

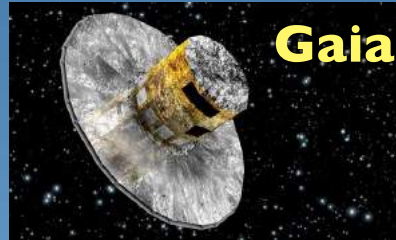
# Big Data Sets in Astronomy - day 2

Željko Ivezić, University of Washington

LSST



SDSS



Gaia



Sao Paulo School of Advanced Science on Learning from Data, July 31 - Aug 2, 2019

# Main Topics:

**1) Introduction and warm-up with astroML**

**2) Density Estimation, Clustering and Classification**

**3) Dimensionality reduction, Regression and Time Series**

I will be switching back and forth between slides and jupyter notebook as we go.

# Motivation for astroML

- **Ever increasing data volume and complexity**
  - SDSS is ~30 TB; LSST will be one SDSS per night, or a total of >100 PB of data (40 billion objects, 30,000 billion forced measurements); also Gaia and many other surveys
  - who and how will do the required data analysis?
- **Sophisticated analysis, need for reproducibility**
  - with the increasing data complexity, analysis becomes more complex, too; what do we do in case of disagreement?
- **Open-source approach improves efficiency**
  - we are not data starved any more!
  - the bottleneck for new results is in human resources (as in “grad students and postdocs”) and analysis tools
  - nobody has an unlimited budget; we should collaborate and share!

# 1) Introduction

## - astroML

If you haven't already, please install astroML by following instructions at:

[https://www.astroml.org/user\\_guide/installation.html](https://www.astroml.org/user_guide/installation.html)

or google for “astroML”, go to

<https://www.astroml.org>

and scroll down to the table of contents.

**Disclaimer:** you may need to run python 2.7 (that is, not python 3.x) as I noticed some errors I didn't understand when testing this morning! Sorry!

# 1) Introduction

## - astroML

To test, start ipython shell and do:

```
[Macintosh-3:~ ivezic$ ipython
[TerminalIPythonApp] WARNING | Config option `ignore_old_config` not recognized
by `TerminalIPythonApp`.
Python 2.7.13 |Anaconda custom (x86_64)| (default, Dec 20 2016, 23:05:08)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

[In [1]: from astroML.datasets import fetch_sdss_spectrum

[In [2]: spec = fetch_sdss_spectrum(1615, 53166, 513)

In [3]: █
```

If there are no error messages, you are good to go!  
If there are problems (e.g. with “GMM”) go to py2.7



**We need more downloads, which you can do by tomorrow:**

Download dataAll.tar.gz (245 MB) as <https://ls.st/> **SLOW!**

Make a directory astroML\_data in your home directory and download dataAll.tar.gz to that directory.

Then unpack it:

```
> cd ~/astroML_data
```

```
> gunzip dataAll.tar.gz (possible done by your machine)
```

```
> tar -xvf dataAll.tar
```

and you should see 4 \*.fit files, 2 \*.npy files and one \*.npz file.

**In addition, clone my directory with lectures (wherever):**

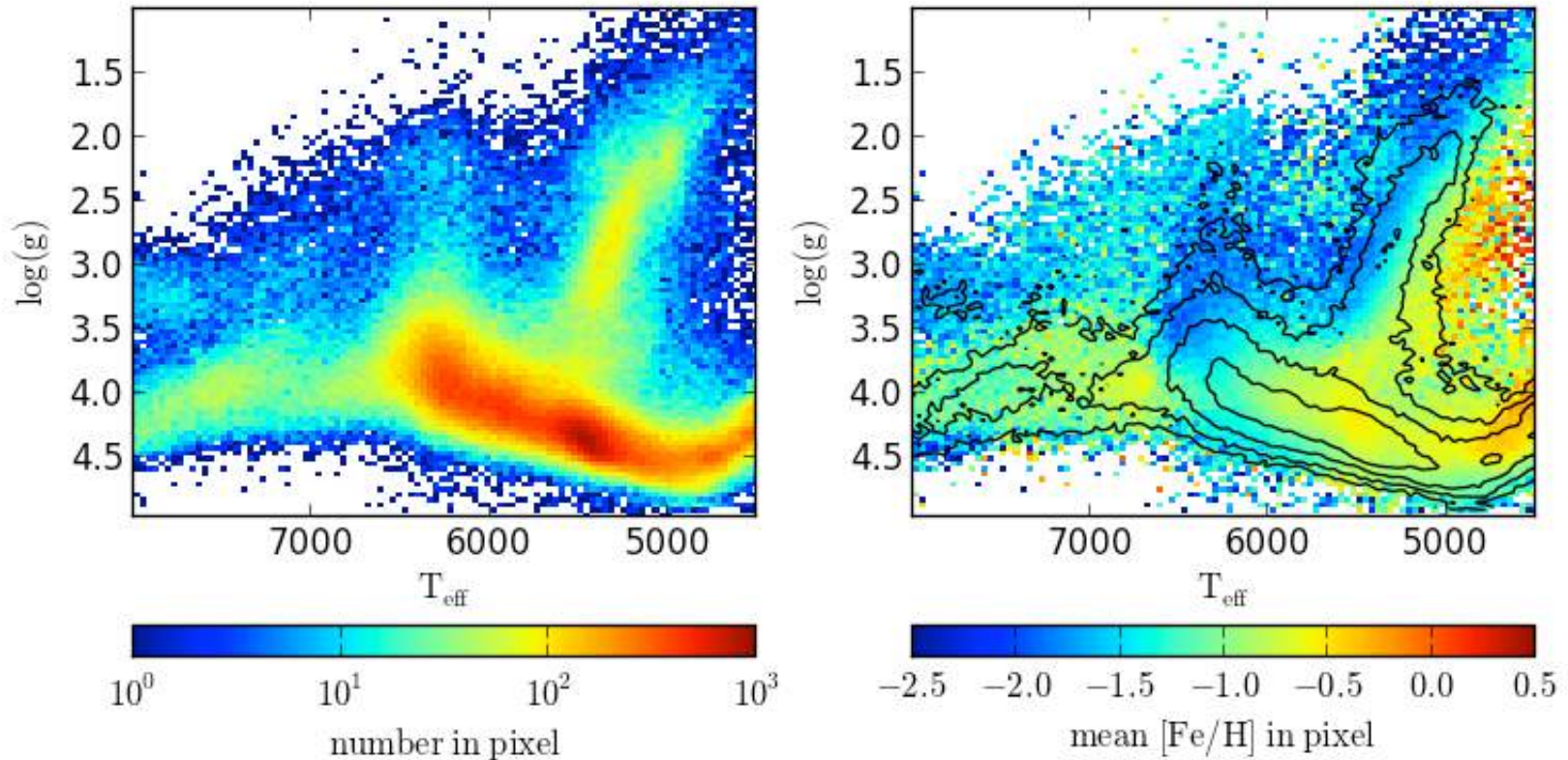
```
> git clone git@github.com:dirac-institute/SPSAS2019.git
```

**ALSO SLOW!**

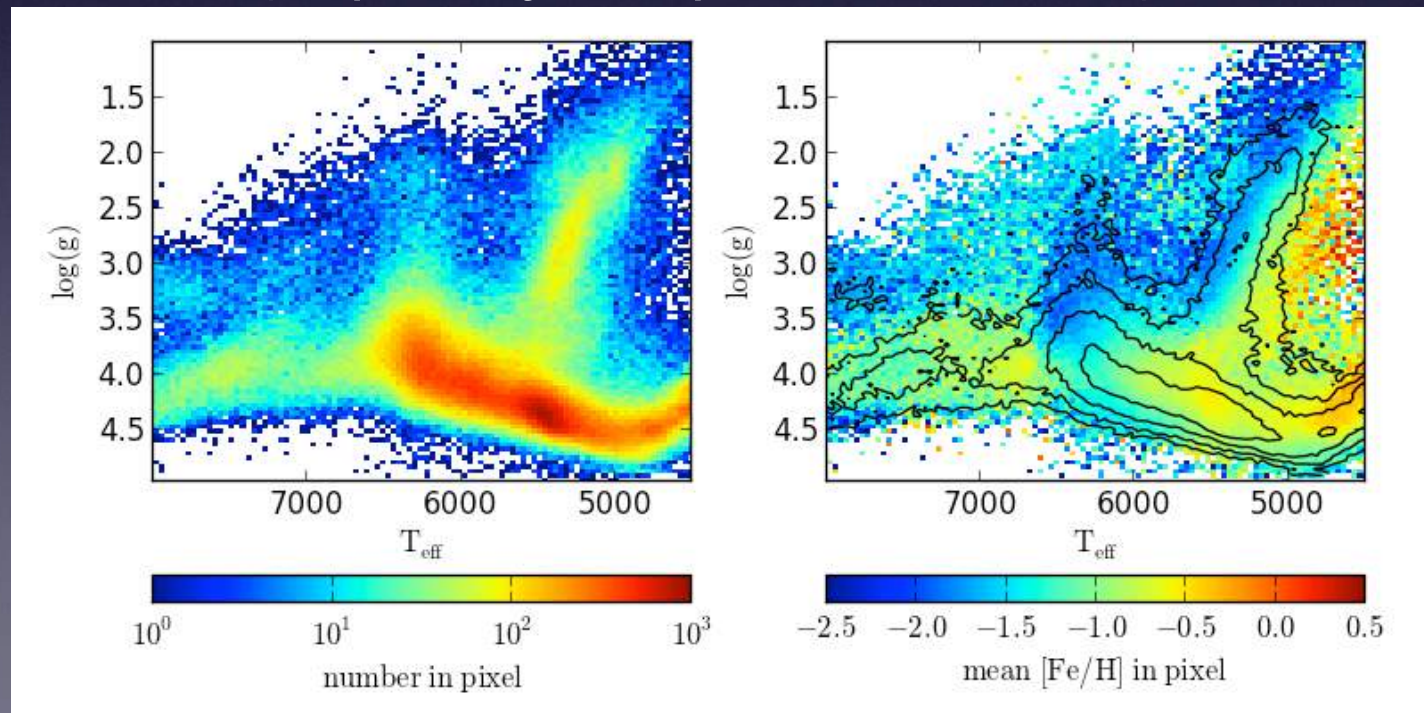
**Test astroML and git repository installation**  
by starting jupyter notebook with file

.../SPSAS2019/lectures/test.ipynb and executing the code there.

It should produce this figure:



- **A Hess diagram coded by a third quantity**
- Hess diagram is an astronomical term for pixelated color-magnitude diagram, where each pixel is coded to display the number of objects in it (also known as “two-dimensional histogram”)
- Of course, the pixels don’t have to be coded by the number of objects in it - we can use instead any **statistic** (a quantity computed from data)





- **A Hess diagram coded by a third quantity**
- Hess diagram is an astronomical term for pixelated color-magnitude diagram, where each pixel is coded to display the number of objects in it (also known as “two-dimensional histogram”)
- Of course, the pixels don’t have to be coded by the number of objects in it - we can use instead any

```
#-----
# Plot the results using the binned_statistic function
from astroML.stats import binned_statistic_2d
N, xedges, yedges = binned_statistic_2d(Teff, logg, FeH,
                                         'count', bins=100)

FeH_mean, xedges, yedges = binned_statistic_2d(Teff, logg, FeH,
                                                'mean', bins=100)
```

- [\\$astroMLdir/astroML/stats/\\_binned\\_statistic.py](#)

## • \$astroMLdir/astroML/stats/\_binned\_statistic.py

```
def binned_statistic(x, values, statistic='mean',  
                    bins=10, range=None):
```

Compute a binned statistic for a set of data.

This is a generalization of a histogram function. A histogram divides the space into bins, and returns the count of the number of points in each bin. This function allows the computation of the sum, mean, median, or other statistic of the values within each bin.

Parameters

-----

**x** : array\_like

A sequence of values to be binned.

**values** : array\_like

The values on which the statistic will be computed. This must be the same shape as x.

**statistic** : string or callable, optional

The statistic to compute (default is 'mean').

The following statistics are available:

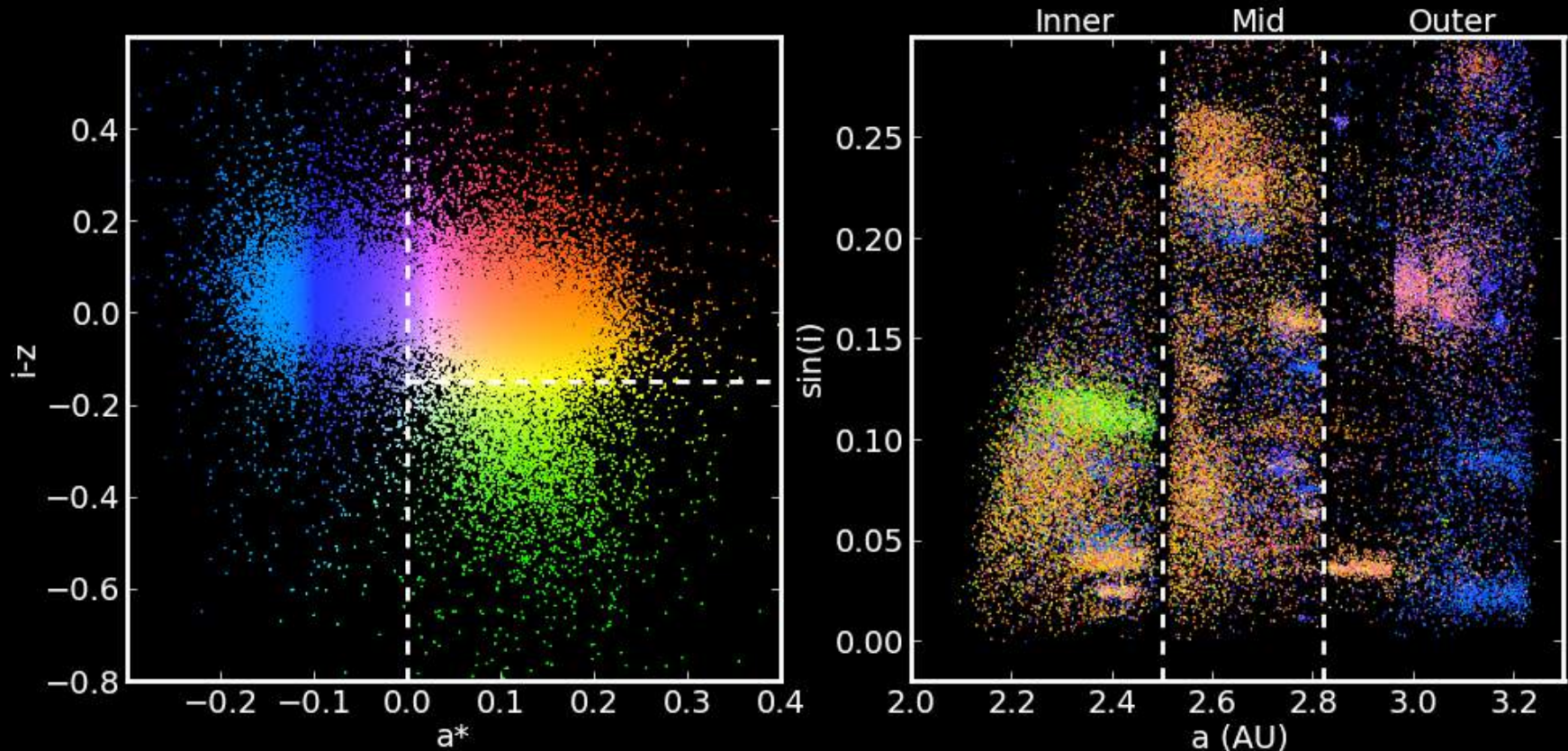
- \* 'mean' : compute the mean of values for points within each bin. Empty bins will be represented by NaN.
- \* 'median' : compute the median of values for points within each bin. Empty bins will be represented by NaN.
- \* 'count' : compute the count of points within each bin. This is identical to an unweighted histogram. 'values' array is not referenced.
- \* 'sum' : compute the sum of values for points within each bin. This is identical to a weighted histogram.
- \* function : a user-defined function which takes a 1D array of values, and outputs a single numerical statistic. This function will be called on the values in each bin. Empty bins will be represented by function([]), or NaN if this returns an error.

**bins** : int or sequence of scalars, optional

If 'bins' is an int, it defines the number of equal-width

You can easily make this plot with astroML

## Visualization of 4-dimensional correlations



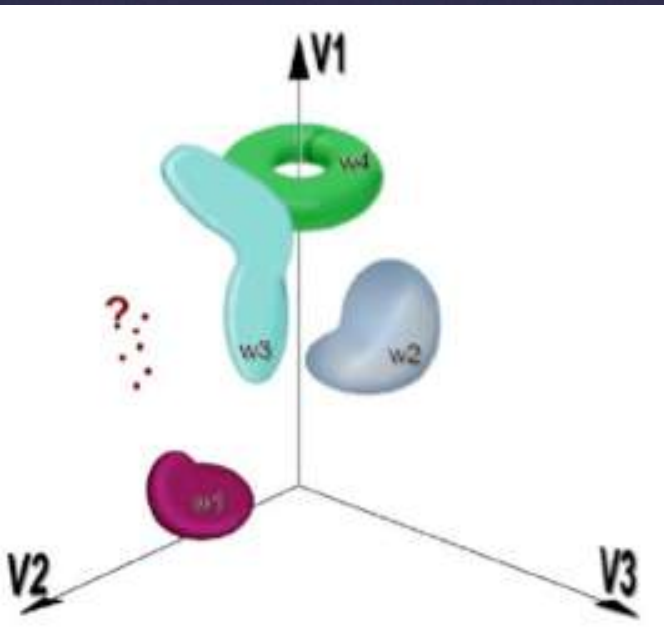
Our goal here is to discuss in detail a few more complex examples.



# Statistical analysis of a massive LSST dataset

- A large (100 PB) database and sophisticated analysis tools: for each of 40 billion objects there will be about 1000 measurements (each with a few dozen measured parameters)

## Data mining and knowledge discovery



- 10,000-D space with 40 billion points
- Characterization of known objects
- Classification of new populations
- Discoveries of unusual objects

Clustering, classification, outliers



# $h(x)$

Possibly the most important single problem in data mining is how to estimate the distribution  $h(x)$  from which values of  $x$  are drawn (or which “generates”  $x$ ). The function  $h(x)$  quantifies the probability that a value lies between  $x$  and  $x + dx$ , equal to  $h(x)dx$ , and is called a probability density function (pdf).  $x$  and  $dx$  can be multi-dimensional.

## A few selected topics for today

- summary statistics and robust statistics for  $f(x)$ , which is our data-based estimate of  $h(x)$
- density estimation
- clustering
- classification

# ● How can we quantify $h(x)$ ?

- Arithmetic mean (also known as the expectation value),

$$\mu = E(x) = \int_{-\infty}^{\infty} xh(x) dx$$

Location parameter

- Variance,

$$V = \int_{-\infty}^{\infty} (x - \mu)^2 h(x) dx$$

- Standard deviation,

$$\sigma = \sqrt{V}$$

Scale parameter

- Skewness,

$$\Sigma = \int_{-\infty}^{\infty} \left( \frac{x - \mu}{\sigma} \right)^3 h(x) dx$$

- Kurtosis,

$$K = \int_{-\infty}^{\infty} \left( \frac{x - \mu}{\sigma} \right)^4 h(x) dx - 3$$

Shape parameters

- $p\%$  quantiles ( $p$  is called a percentile),  $q_p$ ,

$$\frac{p}{100} = \int_{-\infty}^{q_p} h(x) dx$$

Parameters describing cumulative distribution

Location, scale, shape, and other parameters defined for  $h(x)$  can also be computed for **a sample drawn from  $h(x)$** .

In general, when estimating the above quantities for a sample of  $N$  measurements, the integral  $\int_{-\infty}^{\infty} g(x)h(x) dx$  becomes proportional to the sum  $\sum_i^N g(x_i)$ , with the constant of proportionality  $\sim (1/N)$ . For example, the *sample arithmetic mean*,  $\bar{x}$ , and the *sample standard deviation*,  $s$ , can be computed via standard formulas,

When using  $h(x)$ , these parameters are called **population statistics**, and when determined from data  $x_i, i=1 \dots N$ , they are called **sample statistics**.

For example, location parameter  $\mu$  for  $h(x)$  is estimated using **the mean**:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

and scale parameter  $\sigma$  for  $h(x)$  is estimated using **the standard deviation**:

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$$

**Note:** when  $h(x)$  is estimated from data, a different symbol should be used, e.g.  $f(x)$

Location parameter  $\mu$  for  $h(x)$  is estimated using **the mean**:

and scale parameter  $\sigma$  is estimated using **the standard deviation**:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$$

These estimators will NOT be exactly equal to  $\mu$  and  $\sigma$ !  
Each will be scattered around the true values ( $\mu$  and  $\sigma$ ) approximately following Gaussian distributions with the widths (scale parameters) given by:

*the standard error of the mean*

$$\sigma_{\bar{x}} = \frac{s}{\sqrt{N}}$$

*often called “error bar”!*

*error of the standard deviation estimate  $s$ :*

$$\sigma_s = \frac{s}{\sqrt{2(N-1)}}$$



So, given  $x_i, i=1 \dots N$ , we can compute

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i \quad \pm \quad \sigma_{\bar{x}} = \frac{s}{\sqrt{N}} \quad \text{the sample mean}$$

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad \pm \quad \sigma_s = \frac{s}{\sqrt{2(N-1)}} \quad \text{the sample standard deviation}$$

What exactly did we compute?

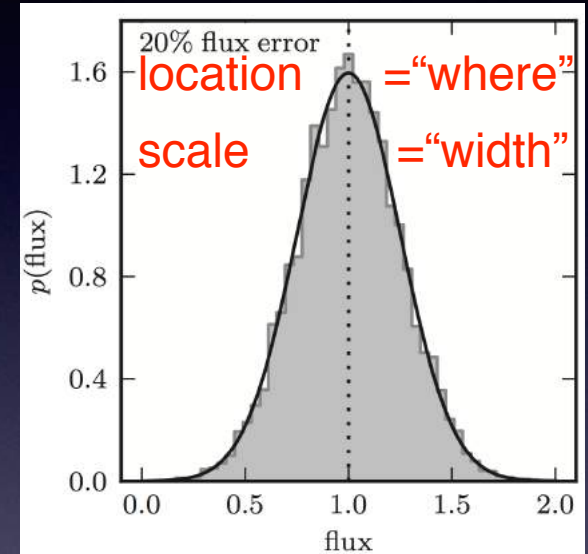
In the majority of practical cases,  $x_i$  represent our  $N$  measurements of some fixed well-defined quantity  $x$ , and our inference about its value is summarized by the sample mean and the standard error of the mean, and assumed Gaussian distribution.

What about standard deviation?

# ● Robust statistics

What is the **mean** wealth in Seattle? Does it depend on the fact that Bill Gates and a few of his friends live in Seattle?

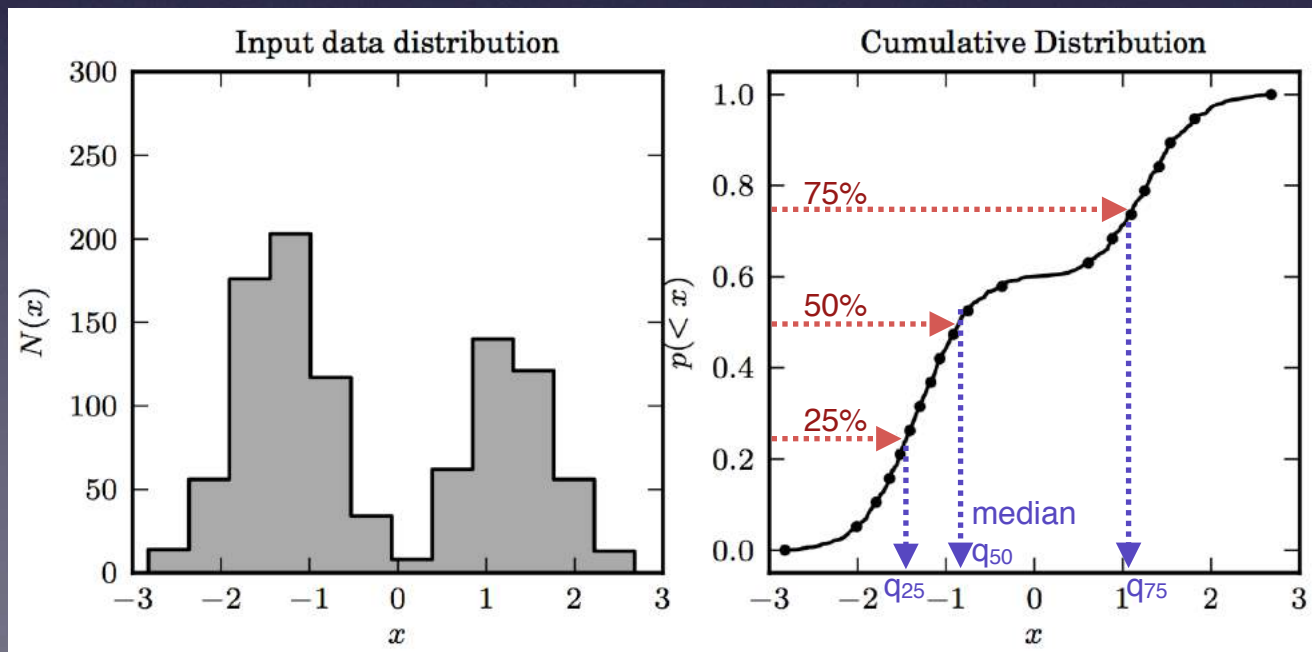
$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$



# ● Robust statistics

What is the **mean** wealth in Seattle? Does it depend on the fact that Bill Gates and a few of his friends live in Seattle?

Median and  $\sigma_G$  are good estimators of location and scale parameters in cases when outliers are present (e.g. “real data”)



## ● Robust statistics

**Task:** given measurements  $x_i$ ,  $i=1\dots N$ , drawn from the Cauchy distribution, find the best estimate of  $\mu$ , let's call it  $\mu^0$ , and its uncertainty,  $\sigma_\mu$

Use the median value of  $x_i$  as an estimate of location,  $\mu^0$   
The scale parameter (“width”) can be estimated from the interquartile range (note:  $q_{50}$  is the median):

$$\sigma_G = 0.7413 (q_{75} - q_{25})$$

In the case of Gaussian,  $\sigma_G$  is equal to standard deviation ( $\sigma$ )

The uncertainty of  $\mu^0$  (i.e. of the median) can be estimated from

$$\sigma_\mu = \sqrt{\frac{\pi}{2N}} \sigma_G$$



## ● Robust statistics

What is the **mean** wealth in Seattle? Does it depend on the fact that Bill Gates and a few of his friends live in Seattle?

Median and  $\sigma_G$  are good estimators of location and scale parameters also in cases when outliers are present (e.g. “real data”)

The price we pay for using the median instead of the mean is 25% larger uncertainty for the former than for the latter (assuming nearly Gaussian distributions).

**This is often good price to pay to avoid catastrophic failures!**

# Gaussian Mixture Models for $h(x)$



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

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

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,  $\alpha_j$ , and its parameters  $\mu_j$  and  $\sigma_j$ . 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](https://arxiv.org/abs/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**, essentially penalized chi2), or **cross-validation**



# ● Bayesian statistics

Posterior pdf for model  $M$  and parameters  $\theta$ , given data  $D$  and prior information  $I$

## Bayes' Theorem:

$$p(M, \theta | D, I) = \frac{p(D | M, \theta, I) p(M, \theta | I)}{p(D | I)}$$

The likelihood of data given  $M$ , with some fixed parameters  $\theta$ , and  $I$ .

Probability of data ("normalization")

Prior

Probability for  $\theta$ , given that  $M$  is true

$$p(M, \theta | I) = p(\theta | M, I) p(M | I)$$

Prior for model  $M$



## Bayesian Information Criterion (BIC)

BIC is an approximation for the probability of a model, given the data, and it corresponds to chi2 penalized for the number of free model parameters,  $k$ :

The BIC is easier to compute and, similarly to AIC, it is based on the maximum value of the data likelihood,  $L^0(M)$ , rather than on its integration over the full parameter space (evidence  $E(M)$  in eq. 5.23). The BIC for a given model  $M$  is computed as

$$\text{BIC} \equiv -2 \ln [L^0(M)] + k \ln(N), \quad (5.35)$$

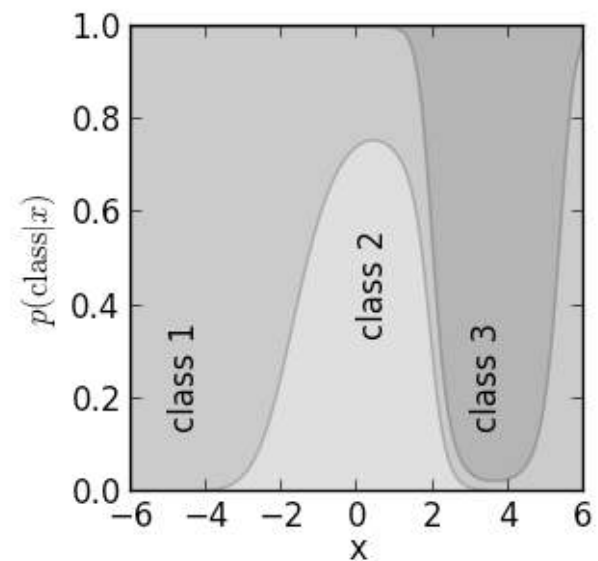
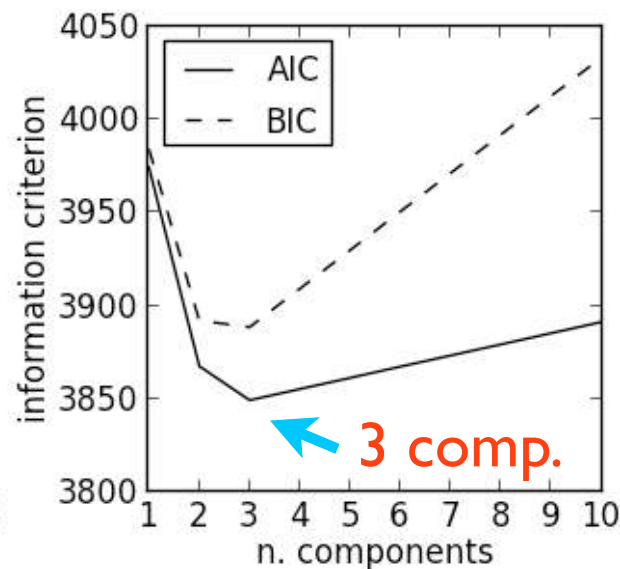
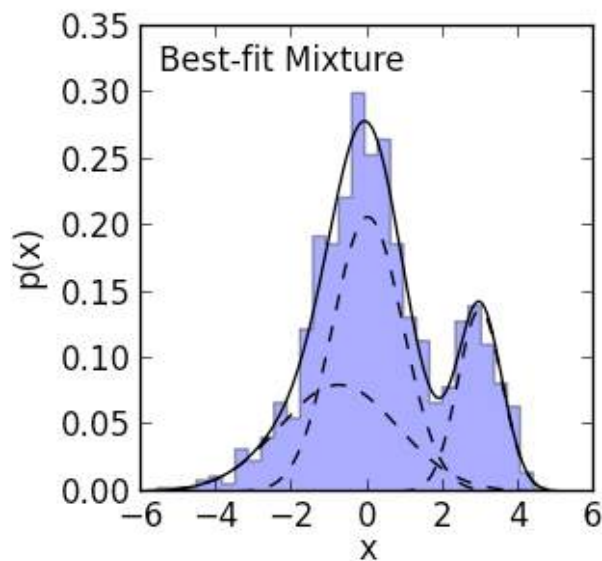
where  $k$  is the number of model parameters and  $N$  is the number of data points. The BIC corresponds to  $-2 \ln[E(M)]$  (to make it consistent with AIC), and can be derived using the approximation for  $E(M)$  given by eq. 5.28 and assuming  $\sigma_\mu \propto 1/\sqrt{N}$ .

BIC is closely related to Aikake Information Criterion (AIC), which is often used by computer scientists (in addition to cross-validation method) to select models.



# Gaussian Mixture Models

**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 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.**

# 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”  $\sigma_j$  from  $\mu_j$  (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  $\theta_j$  now corresponds to  $\mu_j$  or  $\sigma_j$ . By setting the derivatives of  $\ln L$  with respect to  $\mu_j$  and  $\sigma_j$  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  $w_{ij} = p(j|x_i)$

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

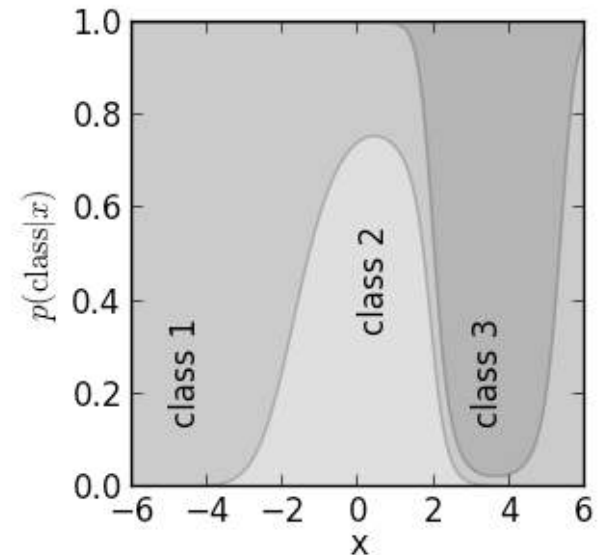
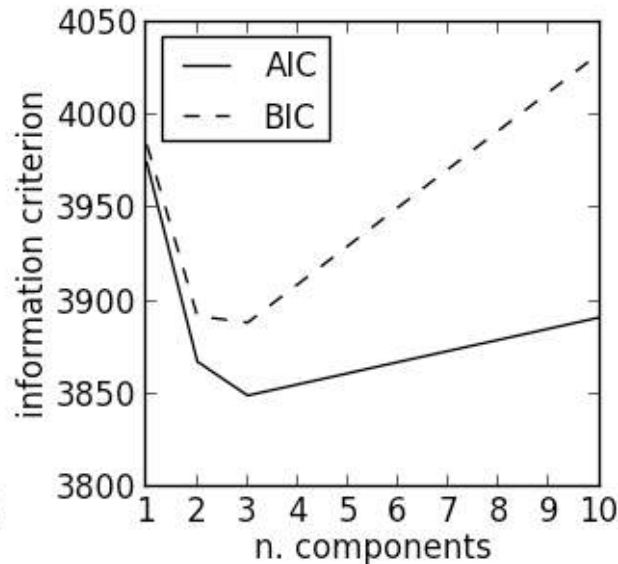
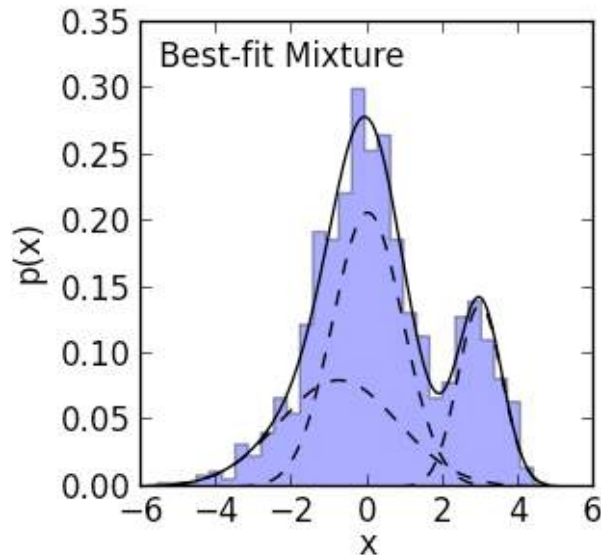
These expressions and eq. 4.21 form the basis of the iterative EM algorithm in the case of Gaussian mixtures. Starting with a guess for  $w_{ij}$ , the values of  $\alpha_j$ ,  $\mu_j$ , and  $\sigma_j$  are estimated using eqs. 4.26–4.28. This is the “maximization” *M-step* which brings the parameters 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  $\sigma_j$  to the sample standard deviation, all  $\alpha_j$  to  $1/M$ , and randomly drawing  $\mu_j$  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  $\alpha_i$ . 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,

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

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:

$$q_{ij} \leftarrow \frac{\alpha_j \mathcal{N}(\mathbf{w}_i | \mathbf{R}_i \mu_j, \mathbf{T}_{ij})}{\sum_j \alpha_j \mathcal{N}(\mathbf{w}_i | \mathbf{R}_i \mu_j, \mathbf{T}_{ij})},$$

$$\mathbf{b}_{ij} \leftarrow \mu_j + \Sigma_j \mathbf{R}_i^T \mathbf{T}_{ij}^{-1} (\mathbf{w}_i - \mathbf{R}_i \mu_j),$$

$$\mathbf{B}_{ij} \leftarrow \Sigma_j - \Sigma_j \mathbf{R}_i^T \mathbf{T}_{ij}^{-1} \mathbf{R}_i \Sigma_j,$$

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

- The maximization (M) step:

$$\alpha_i \leftarrow \frac{1}{N} \sum_j q_{ij},$$

$$\mu_j \leftarrow \frac{1}{q_j} \sum_i q_{ij} \mathbf{b}_{ij},$$

$$\Sigma_j \leftarrow \frac{1}{q_j} \sum_i q_{ij} [(\mu_j - \mathbf{b}_{ij})(\mu_j - \mathbf{b}_{ij}^T) + \mathbf{B}_{ij}],$$

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

# Extreme Deconvolution in high-D (XD)

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

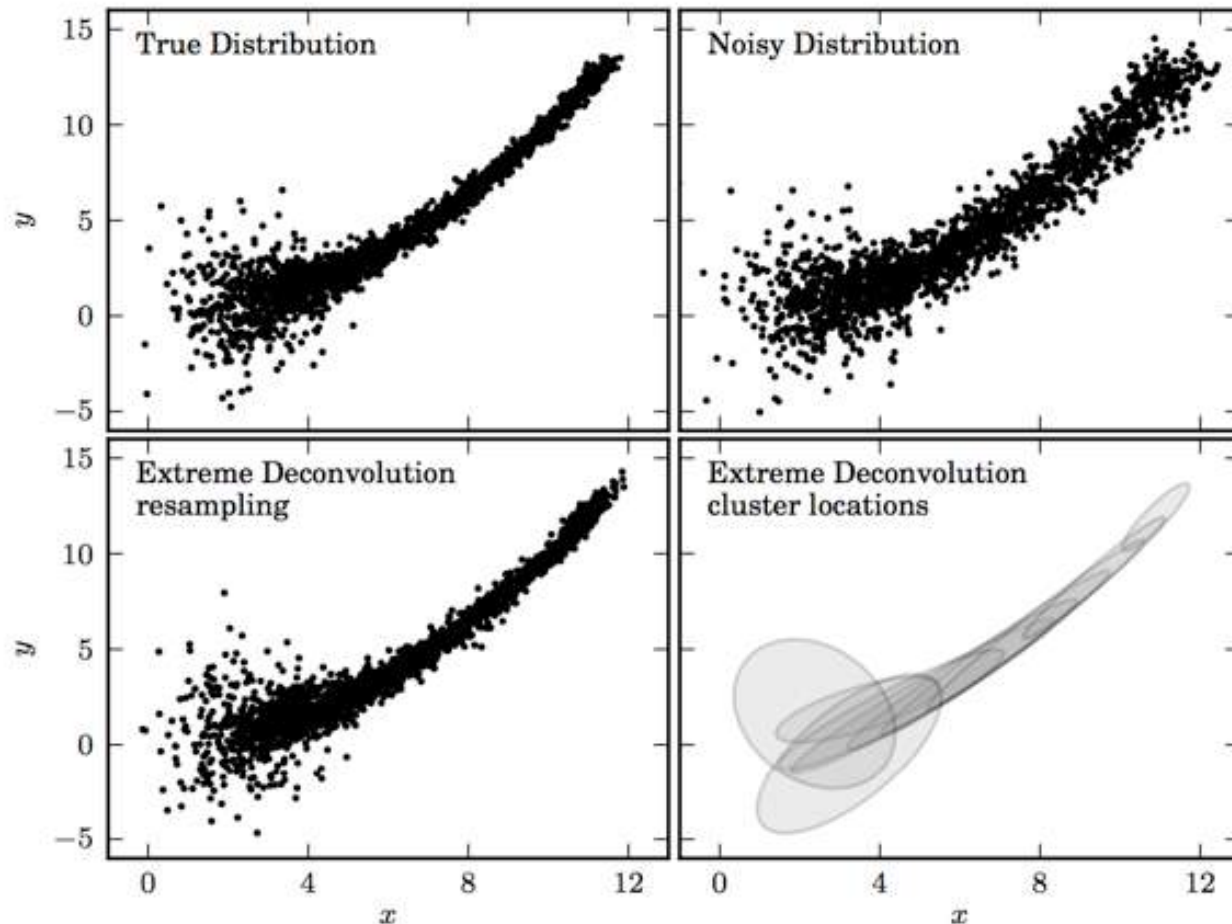


Figure 6.11.: An example of extreme deconvolution showing a simulated two-dimensional distribution of points, where the positions are subject to errors. The top two panels show the distributions with small (left) and large (right) errors. The bottom panels show the densities derived from the noisy sample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



# Clustering (unsupervised classification)

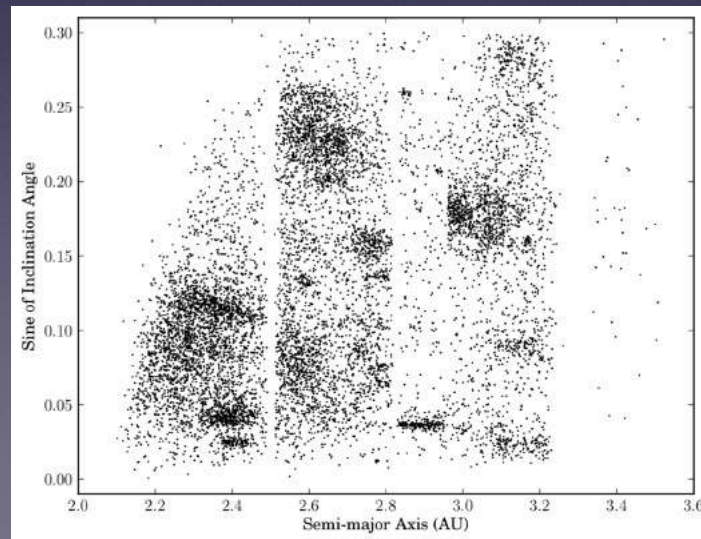
Here “unsupervised” means that there is no prior information about the number and properties of clusters.

“Clustering” in astronomy refers to a number of different aspects of data analysis.

Given a multivariate point data set, we can ask whether it displays any structure, that is, concentrations of points. Alternatively, when a density estimate is available we can search for “overdensities.”

Another way to interpret clustering is to seek a partitioning or segmentation of data into smaller parts according to some criteria.

How many clusters do you “see” in this diagram?



## Clustering with Gaussian Mixture in 2D

- we may need many components (here 100); data should tell us how many...

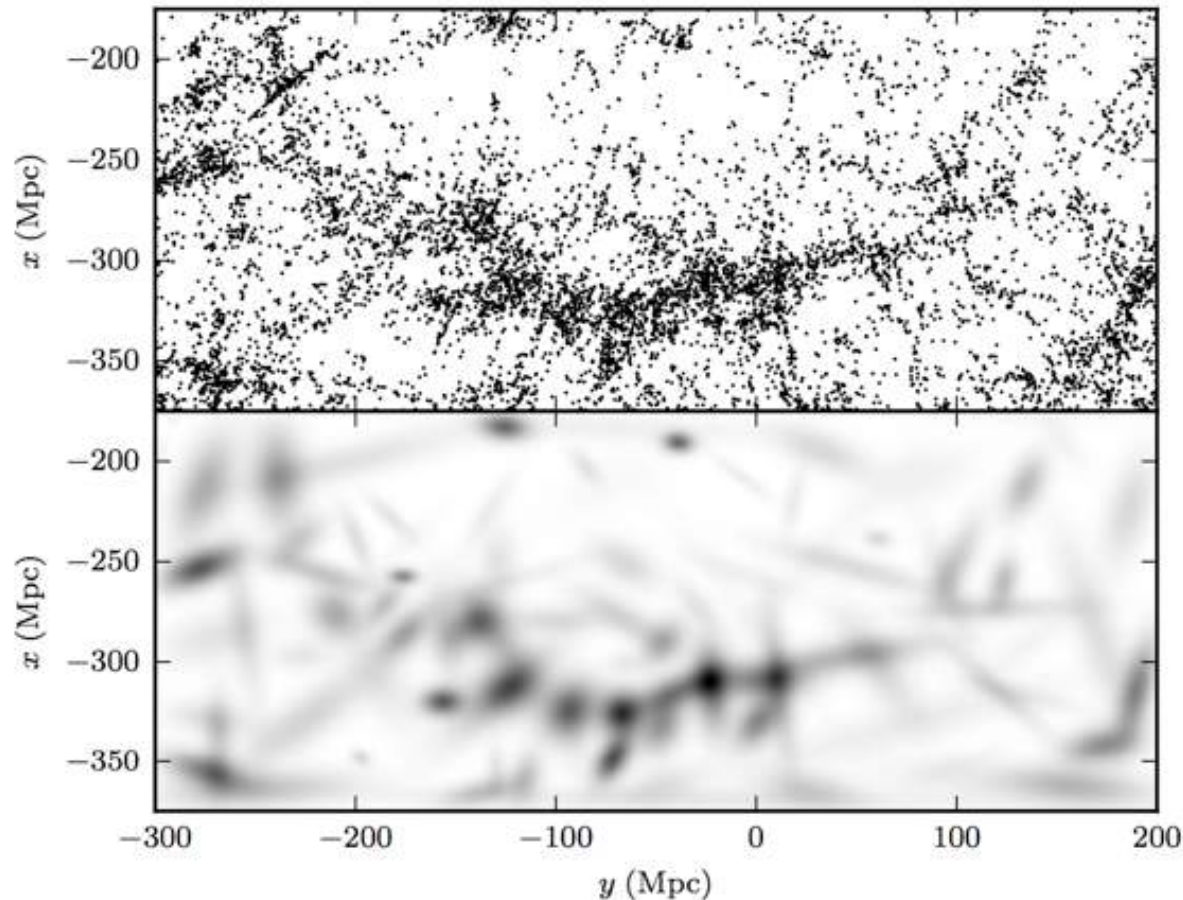


Figure 6.7.: A two-dimensional mixture of 100 Gaussians (bottom) used to estimate the number density distribution of galaxies within the SDSS Great Wall (top). Compare to figures 6.4 and 6.3, where the density for the same distribution is computed using both kernel density and nearest-neighbor-based estimates.

**Break here, go to notebook...**

# Supervised classification

Here “supervised” means that there is prior information about the number and properties of clusters: for a training sample, we know the so-called “class labels” (for each data point in the training sample, we know to which cluster it belongs; characterizing these known clusters is typically easier than finding and characterizing unknown clusters)

There are two basic types of classification methods: **generative classification** methods model the underlying density field (i.e. it relies on density estimation methods), and **discriminative classification** methods which focus on finding the decision boundary which separates classes directly. The former are easier to interpret, the latter often work better in high-D cases.



# Generative classification

Given a set of data  $\{\mathbf{x}\}$  consisting of  $N$  points in  $D$  dimensions, such that  $x_i^j$  is the  $j$ th feature of the  $i$ th point, and a set of discrete labels  $\{y\}$  drawn from  $K$  classes, with values  $y_k$ , Bayes' theorem describes the relation between the labels and features:

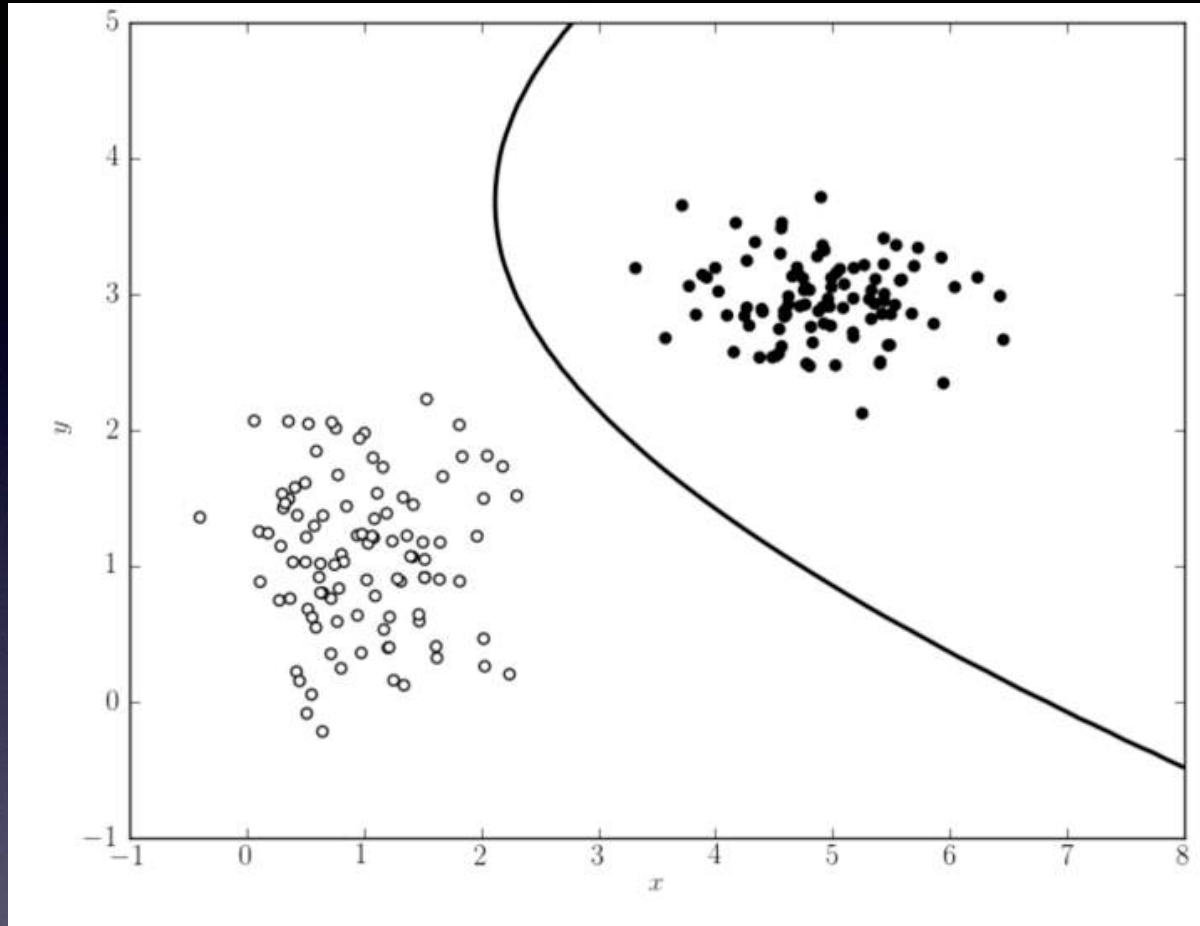
$$p(y_k|\mathbf{x}_i) = \frac{p(\mathbf{x}_i|y_k)p(y_k)}{\sum_i p(\mathbf{x}_i|y_k)p(y_k)}. \quad (9.6)$$

If we knew the full probability densities  $p(\mathbf{x}, y)$  it would be straightforward to estimate the classification likelihoods directly from the data. If we chose not to fully sample  $p(\mathbf{x}, y)$  with our training set we can still define the classifications by drawing from  $p(y|\mathbf{x})$  and comparing the likelihood ratios between classes (in this way we can focus our labeling on the specific, and rare, classes of source rather than taking a brute-force random sample).

In generative classifiers we are modeling the class-conditional densities explicitly, which we can write as  $p_k(\mathbf{x})$  for  $p(\mathbf{x}|y = y_k)$ , where the class variable is, say,  $y_k = 0$  or  $y_k = 1$ . The quantity  $p(y = y_k)$ , or  $\pi_k$  for short, is the probability of any point having class  $k$ , regardless of which point

The task of learning the best classifier then becomes the task of estimating the  $p_k$ 's. This approach means we will be doing multiple separate *density estimates* using many of the techniques introduced

# Discriminative classification



$$\text{completeness} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}},$$

$$\text{contamination} = \frac{\text{false positives}}{\text{true positives} + \text{false positives}},$$

In this simple example,  
perfect separation is  
possible; generally not  
the case!

# Decision boundary

The *decision boundary* between two classes is the set of  $x$  values at which each class is equally likely; that is,

$$\pi_1 p_1(\mathbf{x}) = \pi_2 p_2(\mathbf{x}); \quad (9.14)$$

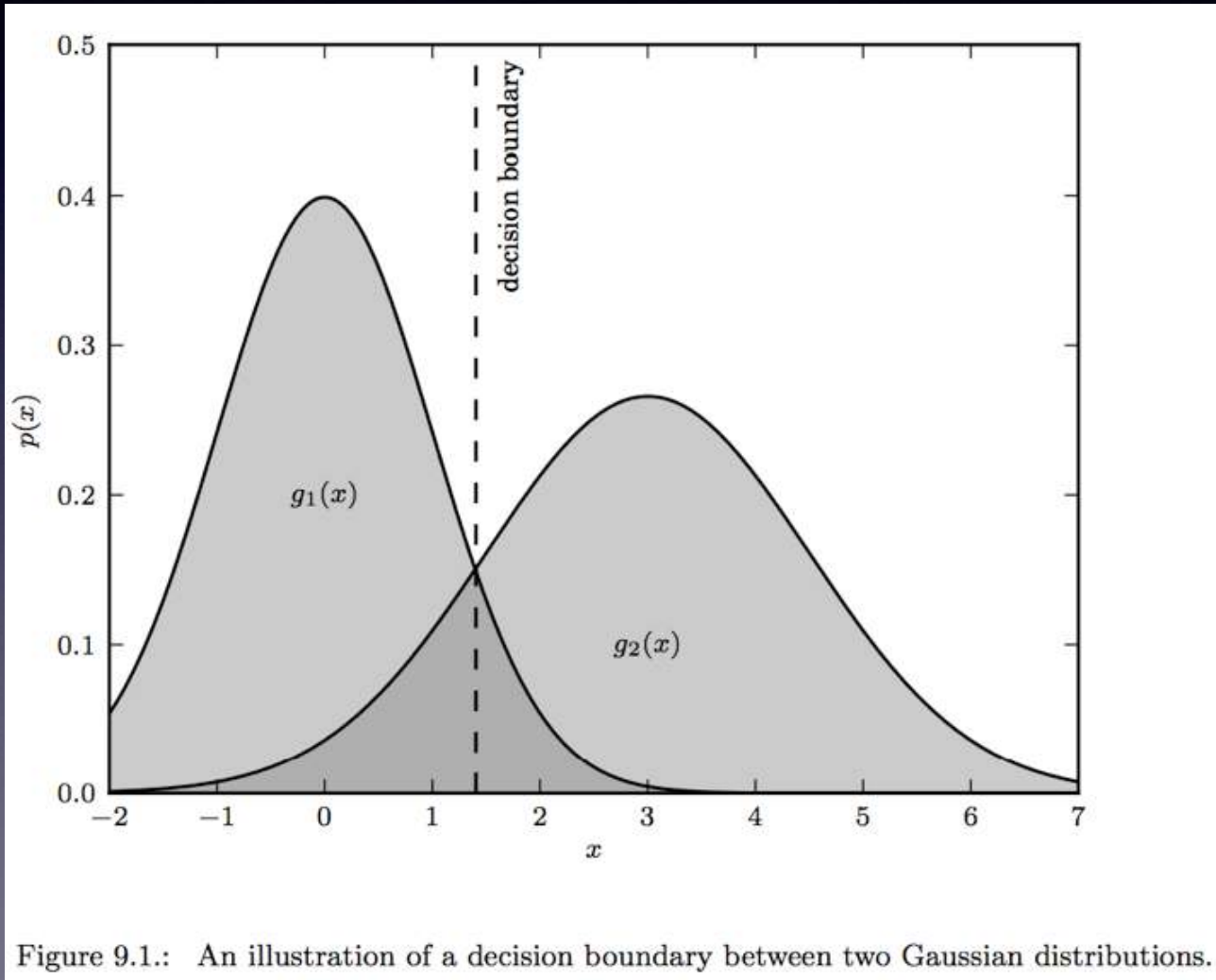



Figure 9.1.: An illustration of a decision boundary between two Gaussian distributions.

# Discriminative classification

discriminant function 

$$g(\mathbf{x}) = f(y|\mathbf{x}) = \int y p(y|\mathbf{x}) dy \quad (9.7)$$

$$= 1 \cdot p(y = 1|\mathbf{x}) + 0 \cdot p(y = 0|\mathbf{x}) = p(y = 1|\mathbf{x}). \quad (9.8)$$

If we now apply Bayes' rule (eq. 3.10), we find (cf. eq. 9.6)

$$g(\mathbf{x}) = \frac{p(\mathbf{x}|y = 1) p(y = 1)}{p(\mathbf{x}|y = 1) p(y = 1) + p(\mathbf{x}|y = 0) p(y = 0)} \quad (9.9)$$

$$= \frac{\pi_1 p_1(\mathbf{x})}{\pi_1 p_1(\mathbf{x}) + \pi_0 p_0(\mathbf{x})}. \quad (9.10)$$

## *Bayes classifier*

Making the discriminant function yield a binary prediction gives the abstract template called a *Bayes classifier*. It can be formulated as

$$\hat{y} = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 1/2, \\ 0 & \text{otherwise,} \end{cases} \quad (9.11)$$

$$= \begin{cases} 1 & \text{if } p(y = 1|\mathbf{x}) > p(y = 0|\mathbf{x}), \\ 0 & \text{otherwise,} \end{cases} \quad (9.12)$$

$$= \begin{cases} 1 & \text{if } \pi_1 p_1(\mathbf{x}) > \pi_0 p_0(\mathbf{x}), \\ 0 & \text{otherwise.} \end{cases} \quad (9.13)$$

This is easily generalized to any number of classes  $K$ , since we can think of a  $g_k(\mathbf{x})$  for each class (in a two-class problem it is sufficient to consider  $g(\mathbf{x}) = g_1(\mathbf{x})$ ). The Bayes classifier is a template in the sense that one can plug in different types of model for the  $p_k$ 's and the  $\pi$ 's.



# Naive Bayes classifier

The Bayes classifier formalism presented above is conceptually simple, but can be very difficult to compute: in practice, the data  $\{\mathbf{x}\}$  above may be in many dimensions, and have complicated probability distributions. We can dramatically reduce the complexity of the problem by making the assumption that all of the attributes we measure are conditionally independent. This means that

$$p(x^i, x^j | y_k) = p(x^i | y_k) p(x^j | y_k), \quad (9.15)$$

$$p(y_k | x^0, x^1, \dots, x^N) = \frac{\prod_i p(x^i | y_k) p(y_k)}{\sum_j \prod_i p(x^i | y_j) p(y_j)}.$$

→ prior for a given class!

recall:

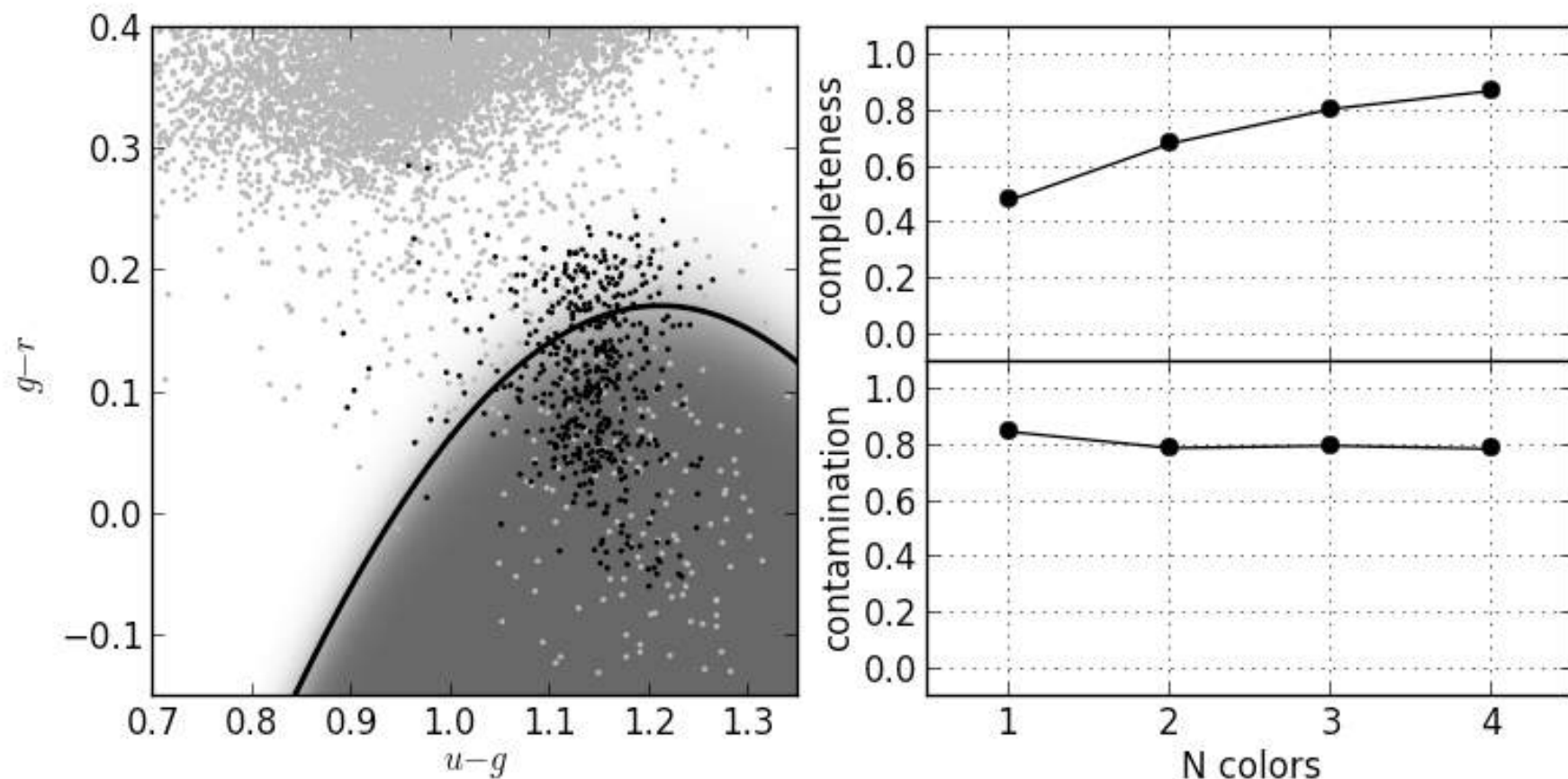
$$p(j | x_i) = \frac{\alpha_j \mathcal{N}(\mu_j, \sigma_j)}{\sum_{j=1}^M \alpha_j \mathcal{N}(\mu_j, \sigma_j)}$$

**Gaussian naive Bayes classifier**  
assumes that the data can be modeled  
by a sum of axis-aligned multi-variate  
gaussians

# Gaussian naive Bayes classification of RR Lyrae using colors

- made this plot by running  
`%run fig_rrlyrae_naivebayes.py`

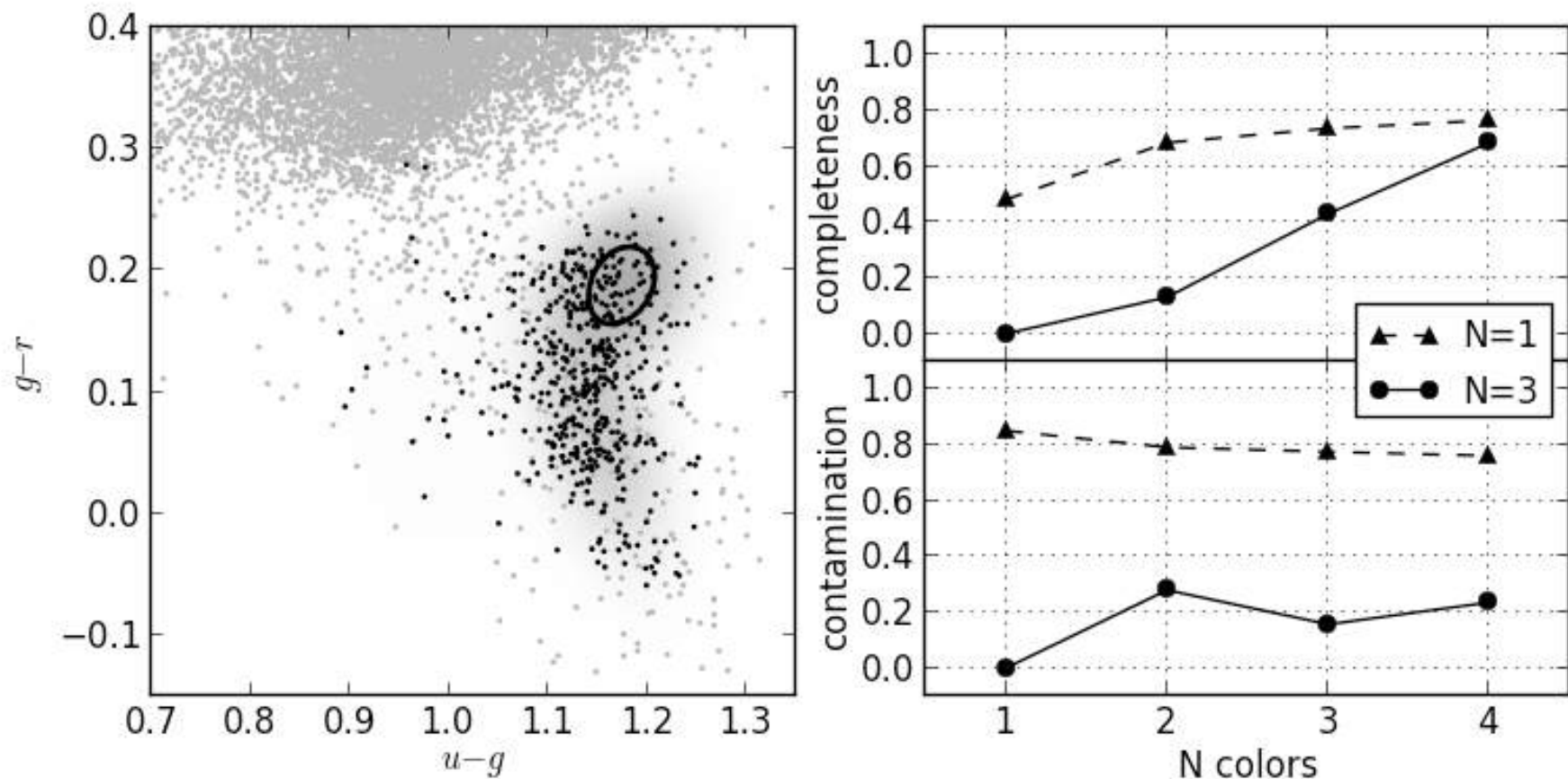
Black dots are 483 RR Lyrae, gray dots are the much larger sample of ~93,000 other stars (for this real-world dataset it is impossible to achieve perfect classification!)



# Gaussian Mixture Bayes classification of RR Lyrae

- made this plot by running  
`%run fig_rrlyrae_GMMbayes.py`

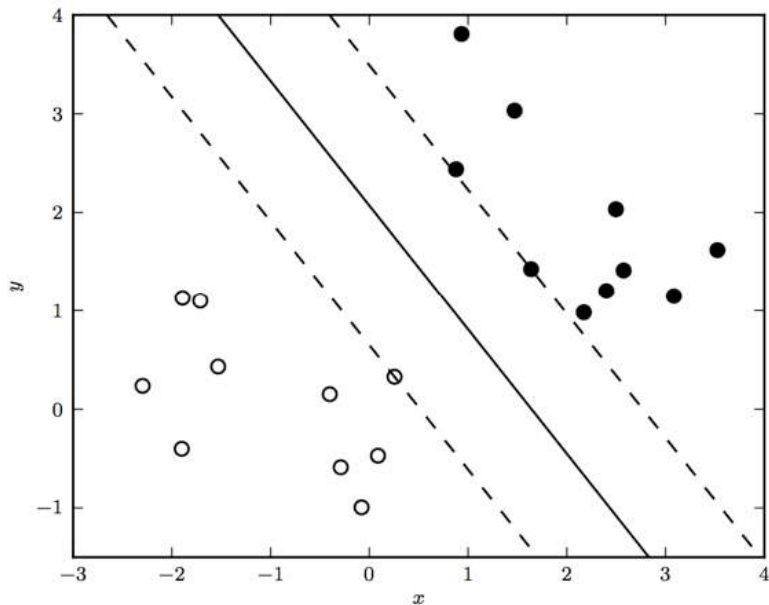
GMM Bayes classifier does **not** assume that gaussian components are aligned with coordinate axes



# Discriminative classification

With nearest-neighbor classifiers we started to see a subtle transition—while clearly related to Bayes classifiers using variable-bandwidth kernel estimators, the class density estimates were skipped in favor of a simple classification decision. This is an example of *discriminative classification*, where we directly model the decision boundary between two or more classes of source. Recall, for  $y \in \{0, 1\}$ , the discriminant function is given by  $g(x) = p(y = 1|x)$ . Once we have it, no matter how we obtain it, we can use the rule

$$\hat{y} = \begin{cases} 1 & \text{if } g(x) > 1/2, \\ 0 & \text{otherwise,} \end{cases} \quad (9.28)$$



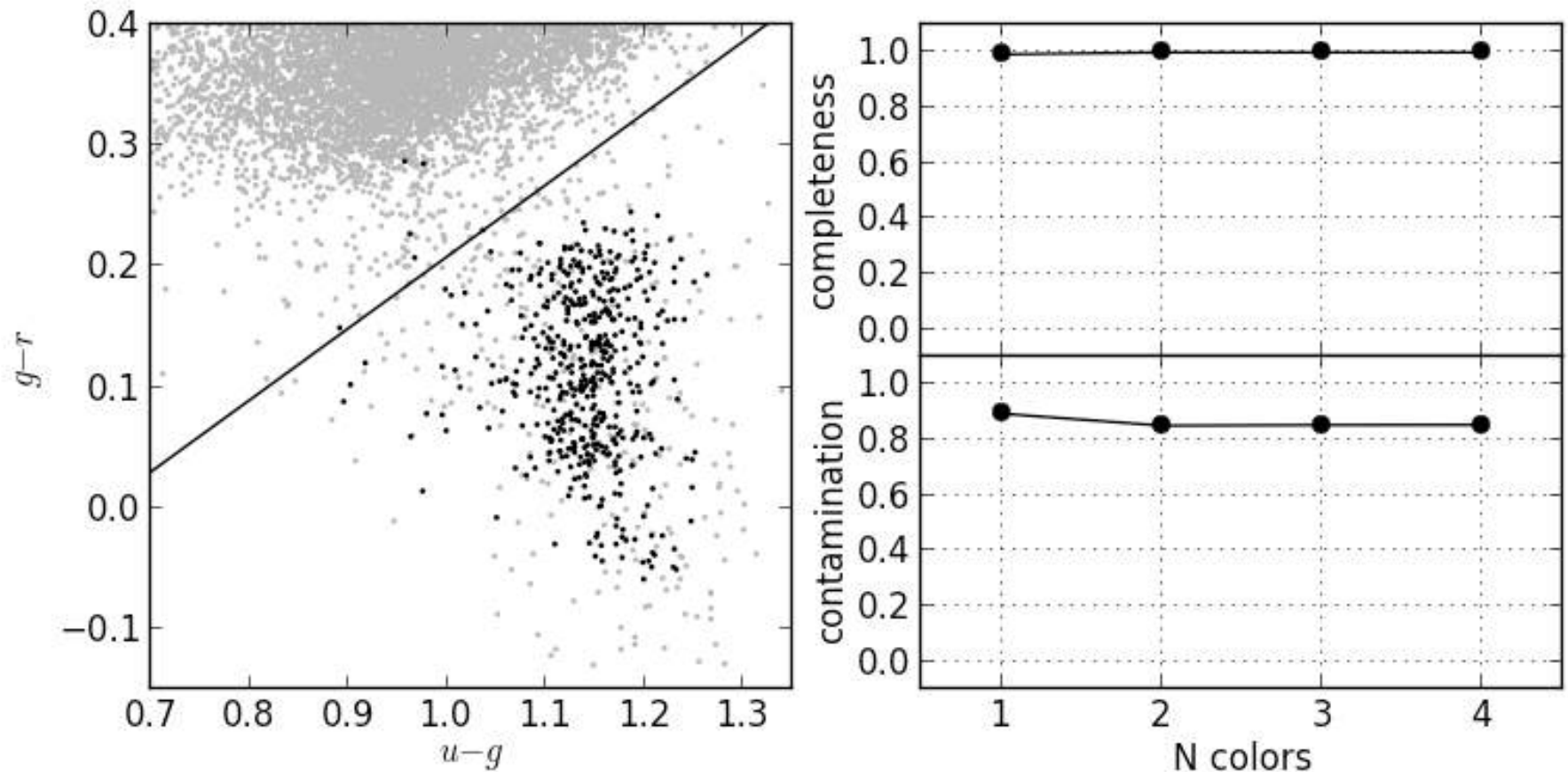
**Left:** the idea behind the Support Vector Machine classifier: a collection of hyperplanes which maximize the class separation

Figure 9.9.: Illustration of SVM. The region between the dashed lines is the *margin*, and the points which the dashed lines touch are called the *support vectors*.



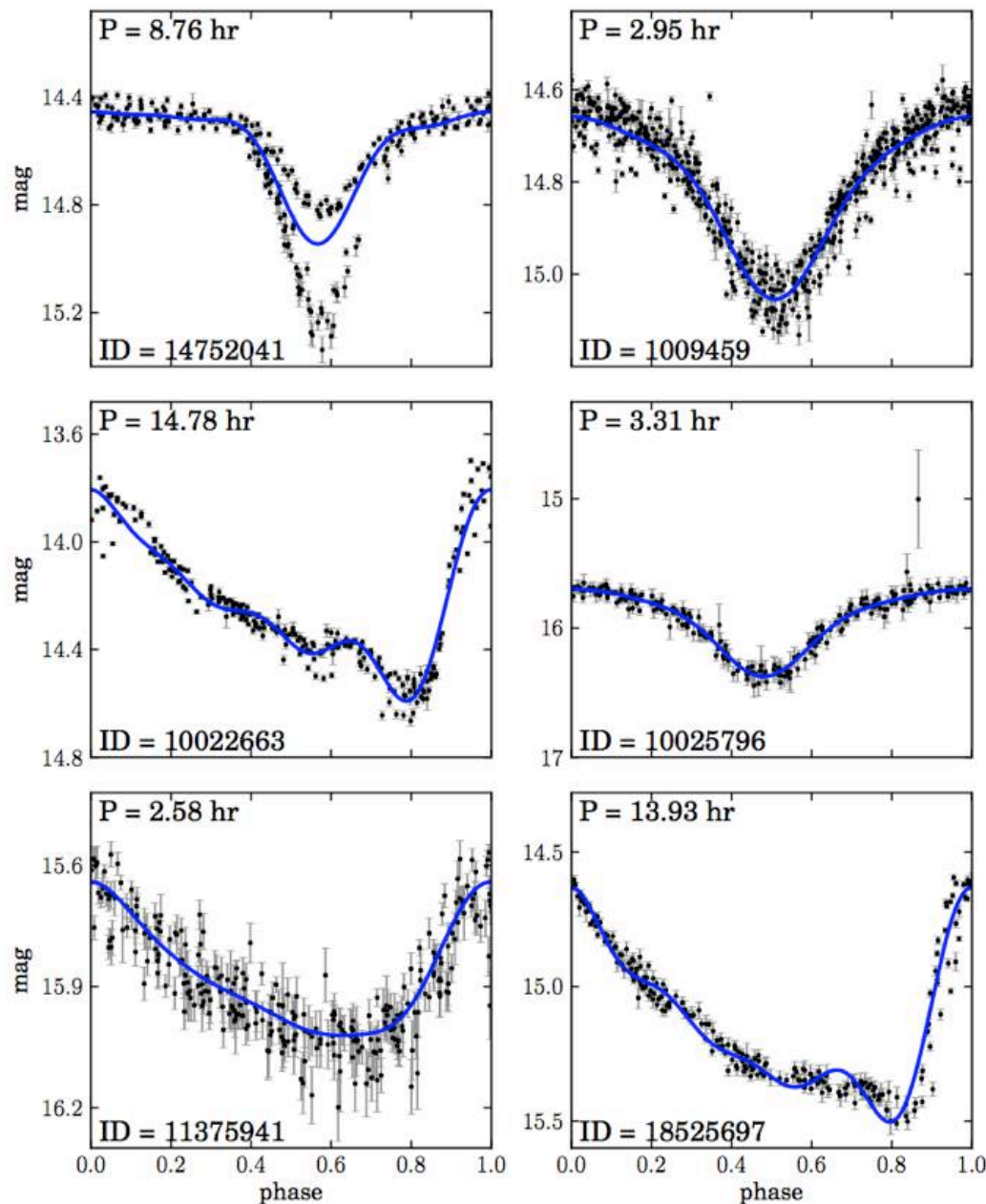
# Support Vector Machine classification of RR Lyrae

- made this plot by running (it takes a few minutes!)  
`%run fig_rrlyrae_svm.py`



SVM is very robust to outliers; note the very high completeness (at the expense of high contamination)

# Light curves (flux vs. time) of variable stars

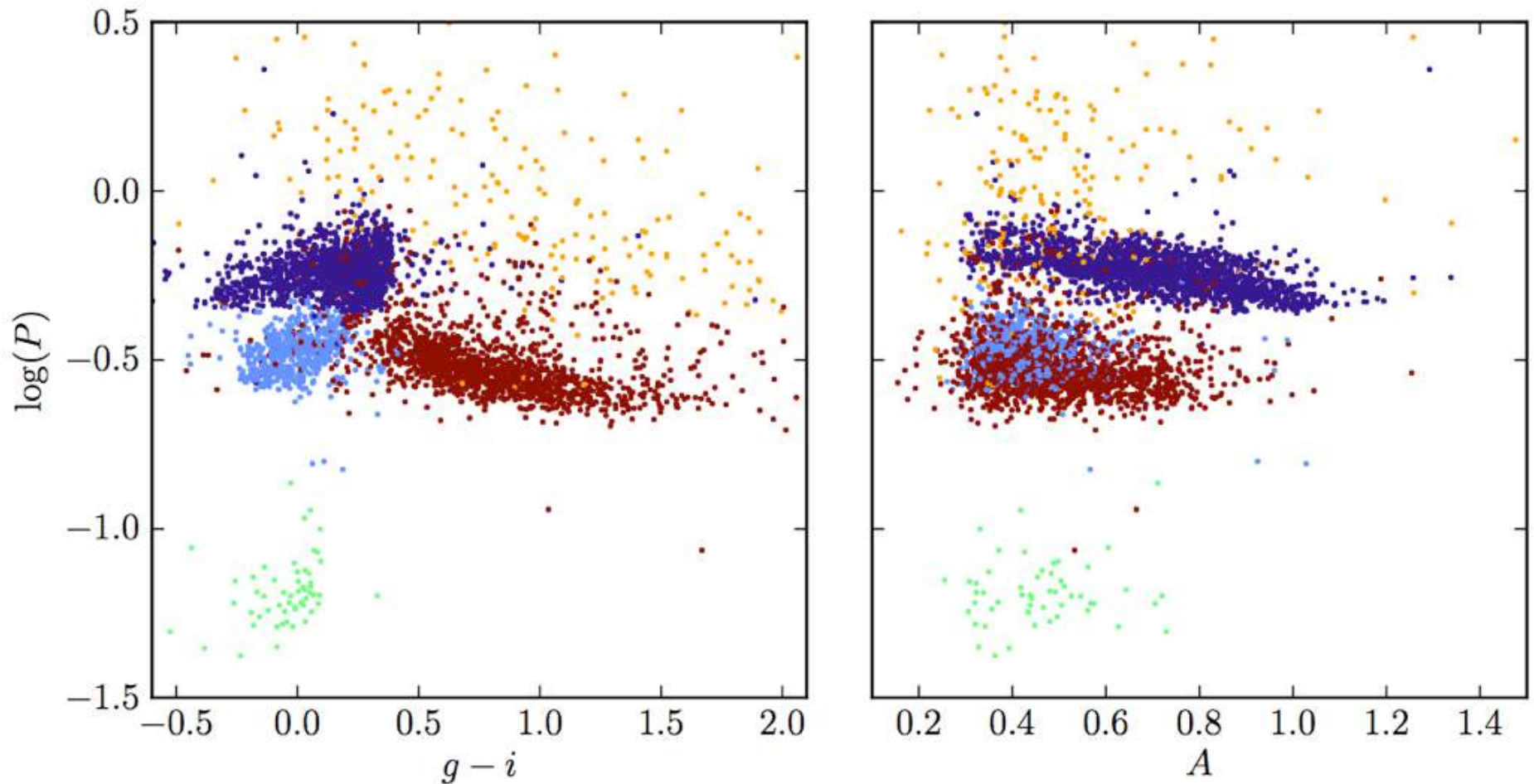


## Light-curve analysis:

- 1) find the best period using periodogram
- 2) for this period, find the best-fit k-term Fourier expansion
- 3) compute the  $\chi^2$ , amplitude, mean flux, mean color, etc.

# SVM classification of variable stars from Palaversa+ 2013

- Based on 7 attributes (4 colors and 3 light curve parameters: period, amplitude, skewness) and 5 input classes (also in astroML - you can experiment later...)



**Break here, go to notebook...**