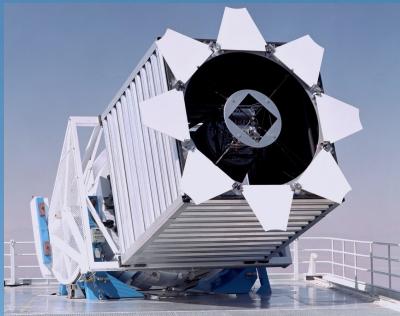


Density estimation and clustering.

Darko Jevremović

LSST



SDSS



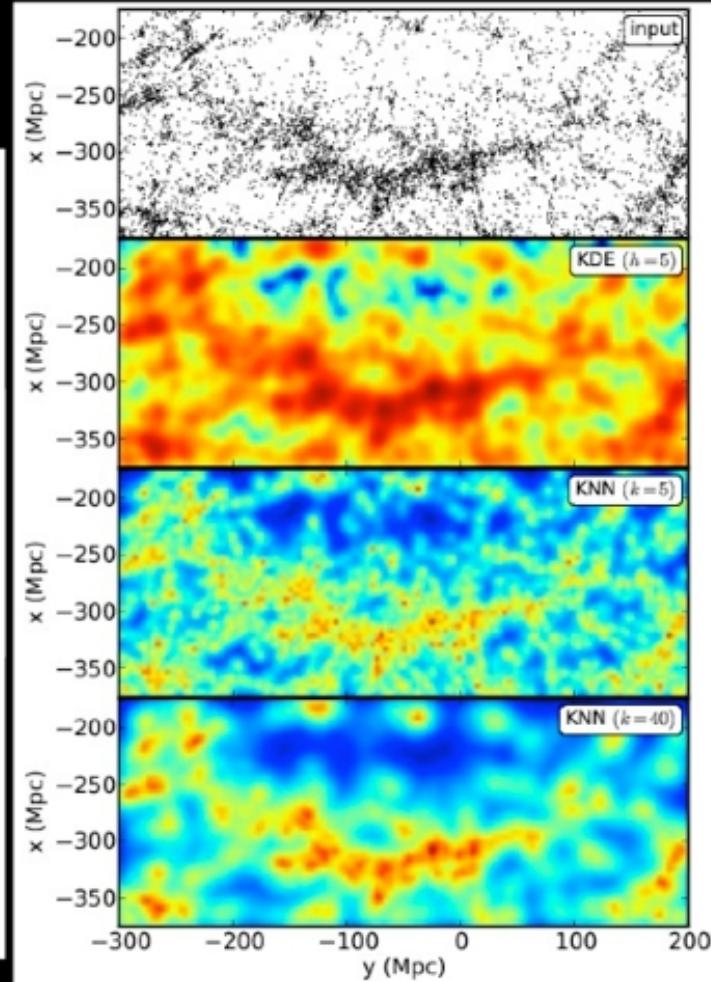
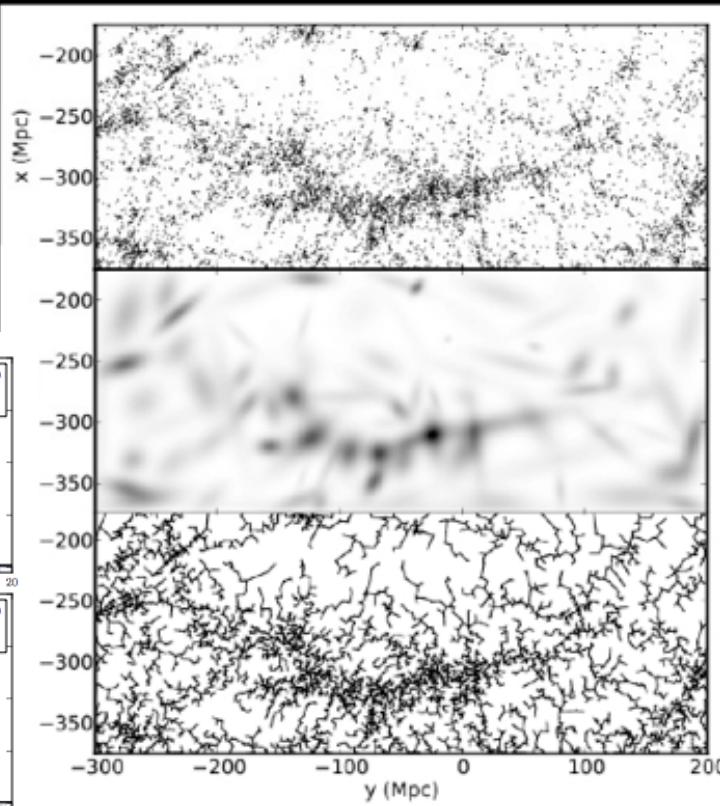
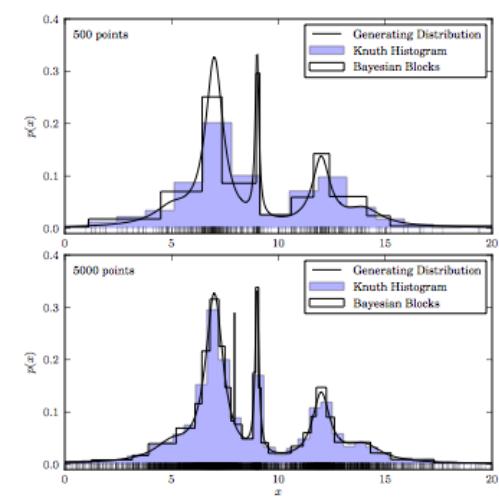
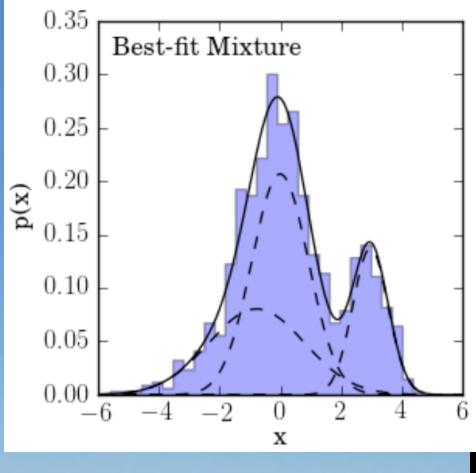
Gaia



Astroinformatics NS

Density Estimation

Clustering and Density Estimation: SDSS Great Wall



Outline

Density estimation is the act of estimating a continuous density field from a discretely sampled set of points drawn from that density field.

- One-dimensional introduction
 - Knuth's histograms
 - Scargle's Bayesian Blocks algorithm
 - Gaussian Mixture models
 - kernel density estimates (KDE)
 - the Wiener filter and connection to KDE
- Density estimation in high-D
 - high-D KDE
 - Bayesian nearest neighbor method
 - Extreme Deconvolution in high-D

What is a histogram?

- ---> Data modeled by a step function

Assuming that we have selected a bin size, Δ_b , the N values of x_i are sorted into M bins, with the count in each bin n_k , $k = 1, \dots, M$. If we want to express the results as a properly normalized $f(x)$, with the values f_k in each bin, then it is customary to adopt

$$f_k = \frac{n_k}{\Delta_b N}. \quad (4.80)$$

The unit for f_k is the inverse of the unit for x_i .

Each estimate of f_k comes with some uncertainty. It is customary to assign “error bars” for each n_k equal to $\sqrt{n_k}$ and thus the uncertainty of f_k is

$$\sigma_k = \frac{\sqrt{n_k}}{\Delta_b N}. \quad (4.81)$$

This practice assumes that n_k are scattered around the true values in each bin (μ) according to a Gaussian distribution, and that error bars enclose the 68% confidence range for the true value. However, when counts are low this assumption of Gaussianity breaks down and the Poisson distribution should be used instead. For example, according to the Gaussian distribution, negative values of μ have nonvanishing probability for small n_k (if $n_k = 1$, this probability is 16%). This is clearly wrong since in counting experiments, $\mu \geq 0$. Indeed, if $n_k \geq 1$, then even $\mu = 0$ is clearly ruled out. Note also that $n_k = 0$ does not necessarily imply that $\mu = 0$: even if $\mu = 1$, counts will be zero in $1/e \approx 37\%$ of cases. Another problem is that the range $n_k \pm \sigma_k$ does not correspond to the 68% confidence interval for true μ when n_k is small. These issues are important when fitting

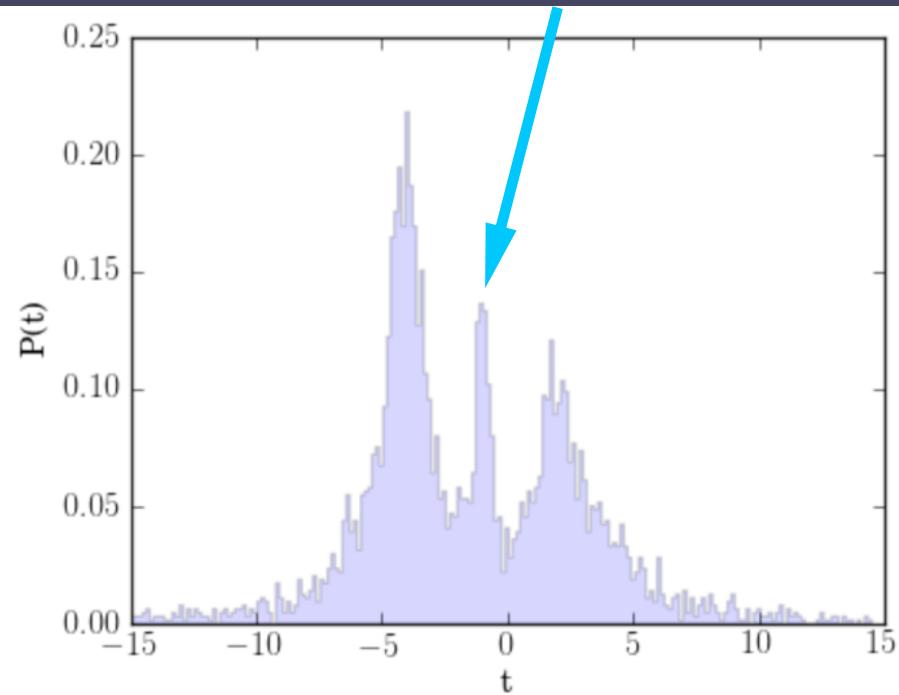
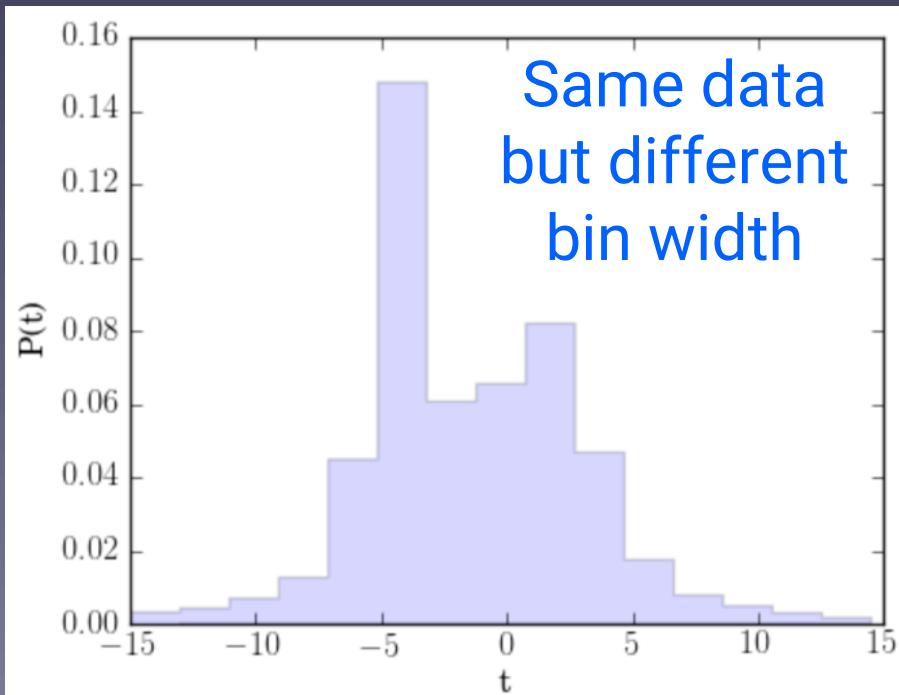
What is a histogram?

- ---> Data modeled by a step function
- How do we determine/estimate/guess the bin width?
- Do all the bins have to have the same width?
- Do we really have to bin data to estimate model parameters?

What is a histogram?

- > Data modeled by a step function
- How do we determine/estimate/guess the bin width?
- Do all the bins have to have the same width?
- Do we really have to bin data to estimate model parameters?

Should we believe the middle peak?



Simple rules for estimating bin width

Various proposed methods for choosing optimal bin width typically suggest a value proportional to some estimate of the distribution's scale, and decreasing with the sample size. The most popular choice is “Scott’s rule” which prescribes a bin width

$$\Delta_b = \frac{3.5\sigma}{N^{1/3}}, \quad (4.78)$$

where σ is the sample standard deviation, and N is the sample size. This rule asymptotically minimizes the mean integrated square error (see eq. 4.14) and assumes that the underlying distribution is Gaussian; see [22]. An attempt to generalize this rule to non-Gaussian distributions is the Freedman–Diaconis rule,

$$\Delta_b = \frac{2(q_{75} - q_{25})}{N^{1/3}} = \frac{2.7\sigma_G}{N^{1/3}}, \quad (4.79)$$

which estimates the scale (“spread”) of the distribution from its interquartile range (see [12]). In the case of a Gaussian distribution, Scott’s bin width is 30% larger than the Freedman–Diaconis bin width. Some rules use the extremes of observed values to estimate the scale of the distribution, which is clearly inferior to using the interquartile range when outliers are present.

Although the Freedman–Diaconis rule attempts to account for non-Gaussian distributions, it is too simple to distinguish, for example, multimodal and unimodal distributions that have the same σ_G .

Knuth's rule for estimating bin width

Knuth shows that the best piecewise constant model has the number of bins, M , which maximizes the following function (up to an additive constant, this is the logarithm of the posterior probability):

$$F(M|\{x_i\}, I) = N \log M + \log \left[\Gamma \left(\frac{M}{2} \right) \right] - M \log \left[\Gamma \left(\frac{1}{2} \right) \right] - \log \left[\Gamma \left(N + \frac{M}{2} \right) \right] + \sum_{k=1}^M \log \left[\Gamma \left(n_k + \frac{1}{2} \right) \right],$$

This is Bayesian model selection of M models (5.107)

where Γ is the gamma function, and n_k is the number of measurements x_i , $i = 1, \dots, N$, which are found in bin k , $k = 1, \dots, M$. Although this expression is more involved than the “rules of thumb” listed in §4.8.1, it can be easily evaluated for an *arbitrary* data set.

Knuth derived eq. 5.107 using Bayesian model selection and treating the histogram as a piecewise constant model of the underlying density function. By assumption, the bin width is constant and the number of bins is the result of model selection. Given the number of bins, M , the model for the underlying pdf is

$$h(x) = \sum_{k=1}^M h_k \Pi(x|x_{k-1}, x_k), \quad (5.108)$$

where the boxcar function $\Pi = 1$ if $x_{k-1} < x \leq x_k$, and 0 otherwise. The M model parameters, h_k , $k = 1, \dots, M$, are subject to normalization constraints, so that there are only $M - 1$ free parameters. The uninformative prior distribution for $\{h_k\}$ is given by

$$p(\{h_k\}|M, I) = \frac{\Gamma(\frac{M}{2})}{\Gamma(\frac{1}{2})^M} \left[h_1 h_2 \dots h_{M-1} \left(1 - \sum_{k=1}^{M-1} h_k \right) \right]^{-1/2}, \quad (5.109)$$

which is known as the Jeffreys prior for the multinomial likelihood. The joint data likelihood is a multinomial distribution (see §3.3.3)

$$p(\{x_i\}|\{h_k\}, M, I) \propto h_1^{n_1} h_2^{n_2} \dots h_M^{n_M}. \quad (5.110)$$

Scargle's Bayesian Blocks algorithm

- Knuth's method assumes constant bin width!
- We can use the same Bayesian machinery to generalize Knuth's model to varying bin width (these are additional model parameters, just like the other ones)

In the Bayesian blocks formalism, the data are segmented into *blocks*, with the borders between two blocks being set by *changepoints*. Using a Bayesian analysis based on Poissonian statistics within each block, an objective function, called the log-likelihood fitness function, can be defined for each block:

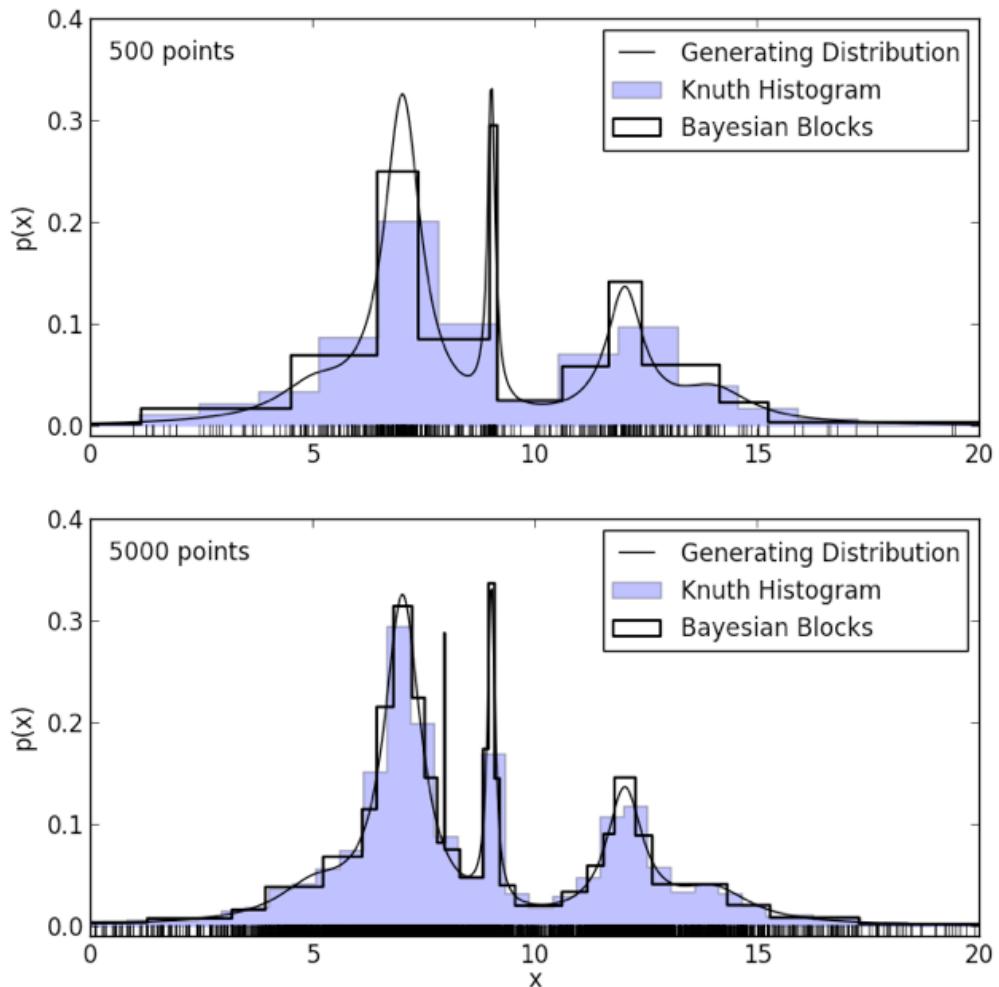
$$F(N_i, T_i) = N_i(\log N_i - \log T_i), \quad (5.113)$$

where N_i is the number of points in block i , and T_i is the width of block i (or the duration, in time-series analysis). Because of the additive nature of log-likelihoods, the fitness function for any set of blocks is simply the sum of the fitness functions for each individual block. This feature allows for the configuration space to be explored quickly using dynamic programming concepts: for more information see [31] or the Bayesian blocks implementation in AstroML.

[31] Scargle, J. D., J. P. Norris, B. Jackson, and J. Chiang (2012).
Studies in astronomical time series analysis. VI.
Bayesian block representations. ArXiv:astro-ph/1207.5578.

Comparing Knuth's rule and Bayesian Blocks

- make this plot by running
`%run fig_bayes_blocks.py`



Note that Knuth's method does not find the narrow peak in the middle for the smaller dataset!

Bayesian Blocks method gives you the best step function that describes your data. It is excellent for low-count data and for time-series analysis! But remember that data errors are not included!!!

Gaussian Mixture Models

The likelihood of a datum x_i for a Gaussian mixture model is given by

$$p(x_i|\boldsymbol{\theta}) = \sum_{j=1}^M \alpha_j \mathcal{N}(\mu_j, \sigma_j), \quad (4.18)$$

where dependence on x_i comes via a Gaussian $\mathcal{N}(\mu_j, \sigma_j)$. The vector of parameters $\boldsymbol{\theta}$ that need to be estimated for a given data set $\{x_i\}$ includes normalization factors for each Gaussian, α_j , and its parameters μ_j and σ_j . It is assumed that the data have negligible uncertainties (e.g., compared

Usually solved using **Expectation Maximization algorithm**
(for a good tutorial see ArXiv:statistics/1105.1476)

How to choose the number of classes?

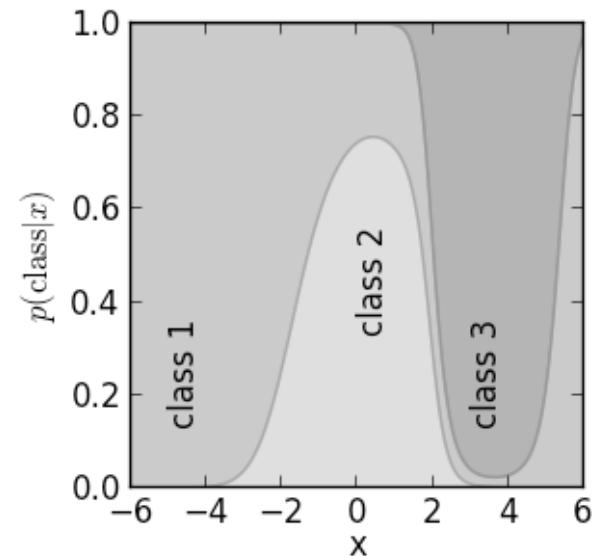
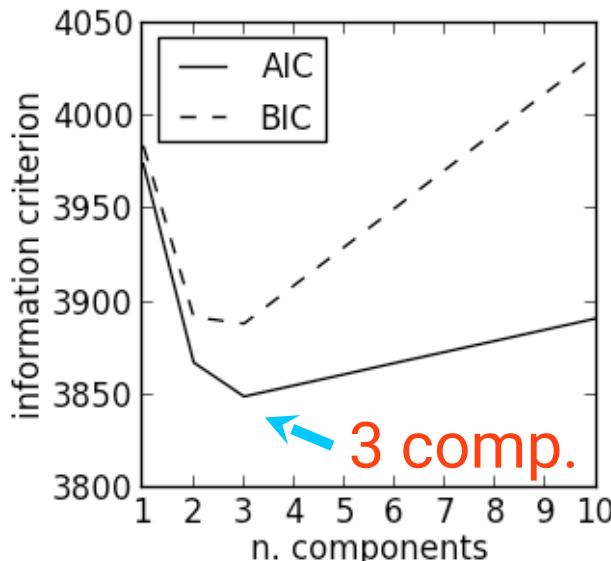
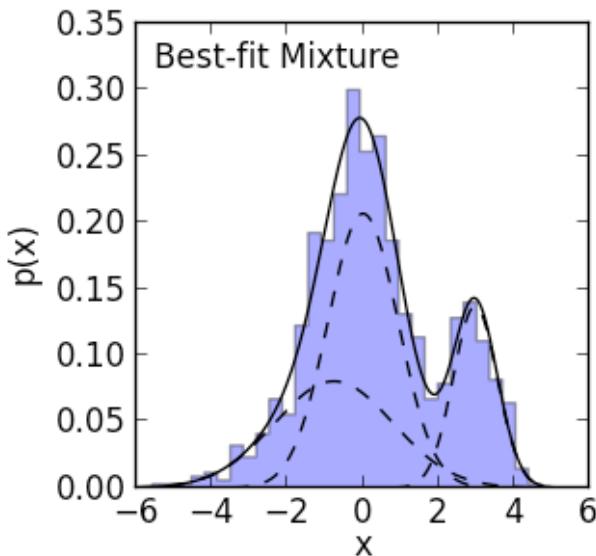
We have assumed in the above discussion of the EM algorithm that the number of classes in a mixture, M , is known. As M is increased, the description of the data set $\{x_i\}$ using a mixture model will steadily improve. On the other hand, a very large M is undesired—after all, $M = N$ will assign a mixture component to each point in a data set. How do we choose M in practice?

Usually determined using Bayesian Information Criterion (**BIC**),
or **cross-validation**

Gaussian Mixture Models

- make this plot by running
`%run fig_GMM_1D.py`

This best GMM fit is also density estimation! The only difference compared to histograms is in the chosen fitting model function!



Uses Expectation
Maximization
method
implemented in
scikit-learn

Bayesian Information
Criterion: it tells you
how many
components data
support!

**Example of
classification:
details next week**

Expectation Maximization Algorithm

- the problem just introduced is equivalent to parameter estimation for a mixture of Gaussians
- A well-known fast and straightforward solution was developed by Dempster, Laird & Rubin (1977)

The likelihood of a datum x_i for a Gaussian mixture model is given by

$$p(x_i|\boldsymbol{\theta}) = \sum_{j=1}^M \alpha_j \mathcal{N}(\mu_j, \sigma_j),$$

There are $(3M-1)$ parameters to estimate.

Since the class labels are not known, for each data value we can only determine the probability that it was generated by class j (sometimes called responsibility, e.g., HTF09). Given x_i , this probability can be obtained for each class using Bayes' rule (see eq. 3.10),

$$\text{shorthand } w_{ij} = p(j|x_i) \quad p(j|x_i) = \frac{\alpha_j \mathcal{N}(\mu_j, \sigma_j)}{\sum_{j=1}^M \alpha_j \mathcal{N}(\mu_j, \sigma_j)}. \quad (4.21)$$

The class probability $p(j|x_i)$ is small when x_i is not within “a few” σ_j from μ_j (assuming that x_i is close to some other mixture component). Of course, $\sum_{j=1}^M p(j|x_i) = 1$. This probabilistic

Expectation Maximization Algorithm

An iterative two-step (E and M) algorithm:

$$\frac{\partial \ln L}{\partial \theta_j} = - \sum_{i=1}^N w_{ij} \frac{\partial}{\partial \theta_j} \left[\ln \sigma_j + \frac{(x_i - \mu_j)^2}{2 \sigma_j^2} \right], \quad (4.25)$$

where θ_j now corresponds to μ_j or σ_j . By setting the derivatives of $\ln L$ with respect to μ_j and σ_j to zero, we get the estimators (this derivation is discussed in more detail in §5.6.1)

$$\mu_j = \frac{\sum_{i=1}^N w_{ij} x_i}{\sum_{i=1}^N w_{ij}}, \quad (4.26)$$

$$\sigma_j^2 = \frac{\sum_{i=1}^N w_{ij} (x_i - \mu_j)^2}{\sum_{i=1}^N w_{ij}}, \quad (4.27)$$

and from the normalization constraint,

shorthand $w_{ij} = p(j|x_i)$

$$\alpha_j = \frac{1}{N} \sum_{i=1}^N w_{ij}. \quad (4.28)$$

These expressions and eq. 4.21 form the basis of the iterative EM algorithm in the case of Gaussian mixtures. Starting with a guess for w_{ij} , the values of α_j , μ_j , and σ_j are estimated using eqs. 4.26–4.28. This is the “maximization” *M-step* which brings the parameters closer toward the local maximum. In the subsequent “expectation” *E-step*, w_{ij} are updated using eq. 4.21. The algorithm is not sensitive to the initial guess of parameter values. For example, setting all σ_j to the sample standard deviation, all α_j to $1/M$, and randomly drawing μ_j from the observed $\{x_i\}$ values, typically works well in practice (see HTF09).

Gaussian Mixture Models with Errors

The previous example assumed that uncertainty in data values (x) was negligible. What do we do when this is not the case?

For example, in case of homoscedastic Gaussian measurement errors, the three GMM components from the last example would be broadened (the measurement error would be added in quadrature to their intrinsic widths).

A recently introduced application of GMM with errors in astronomical context was dubbed “**Extreme Deconvolution**” (see Bovy et al. 2009, ArXiv:0905.2979)

Extreme Deconvolution works in multi-dimensional spaces too, and naturally treats noisy, heterogeneous, and incomplete data.

More details and a high-D example later...

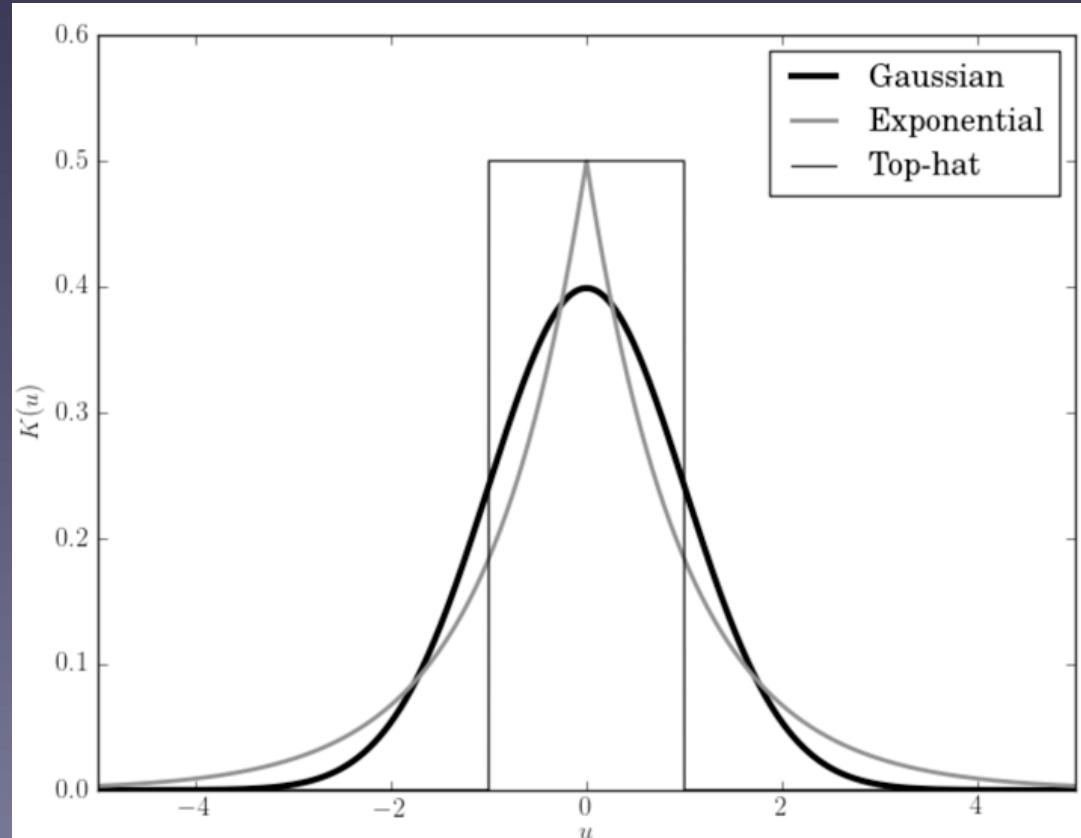
Kernel Density Estimates (KDE)

the underlying pdf is

$$h(x) = \sum_{k=1}^M h_k \Pi(x|x_{k-1}, x_k),$$

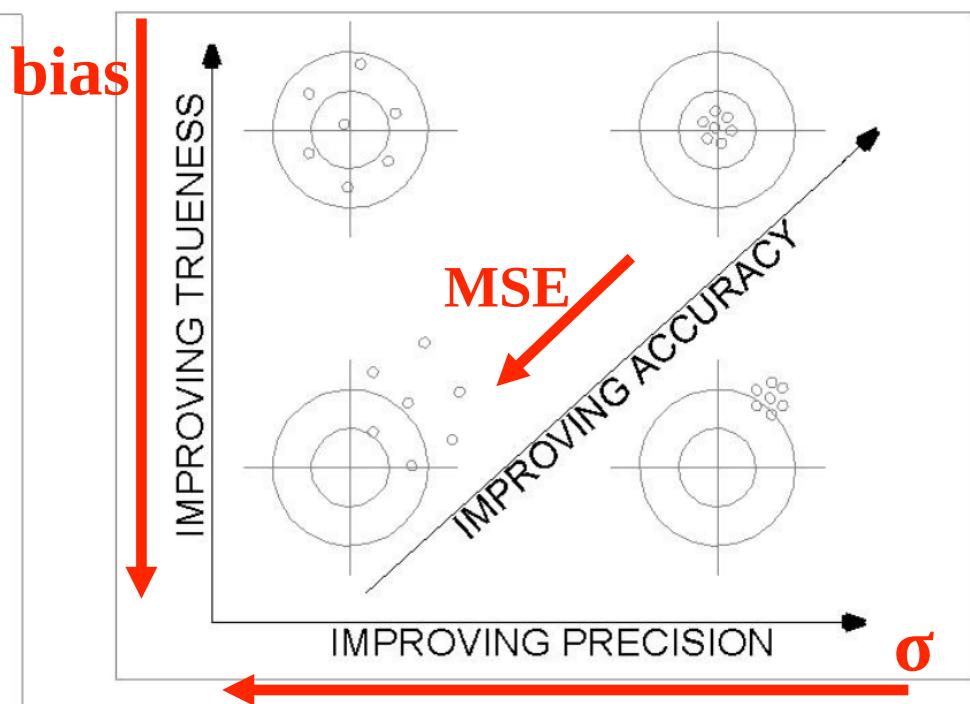
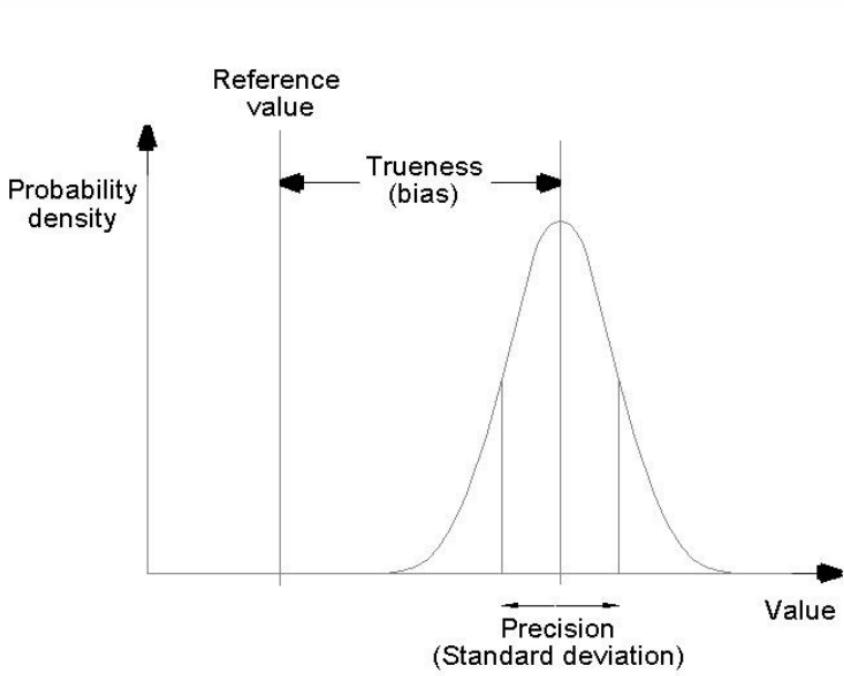
where the boxcar function $\Pi = 1$ if $x_{k-1} < x \leq x_k$, and 0 otherwise.

When computing histogram, we replace each object by the boxcar (top-hat) function. This function is called **kernel**. But of course we can use any other function (well, non-negative, normalized to 1, zero-mean, finite variance). This is the main idea of the KDE; it also works in high-D spaces.



• Measurements with known errors

Let's assume that we have N measurements x_i , and that for each measurement we know the corresponding error distribution, that is, the expected distribution of x_i around the true value μ (which we want to estimate)

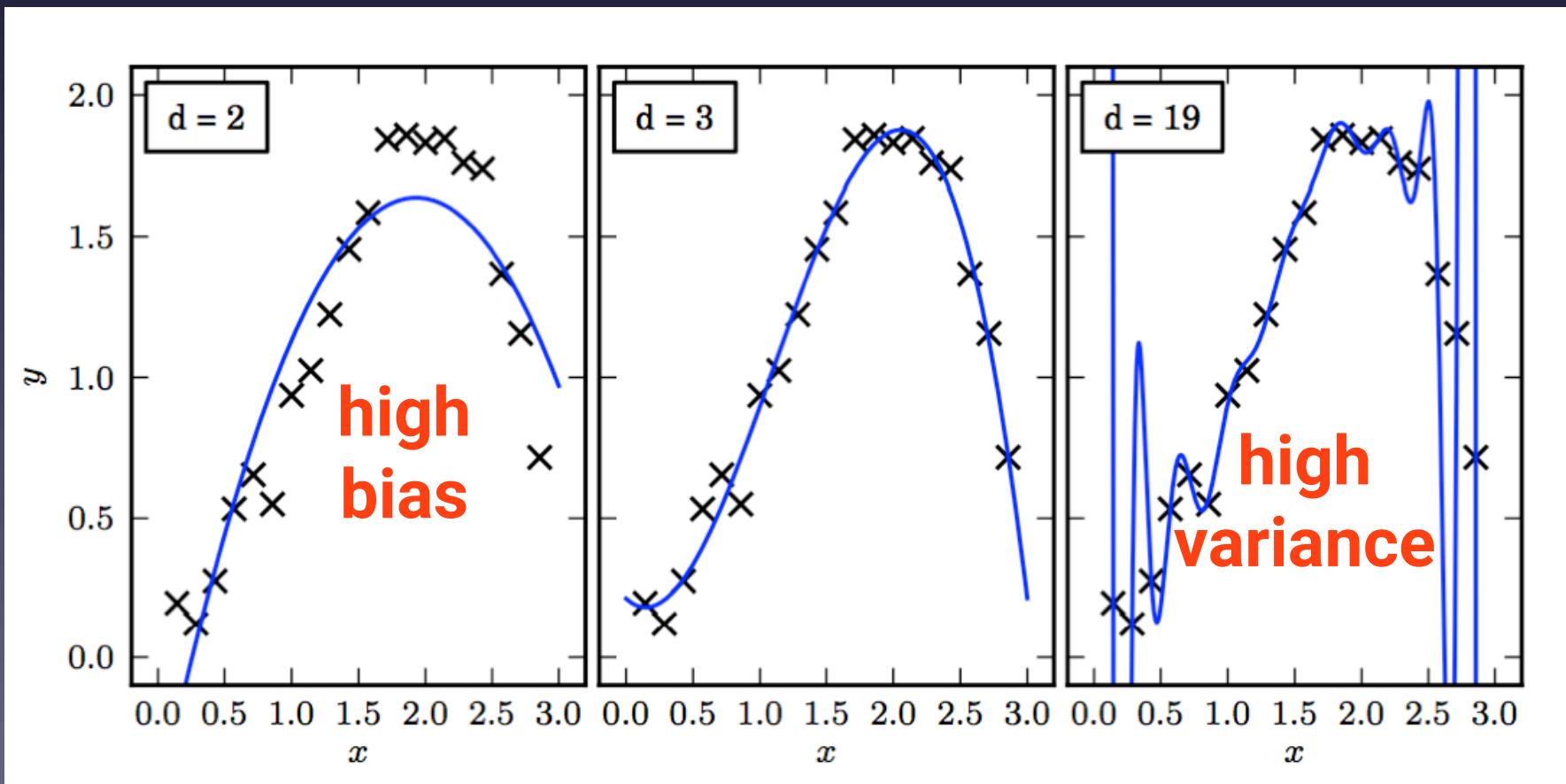


Mean Squared Error:

$$\text{MSE} = V + \text{bias}^2$$

(V =variance= σ^2)

Bias vs. variance tradeoff



high bias: “can’t fit” data points

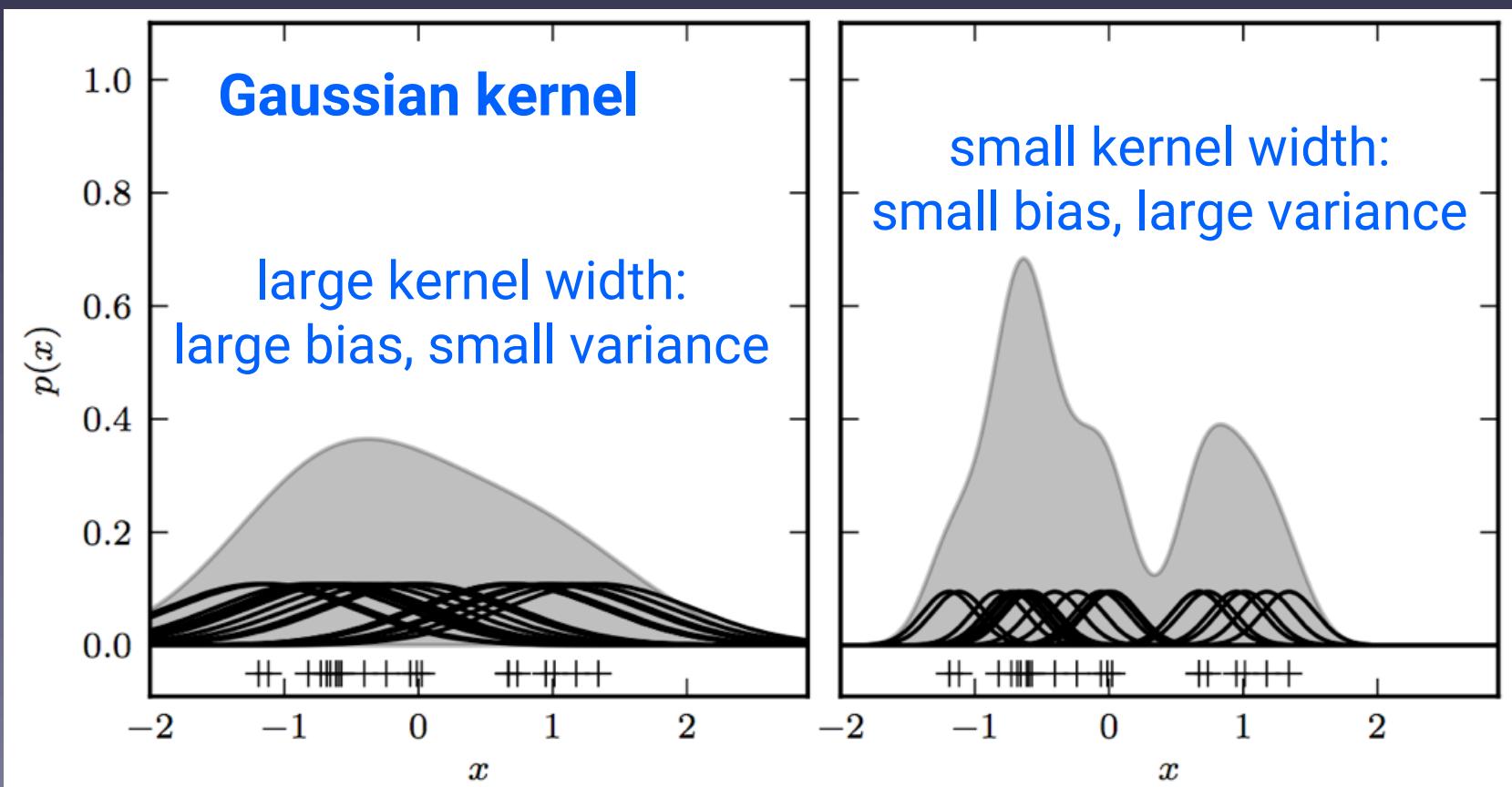
high variance: “oscillations” between data points (how would we know? c.f. cross-validation method: “leave one out”)

Kernel Density Estimates (KDE)

the underlying pdf is

$$h(x) = \sum_{k=1}^M h_k \Pi(x|x_{k-1}, x_k),$$

where the boxcar function $\Pi = 1$ if $x_{k-1} < x \leq x_k$, and 0 otherwise.



Kernel Density Estimates (KDE)

How do we choose the kernel width?

That is, the kernel width is a free parameter to be determined from data (despite KDE being called a non-parametric method)

Ideally we would select a kernel that has h as small as possible. If h becomes too small we increase the variance of the density estimation. If h is too large then the variance decreases but at the expense of the bias in the derived density. The optimal kernel function, in terms of minimum variance, turns out to be

$$K(x) = \frac{3}{4} (1 - x^2) \quad (6.9)$$

for $|x| \leq 1$ and 0 otherwise; see [37]. This function is called the Epanechnikov kernel.

We choose the kernel width by **cross-validation**:

Cross-validation can be used for any cost function (see §8.11); we just have to be able to evaluate the cost on *out-of-sample* data (i.e., points not in the training set). If we consider the likelihood cost for KDE, for which we have the leave-one-out *likelihood cross-validation*, then the cost is simply the sum over all points in the data set (i.e., $i = 1, \dots, N$) of the log of the likelihood of the density, where the density, $\hat{f}_{h,-i}(x_i)$, is estimated leaving out the i th data point. This can be written as

$$\text{CV}_l(h) = \frac{1}{N} \sum_{i=1}^N \log \hat{f}_{h,-i}(x_i), \quad (6.5)$$

Kernel Density Estimates (KDE)

The optimal KDE bandwidth decreases at the rate $\mathcal{O}(N^{-1/5})$ (in a one-dimensional problem), and the error of the KDE using the optimal bandwidth converges at the rate $\mathcal{O}(N^{-4/5})$; it can be shown that histograms converge at a rate $\mathcal{O}(N^{-2/3})$; see [35]. KDE is, therefore, theoretically superior to the histogram as an estimator of the density. It can also be shown that there does not exist a density estimator that converges faster than $\mathcal{O}(N^{-4/5})$ (see Wass10).

astroML implementation is very easy to use:

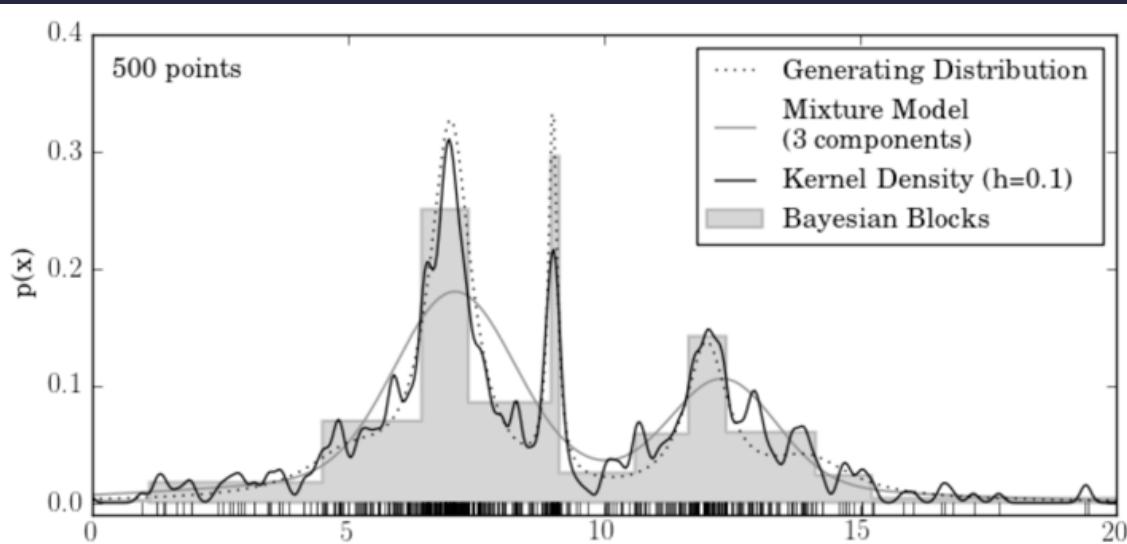
AstroML contains an implementation of kernel density estimation in D dimensions using the above kernels:

```
import numpy as np
from astroML.density_estimation import KDE

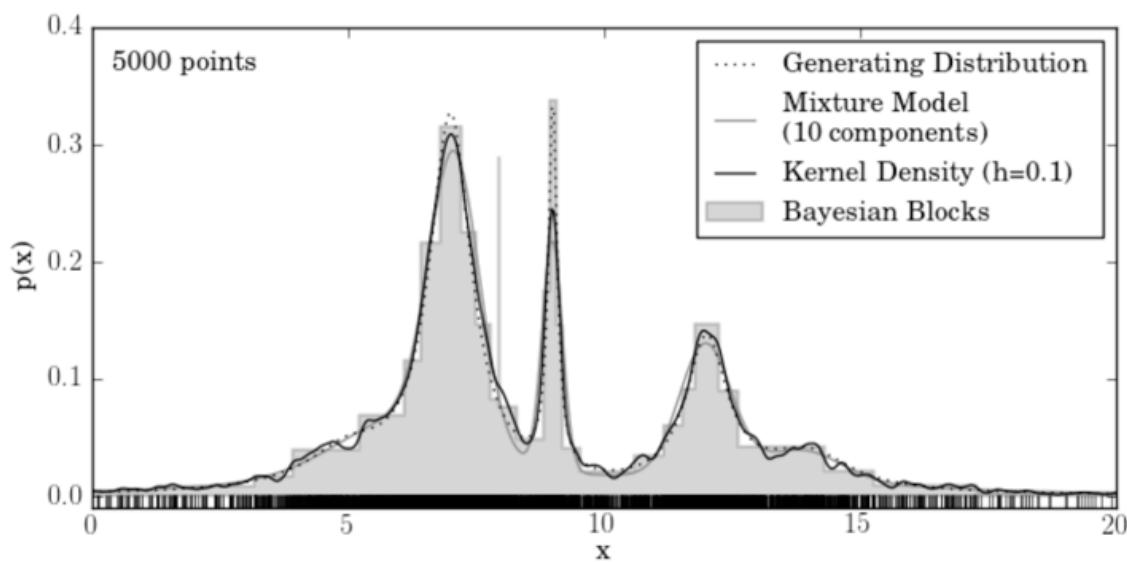
X = np.random.normal(size=(1000, 2)) # 1000 points in 2 dims
kde = KDE('gaussian', h=0.1) # select the gaussian kernel
kde.fit(X) # fit the model to the data
dens = kde.eval(X) # evaluate the model at the data
```

Comparing different methods:

- you **could** make this plot by running
`%run fig_GMM_density_estimation.py`



Key point: for large datasets, all methods result in similar estimates



puzzle: note the spike at $x \sim 8$ for the BB algorithm in the bottom panel

Data smoothing

- directly related to density estimation
- High-pass filtering: estimating the baseline
- Low-pass filtering: data smoothing

Data smoothing

The power spectrum for common Gaussian noise is flat and will extend to frequencies as high as the Nyquist limit, $f_N = 1/(2\Delta t)$. If the data are band limited to a lower frequency, $f_c < f_N$, then they can be smoothed without much impact by suppressing frequencies $|f| > f_c$. Given a filter in frequency space, $\Phi(f)$, we can obtain a smoothed version of data by taking the inverse Fourier transform of

$$\hat{Y}(f) = Y(f) \Phi(f), \quad (10.19)$$

where $Y(f)$ is the discrete Fourier transform of data. At least in principle, we could simply set $\Phi(f)$ to zero for $|f| > f_c$, but this approach would result in ringing (i.e., unwanted oscillations) in the signal. Instead, the optimal filter for this purpose is constructed by minimizing the MISE between $\hat{Y}(f)$ and $Y(f)$ (for detailed derivation see NumRec) and is called the Wiener filter:

$$\Phi(f) = \frac{P_S(f)}{P_S(f) + P_N(f)}. \quad (10.20)$$

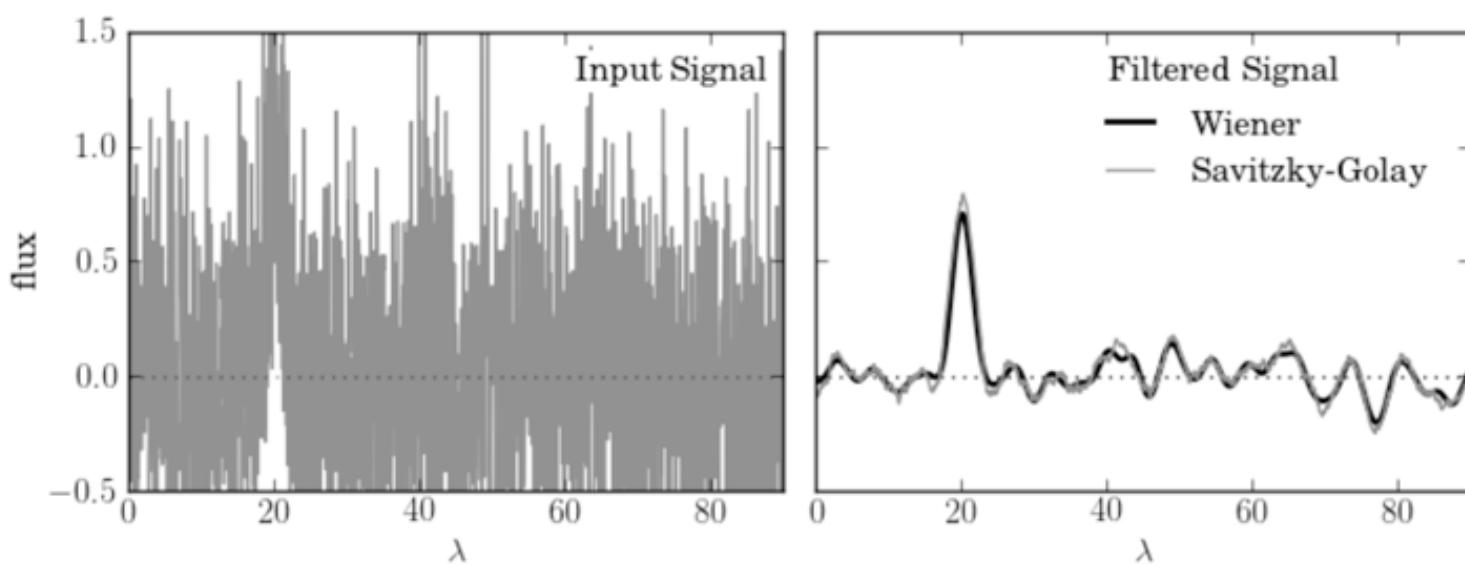
Here $P_S(f)$ and $P_N(f)$ represent components of a two-component (signal and noise) fit to the PSD of input data, $\text{PSD}_Y(f) = P_S(f) + P_N(f)$, which holds as long as the signal and noise are uncorrelated. Given some assumed form of signal and noise, these terms can be determined from a fit to the observed PSD, as illustrated by the example shown in figure 10.10. Even when the fidelity of the PSD fit is not high, the resulting filter performs well in practice (the key features are that $\Phi(f) \sim 1$ at small frequencies and that it drops to zero at high frequencies for a band-limited signal).

Data smoothing

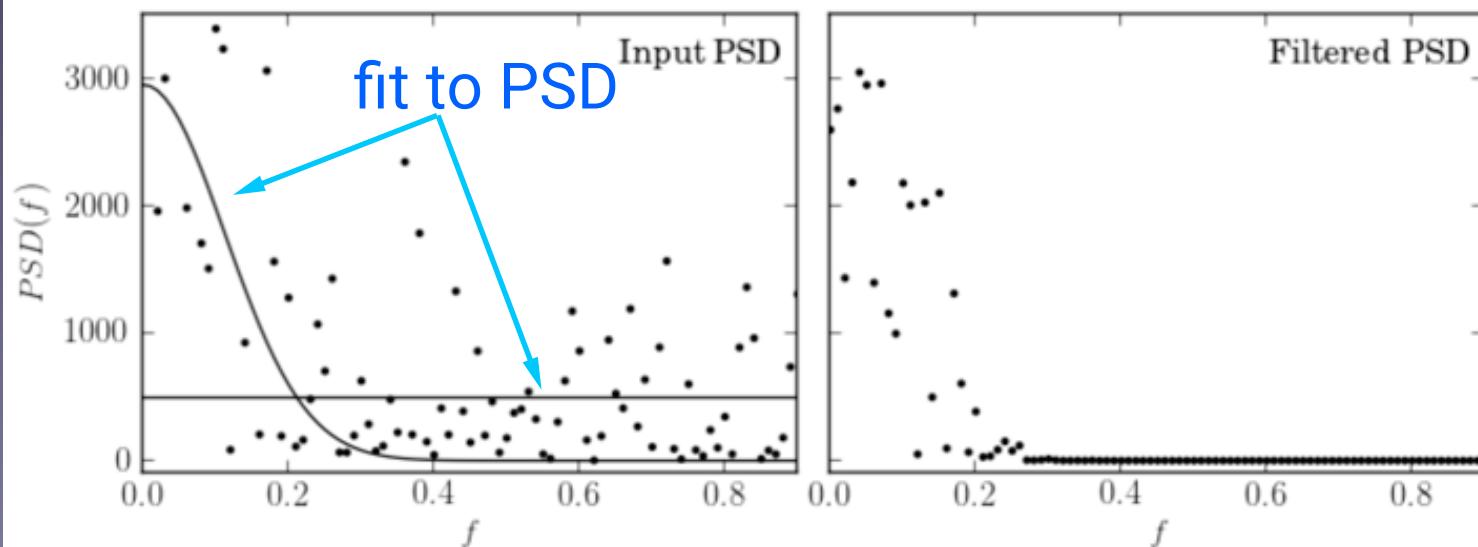
- make this plot by running
%run fig_wiener_filter.py

Savitzky-Golay filter:

local polynomial regression (here 4)
works for unevenly sampled data!



if error, add
this to call in
line 40:
use_fft=False



KDE - Wiener filter connection

- make this plot by running
`%run fig_wiener_KDE.py`

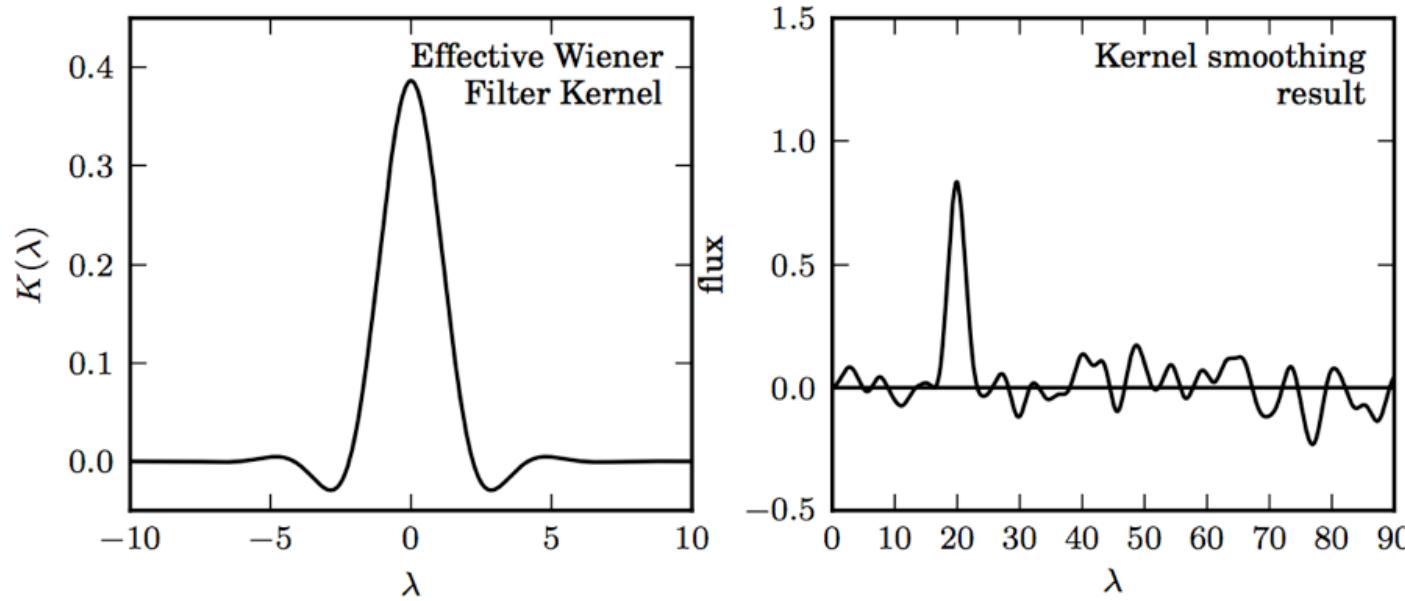


Figure 10.11.: The left panel shows the inverse Fourier transform of the Wiener filter $\Phi(f)$ applied in figure 10.10. By the convolution theorem, the Wiener-filtered result is equivalent to the convolution of the unfiltered signal with the kernel shown above, and thus Wiener filtering and kernel density estimation (KDE) are directly related. The right panel shows the data smoothed by this kernel, which is equivalent to the Wiener filter smoothing in figure 10.10.

Unlike FFT needed for Wiener filtering, KDE of course also works with unevenly sampled data!

Outline

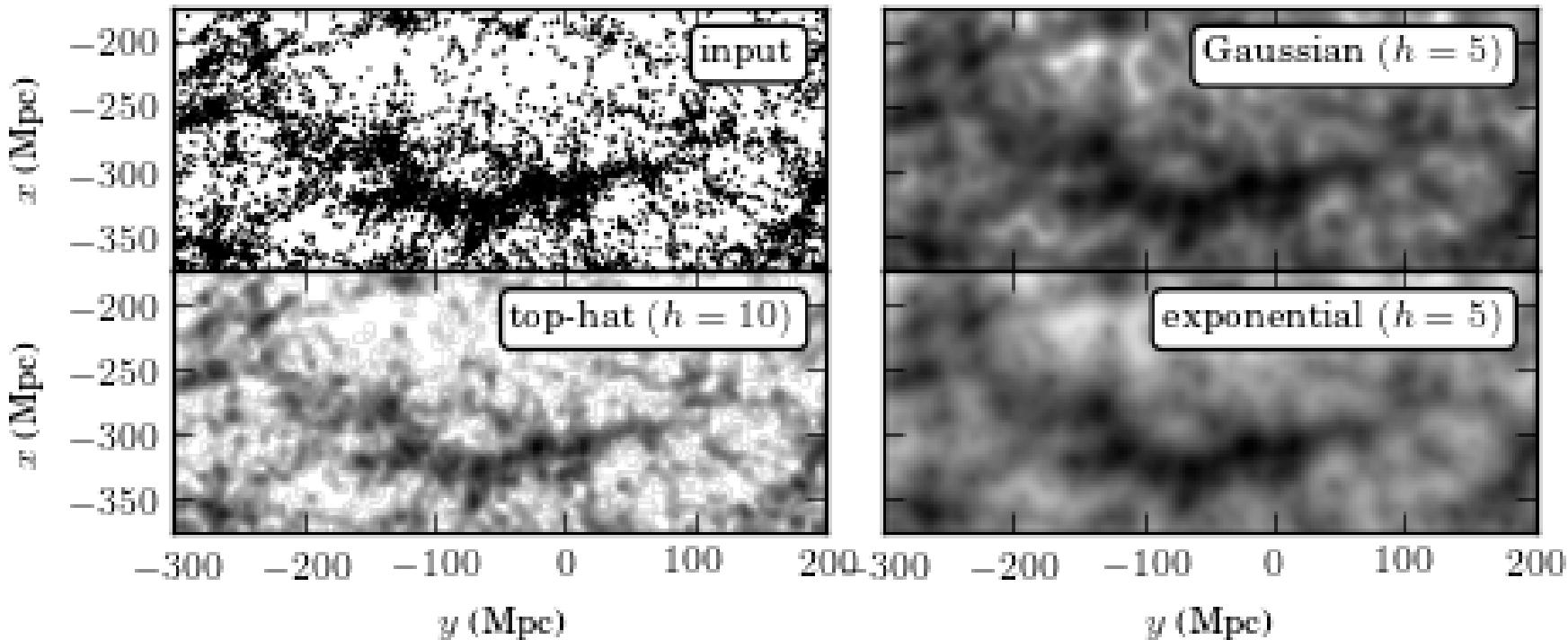
Density estimation is the act of estimating a continuous density field from a discretely sampled set of points drawn from that density field.

- One-dimensional introduction
 - Knuth's histograms
 - Scargle's Bayesian Blocks algorithm
 - Gaussian Mixture models
 - kernel density estimates (KDE)
 - the Wiener filter and connection to KDE
- Density estimation in high-D
 - high-D KDE
 - Bayesian nearest neighbor method
 - Extreme Deconvolution in high-D

KDE in 2-D case

- you **could** make this plot by running
`%run fig_great_wall_KDE.py`

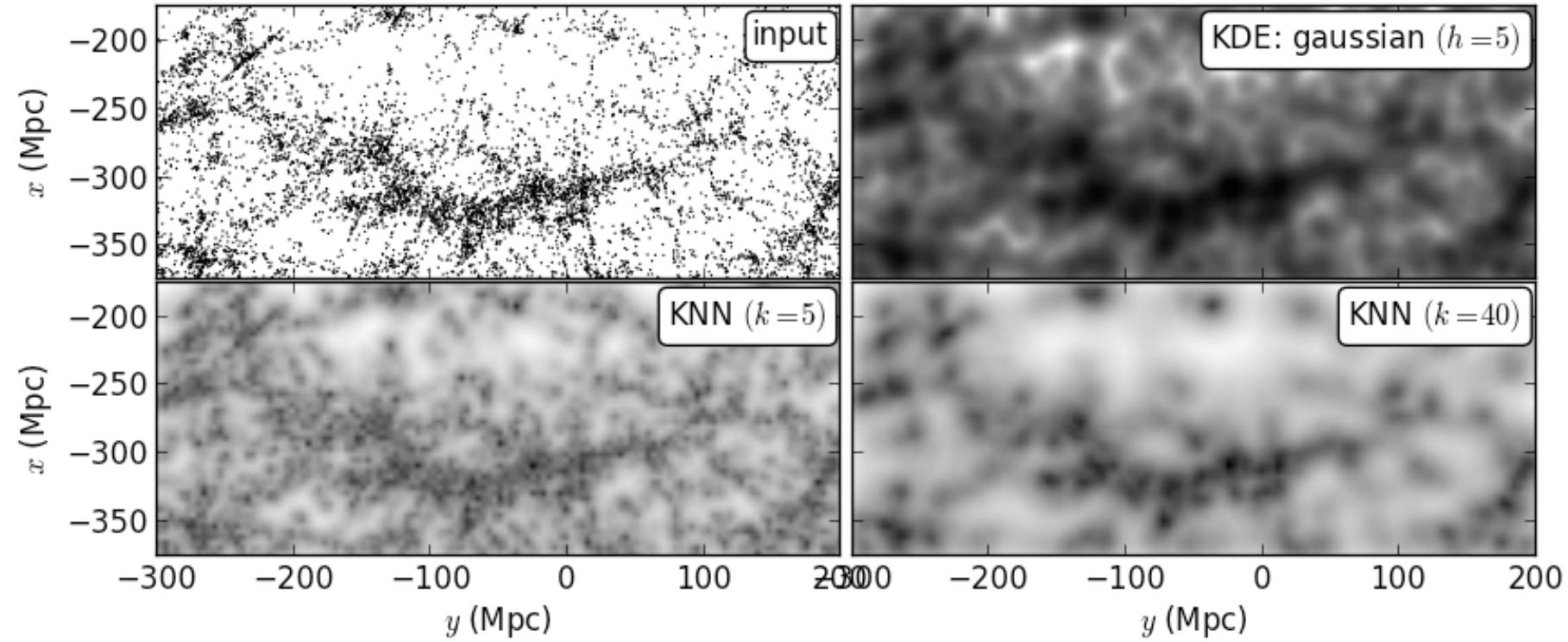
SDSS Great Wall



“SDSS Great Wall” is the largest known concentration of galaxies (Gott et al. 2005, ApJ 624, 463)

KDE in 2-D case

- make this plot by running
`%run fig_great_wall.py`



KNN here stands for “K nearest neighbors”: a Bayesian method good for sparsely sampled data (Ivezić et al. 2005, AJ 129, 1096)

Bayesian K Nearest Neighbors method

The K-th Nearest Neighbor Method (Dressler 1980, ApJ 236, 351):

Another often used and simple density estimation technique is based on the distribution of nearest neighbors. For each point (e.g., a pixel location on the two-dimensional grid) we can find the distance to the K th-nearest neighbor, d_K . In this method, originally proposed in an astronomical context by Dressler et al. [11], the implied point density at an arbitrary position x is estimated as

$$\hat{f}_K(x) = \frac{K}{V_D(d_K)}, \quad (6.13)$$

where the volume V_D is evaluated according to the problem dimensionality, D (e.g., for $D = 2$, $V_2 = \pi d^2$; for $D = 3$, $V_3 = 4\pi d^3/3$; for higher dimensions, see eq. 7.3). The simplicity of this estimator is a consequence of the assumption that the underlying density field is locally constant. In practice, the method is even simpler because one can compute

$$\hat{f}_K(x) = \frac{C}{d_K^D}, \quad (6.14)$$

This method is biased and has a large variance for small K .

Bayesian K Nearest Neighbors method

$$\hat{f}_K(x) = \frac{C}{d_K^D}, \quad (6.14)$$

and evaluate the scaling factor C at the end by requiring that the sum of the product of $\hat{f}_K(x)$ and pixel volume is equal to the total number of data points. The error in $\hat{f}_K(x)$ is $\sigma_f = K^{1/2}/V_D(d_K)$, and the fractional (relative) error is $\sigma_f/\hat{f} = 1/K^{1/2}$. Therefore, the fractional accuracy increases with K at the expense of the spatial resolution (the effective resolution scales with $K^{1/D}$). In practice, K should be at least 5 because the estimator is biased and has a large variance for smaller K ; see [7].

This general method can be improved (the error in \hat{f} can be decreased without a degradation in the spatial resolution, or alternatively the resolution can be increased without increasing the error in \hat{f}) by considering distances to *all* K nearest neighbors instead of only the distance to the K th-nearest neighbor; see [18]. Given distances to all K neighbors, d_i , $i = 1, \dots, K$,

$$\hat{f}_K(x) = \frac{C}{\sum_{i=1}^K d_i^D}. \quad (6.15)$$

Derivation of eq. 6.15 is based on Bayesian analysis, as described in [18]. The proper normalization when computing local density without regard to overall mean density is

$$C = \frac{K(K+1)}{2V_D(1)}. \quad (6.16)$$

This method is very good for sparsely sampled data, and better than e.g. KDE with a Gaussian kernel for $K \sim 10$ (details: Ivezić+ 2005, AJ 129, 1096)

Extreme Deconvolution in high-D

- **But let's first warm up with 1-D case**
- **Two basic assumptions:**
 - a) the sampled population distribution is a single Gaussian, and
 - b) data have known heteroscedastic errors:

In order to proceed with parameter estimation in this situation, we shall assume that data were drawn from an intrinsic $\mathcal{N}(\mu, \sigma)$ distribution, and that measurement errors are also Gaussian and described by the known width e_i . Starting with an analog of eq. 5.52,

$$p(\{x_i\}|\mu, \sigma, I) = \prod_{i=1}^N \frac{1}{\sqrt{2\pi}(\sigma^2 + e_i^2)^{1/2}} \exp\left(\frac{-(x_i - \mu)^2}{2(\sigma^2 + e_i^2)}\right), \quad (5.63)$$

and following the same steps as above (with uniform priors for μ and σ), we get an analog of eq. 5.56:

$$L_p = \text{constant} - \frac{1}{2} \sum_{i=1}^N \left(\ln(\sigma^2 + e_i^2) + \frac{(x_i - \mu)^2}{\sigma^2 + e_i^2} \right). \quad (5.64)$$

The solution for best-fit parameters cannot be expressed in a closed form any more (it can for homoscedastic errors)

- Of course, we can evaluate posterior pdf by brute force but this doesn't work well in high-D cases:

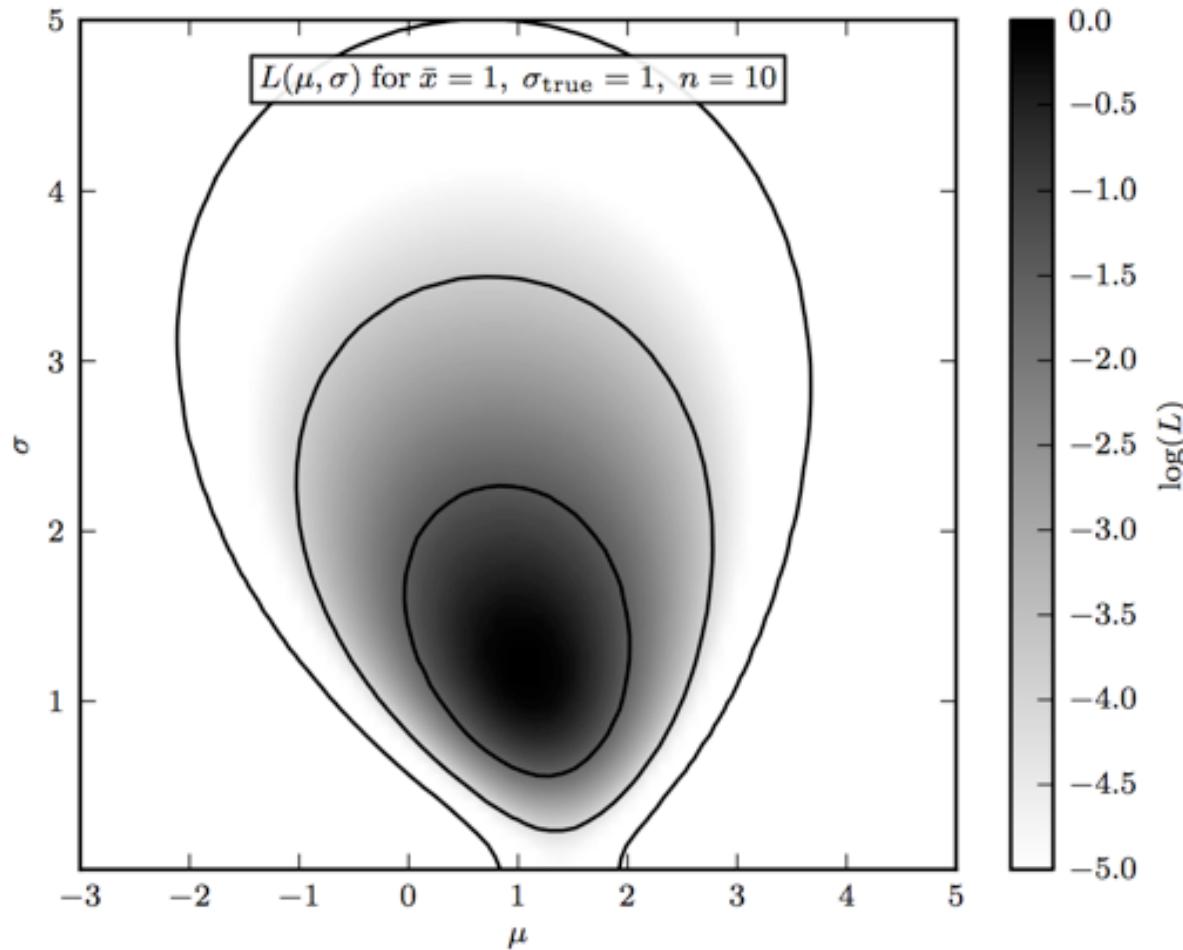


Figure 5.7.: The logarithm of the posterior probability density function for μ and σ , $L_p(\mu, \sigma)$, for a Gaussian distribution with heteroscedastic Gaussian measurement errors (sampled uniformly from the 0–3 interval), given by eq. 5.64. The input values are $\mu = 1$ and $\sigma = 1$, and a randomly generated sample has 10 points. Note that the posterior pdf is not symmetric with respect to the $\mu = 1$ line, and that the outermost contour, which encloses the region that contains 0.997 of the cumulative (integrated) posterior probability, allows solutions with $\sigma = 0$.

Expectation Maximization Algorithm

- the problem just introduced is equivalent to parameter estimation for a mixture of Gaussians
- A well-known fast and straightforward solution was developed by Dempster, Laird & Rubin (1977)

The likelihood of a datum x_i for a Gaussian mixture model is given by

$$p(x_i|\boldsymbol{\theta}) = \sum_{j=1}^M \alpha_j \mathcal{N}(\mu_j, \sigma_j),$$

There are $(3M-1)$ parameters to estimate.

Since the class labels are not known, for each data value we can only determine the probability that it was generated by class j (sometimes called responsibility, e.g., HTF09). Given x_i , this probability can be obtained for each class using Bayes' rule (see eq. 3.10),

$$\text{shorthand } w_{ij} = p(j|x_i) \quad p(j|x_i) = \frac{\alpha_j \mathcal{N}(\mu_j, \sigma_j)}{\sum_{j=1}^M \alpha_j \mathcal{N}(\mu_j, \sigma_j)}. \quad (4.21)$$

The class probability $p(j|x_i)$ is small when x_i is not within “a few” σ_j from μ_j (assuming that x_i is close to some other mixture component). Of course, $\sum_{j=1}^M p(j|x_i) = 1$. This probabilistic

Expectation Maximization Algorithm

An iterative two-step (E and M) algorithm:

$$\frac{\partial \ln L}{\partial \theta_j} = - \sum_{i=1}^N w_{ij} \frac{\partial}{\partial \theta_j} \left[\ln \sigma_j + \frac{(x_i - \mu_j)^2}{2 \sigma_j^2} \right], \quad (4.25)$$

where θ_j now corresponds to μ_j or σ_j . By setting the derivatives of $\ln L$ with respect to μ_j and σ_j to zero, we get the estimators (this derivation is discussed in more detail in §5.6.1)

$$\mu_j = \frac{\sum_{i=1}^N w_{ij} x_i}{\sum_{i=1}^N w_{ij}}, \quad (4.26)$$

$$\sigma_j^2 = \frac{\sum_{i=1}^N w_{ij} (x_i - \mu_j)^2}{\sum_{i=1}^N w_{ij}}, \quad (4.27)$$

and from the normalization constraint,

shorthand $w_{ij} = p(j|x_i)$

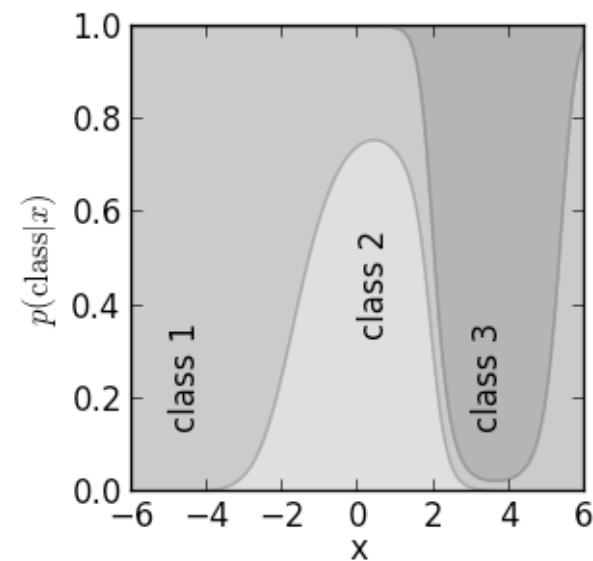
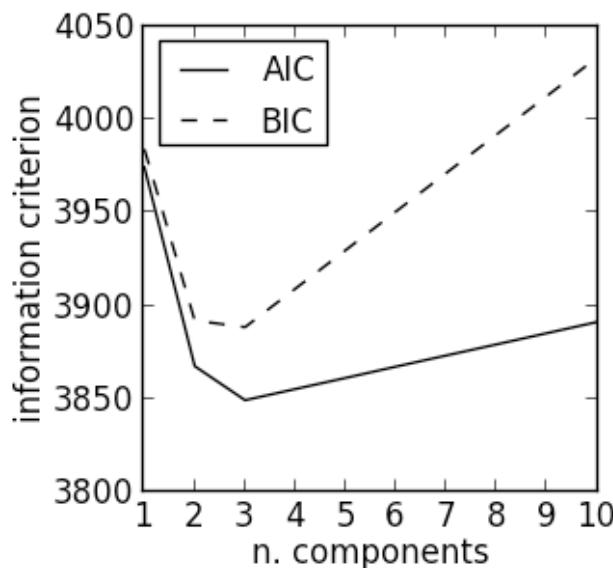
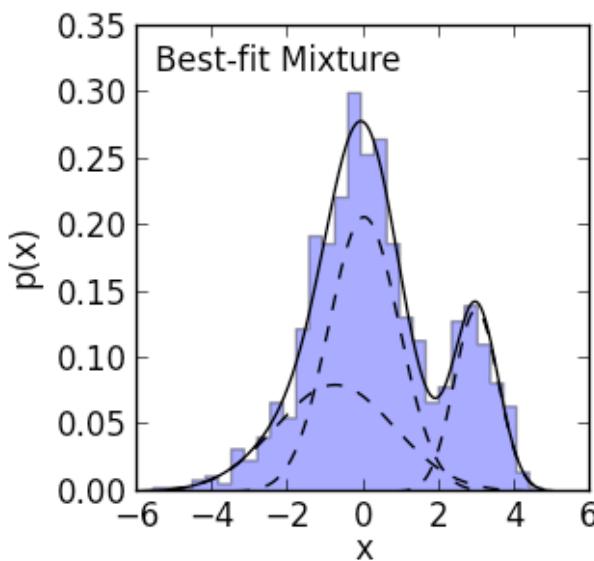
$$\alpha_j = \frac{1}{N} \sum_{i=1}^N w_{ij}. \quad (4.28)$$

These expressions and eq. 4.21 form the basis of the iterative EM algorithm in the case of Gaussian mixtures. Starting with a guess for w_{ij} , the values of α_j , μ_j , and σ_j are estimated using eqs. 4.26–4.28. This is the “maximization” *M-step* which brings the parameters closer toward the local maximum. In the subsequent “expectation” *E-step*, w_{ij} are updated using eq. 4.21. The algorithm is not sensitive to the initial guess of parameter values. For example, setting all σ_j to the sample standard deviation, all α_j to $1/M$, and randomly drawing μ_j from the observed $\{x_i\}$ values, typically works well in practice (see HTF09).

Expectation Maximization Algorithm

Scikit-learn contains an EM algorithm for fitting N -dimensional mixtures of Gaussians:

```
>>> import numpy as np
>>> from sklearn.mixture import GMM
>>> X = np.random.normal(size=(100, 1)) # 100 points in 1 dim
>>> model = GMM(2) # two components
>>> model.fit(X)
>>> model.means_ # the locations of the best-fit components
array([[-0.05786756],
       [ 0.69668864]])
```



Extreme Deconvolution in high-D (XD)

Bovy, Hogg & Roweis 2011 (arXiv:0905.2979)

- **Two basic assumptions:**
 - a) the sampled population distribution is a arbitrary mixture of high-D Gaussian components, and
 - b) data have heteroscedastic errors with known covariance matrix

being α_i . Thus, the pdf of \mathbf{x} is given as

$$p(\mathbf{x}) = \sum_j \alpha_j \mathcal{N}(\mathbf{x}|\mu_j, \Sigma_j), \quad (6.18)$$

where, recalling eq. 3.97,

$$\mathcal{N}(\mathbf{x}|\mu_j, \Sigma_j) = \frac{1}{\sqrt{(2\pi)^D \det(\Sigma_j)}} \exp \left(-\frac{1}{2} (\mathbf{x} - \mu_j)^T \Sigma_j^{-1} (\mathbf{x} - \mu_j) \right). \quad (6.19)$$

Extreme deconvolution generalizes the EM approach to a case with measurement errors. More explicitly, one assumes that the noisy observations \mathbf{x}_i and the true values \mathbf{v}_i are related through

$$\mathbf{x}_i = \mathbf{R}_i \mathbf{v}_i + \epsilon_i, \quad (6.20)$$

XD is a very powerful method: it can treat incomplete and heterogeneous data

Extreme Deconvolution in high-D (XD)

Bovy, Hogg & Roweis 2011 (arXiv:0905.2979)

- The expectation (E) step:

$$\begin{aligned} q_{ij} &\leftarrow \frac{\alpha_j \mathcal{N}(\mathbf{w}_i | \mathbf{R}_i \boldsymbol{\mu}_j, \mathbf{T}_{ij})}{\sum_j \alpha_k \mathcal{N}(\mathbf{w}_i | \mathbf{R}_i \boldsymbol{\mu}_k, \mathbf{T}_{ij})}, \\ \mathbf{b}_{ij} &\leftarrow \boldsymbol{\mu}_j + \boldsymbol{\Sigma}_j \mathbf{R}_i^T \mathbf{T}_{ij}^{-1} (\mathbf{w}_i - \mathbf{R}_i \boldsymbol{\mu}_j), \\ \mathbf{B}_{ij} &\leftarrow \boldsymbol{\Sigma}_j - \boldsymbol{\Sigma}_j \mathbf{R}_i^T \mathbf{T}_{ij}^{-1} \mathbf{R}_i \boldsymbol{\Sigma}_j, \end{aligned}$$

where $\mathbf{T}_{ij} = \mathbf{R}_i \boldsymbol{\Sigma}_j \mathbf{R}_i^T + \mathbf{S}_i$.

- The maximization (M) step:

$$\begin{aligned} \alpha_i &\leftarrow \frac{1}{N} \sum_i q_{ij}, \\ \boldsymbol{\mu}_j &\leftarrow \frac{1}{q_j} \sum_i q_{ij} \mathbf{b}_{ij}, \\ \boldsymbol{\Sigma}_j &\leftarrow \frac{1}{q_j} \sum_i q_{ij} [(\boldsymbol{\mu}_j - \mathbf{b}_{ij})(\boldsymbol{\mu}_j - \mathbf{b}_{ij}^T) + \mathbf{B}_{ij}], \end{aligned}$$

where $q_j = \sum_i q_{ij}$.

Extreme Deconvolution in high-D (XD)

Bovy, Hogg & Roweis 2011 (arXiv:0905.2979)

The iteration of these steps increases the likelihood of the observations \mathbf{w}_i , given the model parameters. Thus, iterating until convergence, one obtains a solution that is a local maximum of the likelihood. This method has been used with success in quasar classification, by estimating the densities of quasar and nonquasar objects from flux measurements; see [5]. Details of the use of XD, including methods to avoid local maxima in the likelihood surface, can be found in [6].

AstroML contains an implementation of XD which has a similar interface to GMM in Scikit-learn:

```
import numpy as np
from astroML.density_estimation import XDGMM

X = np.random.normal(size=(1000, 1)) # 1000 pts in 1 dim
Xerr = np.random.random((1000, 1, 1)) # 1000 1x1 covariance matrices
xdgmm = XDGMM(n_components=2)
xdgmm.fit(X, Xerr) # fit the model
logp = xdgmm.logprob_a(X, Xerr) # evaluate probability
X_new = xdgmm.sample(1000) # sample new points from distribution
```

Extreme Deconvolution in high-D (XD)

- make this plot by running
`%run fig_XD_example.py`

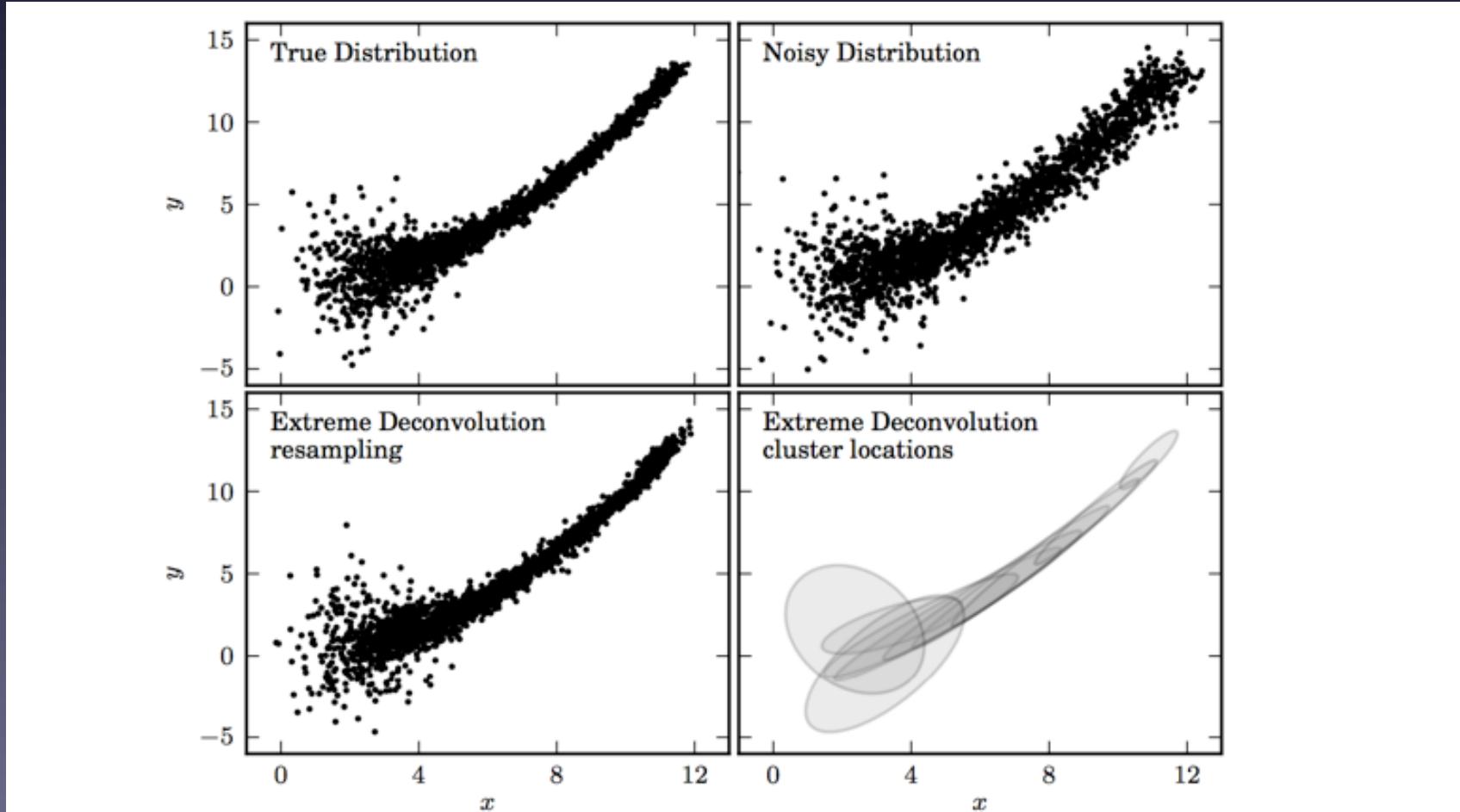
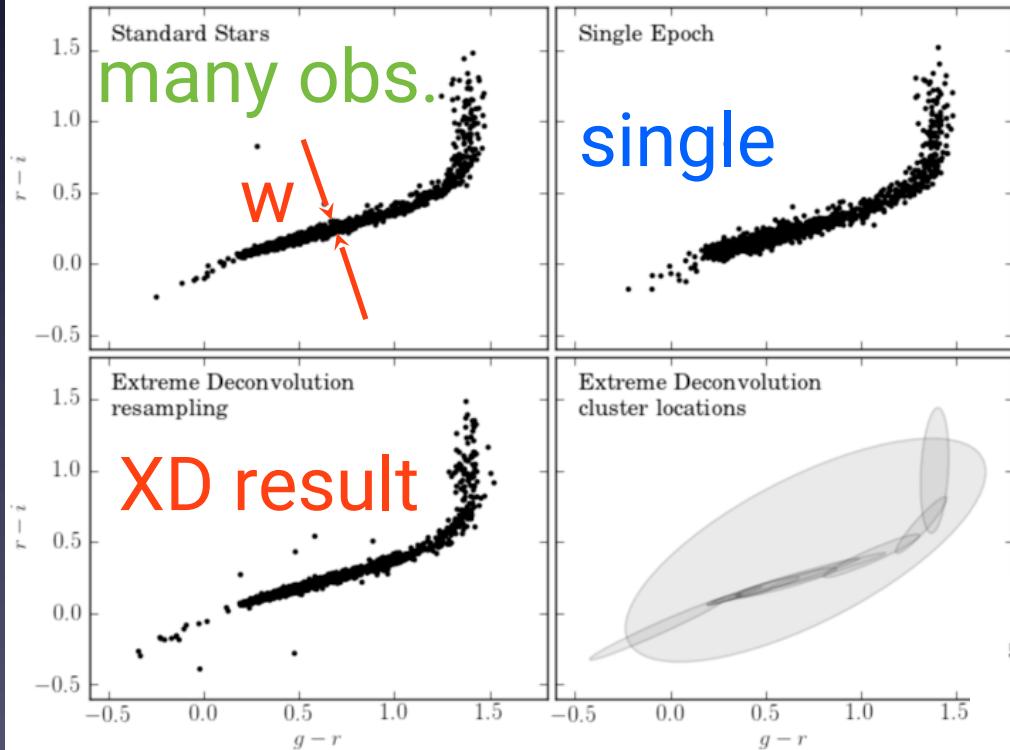


Figure 6.11.: An example of extreme deconvolution showing a simulated two-dimensional distribution of points, where the positions are subject to errors. The top two panels show the distributions with small (left) and large (right) errors. The bottom panels show the densities derived from the noisy sample (top-right panel) using extreme deconvolution; the resulting distribution closely matches that shown in the top-left panel.

Extreme Deconvolution in high-D (XD)



XD result: the intrinsic width of the $r - i$ vs. $g - r$ stellar locus is ~ 0.01 mag., in agreement with high-SNR averaged multi-epoch photometry

XD is a very powerful method: if GMM is able to model the observed distribution (usually OK)

