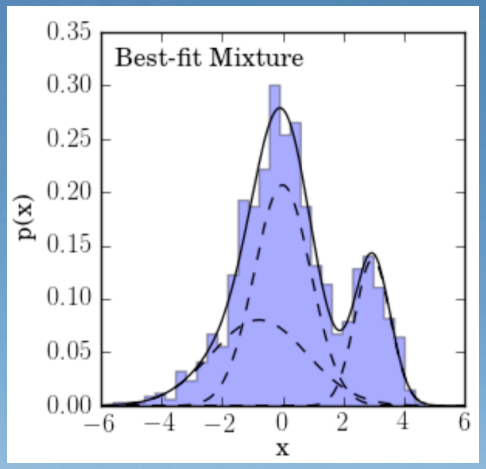
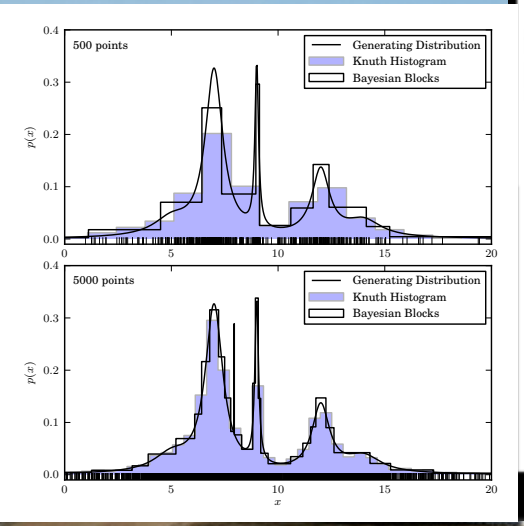
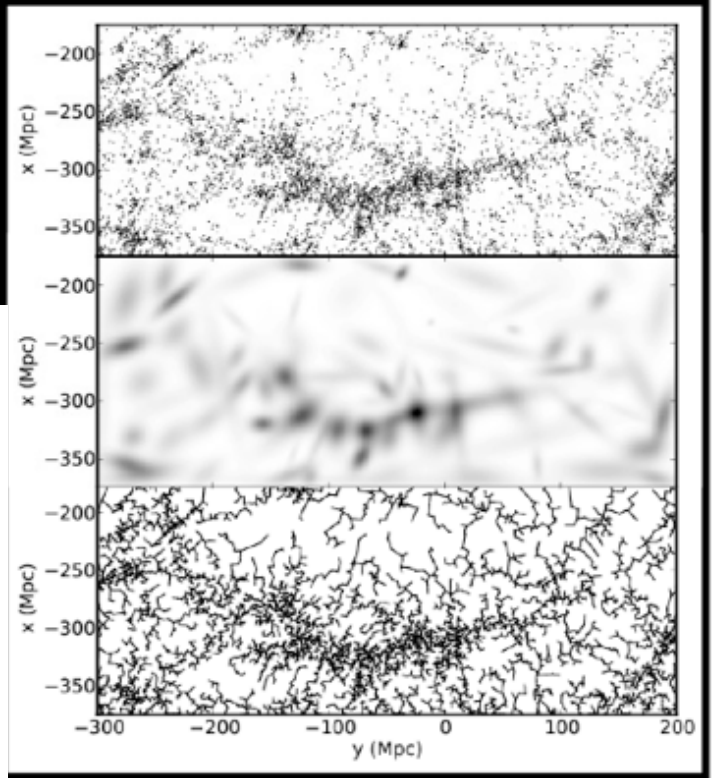
Lecture 2: Density Estimation and Dimensionality Reduction

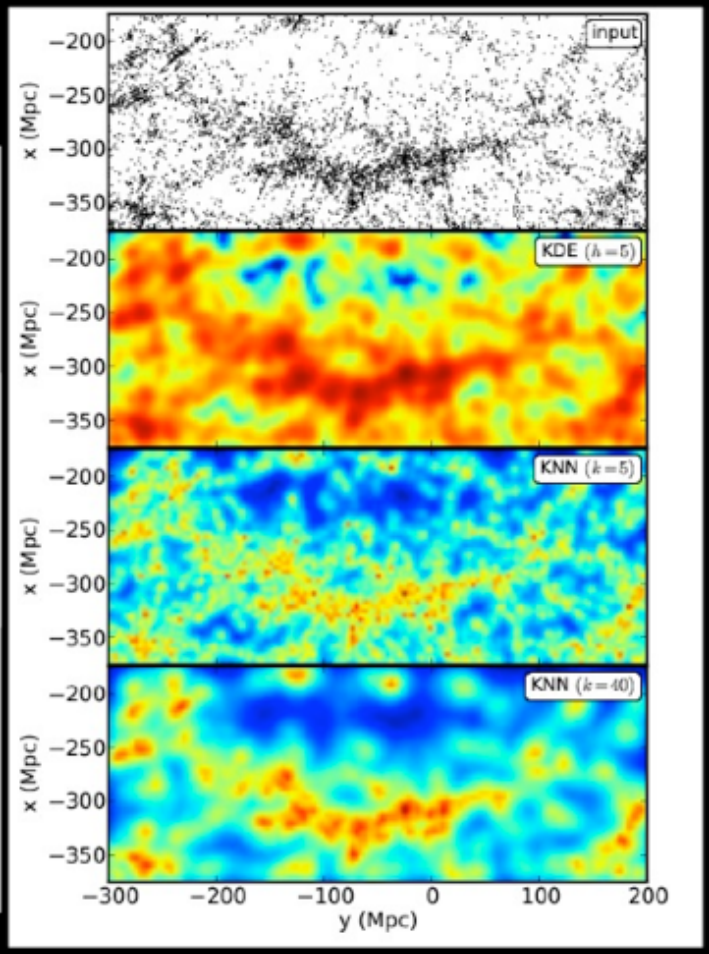
Željko Ivezić, University of Washington





Clustering and Density Estimation: SDSS Great Wall





Outline 1

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

Outline 2

- Dimensionality Reduction
  - Principal Component Analysis
  - Non-negative Matrix Factorization
  - Independent Component Analysis
  - Manifold learning (Locally Linear Embedding)
- (Regression)
  - (Gaussian) errors in both variables
  - regression with non-Gaussian errors and/or outliers
  - learning curves

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, $k = 1, \ldots, M$. 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

$$f_k = \frac{n_k}{\Delta_b N}. \tag{4.80}$$

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

$$\sigma_k = \frac{\sqrt{n_k}}{\Delta_b N}. \tag{4.81}$$

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

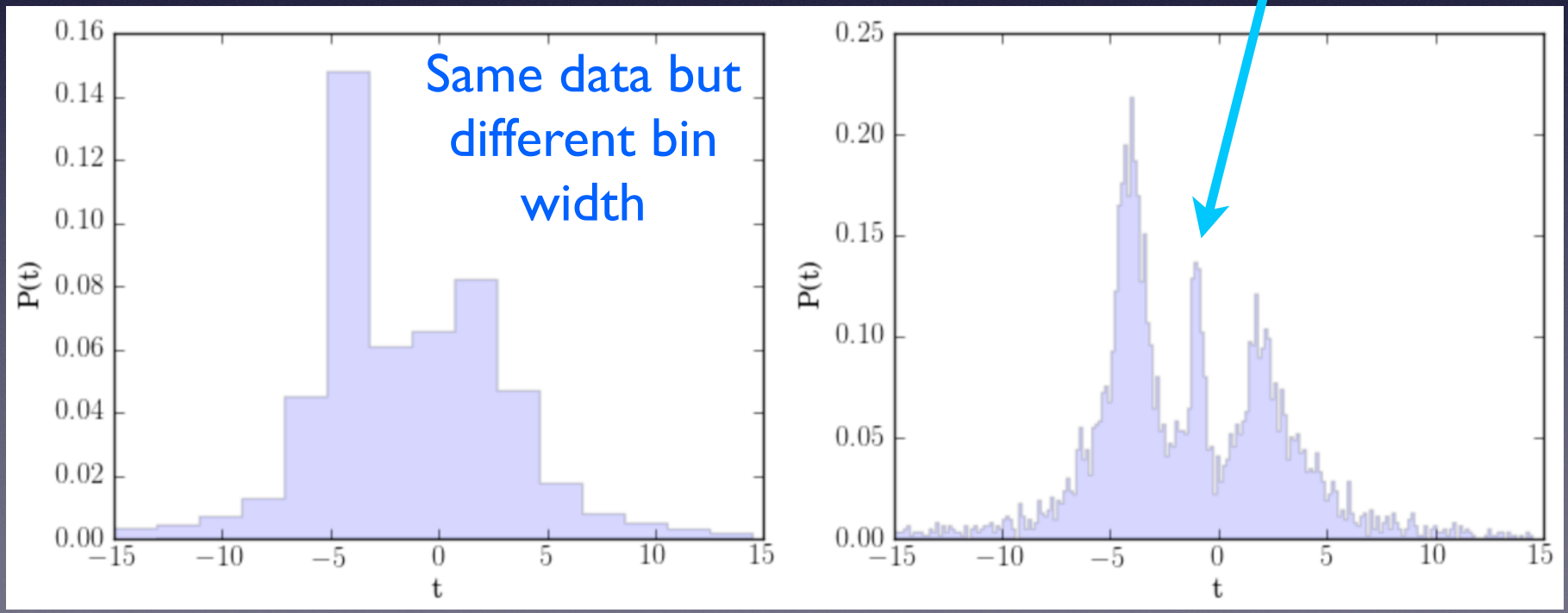
What is a histogram?

- ---> 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?

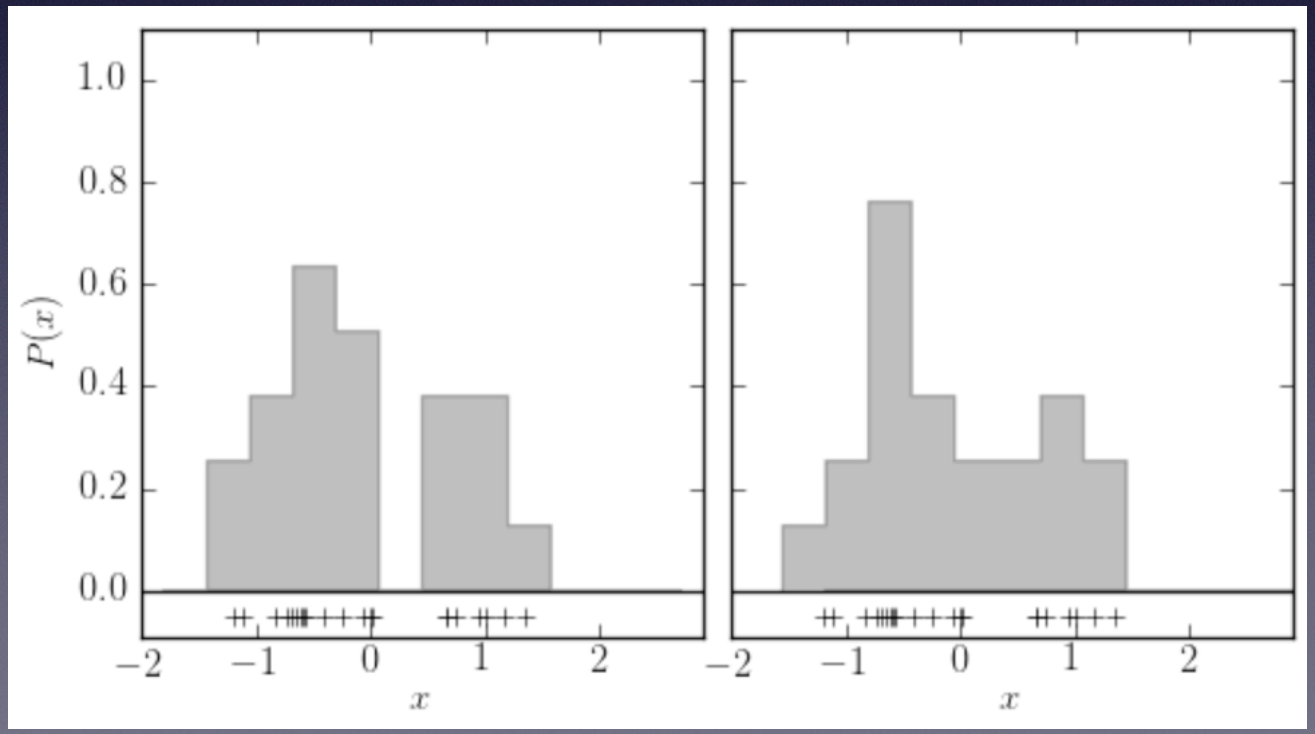
- ---> Data modeled by a step function
- How do we determine/estimate/guess the bin width?
- Do all the bins have to have the same width?
- Do we really have to bin data to estimate model parameters?

Should we believe the middle peak?



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?



Despite the same bin width, a small offset in bin placement can give very different impressions about data behavior

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

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

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,

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

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):

data

$$F(M|\{x_i\},I)) = N\,\log M + \log\left[\Gamma\left(\frac{M}{2}\right)\right] - M\log\left[\Gamma\left(\frac{1}{2}\right)\right] - \log\left[\Gamma\left(N+\frac{M}{2}\right)\right] + \sum_{k=1}^{M}\log\left[\Gamma\left(n_k+\frac{1}{2}\right)\right], \quad (5.107)$$

This is Bayesian model selection of M models

where Γ is the gamma function, and n_k is the number of measurements x_i, $i = 1,\ldots,N$, which are found in bin k, $k = 1,\ldots,M$. 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 $x_{k-1} < x \le x_k$, and 0 otherwise. The M model parameters, h_k, $k = 1,\ldots,M$, are subject to normalization constraints, so that there are only $M-1$ free parameters. The uninformative prior distribution for $\{h_k\}$ is given by

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

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

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

Knuth's rule for estimating bin width

The posterior pdf for model parameters h_k is obtained by multiplying the prior and data likelihood. The posterior probability for the number of bins M is obtained by marginalizing the posterior pdf over all h_k. This marginalization includes a series of nested integrals over the $(M-1)$-dimensional parameter space, and yields eq. 5.107; details can be found in Knuth's paper.

Knuth also derived the posterior pdf for h_k, and summarized it by deriving its expectation value and variance. The expectation value is

Classic result →

$$h_k = \frac{n_k + \frac{1}{2}}{N + \frac{M}{2}}, \qquad (5.111)$$

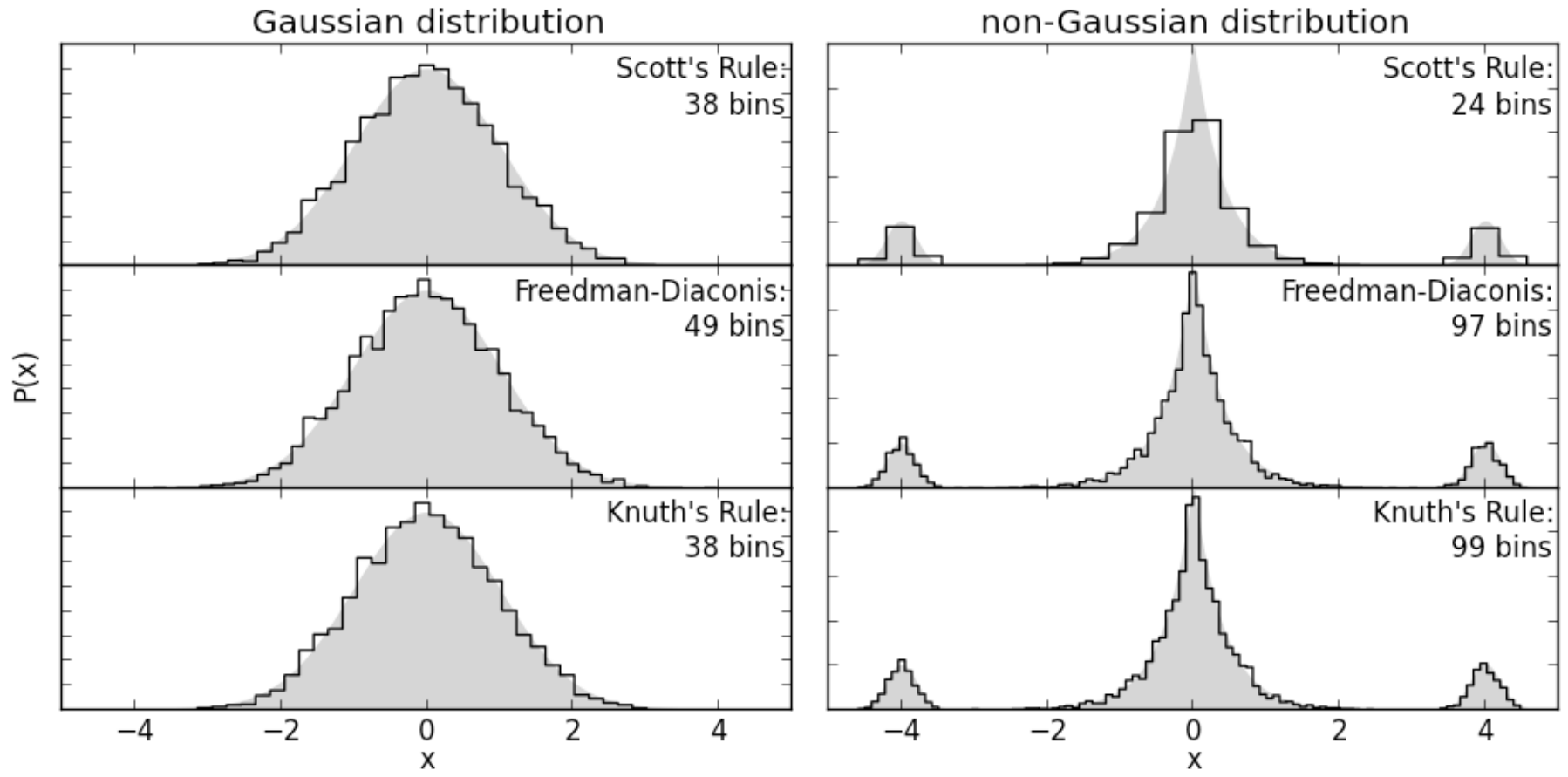
h_k is not 0 even when n_k=0 !

The usefulness of Knuth's analysis and the result summarized by eq. 5.107 goes beyond finding the optimal bin size. The method is capable of recognizing substructure in data and, for example, it results in $M = 1$ when the data are consistent with a uniform distribution, and suggests more bins for a multimodal distribution than for a unimodal distribution even when both samples have the same size and σ_G (again, eq. 5.112 is an approximation valid only for unimodal centrally concentrated distributions; if in doubt, use eq. 5.107; for the latter, see the Python code used to generate figure 5.20).

Lastly, remember that Knuth's derivation assumed that the uncertainty of each x_i is negligible.

Comparing simple rules and Knuth's rule

- make this plot by running
  %run fig_hist_binsize.py



- the work horse code lives in
  $astroMLdir/astroML/density_estimation/hist_tools.py

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:

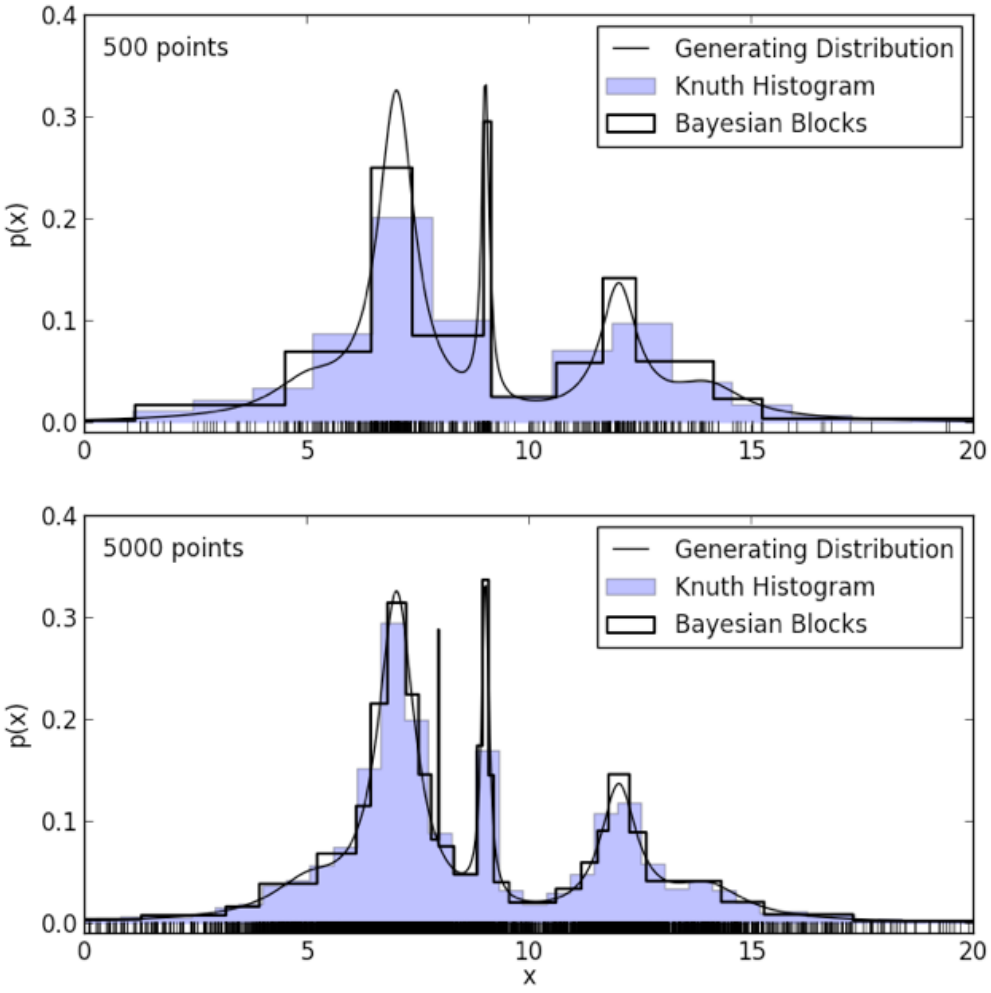
$$F(N_i, T_i) = N_i(\log N_i - \log T_i), \tag{5.113}$$

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

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

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?

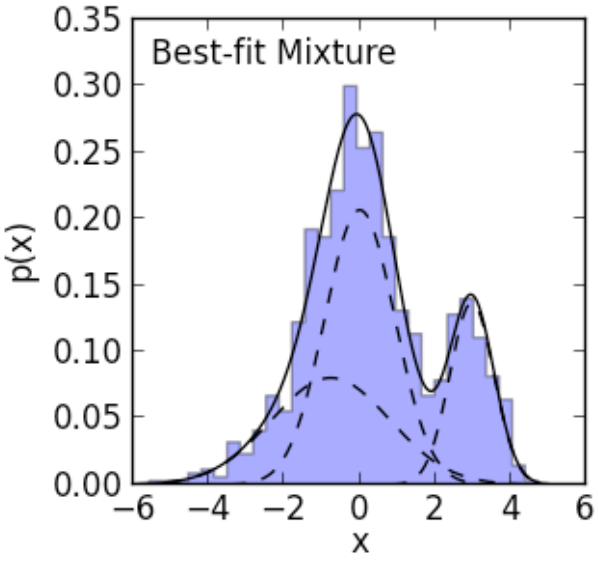
We have assumed in the above discussion of the EM algorithm that the number of classes in a mixture, M, is known. As M is increased, the description of the data set $\{x_i\}$ using a mixture model will steadily improve. On the other hand, a very large M is undesired—after all, $M = N$ will assign a mixture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

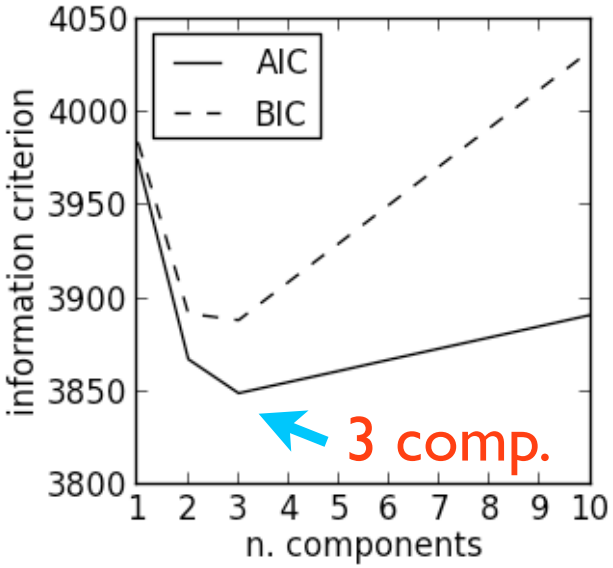
Usually determined using Bayesian Information Criterion (**BIC**), or **cross-validation**

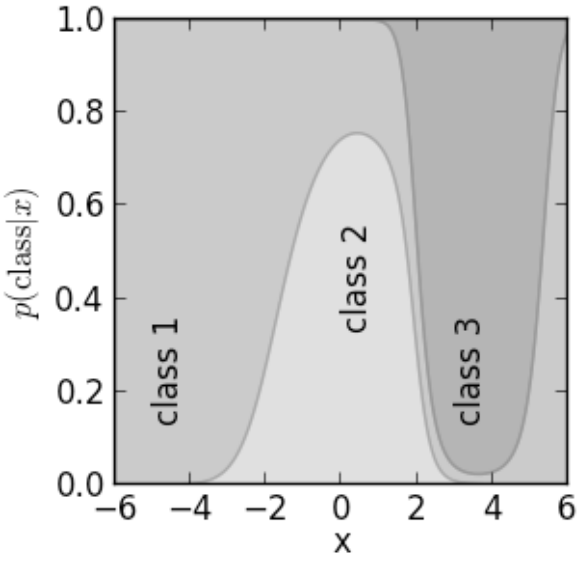
Gaussian Mixture Models

- 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

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 homescedastic 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...

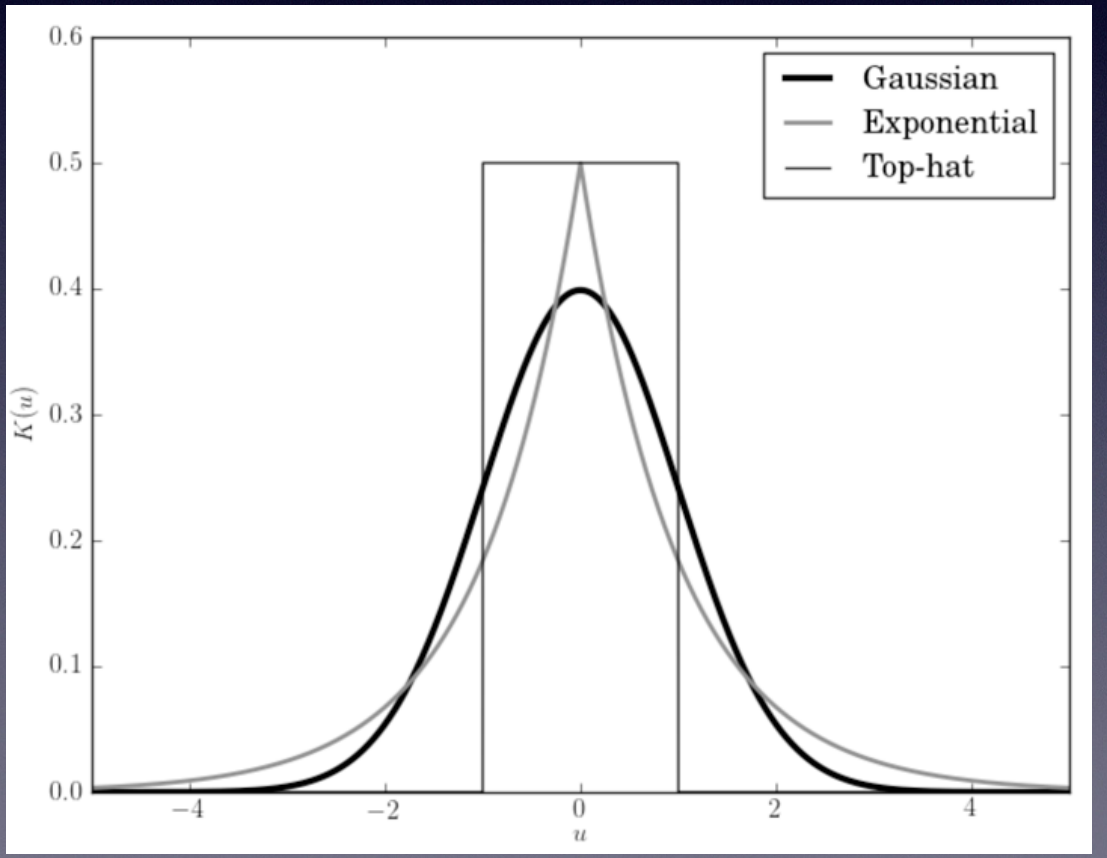
Kernel Density Estimates (KDE)

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.

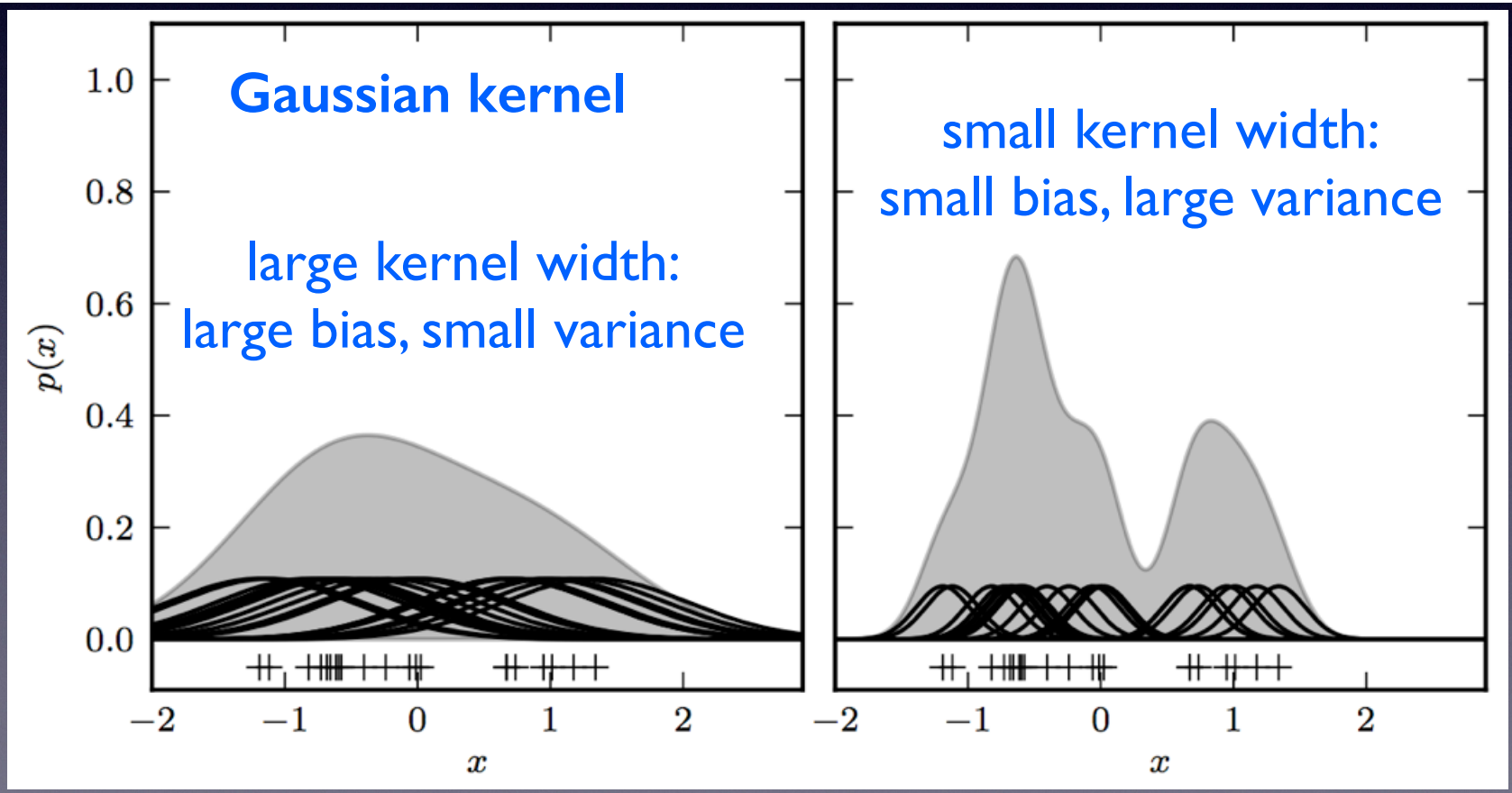


Kernel Density Estimates (KDE)

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

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

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., $i = 1, \ldots, N$) of the log of the likelihood of the density, where the density, $\widehat{f}_{h,-i}(x_i)$, is estimated leaving out the ith data point. This can be written as

$$\mathrm{CV}_l(h) = \frac{1}{N}\sum_{i=1}^{N} \log \widehat{f}_{h,-i}(x_i), \tag{6.5}$$

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
```

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).

What to do when data have uncertainties?

Deconvolution KDE (Stefanski & Raymond 1990):

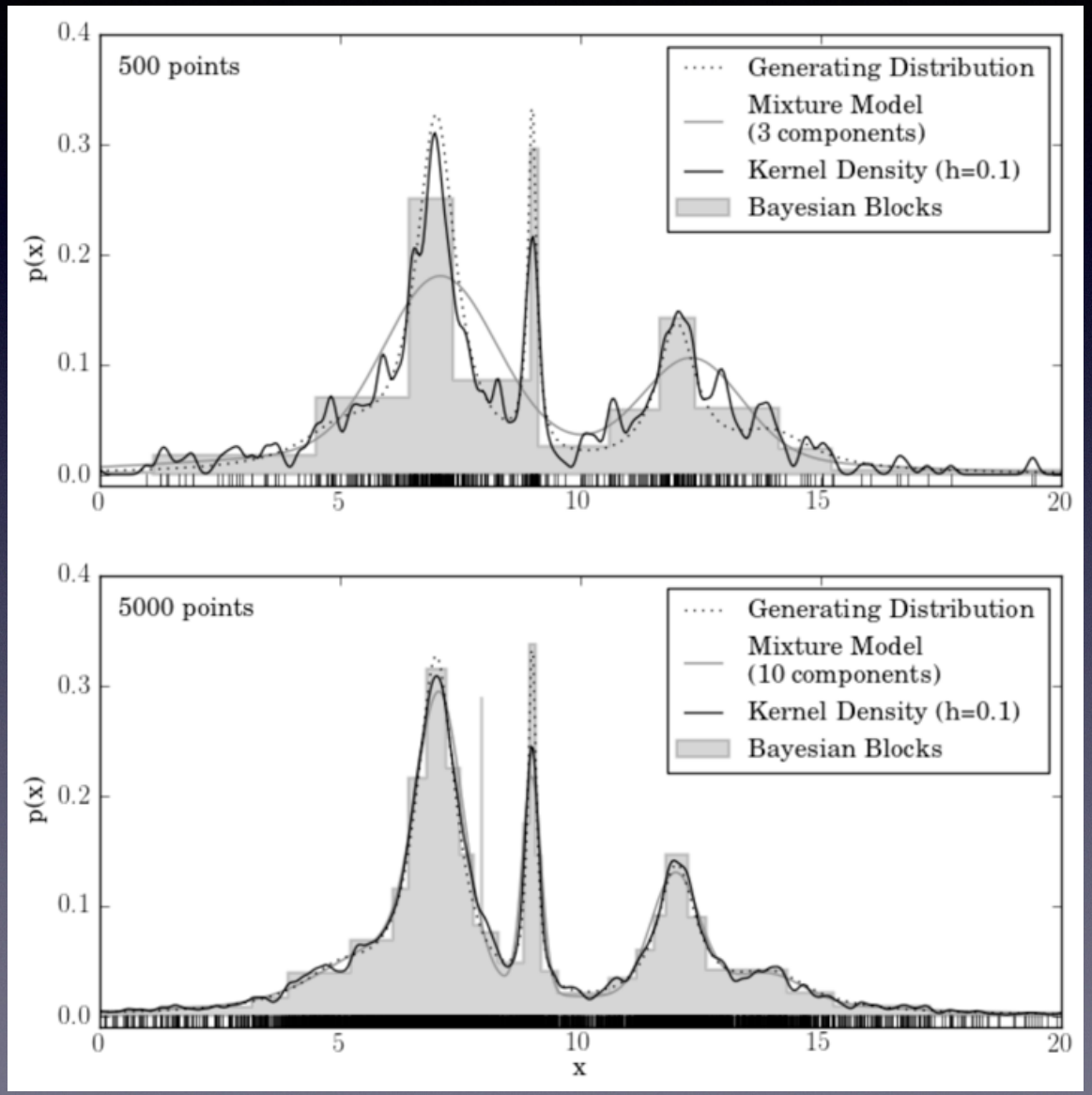
1. Find the kernel density estimate of the observed data, $f(x)$, and compute the Fourier transform $F(k)$.

2. Compute the Fourier transform $G(k)$ of the noise distribution $g(x)$.

3. From eq. 6.10 and the convolution theorem, the Fourier transform of the true distribution $h(x)$ is given by $H(k) = F(k)/G(k)$. The underlying noise-free pdf $h(x)$ can be computed via the inverse Fourier transform of $H(k)$.

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~8 for the BB algorithm in the bottom panel

this was a bug (fixed in latest astroML)

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

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

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:

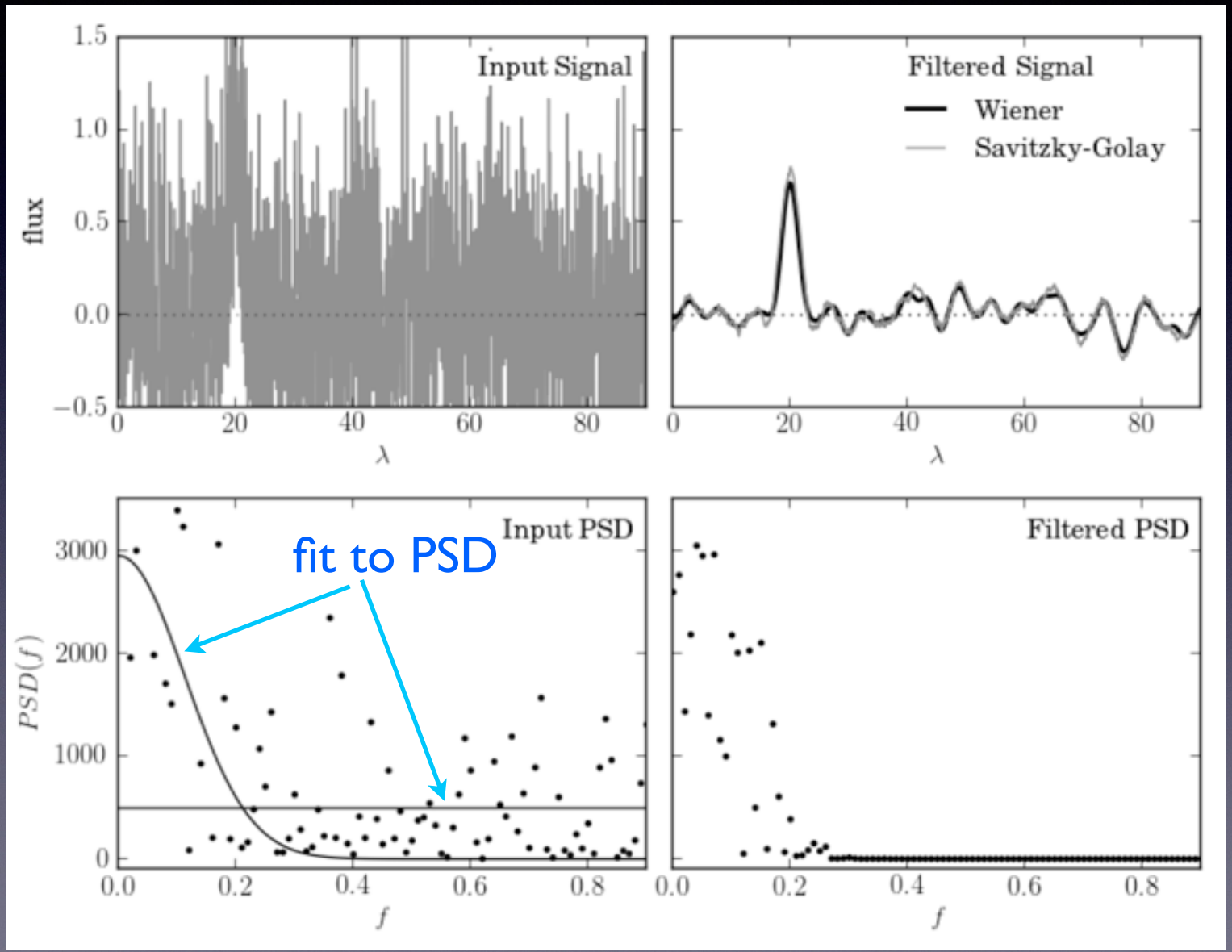
$$\Phi(f) = \frac{P_S(f)}{P_S(f) + P_N(f)}. \tag{10.20}$$

Here $P_S(f)$ and $P_N(f)$ represent components of a two-component (signal and noise) fit to the PSD of input data, $\mathrm{PSD}_Y(f) = P_S(f) + P_N(f)$, 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

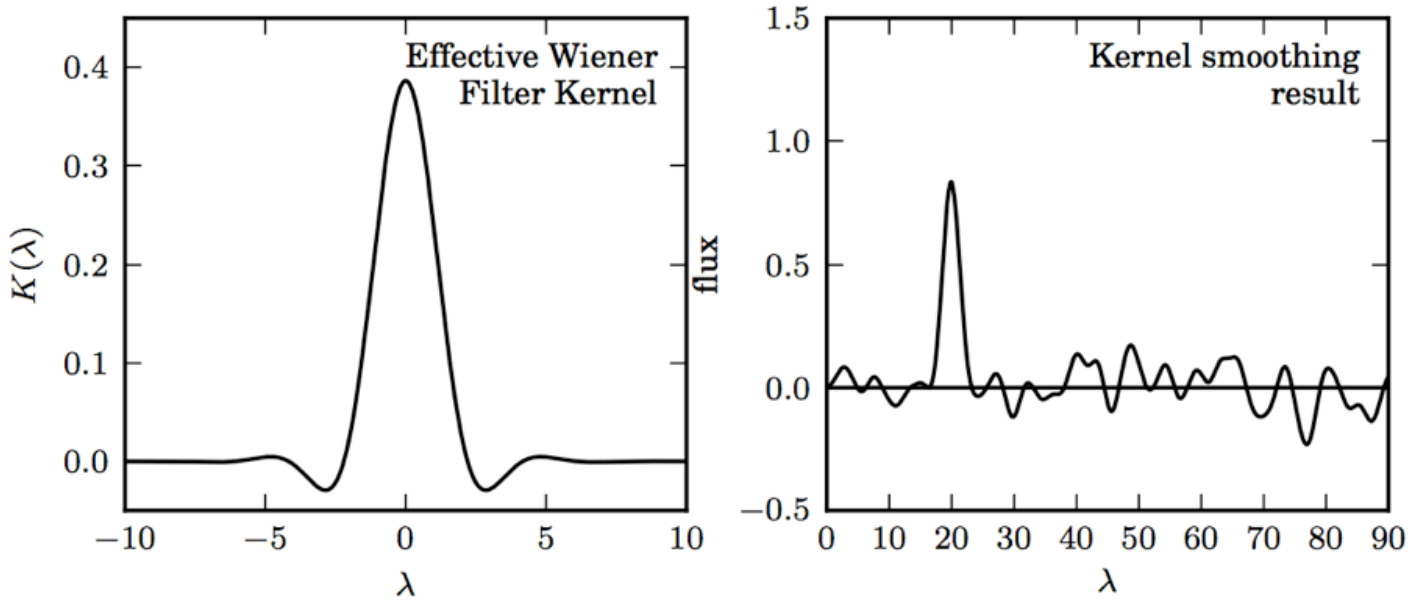


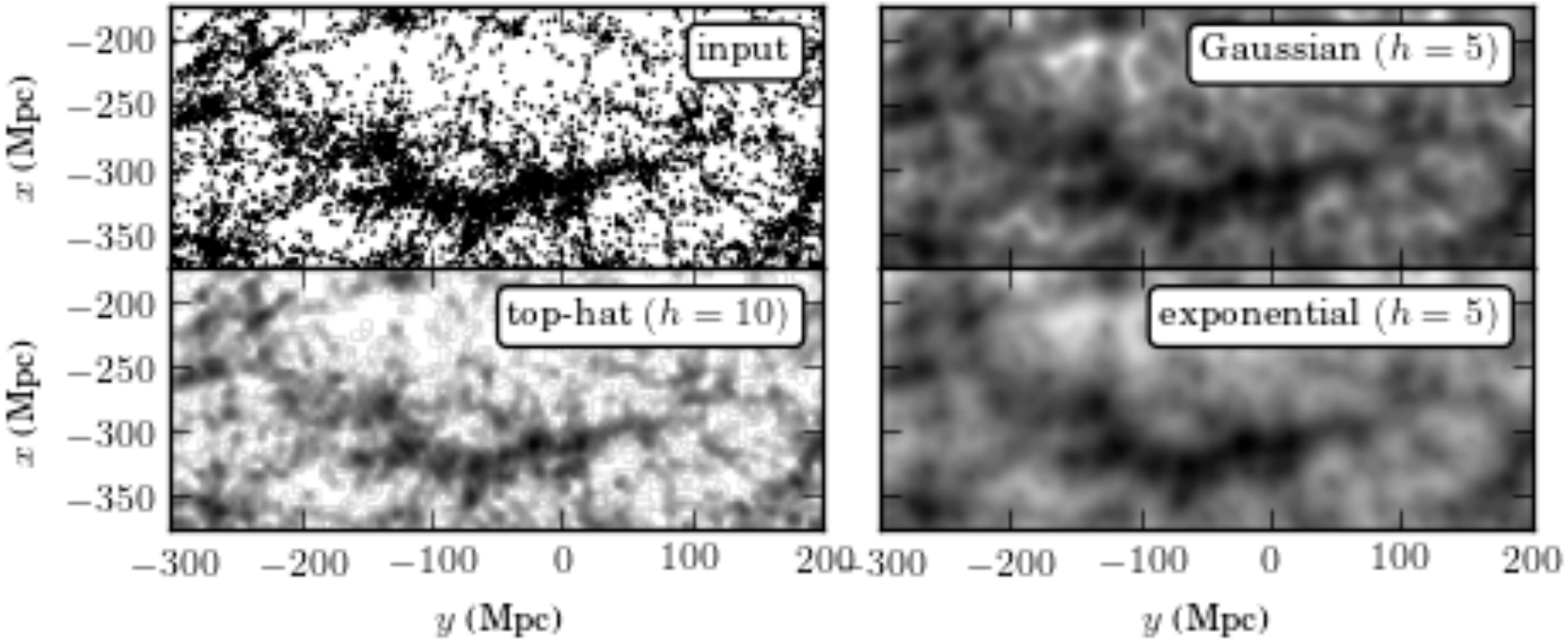
Figure 10.11.: The left panel shows the inverse Fourier transform of the Wiener filter $\Phi(f)$ applied in figure 10.10. By the convolution theorem, the Wiener-filtered result is equivalent to the convolution of the unfiltered signal with the kernel shown above, and thus Wiener filtering and kernel 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!

KDE in 2-D case

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

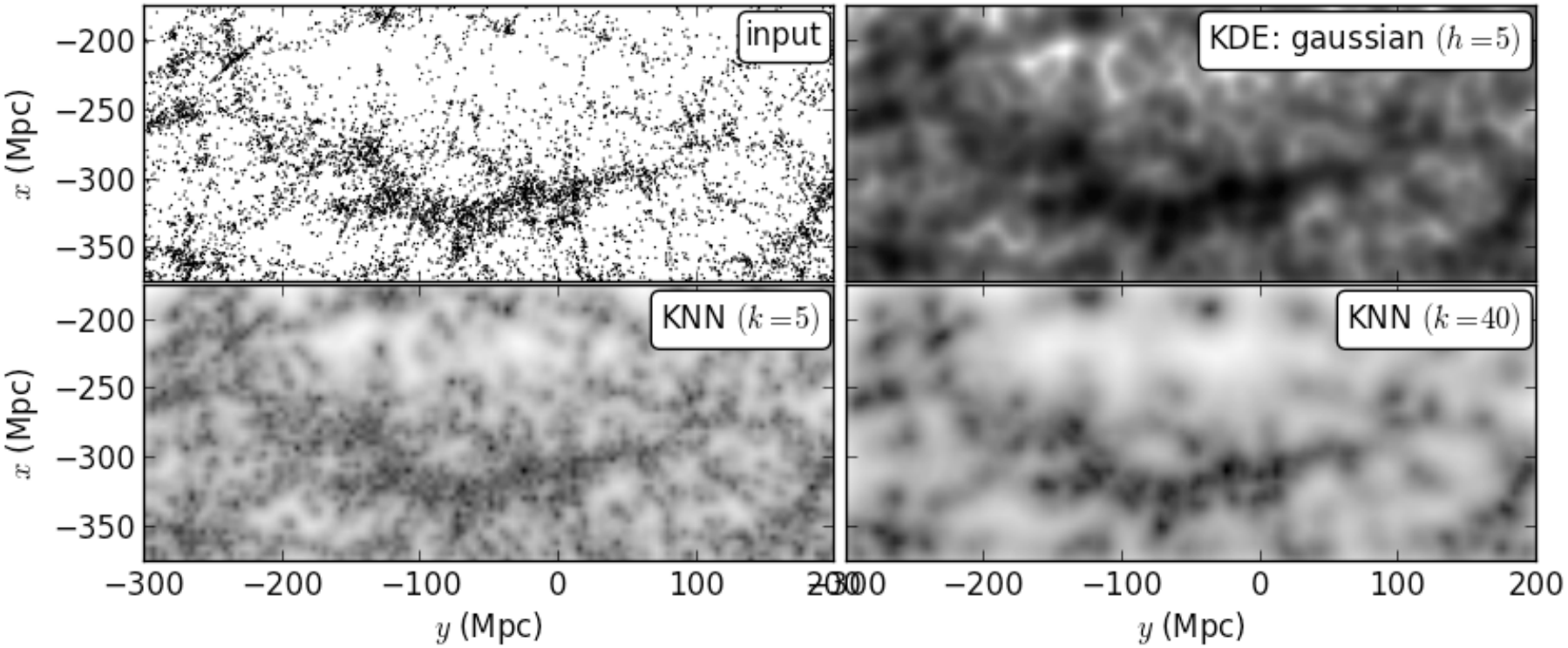
SDSS Great Wall



"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 Kth-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

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

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

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

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

Bayesian K Nearest Neighbors method

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

and evaluate the scaling factor C at the end by requiring that the sum of the product of $\widehat{f}_K(x)$ and pixel volume is equal to the total number of data points. The error in $\widehat{f}_K(x)$ is $\sigma_f = K^{1/2}/V_D(d_K)$, and the fractional (relative) error is $\sigma_f/\widehat{f} = 1/K^{1/2}$. 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 $\widehat{f}$ can be decreased without a degradation in the spatial resolution, or alternatively the resolution can be increased without increasing the error in $\widehat{f}$) by considering distances to *all* K nearest neighbors instead of only the distance to the Kth-nearest neighbor; see [18]. Given distances to all K neighbors, d_i, $i = 1, \ldots, K$,

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

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

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

This method is very good for sparsely sampled data, and better than e.g. KDE with a Gaussian kernel for K~10 (details: Ivezic+ 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:**

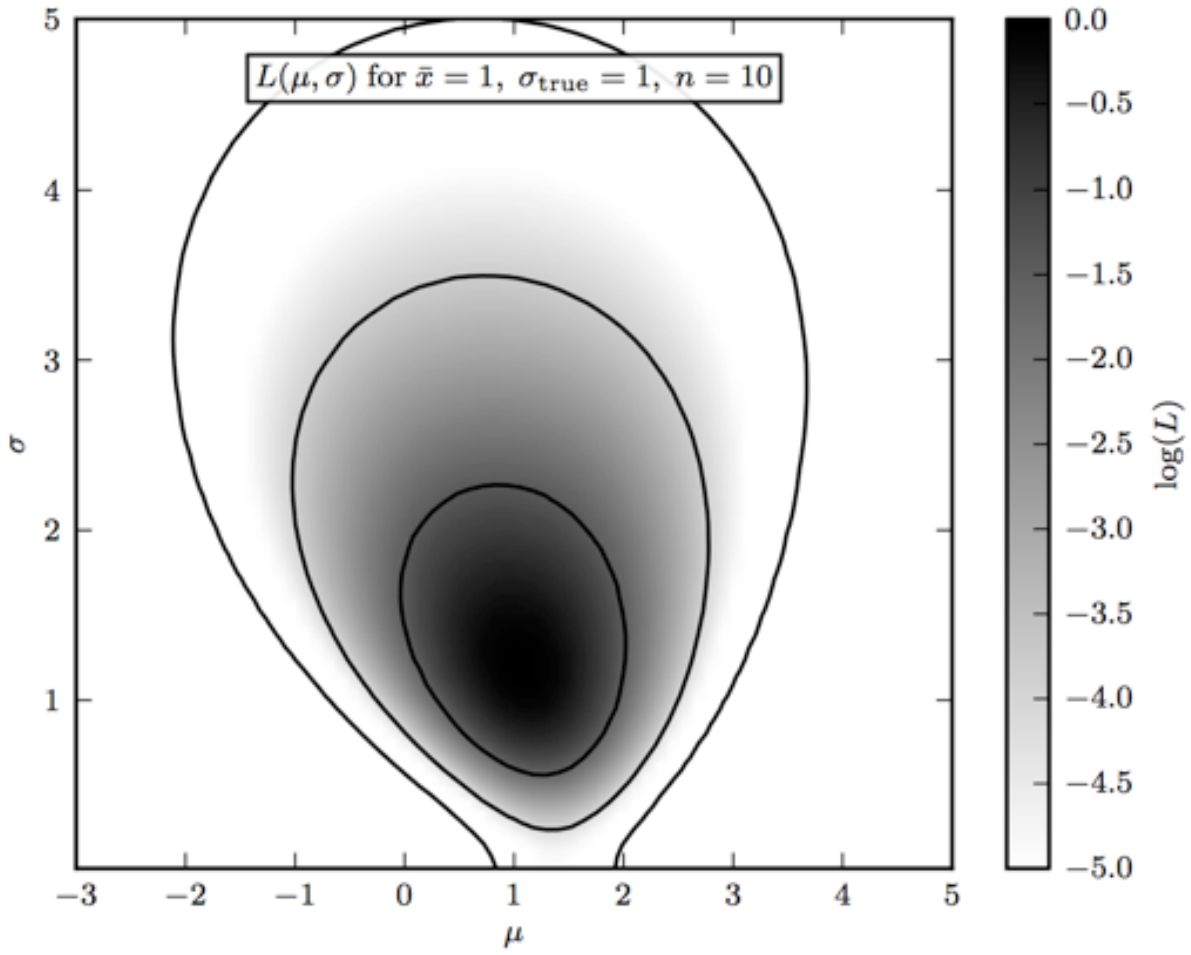


Figure 5.7.: The logarithm of the posterior probability density function for μ and σ, $L_p(\mu, \sigma)$, for a Gaussian distribution with heteroscedastic Gaussian measurement errors (sampled uniformly from the 0–3 interval), given by eq. 5.64. The input values are $\mu = 1$ and $\sigma = 1$, and a randomly 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 straighforward 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),

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

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

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:

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

where θ_j now corresponds to μ_j or σ_j. By setting the derivatives of lnL 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)

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

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

and from the normalization constraint,

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

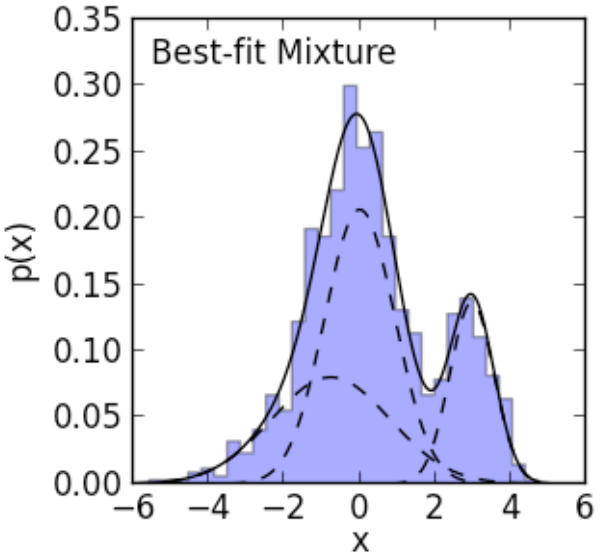
$$\alpha_j = \frac{1}{N} \sum_{i=1}^{N} w_{ij}. \tag{4.28}$$

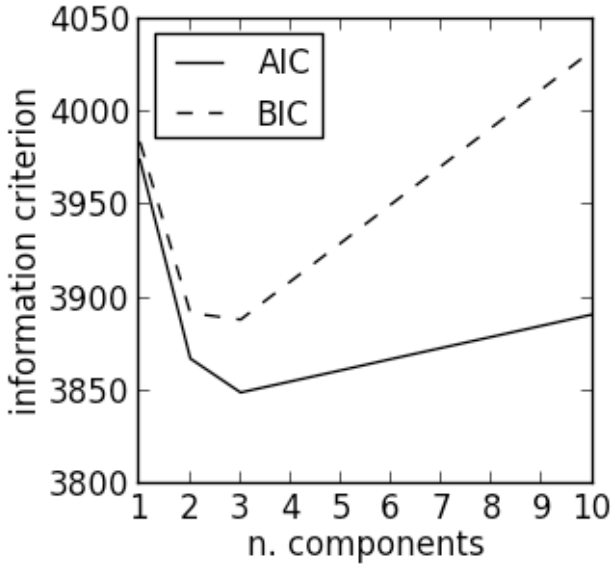
These expressions and eq. 4.21 form the basis of the iterative EM algorithm in the case of Gaussian mixtures. Starting with a guess for w_{ij}, the values of α_j, μ_j, and σ_j are estimated using eqs. 4.26–4.28. This is the "maximization" *M-step* which brings the parameters closer toward the local maximum. 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).

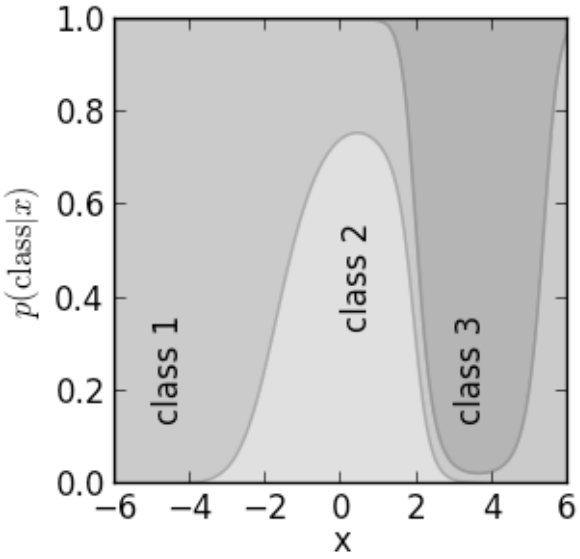
Expectation Maximization Algorithm

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

$$p(\mathbf{x}) = \sum_j \alpha_j \mathcal{N}(\mathbf{x}|\mu_j, \mathbf{\Sigma}_j)\,, \tag{6.18}$$

where, recalling eq. 3.97,

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

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

$$\mathbf{x}_i = \mathbf{R}_i \mathbf{v}_i + \epsilon_i, \tag{6.20}$$

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)

- **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

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

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

$$p(\mathbf{x}) = \sum_j \alpha_j \mathcal{N}(\mathbf{x}|\mu_j, \mathbf{\Sigma}_j)\,, \tag{6.18}$$

where, recalling eq. 3.97,

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

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

$$\mathbf{x}_i = \mathbf{R}_i \mathbf{v}_i + \epsilon_i, \tag{6.20}$$

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

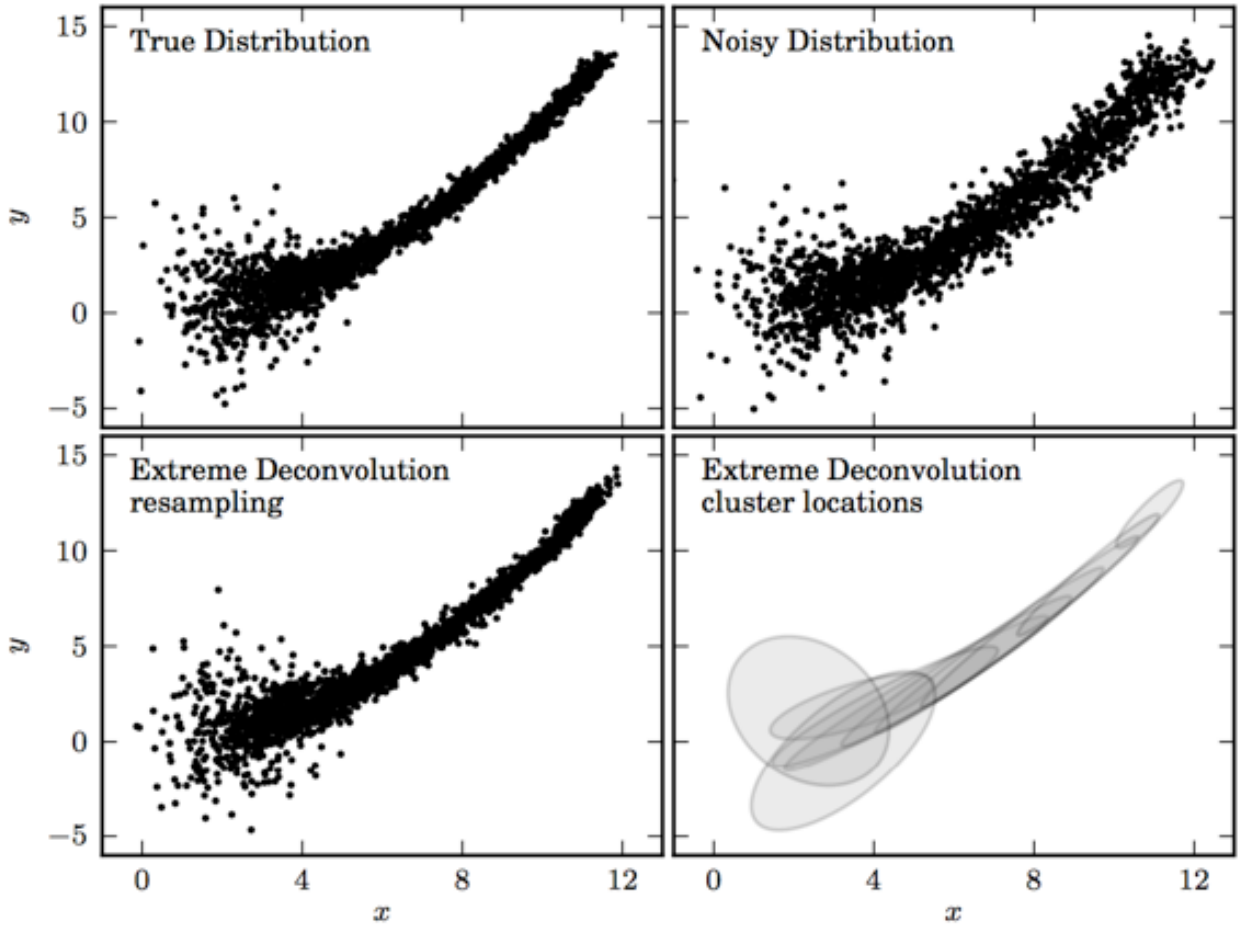
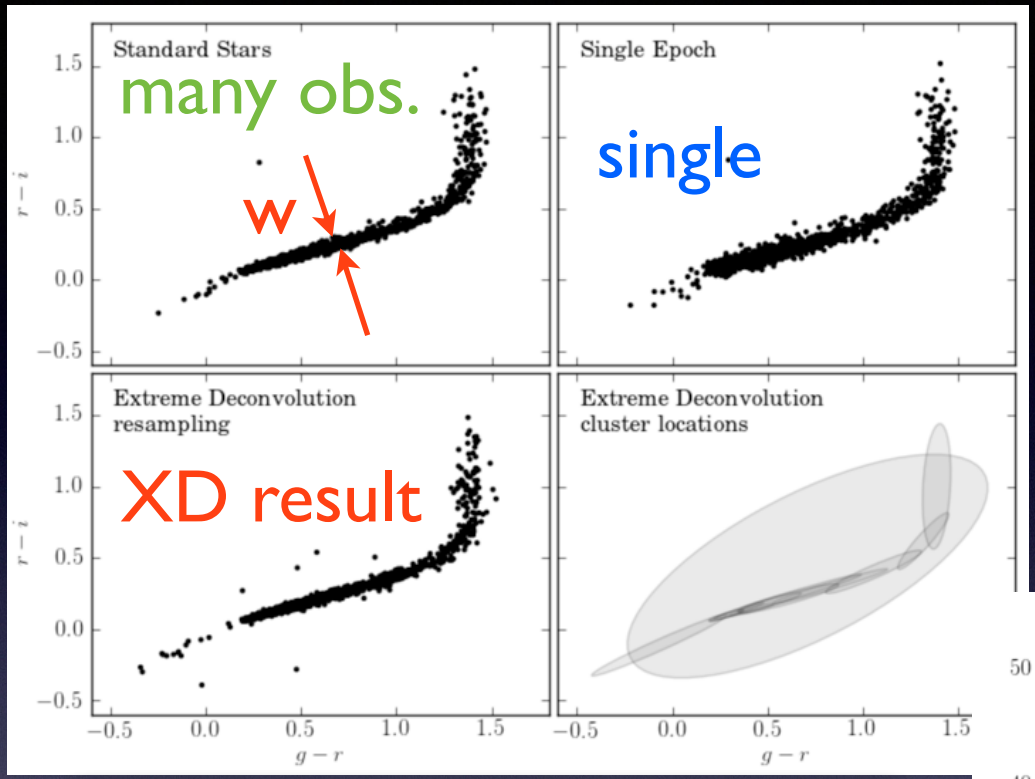
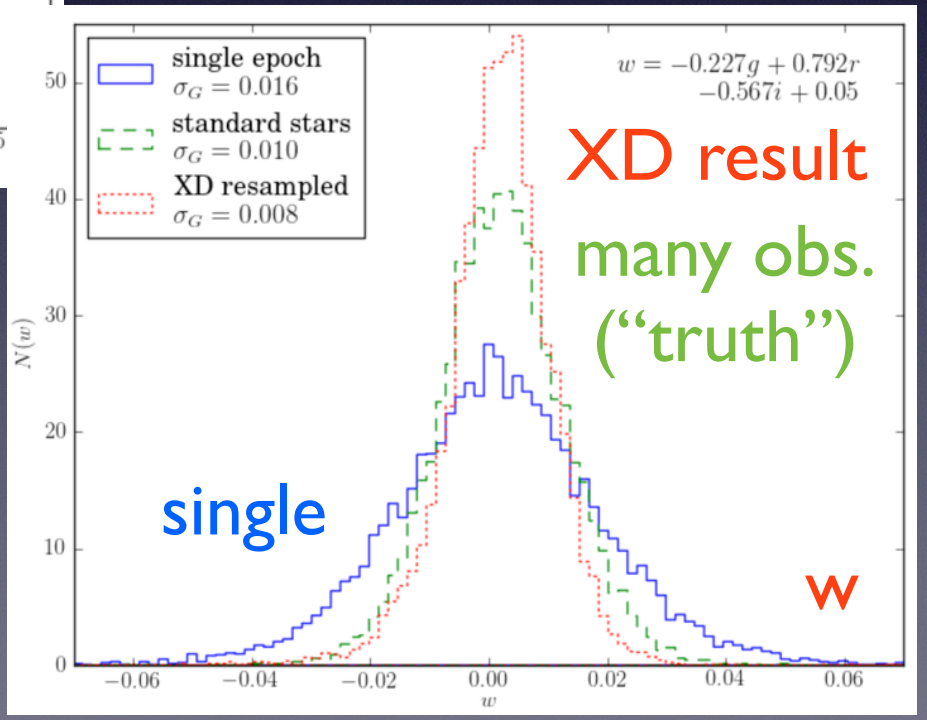


Figure 6.11.: An example of extreme deconvolution showing a simulated two-dimensional distribution of points, where the positions are subject to errors. The top two panels show the distributions with small (left) and large (right) errors. The bottom panels show the densities derived from the noisy sample (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 is a very powerful method: if GMM is able to model the observed distribution (usually OK)



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

Outline 2

- Dimensionality Reduction
  - Principal Component Analysis
  - Non-negative Matrix Factorization
  - Independent Component Analysis
  - Manifold learning (Locally Linear Embedding)
- (Regression)
  - (Gaussian) errors in both variables
  - regression with non-Gaussian errors and/or outliers
  - learning curves

Principal Component Analysis (PCA)

Motivation:

2D case: what is the direction of largest variance in the data? Equivalently, what is the rotation of the coordinate system in which results in no covariance between the variables?

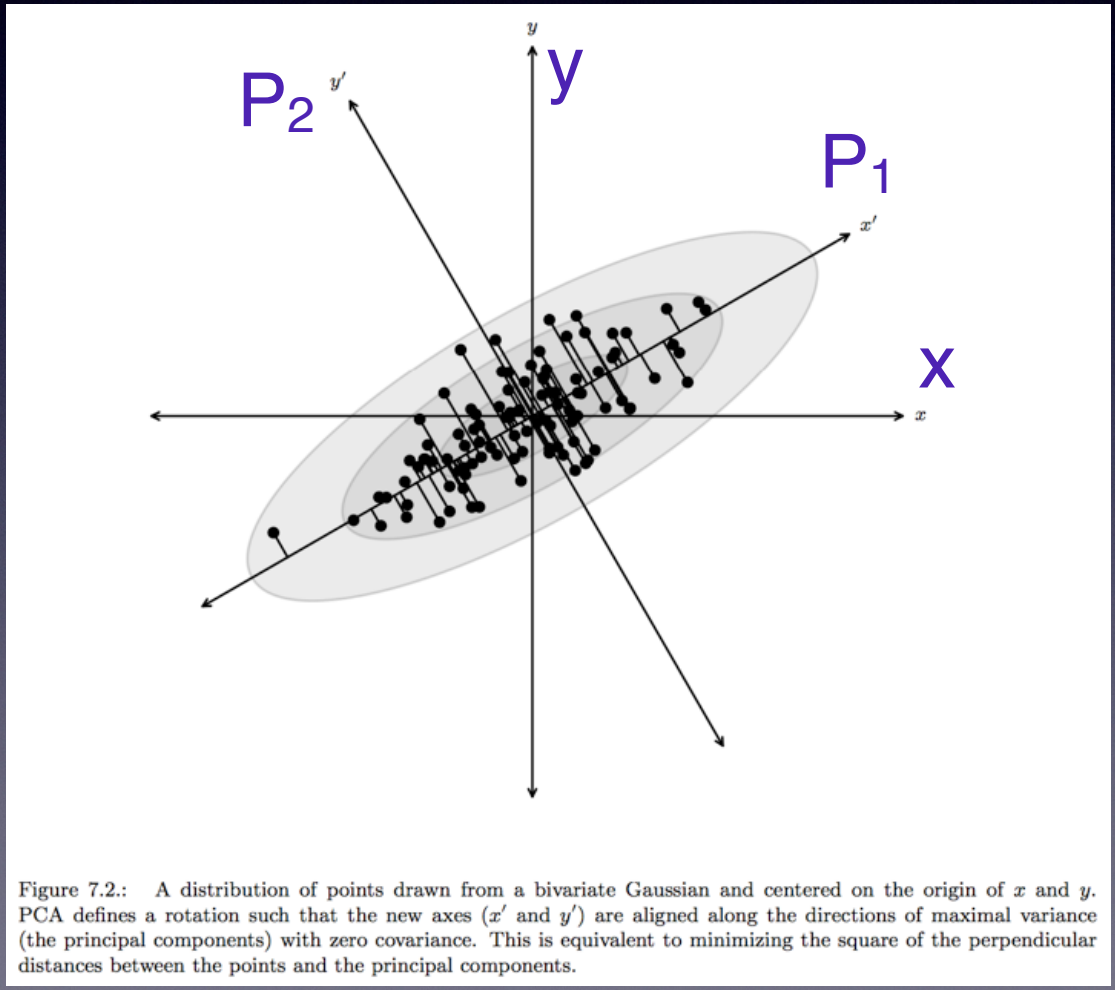


Figure 7.2.: A distribution of points drawn from a bivariate Gaussian and centered on the origin of x and y. PCA defines a rotation such that the new axes (x' and y') are aligned along the directions of maximal variance (the principal components) with zero covariance. This is equivalent to minimizing the square of the perpendicular distances between the points and the principal components.

The two variables x and y have a non-vanishing covariance; the covariance in the P_1 vs. P_2 coordinate system is 0 by construction (recall discussion of 2-D Gaussian in Lecture 1)

Principal Component Analysis (PCA)

Motivation:

High-D case (driven by the "curse of dimensionality"): there are ~4000 data points in SDSS spectra. Can we represent these spectra as linear combinations of **much smaller** number of eigen-components?

The curse of dimensionality:
in high-D space, the ratio of the volume of the unit hyper-sphere and the volume of the hyper-cube that encloses it goes to 0 with as D increases

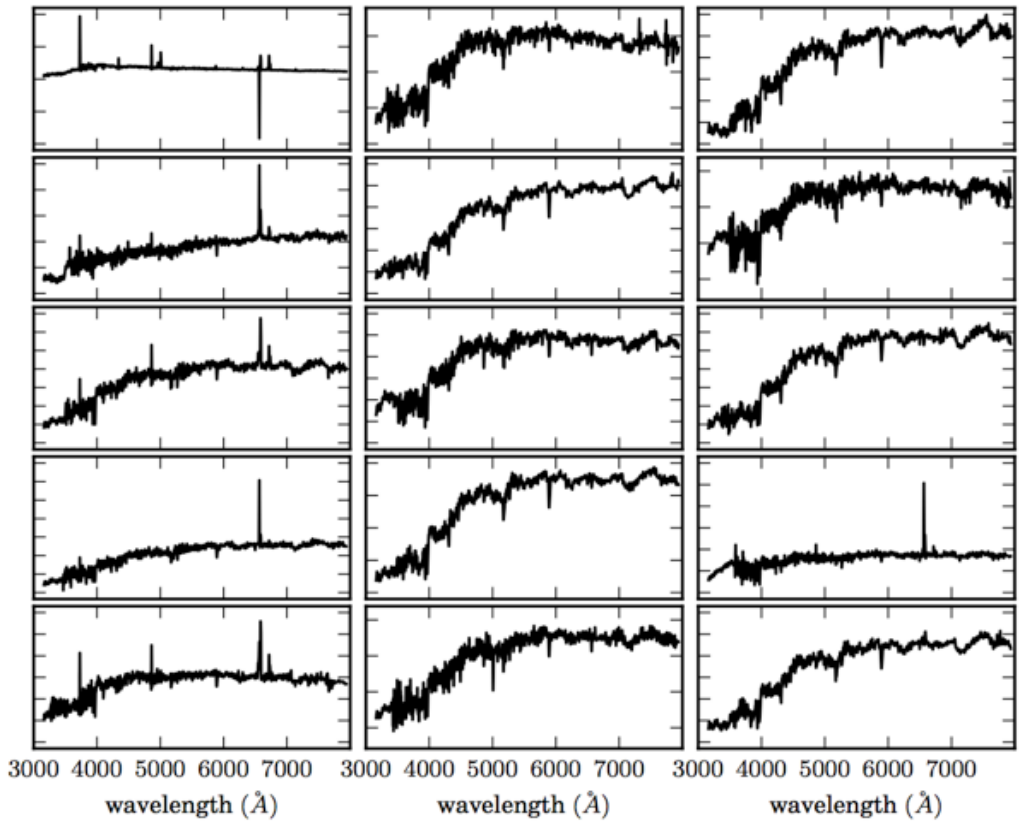


Figure 7.1.: A sample of 15 galaxy spectra selected from the SDSS spectroscopic data set (see §1.5.5). These spectra span a range of galaxy types, from star-forming to passive galaxies. Each spectrum has been shifted to its rest frame and covers the wavelength interval 3000–8000 Å. The specific fluxes, $F_\lambda(\lambda)$, on the ordinate axes have an arbitrary scaling.

Principal Component Analysis (PCA)

sometimes called Karhunen-Loeve and Hotelling transform

How is it done:

Data matrix if formed by N sets (objects) of K measured features (attributes); e.g. K~4000 for SDSS spectra of, say, N=100,000 quasars; we subtract the mean of each feature (and sometimes normalized by their st. deviation, called whitening; in case of spectra, flux is normalized to 1 to avoid uninteresting correlations with source brightness)

Using matrix algebra, the first eigen-component is found by maximizing variance and other eigen-components by requiring covariance to vanish.

There are different approaches, whose performance depends on N and K.

Principal Component Analysis (PCA)

sometimes called Karhunen-Loeve and Hotelling transform

How is it done:

Different matrix algebra approaches to PCA:

1) Singular Value Decomposition of the data matrix:
efficient in most practical cases (exceptions below)

2) Eigenvalue Decomposition of the covariance matrix:
efficient for $N >> K$

3) Eigenvalue Decomposition of the correlation matrix:
efficient for $K >> N$

Principal Component Analysis (PCA)

How is it done:

PCA can be performed easily using Scikit-learn:

```
import numpy as np
from sklearn.decomposition import PCA

X = np.random.normal(size=(100, 3))  # 100 points in 3 dimensions
R = np.random.random((3, 10))  # projection matrix
X = np.dot(X, R)  # X is now 10-dim, with 5 intrinsic dims
pca = PCA(n_components=4)  # n_components can be optionally set
pca.fit(X)
comp = pca.transform(X)  # compute the subspace projection of X

mean = pca.mean_  # length 10 mean of the data
components = pca.components_  # 4 x 10 matrix of components
var = pca.explained_variance_  # the length 4 array of eigenvalues
```

Principal Component Analysis (PCA)

Each spectrum $\boldsymbol{x}_i(k)$ can be described by

$$\boldsymbol{x}_i(k) = \boldsymbol{\mu}(k) + \sum_j^R \theta_{ij} \boldsymbol{e}_j(k), \tag{7.17}$$

where i represents the number of the input spectrum, j represents the number of the eigenspectrum, and, for the case of a spectrum, k represents the wavelength. Here, $\boldsymbol{\mu}(k)$ is the mean spectrum and θ_{ij} are the linear expansion coefficients derived from

$$\theta_{ij} = \sum_k \boldsymbol{e}_j(k)(\boldsymbol{x}_i(k) - \boldsymbol{\mu}(k)). \tag{7.18}$$

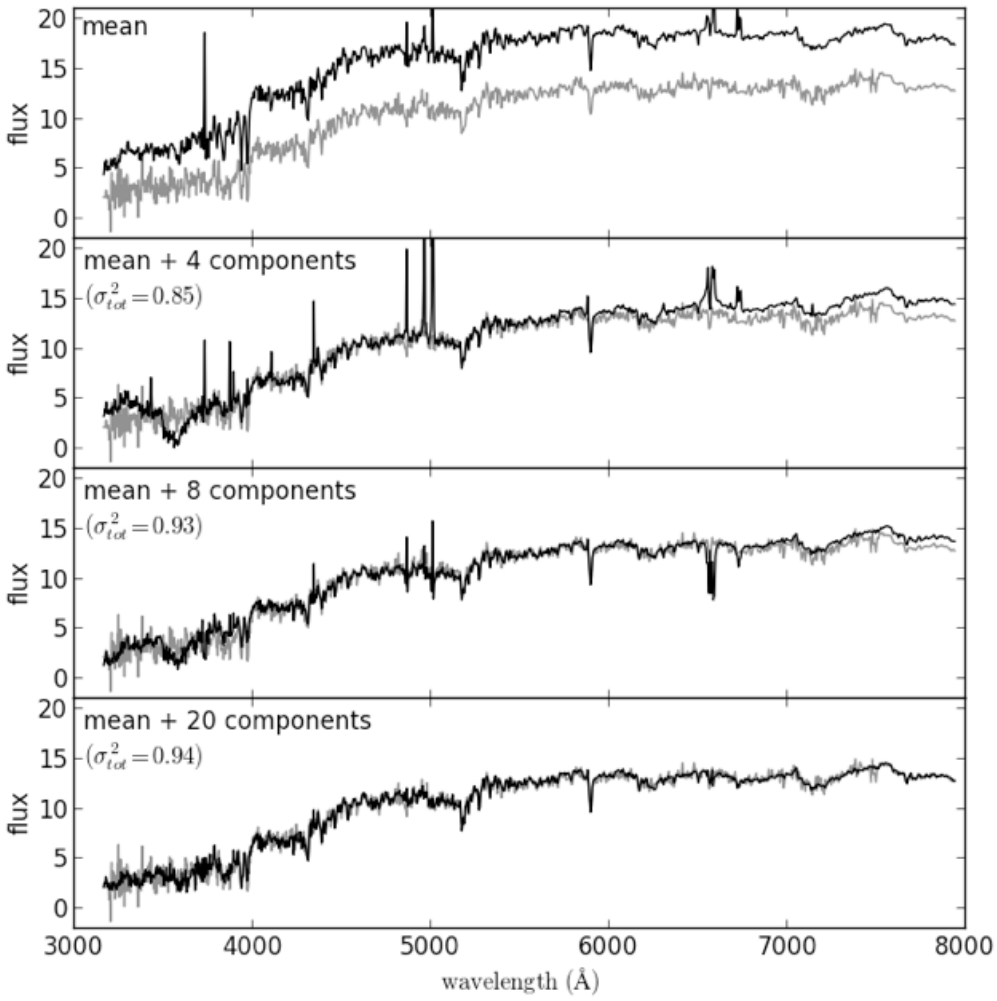
R is the total number of eigenvectors (given by the rank of X, $\min(N,K)$). If the summation is over all eigenvectors, the input spectrum is fully described with no loss of information. Truncating this expansion (i.e., $r < R$),

$$\boldsymbol{x}_i(k) = \sum_i^{r<R} \theta_i \boldsymbol{e}_i(k), \tag{7.19}$$

will exclude those eigencomponents with smaller eigenvalues. These components will, predominantly, reflect the noise within the data set. This is reflected in figure 7.5: truncating the recon-

Principal Component Analysis (PCA)

- make this plot by running
  %run fig_spec_reconstruction.py



How to choose the number of eigencomponents?

Mean spectrum

4 eigencomponents

8 eigencomponents

20 eigencomponents

Principal Component Analysis (PCA)

How to choose the number of eigencomponents?

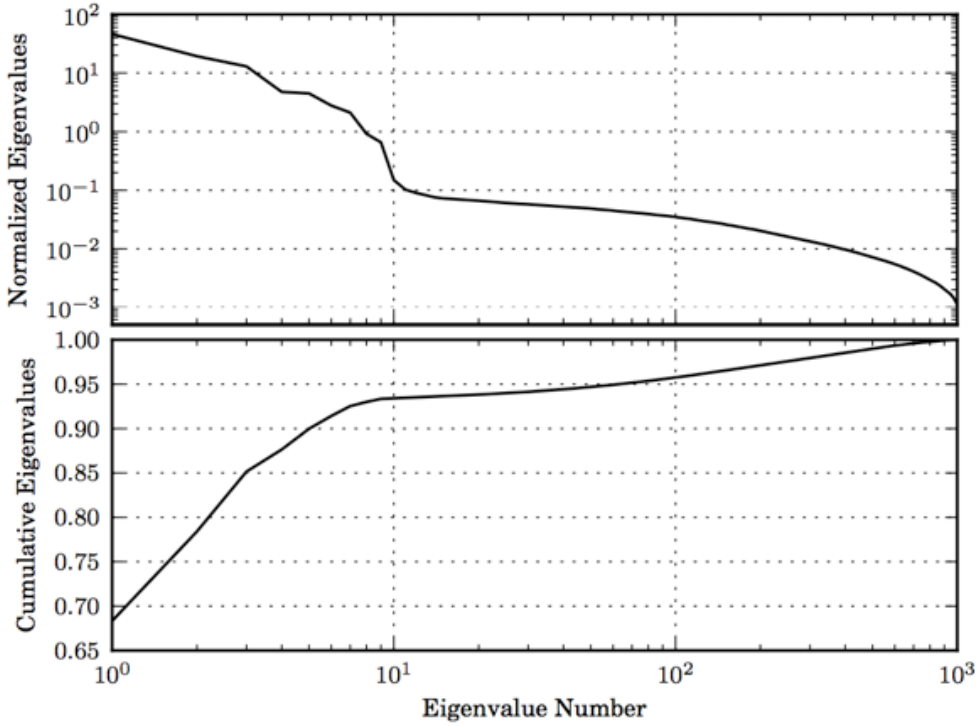


Figure 7.4.: The eigenvalues for the PCA decomposition of the SDSS spectra described in §7.3.2. The top panel shows the decrease in eigenvalue as a function of the number of eigenvectors, with a break in the distribution at ten eigenvectors. The lower panel shows the cumulative sum of eigenvalues normalized to unity. 94% of the variance in the SDSS spectra can be captured using the first ten eigenvectors.

There is no universal method, but typically we require that some fraction of data variance is captured by truncated series.

This plot (called the scree plot) shows that the first ten eigencomponents already capture about 94% of the variance in the dataset. Higher terms attempt to describe high-frequency measurement noise.

Principal Component Analysis (PCA)

A few more points about PCA

It can treat missing data by introducing weights for each point (Connolly & Szalay, 1999, AJ 117, 2052).

If the dataset cannot fit in memory, then PCA computation can be challenging. One way to address this is the use of iterative online algorithms.

In some problems, linear decomposition might not be appropriate (e.g. a set of Planck functions with varying temperature)

Eigencomponents can be negative, which in some applications provides only limited insight into underlying physics (e.g. galaxy spectra).

Non-negative Matrix Factorization (NMF)

Attempts to address the fact that PCA eigencomponents can be negative even when we have physical reasons (a priori knowledge) to expect them to be non-negative

Assumes that the data matrix X is a product of two positive matrices, $X=Y*W$

Solved iteratively for Y and W by minimizing the reconstruction error $||(X-WY)^2||$; sometimes problems with local minima are encountered (Lee & Seung, 2001)

We will see an example after introducing ICA

Independent Component Analysis (ICA)

Also known as the “coctail party problem”, it assumes that the signal is a linear combination of components:

Each spectrum, $x_i(k)$, can now be described by

$$x_1(k) = a_{11}s_1(k) + a_{12}s_2(k) + a_{13}s_3(k) + \cdots, \quad (7.34)$$
$$x_2(k) = a_{21}s_1(k) + a_{22}s_2(k) + a_{23}s_3(k) + \cdots, \quad (7.35)$$
$$x_3(k) = a_{31}s_1(k) + a_{32}s_2(k) + a_{33}s_3(k) + \cdots, \quad (7.36)$$

where $s_i(k)$ are the individual stellar spectra and a_{ij} the appropriate mixing amplitudes. In matrix format we can write this as

$$X = AS, \quad (7.37)$$

where X and S are matrices for the set of input spectra and stellar spectra, respectively. Extracting these signal spectra is equivalent to estimating the appropriate weight matrix, W, such that

$$S = WX. \quad (7.38)$$

This appears very similar to PCA, but...

Independent Component Analysis (ICA)

The principle that underlies ICA comes from the observation that the input signals, $s_i(k)$, should be statistically independent. Two random variables are considered statistically independent if their joint probability distribution, $f(x, y)$, can be fully described by a combination of their marginalized probabilities, that is,

$$f(x^p, y^q) = f(x^p) f(y^q), \tag{7.39}$$

where p and q represent arbitrary higher-order moments of the probability distributions. For the case of PCA, $p = q = 1$ and the statement of independence simplifies to the weaker condition of uncorrelated data (see §7.3.1 on the derivation of PCA).

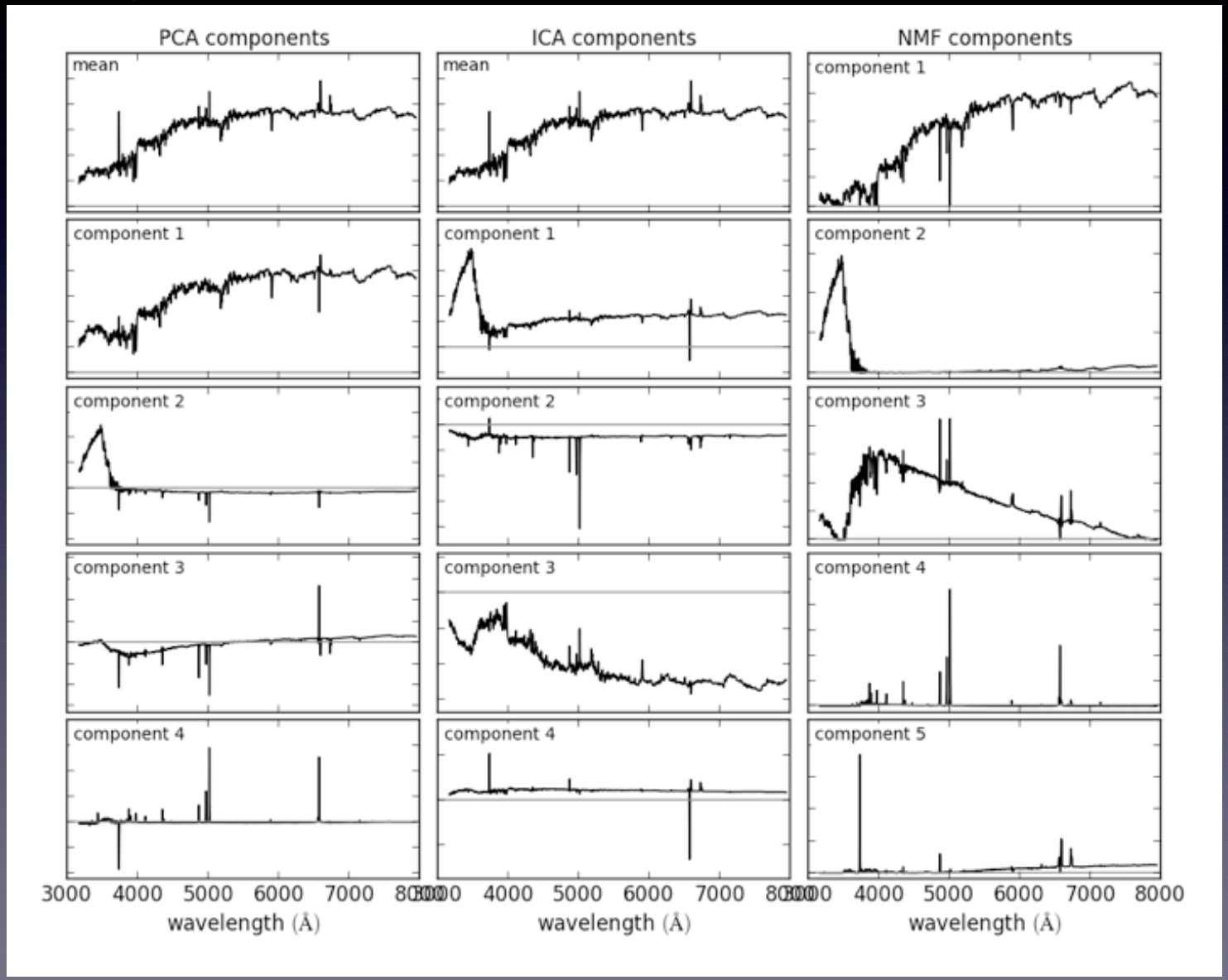
In most implementations of ICA algorithms the requirement for statistical independence is expressed in terms of the non-Gaussianity of the probability distributions. The rationale for this is that the sum of any two independent random variables will always be more Gaussian than either

Let's now compare PCA, NMF and ICA using our sample of SDSS galaxy spectra.

Comparison of PCA, ICA and NMF

- make this plot by running (it takes a minute)
  %run fig_spec_decompositions.py

Which one is the most astrophysical?



Manifold Learning

What do we do if the assumption of linearity does not hold?

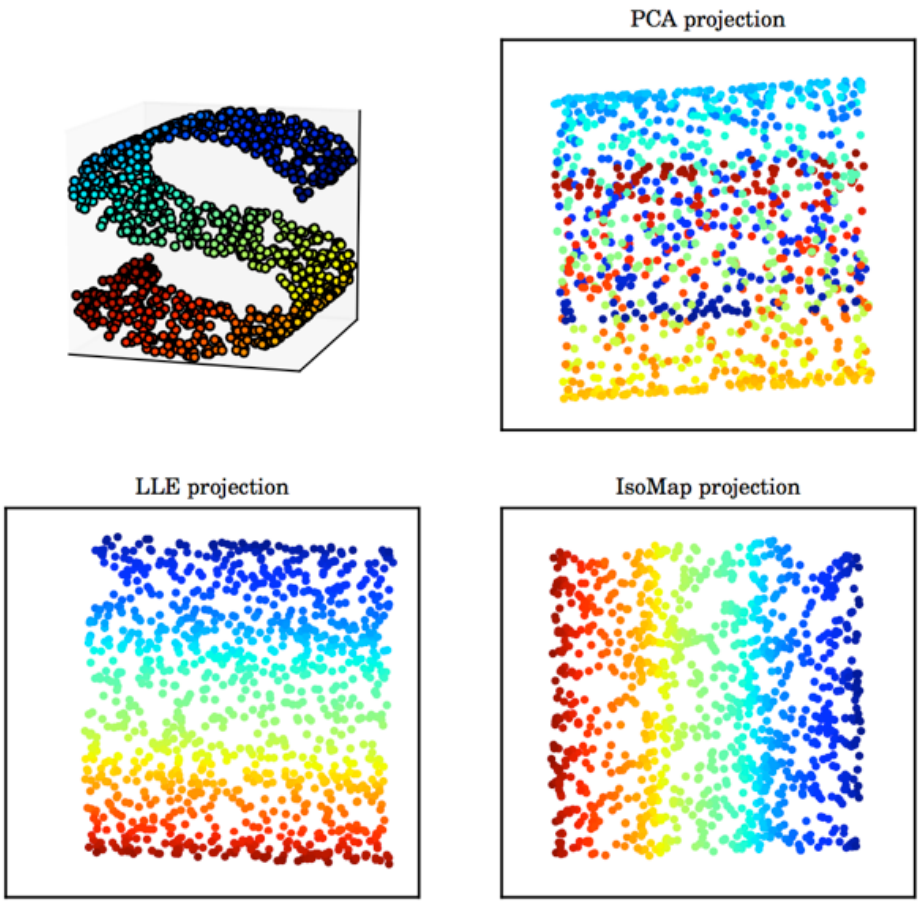


Figure 7.8.: A comparison of PCA and manifold learning. The top-left panel shows an example S-shaped data set (a two-dimensional manifold in a three-dimensional space). PCA identifies three principal components within the data. Projection onto the first two PCA components results in a mixing of the colors along the manifold. Manifold learning (LLE and IsoMap) preserves the local structure when projecting the data, preventing the mixing of the colors.

Here we have a 2D manifold in 3D space. PCA and other linear methods cannot uncover this structure.

Manifold learning techniques "unfold" or "unwrap" this manifold so that its structure becomes clear

Manifold Learning

Locally Linear Embedding

One of more popular techniques among a number of recent methods for non-linear dimensionality reduction.

In non-linear problems, it can have significantly better performance than PCA; e.g. it takes only two components to describe the same fraction of variance in SDSS galaxy spectra that requires dozens of PCA components (Vanderplas & Connolly 2009, AJ, 138, 1365)

The local manifold is determined by analyzing the nearest neighbor distribution around a given point; in some sense, one can think of a tangential hyper-plane approximation in high-D geometry

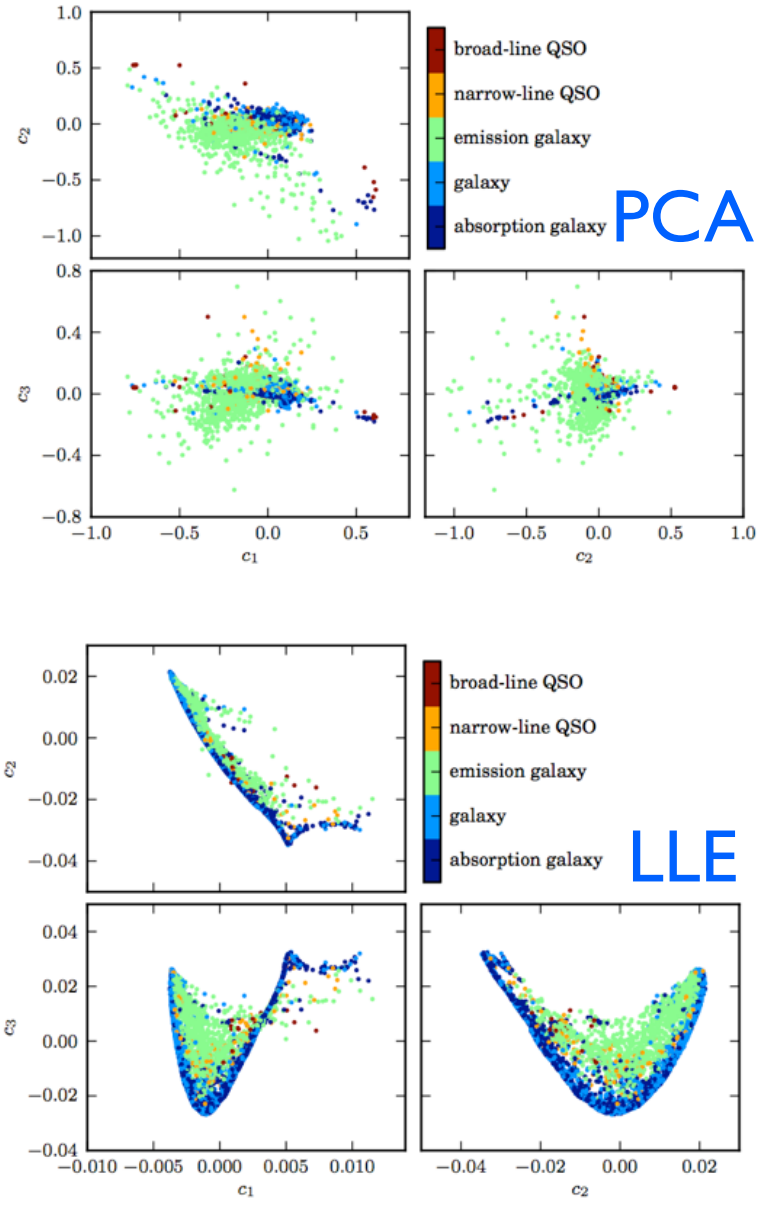


Figure 7.9.: A comparison of the classification of quiescent galaxies and sources with strong line emission using LLE and PCA. The top panel shows the segregation of galaxy types as a function of the first three PCA components. The lower panel shows the segregation using the first three LLE dimensions. The preservation of locality in LLE enables nonlinear features within a spectrum (e.g., variation in the width of an emission line) to be captured with fewer components. This results in better segregation of spectral types with fewer dimensions.

SDSS galaxy spectra

Similarly to the "S curve" example:

PCA: the structure in eigencoefficients is scrambled

LLE: the structure in eigencoefficients preserves information (from independent classification)

To reproduce: astroML, book figures, chapter 9

Which dimensionality reduction technique to use in practice?

Simple summary. Table 7.1 is a simple summary of the trade-offs along our axes of accuracy, interpretability, simplicity, and speed in dimension reduction methods, expressed in terms of high (H), medium (M), and low (L) categories.

Table 7.1.: Summary of the practical properties of the main dimensionality reduction techniques.

Method	Accuracy	Interpretability	Simplicity	Speed
Principal component analysis	H	H	H	H
Locally linear embedding	H	M	H	M
Nonnegative matrix factorization	H	H	M	M
Independent component analysis	M	M	L	L

Regression

Motivation:

While regression was already addressed in earlier lectures, there are a few exceedingly useful tools in astroML that I want you to be aware of:

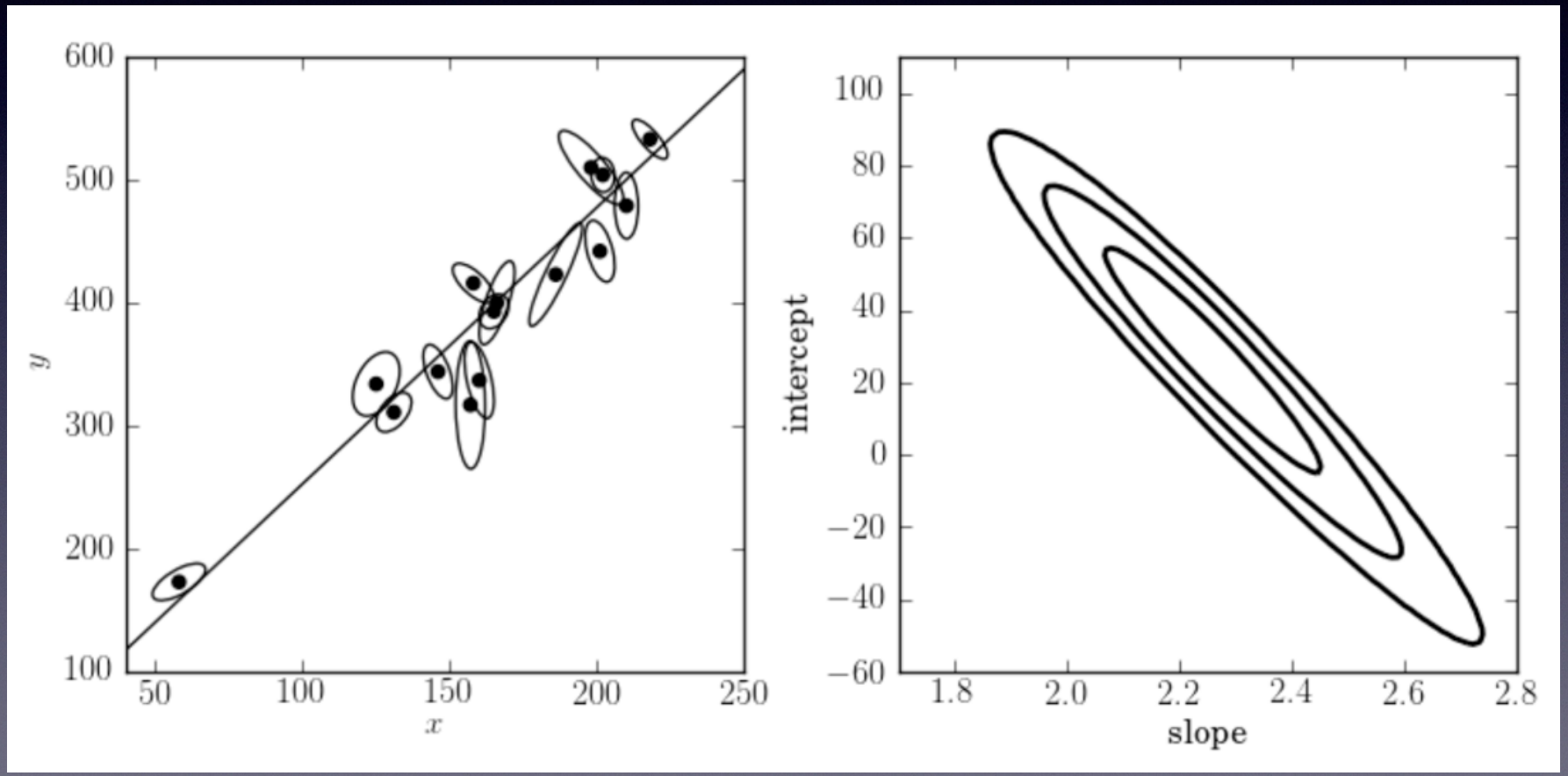
– (Gaussian) errors in both variables
– regression with non-Gaussian errors and/or outliers
– learning curves

Gaussian errors in both variables

(see Hogg, Bovy & Lang, 2010, arXiv:1008.4686)

Example code:

http://www.astroml.org/book_figures/chapter8/fig_total_least_squares.html



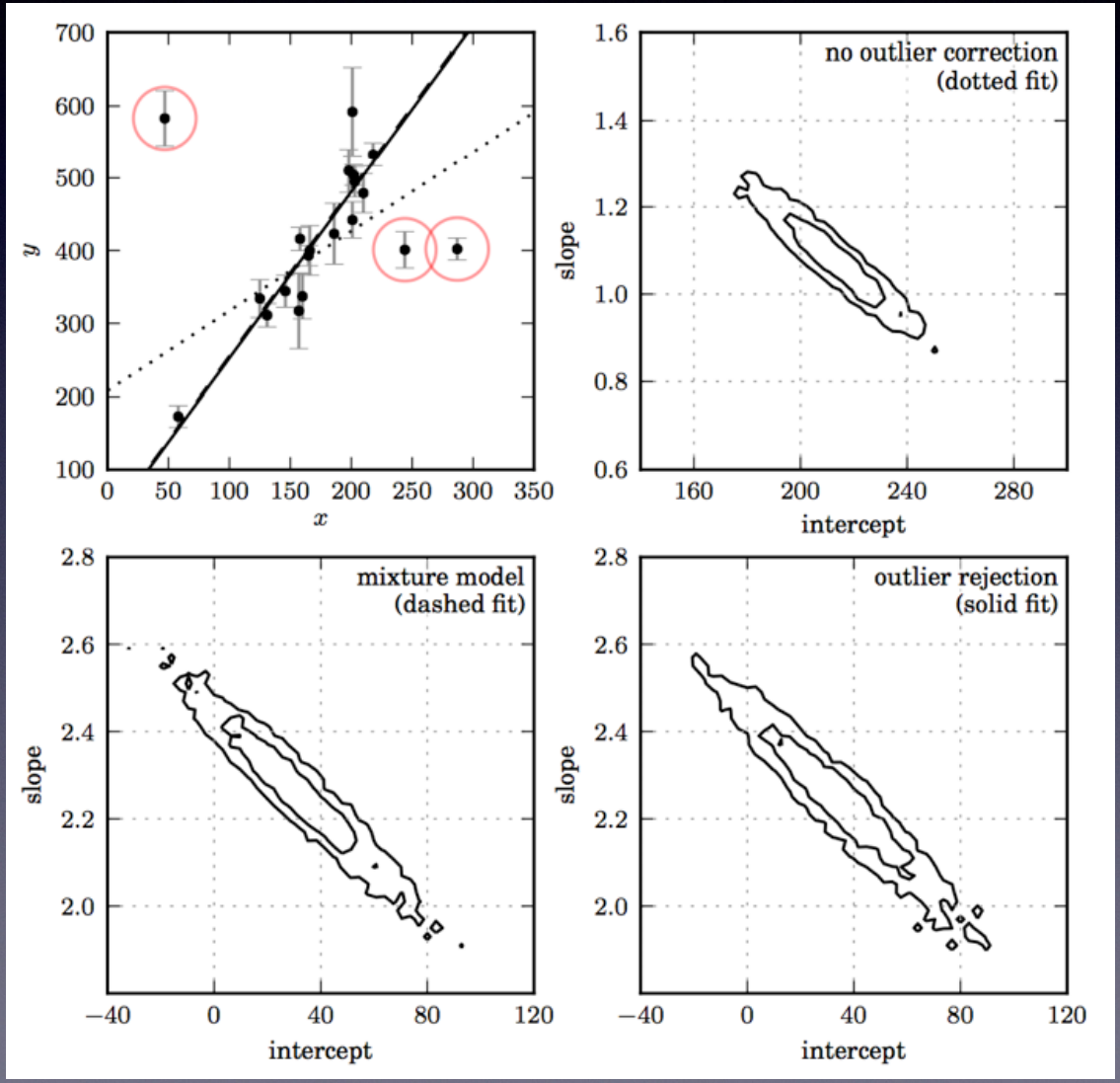
Regression with non-Gaussian errors and outliers (see Hogg, Bovy & Lang, 2010, arXiv:1008.4686)

Example code:

http://www.astroml.org/book_figures/chapter8/fig_outlier_rejection.html

The code uses MCMC

It's easy to change the outlier model (the mixture likelihood)...

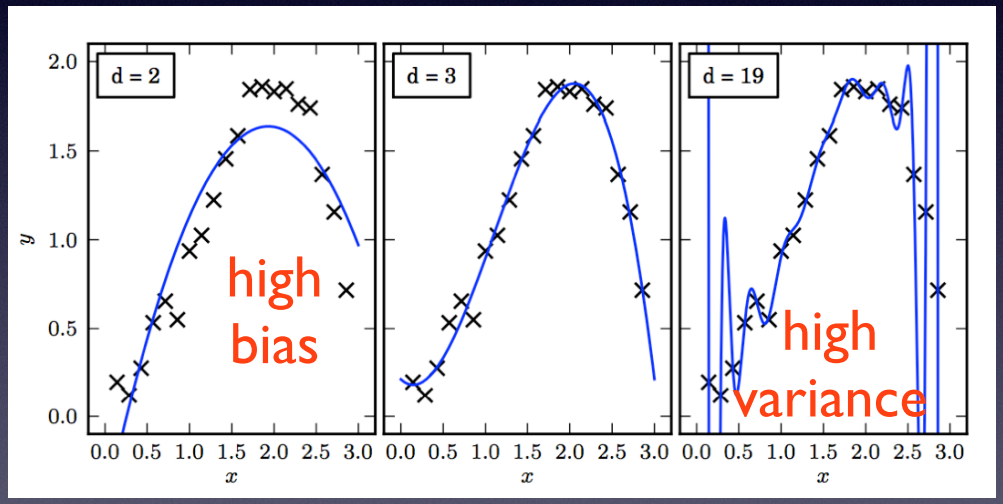


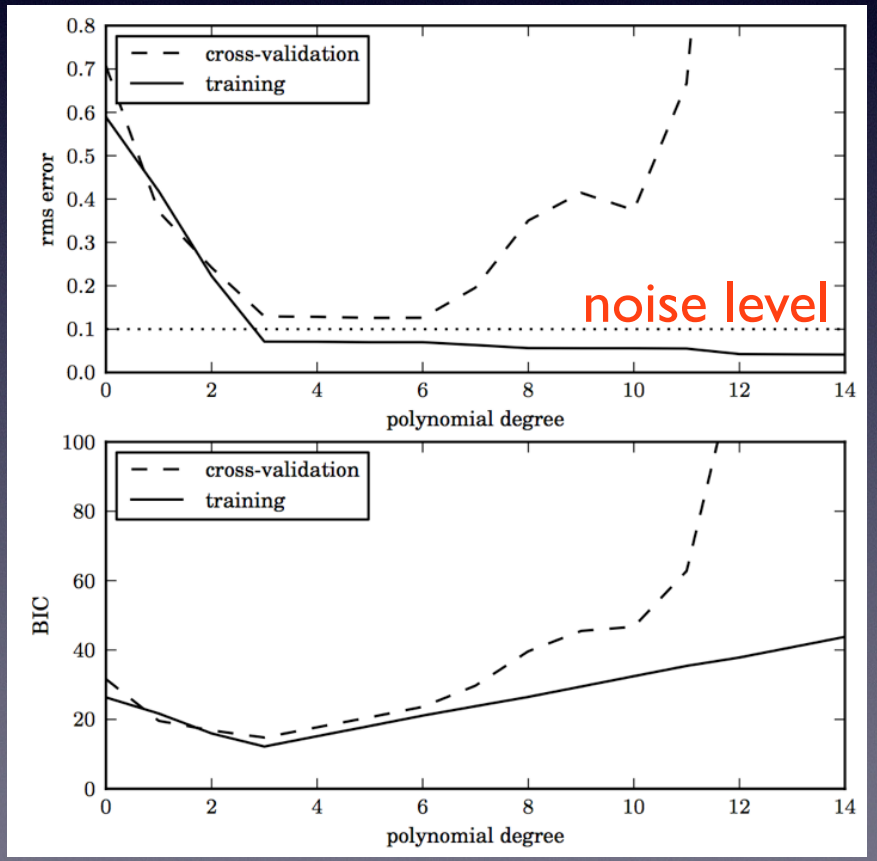
Learning curves

Example code:

http://www.astroml.org/book_figures/chapter8/fig_cross_val.html

Cross-validation was already addressed so here I only briefly address the concept of learning curves:





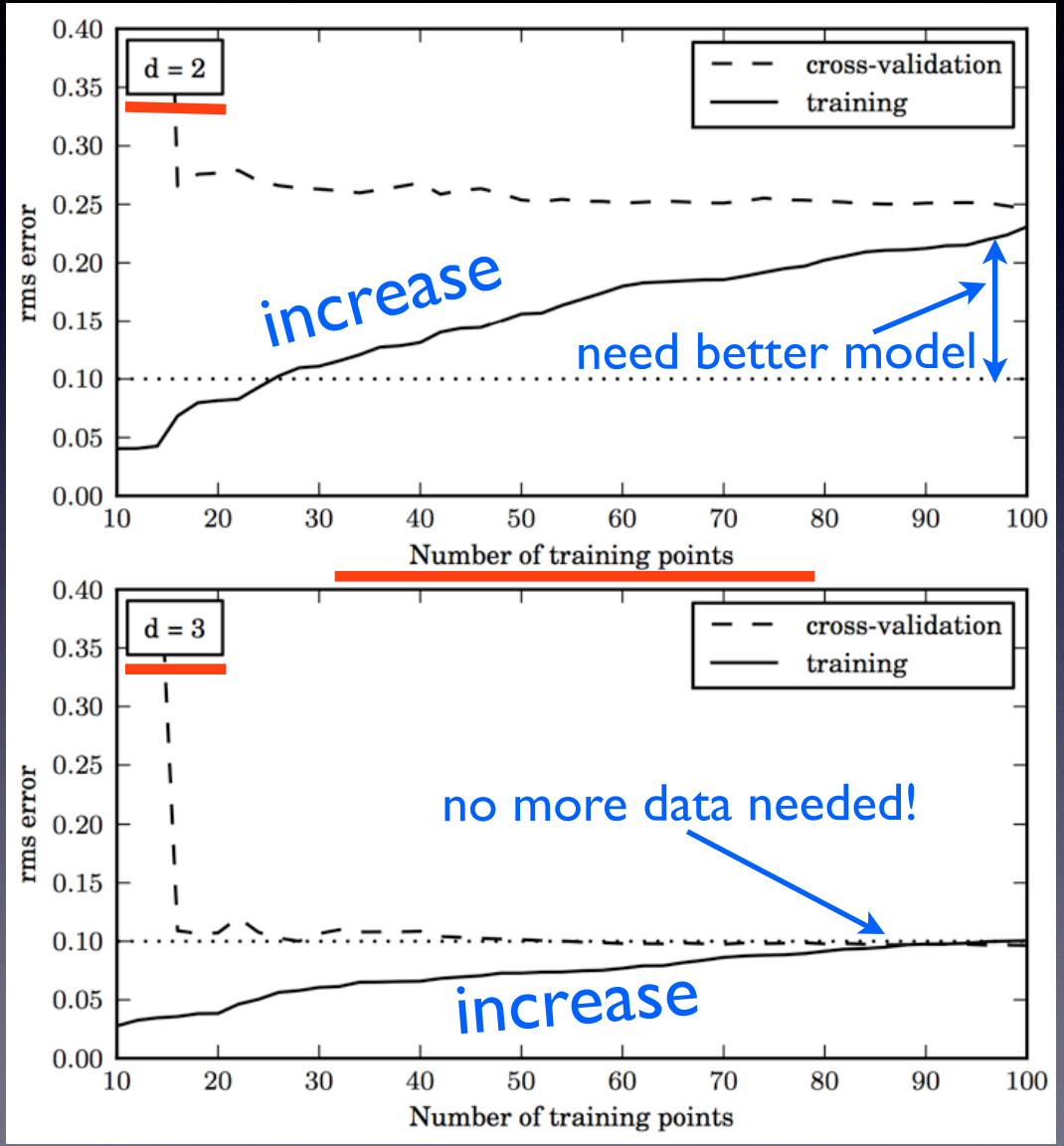
This is an example of how to use cross-validation and/or BIC to choose the order of fitting polynomial (3 here)

Learning curves

In the previous example, the minimum cross-validation rms was similar to the intrinsic noise level. What if this is not the case?

We can improve data vs. model agreement:
- get more data to better constrain model parameters
- use more complex model
- measure more attributes

The guidance for how to improve **our data** can be provided by "learning curves":



Learning curves

In the previous example, the minimum cross-validation rms was similar to the intrinsic noise level. What if this is not the case?

1. **The training error and cross-validation error have converged**. In this case, increasing the number of training points under the same model is futile: the error cannot improve further. This indicates a model error dominated by bias (i.e., it is underfitting the data). For a high-bias model, the following approaches may help:

   - Add additional features to the data.
   - Increase the model complexity.
   - Decrease the regularization.

2. **The training error is much smaller than the cross-validation error**. In this case, increasing the number of training points is likely to improve the model. This condition indicates that the model error is dominated by variance (i.e., it is overfitting the data). For a high-variance model, the following approaches may help:

   - Increase the training set size.
   - Decrease the model complexity.
   - Increase the amplitude of the regularization.

The Vision:

- Maintain a coherent & well-written set of examples of data processing in python *using real data.*
- Offer a standard repository of high-performance python code for astronomy.
- Complement the effort of the AstroPy team.
- Let the cream rise to the top: useful code – once battle-tested – can be moved upstream and become useful in other fields:
  - Ball Tree & two-point statistics (scikit-learn 0.10)
  - Minimum Spanning Tree (scipy 0.12)
  - binned_statistics (scipy 0.12)

Big Goal: community involvement! astroML should be much more than simply a textbook software package. It should become a community repository.

YOU can be part of that vision, too!