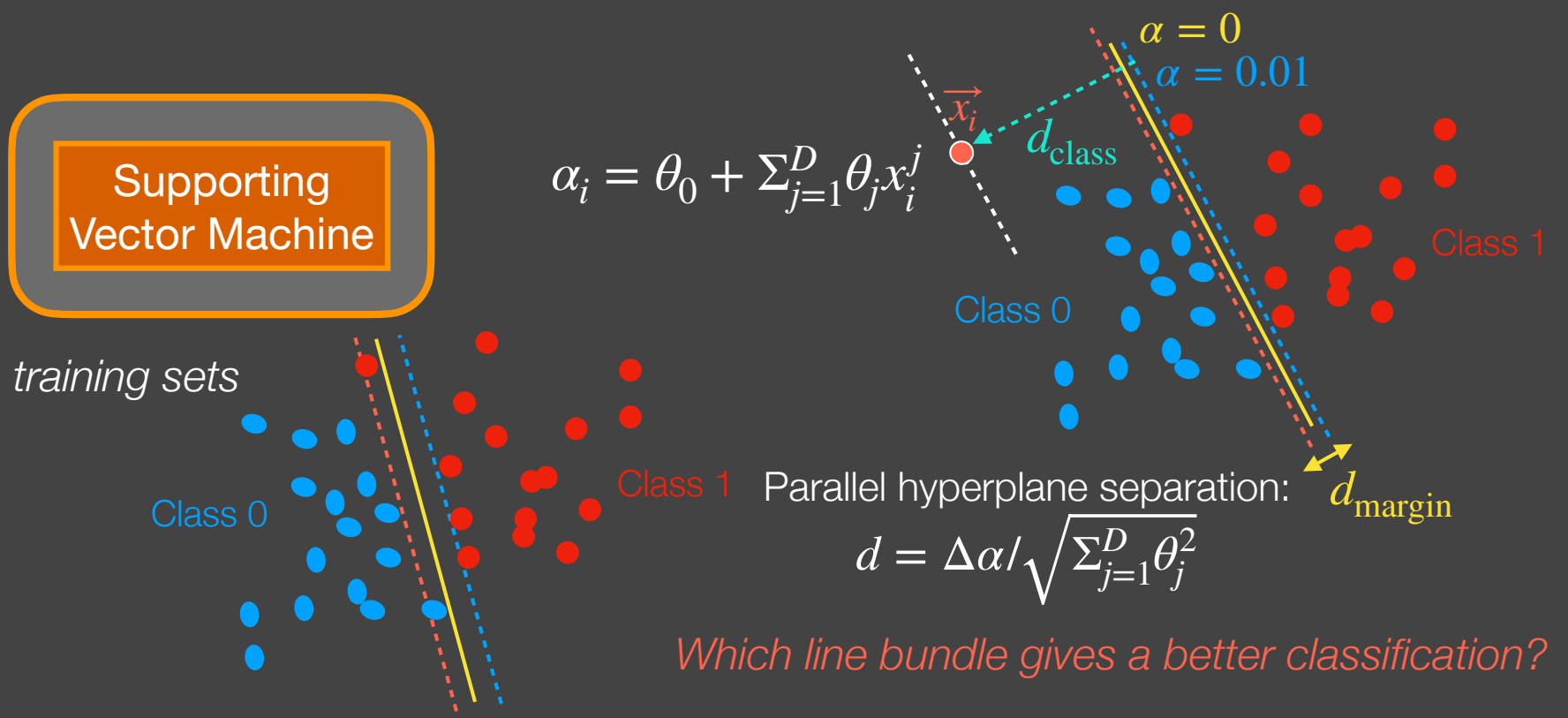
Or minimize this entropy:

$$S = -\Sigma_i^N \left\{ y_k(x_i) \log(p_i) + [1 - y_k(x_i)] \log(1 - p_i) \right\},$$

https://www.youtube.com/watch?v=jbluHIgBmBo

Supporting Vector Machine

$\alpha_i = \theta_0 + \Sigma_{j=1}^{D} \theta_j x_i^j$



Parallel hyperplane separation:

$$d = \Delta\alpha / \sqrt{\Sigma_{j=1}^{D} \theta_j^2}$$

Which line bundle gives a better classification?

Recall L_2 norm in model regression:

$$\ln[p(\vec{\theta} \mid \{x_i, y_i\}, I)] \sim -\Sigma_{i=1}^{N} \frac{[y_i - f(x_i \mid \vec{\theta})]^2}{2\sigma_i^2}$$

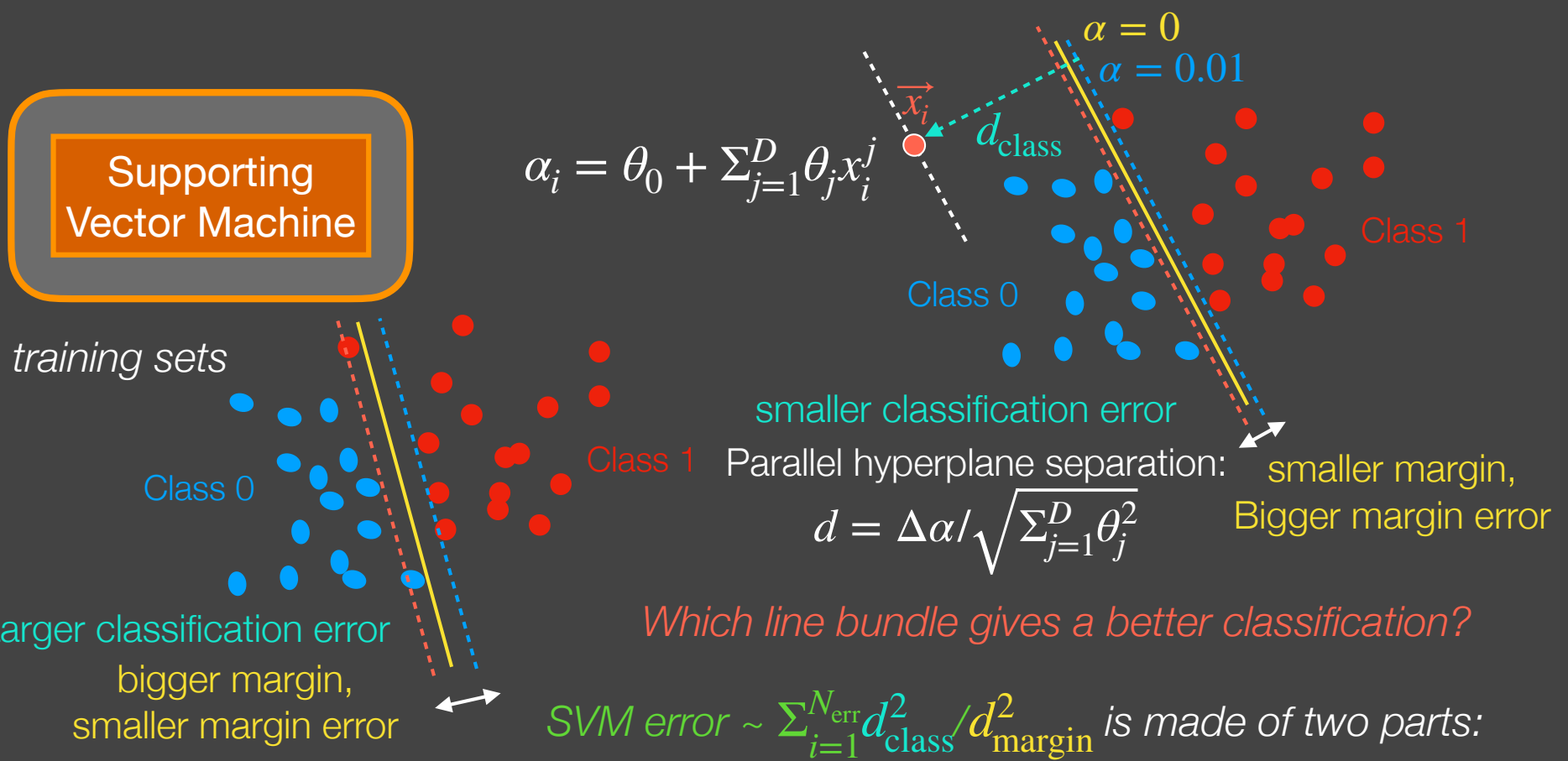
Maximize likelihood estimate

Minimize Least square estimate

Likewise, we can construct SVM error function in a similar fashion for all mis-identified points (in training dataset) under given parameter set $\{\theta_{j=0,1,2...D}\}$, which is equivalent to a *renormalized* summed distance error square $\sim \Sigma_{i=1}^{N_{\text{err}}} d_{\text{class}}^2 / d_{\text{margin}}^2$, where N_{err} is the total number of mis-identified points.

Supporting Vector Machine

$$\alpha_i = \theta_0 + \Sigma_{j=1}^{D} \theta_j x_i^j$$



Which line bundle gives a better classification?

SVM error ~ $\Sigma_{i=1}^{N_{\text{err}}} d_{\text{class}}^2 / d_{\text{margin}}^2$ *is made of two parts:*

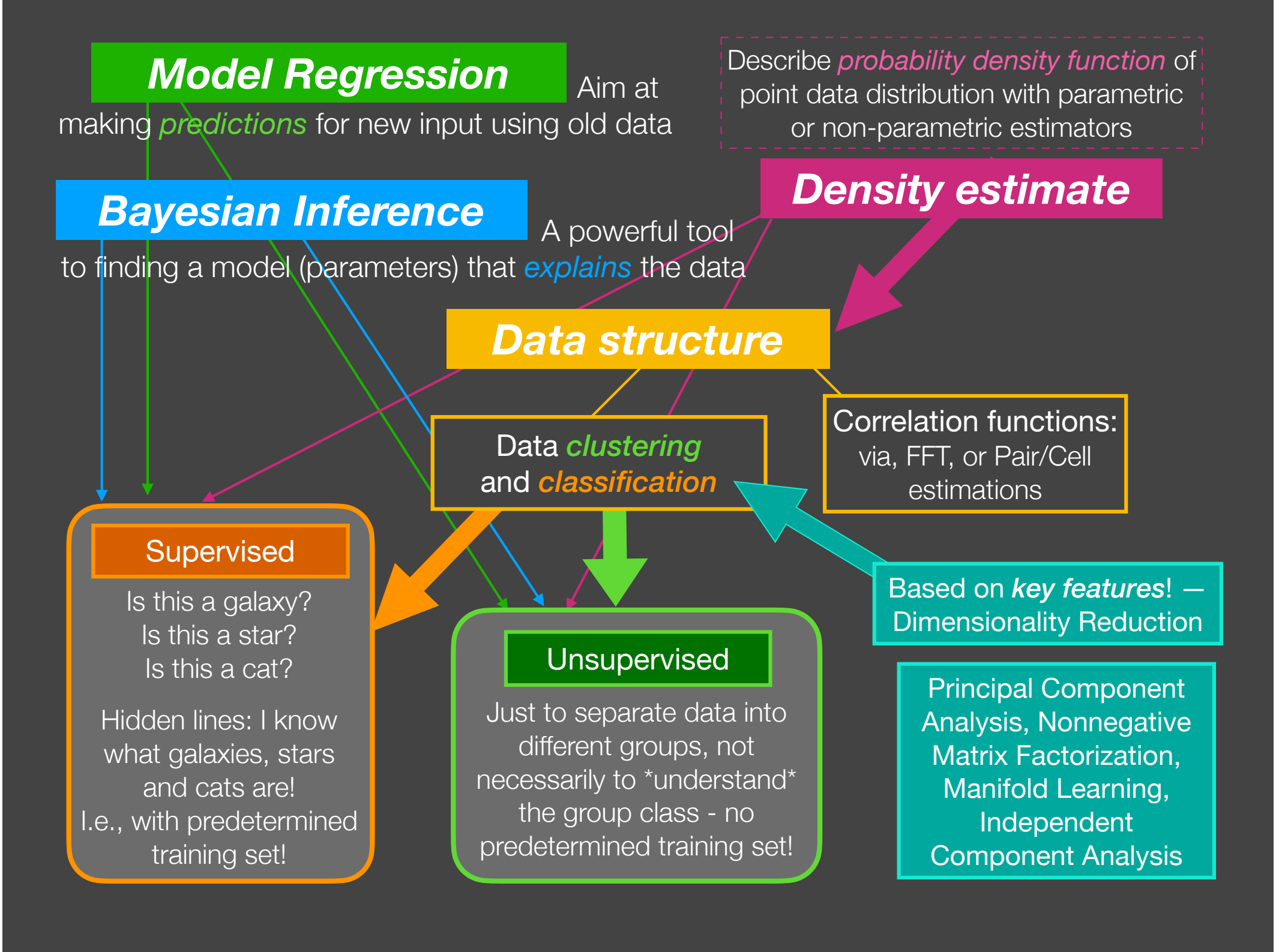
SVM error = ***C*** *x classification_error* + *margin_error*

The best classification hyper-plane (given by a set of $\theta_{j=0,1,2...D}$) will minimize the SVM error.

The margin error is equivalent to a regularization term! Adjusted through *C*.

C small: margin error dominates -> seek for a safe margin!
C big: classification error dominates -> seek for "tight" classification!

https://www.youtube.com/watch?v=Lpr__X8zuE8


Model Regression
Aim at making predictions for new input using old data
Describe probability density function of point data distribution with parametric or non-parametric estimators
Density estimate
Bayesian Inference
A powerful tool to finding a model (parameters) that explains the data
Data structure
Data clustering and classification
Correlation functions: via, FFT, or Pair/Cell estimations
Supervised
Is this a galaxy?
Is this a star?
Is this a cat?
Hidden lines: I know what galaxies, stars and cats are!
I.e., with predetermined training set!
Unsupervised
Just to separate data into different groups, not necessarily to *understand* the group class - no predetermined training set!
Based on key features! — Dimensionality Reduction
Principal Component Analysis, Nonnegative Matrix Factorization, Manifold Learning, Independent Component Analysis