Lecture 4 - Inference, Classification, dimensional reduction

See https://github.com/jbrinchmann/MLD2025 as usual

Lectures/Lecture 4 has the PDF for today

Classical versus Bayesian inference

Frequentist (classical)

- Probabilities refer to *relative frequencies of events*. They are objective properties of the real world.
- Parameters are *fixed, unknown constants*. Because they are not fluctuating, probability statements about parameters are meaningless.
- Statistical procedures should have well-defined long-run frequency properties.

Bayesian

- Probability describes the degree of subjective belief, not the limiting frequency. Probability statements can be made about things *other than data,* including model parameters and models themselves.
- Inferences about a parameter are made by producing its probability distribution - *this distribution quantifies the uncertainty of our knowledge about that parameter*. Various point estimates, such as expectation value may then be extracted from this distribution

(based on Statistics, Data mining & Machine Learning in Astronomy)

Classical inference

The broad idea

We want to compare data, $\{y_i\}$, to a model $M(a, b, \ldots)$

To do so we need:

A way to compare our data to the model (cost functions)

e.g.:

$$\int \left(f(x) - f_{\text{true}}(x)\right)^2 \, dx$$

$$\left|f(x) - f_{\text{true}}(x)\right|$$

A way to rank/compare different models Likelihood ratios, information criteria (AIC) etc.

A way to assess whether the fit we obtained is "good" e.g. χ^2 tests

This is arguably the approach most widely used in Machine Learning/Data Mining.

Maximum Likelihood

How were the data generated?

$p(D|M)$

1. Define a likelihood of the data given a model

   I will write the parameters of the model $\boldsymbol{\theta}$ and the model $M(\boldsymbol{\theta})$

   sometimes $\hat{}$

2. Find $\boldsymbol{\theta} = \boldsymbol{\theta}^0$ that maximise $p(D\,|\,M)$

   point estimates

   For this we use a minimization routine (e.g. within **`scipy.optimize`** such as curvefit, minimize or nnls)

3. Find confidence levels for $\boldsymbol{\theta}^0$

$$\sigma^2_{j,k} = -\left.\frac{d^2 \ln L}{d\theta_j d\theta_k}\right|_{\theta=\theta_0}$$

ML - Simple example - Gaussian errors

$\boldsymbol{\theta} = (\alpha, \beta)$

Model y as $y_i = \alpha + \beta x_i + \epsilon_i$ $\epsilon_i \sim N(0, \sigma_i^2)$

Likelihood of data:

$$P(x_i, y_i, \sigma_i | \alpha, \beta) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-[y_i - (\alpha + \beta x_i)]^2 / 2\sigma_i^2}$$

ML - Simple example - Gaussian errors

$\boldsymbol{\theta} = (\alpha, \beta)$

Model y as $$y_i = \alpha + \beta x_i + \epsilon_i \qquad \epsilon_i \sim N(0, \sigma_i^2)$$

Likelihood of data:

$$P(x_i, y_i, \sigma_i | \alpha, \beta) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-[y_i - (\alpha + \beta x_i)]^2 / 2\sigma_i^2}$$

Full Likelihood $$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\sigma} | \alpha, \beta) = \prod_i P(x_i, y_i, \sigma_i | \alpha, \beta)$$

ML - Simple example - Gaussian errors

$\boldsymbol{\theta} = (\alpha, \beta)$

Model y as $$y_i = \alpha + \beta x_i + \epsilon_i \qquad \epsilon_i \sim N(0, \sigma_i^2)$$

Likelihood of data:

$$P(x_i, y_i, \sigma_i | \alpha, \beta) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-[y_i - (\alpha + \beta x_i)]^2 / 2\sigma_i^2}$$

Full Likelihood $$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\sigma} | \alpha, \beta) = \prod_i P(x_i, y_i, \sigma_i | \alpha, \beta)$$

So log likelihood: $$\ln L = -\frac{1}{2} \ln 2\pi - \frac{1}{2} \sum_i \frac{[y_i - (\alpha + \beta x_i)]^2}{\sigma_i^2}$$

ML - Simple example - Gaussian errors

$\boldsymbol{\theta} = (\alpha, \beta)$

Model y as $$y_i = \alpha + \beta x_i + \epsilon_i \qquad \epsilon_i \sim N(0, \sigma_i^2)$$

Likelihood of data:

$$P(x_i, y_i, \sigma_i | \alpha, \beta) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-[y_i - (\alpha + \beta x_i)]^2 / 2\sigma_i^2}$$

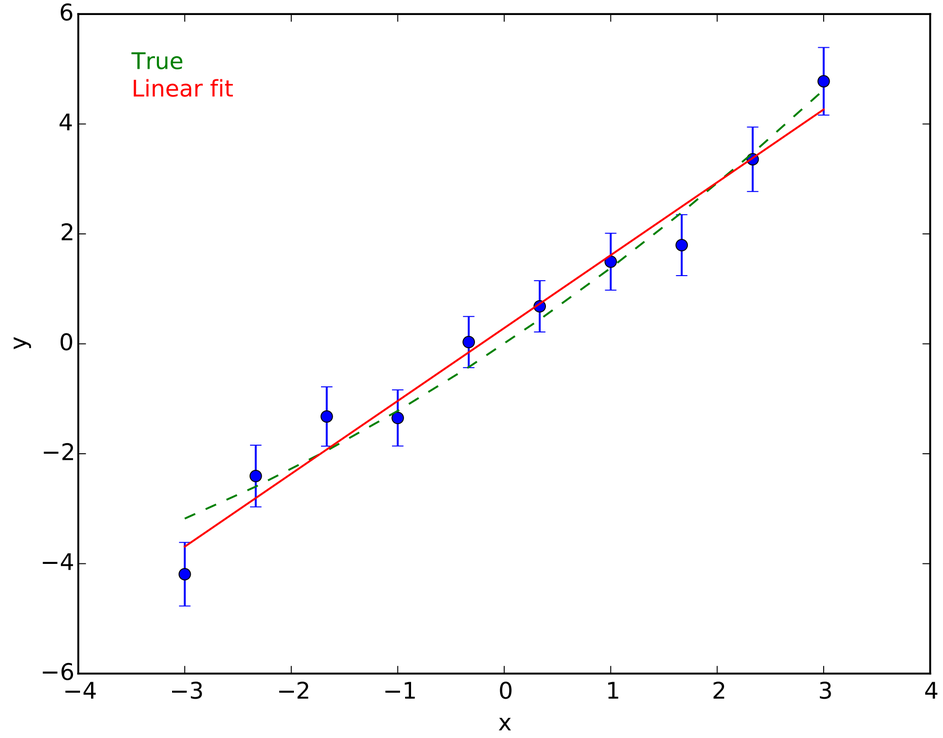
Full Likelihood $$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\sigma} | \alpha, \beta) = \prod_i P(x_i, y_i, \sigma_i | \alpha, \beta)$$

So log likelihood: $$\ln L = -\frac{1}{2} \ln 2\pi - \frac{1}{2} \sum_i \frac{[y_i - (\alpha + \beta x_i)]^2}{\sigma_i^2}$$

Minimisation of this is the same as least squares minimisation.

ML - Simple example - Gaussian errors, standard function

```
from scipy.optimize import curve_fit

def func_line(x, a, b):
    return a + b*x

pars, cov = curve_fit(func_line, x, y_obs)
```



This is equivalent to the regression solution you have seen before:

```
from astroML.linear_model import LinearRegression
m = LinearRegression()
m.fit(x[:, None], y_obs, sigma)
a, b = m.coef_
```

ML - Simple example - explicit likelihood

```
def neglnL(theta, x, y, yerr):
    a, b = theta
    model = b * x + a
    inv_sigma2 = 1.0/(yerr**2)

    return 0.5*(np.sum((y-model)**2*inv_sigma2))
```

$$-\ln L = \frac{1}{2}\sum_i \frac{\left[y_i - (\alpha + \beta x_i)\right]^2}{\sigma_i^2}$$

```
import scipy.optimize as op
result = op.minimize(neglnL, [1.0, 0.0], args=(x, y_obs, sigma))
a_ml, b_ml = result["x"]
```

More work in this case - but a much more general approach.

Bayesian inference

Bayesian statistics

This allows us to calculate the likelihood distribution of parameters

$$p(\text{Model}|\text{Data}) = p(M|D) = \frac{p(D|M)p(M)}{p(D)}$$

The key part here is that we need to specify what **prior** information we have on the model parameters.

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

The overall plan of attack

- Define the likelihood - the best is a generative one that mimic how you think the data were created. $p(D|\boldsymbol{\theta}, M, I)$
- Decide on the prior - ie. what range of model parameters are likely. $p(\boldsymbol{\theta}, M|I)$
- Use Bayes' theorem to calculate the posterior likelihood distribution. $p(M|D, I)$
- To calculate the posterior distribution we often use Markov Chain Monte Carlo calculations. MCMC

The result of the calculation can be summarised - the maximum of $p(M|D,I)$ gives the maximum a posteriori (**MAP**) estimate - means, medians are also reasonable options.

A practical example - line fitting - model

Generative model: $y_i = \alpha + \beta x_i + \epsilon_i \qquad \epsilon_i \sim N(0, \sigma_i^2)$

So log likelihood: $$\ln L = -\frac{1}{2}\ln 2\pi - \sum_i \ln \sigma_i - \frac{1}{2}\sum_i \frac{[y_i - (\alpha + \beta x_i)]^2}{\sigma_i^2}$$

(note that since σ_i are known, the first two terms are constants and can be ignored for maximisation

```
def lnL(theta, x, y, yerr):
    a, b = theta
    model = b * x + a
    inv_sigma2 = 1.0/(yerr**2)

    return -0.5*(np.sum((y-model)**2*inv_sigma2))
```

A practical example - line fitting - prior

Prior:

$$p(a, b, M|I) = \begin{cases} \text{const}, & \text{if } a \in [-5, 5] \text{ and } b \in [-10, 10]; \\ 0 & \text{otherwise} \end{cases}$$

```
def lnprior(theta):
    a, b = theta
    if -5.0 < a < 5.0 and -10.0 < b < 10.0:
        return 0.0
    return -np.inf
```

(the -np.inf is because p=0 means ln p = -∞)

A practical example - line fitting - posterior

Putting it together: $p(D|a,b,M,I)p(a,b,M|I)$

```
def lnprob(theta, x, y, yerr):
    """
    The likelihood to include in the MCMC.
    """
    lp = lnprior(theta)
    if not np.isfinite(lp):
        return -np.inf
    return lp + lnL(theta, x, y, yerr)
```

Note that I ignore the normalisation $p(D|I)$

A practical example - line fitting

```
import emcee

# Use ML to get a starting point.
# result = run_ml()
p_init = np.array([ 0.28233725,  1.31299656])

# Set up the properties of the problem.
ndim, nwalkers = 2, 100

# Setup a number of initial positions.
pos = [p_init + 1e-4*np.random.randn(ndim) for i in range(nwalkers)]

# Create the sampler.
sampler = emcee.EnsembleSampler(nwalkers, ndim, lnprob, args=(x, y_obs, sigma))

# Run the process.
sampler.run_mcmc(pos, 500)
```

A practical example - importing the package

```
import emcee
```

We will use the MCMC Hammer library (emcee) -
http://dan.iel.fm/emcee/current/

it is a pure Python implementation (as pymc3), flexible and reasonably fast - for challenging problems other packages (MultiNest, BUGS, JAGS, STAN) might be better choices but it is well worth starting with this as the learning curve is less steep.

On your laptop or in Google Colab you can install it with:

pip install emcee
pip install corner

A practical example - starting position

```
# Use ML to get a starting point.
# result = run_ml()
p_init = np.array([ 0.28233725,  1.31299656])
```

The MCMC process will randomly (but cleverly) step around in your parameter space. To do this well you need a good starting position. A maximum-likelihood solution gives a good starting point.

I used scipy.optimize.minimize here. I set the result to a variable p_init.

A practical example - line fitting

```
# Set up the properties of the problem.
ndim, nwalkers = 2, 100
```

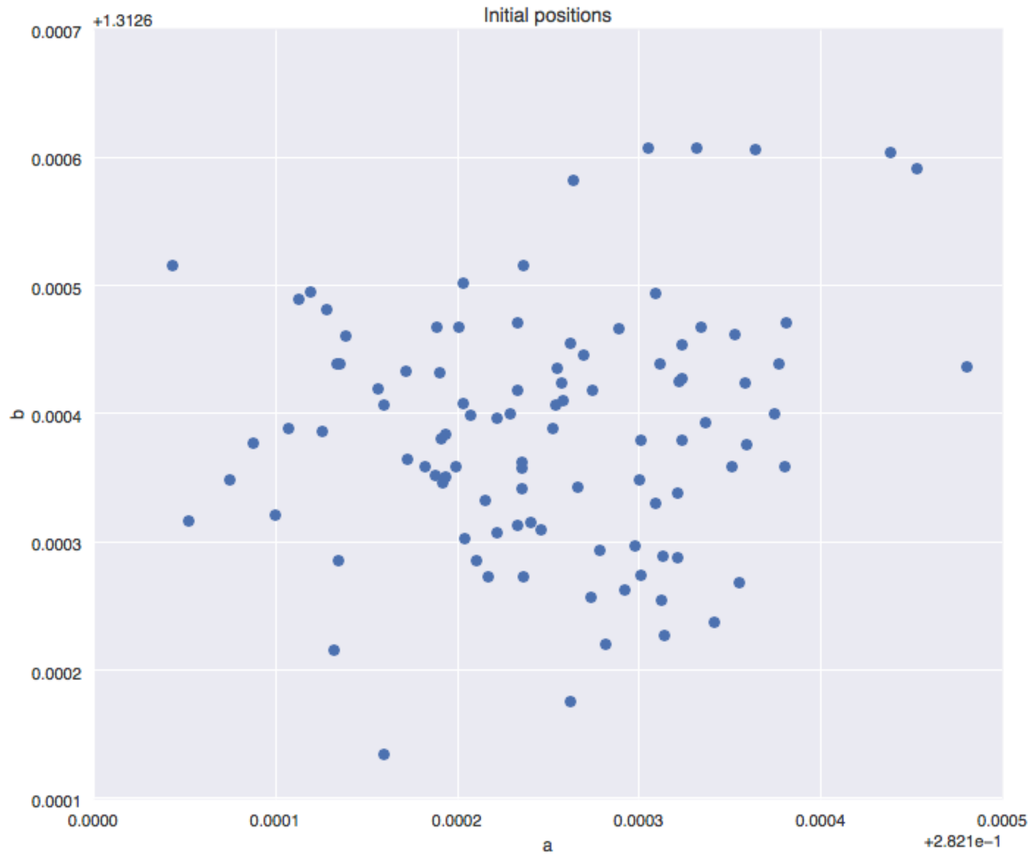
We have two parameters - so the dimensionality is 2

And we also need “walkers” - view these as random processes that explore your parameter space along different paths. The **emcee** documentation recommends using as many as you can get away with.

A practical example - line fitting

```
# Setup a number of initial positions.
pos = [p_init + 1e-4*np.random.randn(ndim) for i in range(nwalkers)]
```

This creates a set of different starting positions - one for each walker.



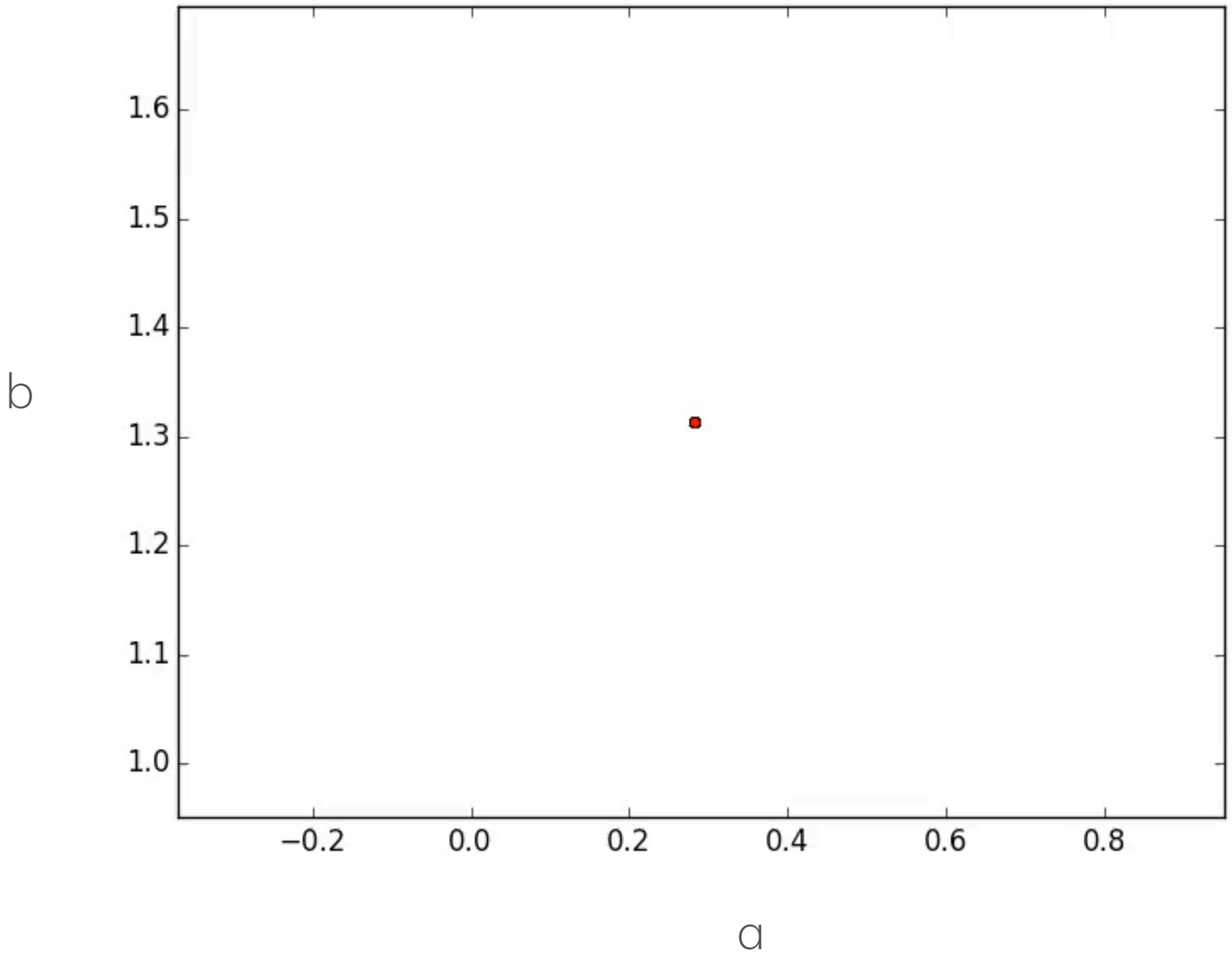
A practical example - line fitting

```
# Create the sampler.
sampler = emcee.EnsembleSampler(nwalkers, ndim, lnprob, args=(x, y_obs, sigma))
```

This sets up the MCMC process - we give the function `lnprob` and the data as a tuple.

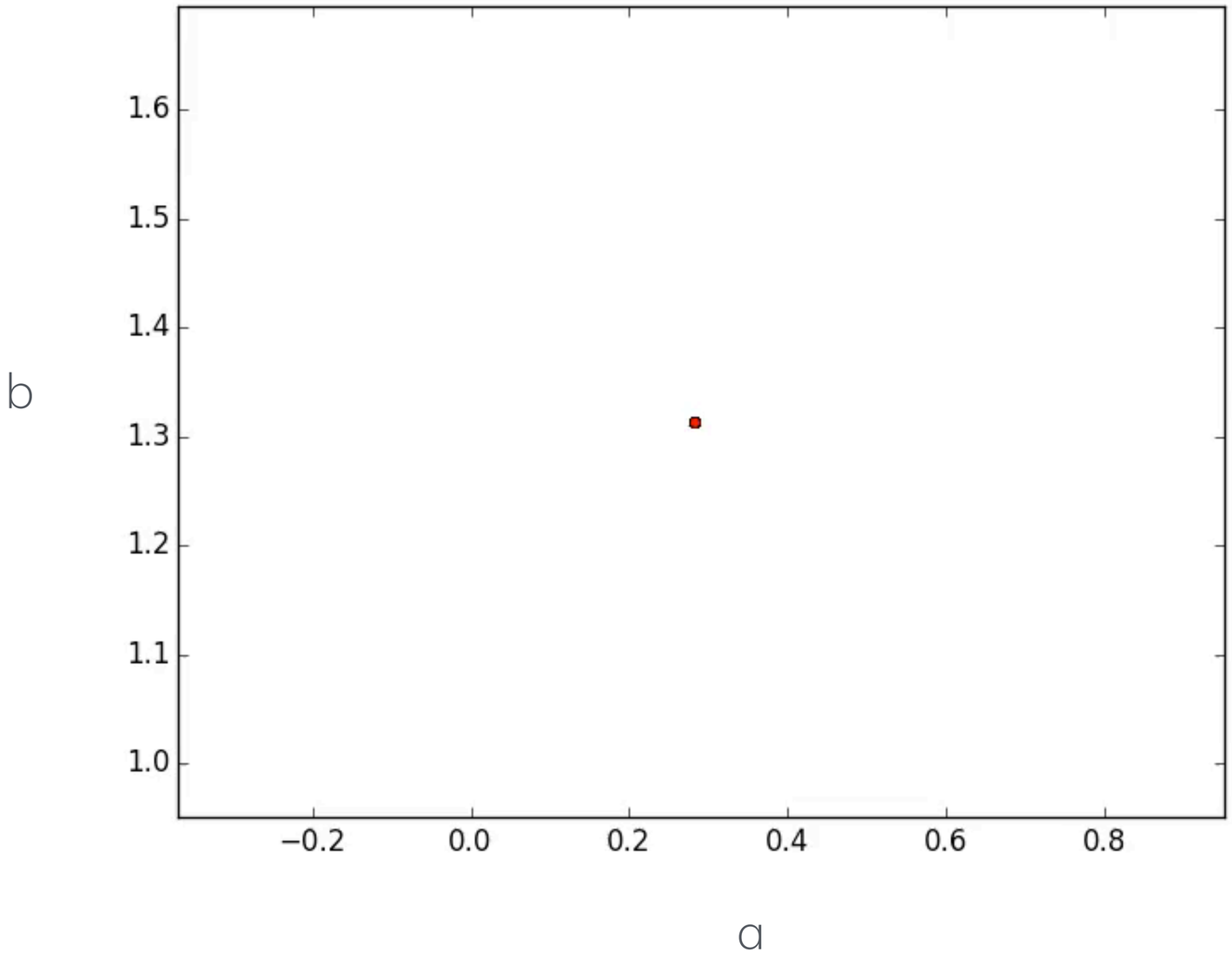
A practical example - line fitting

```
# Run the process.
sampler.run_mcmc(pos, 500)
```



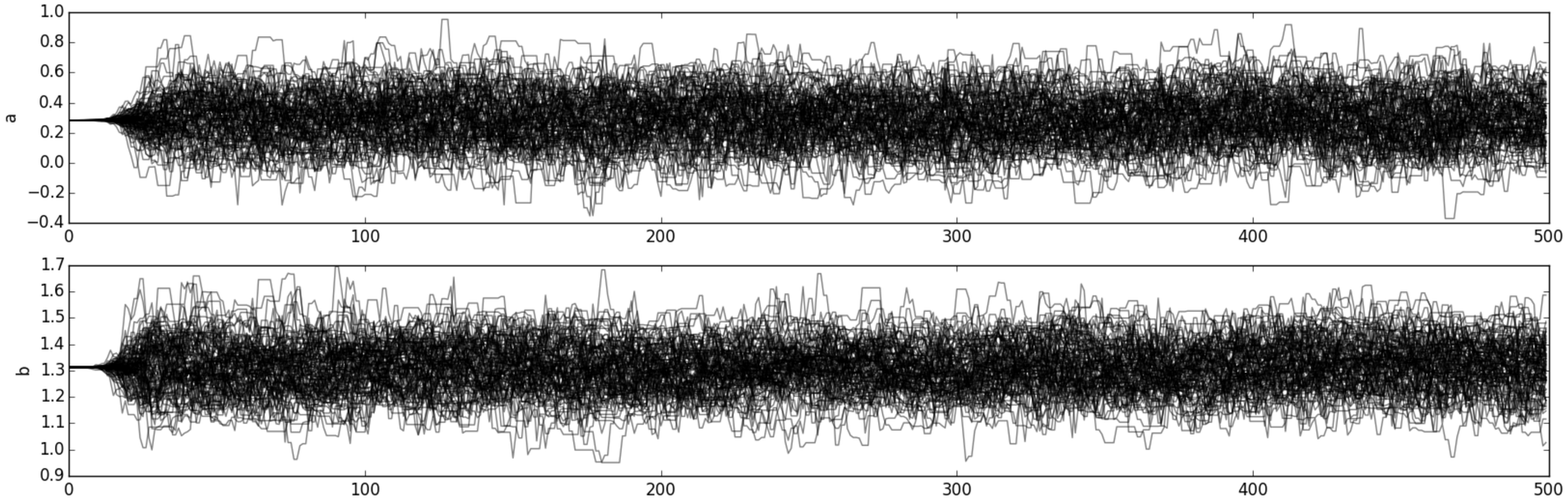
A practical example - line fitting

```
# Run the process.
sampler.run_mcmc(pos, 500)
```



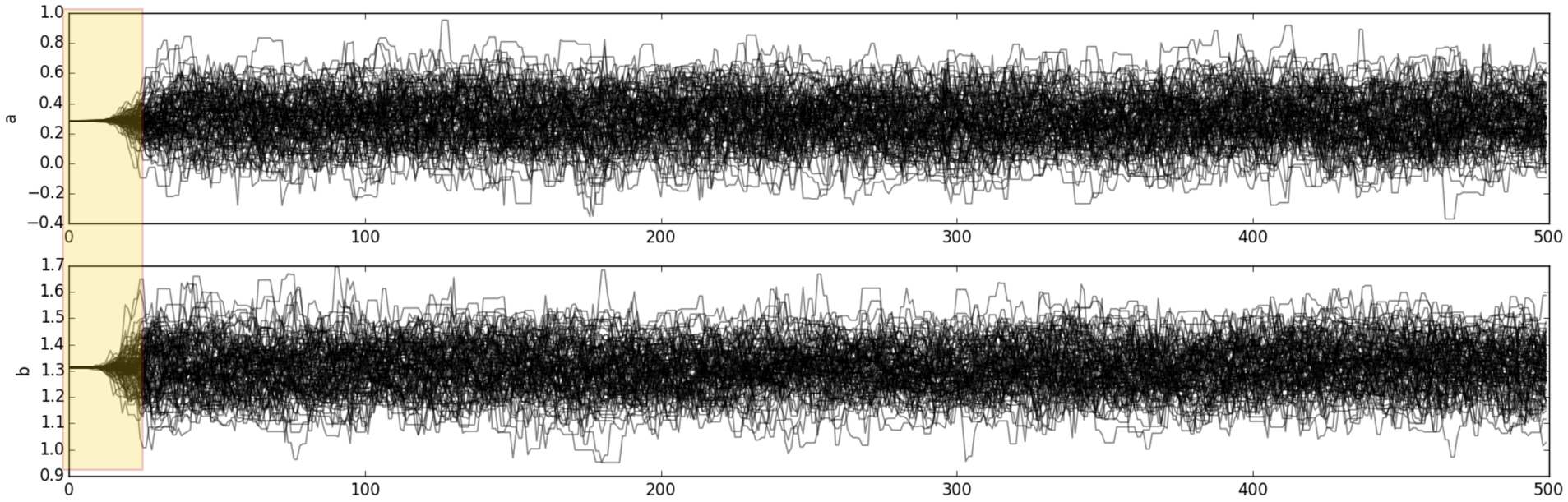
Look at chain behaviour

```
labels = [‘a’, ‘b’]
chain = sampler.chain
for i_dim in range(2):
    plt.subplot(2,1,i_dim+1)
    plt.ylabel(labels[i_dim])

    for i in range(100):
        plt.plot(chain[i,:,i_dim],color='black', alpha=0.5)
```



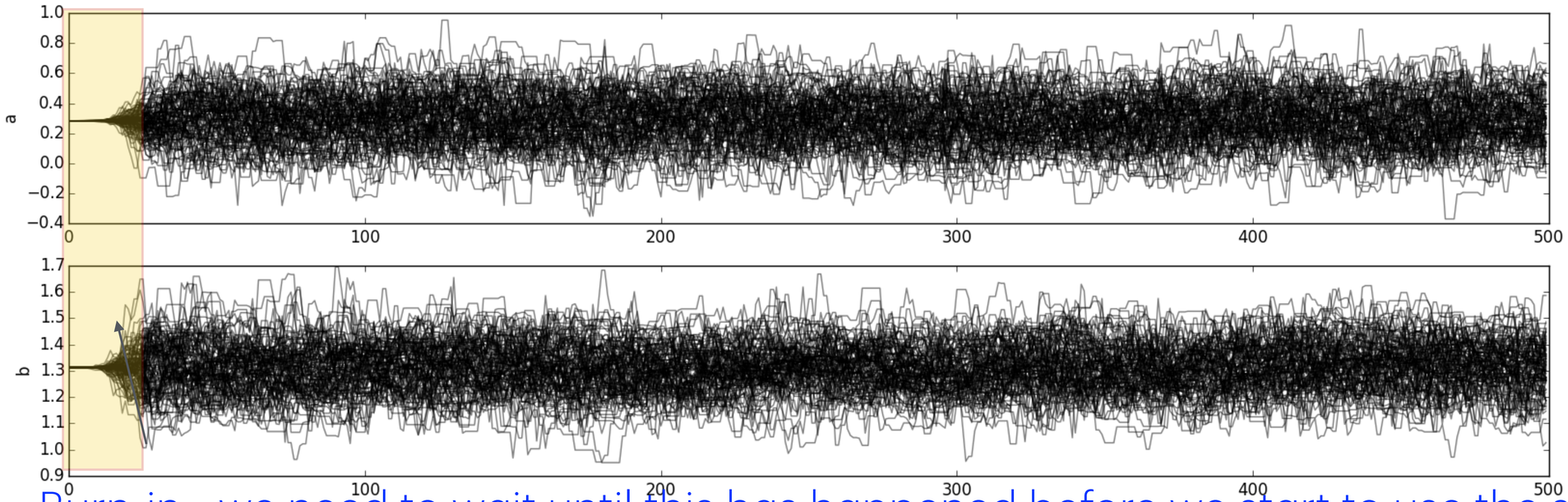
Look at chain behaviour

```
labels = [‘a’, ‘b’]
chain = sampler.chain
for i_dim in range(2):
    plt.subplot(2,1,i_dim+1)
    plt.ylabel(labels[i_dim])

    for i in range(100):
        plt.plot(chain[i,:,i_dim],color='black', alpha=0.5)
```



Look at chain behaviour

```
labels = [‘a’, ‘b’]
chain = sampler.chain
for i_dim in range(2):
    plt.subplot(2,1,i_dim+1)
    plt.ylabel(labels[i_dim])

    for i in range(100):
        plt.plot(chain[i,:,i_dim],color='black', alpha=0.5)
```



Burn-in - we need to wait until this has happened before we start to use the samples.

A practical example - showing the result

Let us extract the samples from 50 onwards and collapse the different walkers

```
samples = sampler.chain[:, 50:, :].reshape((-1, 2))
```

We then show the result using the corner package:

```
import corner

fig = corner.corner(samples, labels=["$a$", “$b$"],
                    truths=[0.0, 1.3], quantiles=[0.16, 0.84])
fig.show()
```

A practical example - showing the result

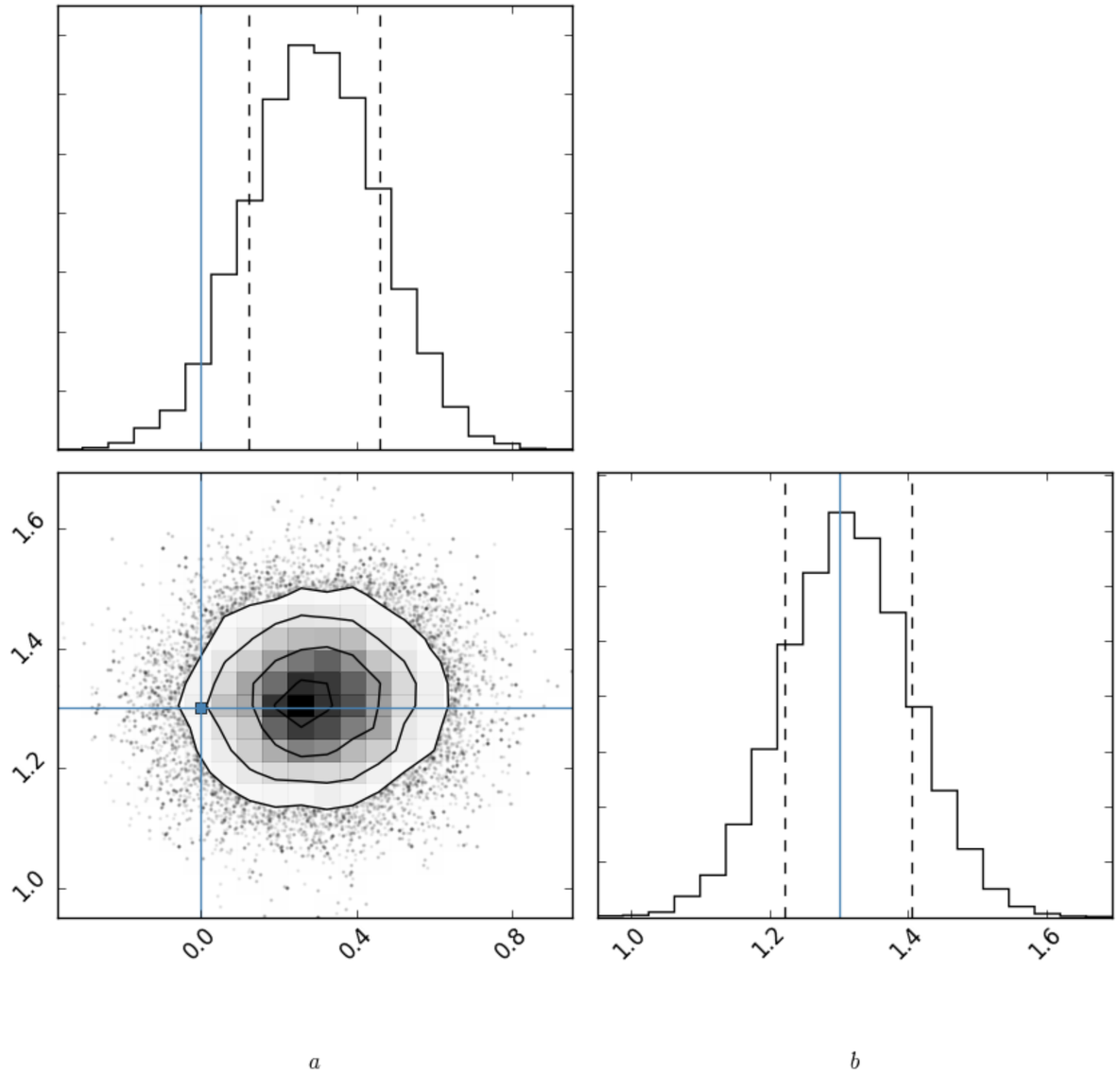
Let us extract

```
samples = s
```

nt walkers

We then show

```
import corn

fig = corne

fig.show()
```




```
16, 0.84])
```

Bayesian model selection

This is arguably the most rigorous but has its own problems.

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

What we want here is $p(M \,|\, D, I)$ so that we can compare against a different model. Thus we need to integrate out θ

This can be very challenging to impossible in multi-D situations.

Credible intervals vs confidence intervals

Frequentist view of the measurement process:

The probability is related to the frequency of repeated events.

In this view, models parameters are fixed and data are random, so we do not talk about the probability of the model parameter. The confidence interval is then random.

Bayesian way to look at it:

The probability is related to the degree of certainty about values.

In this view, model parameters are random (have a distribution) and data are fixed. Thus we can talk about the probability distribution of a parameter.

Credible intervals vs confidence intervals

Frequentists: confidence intervals

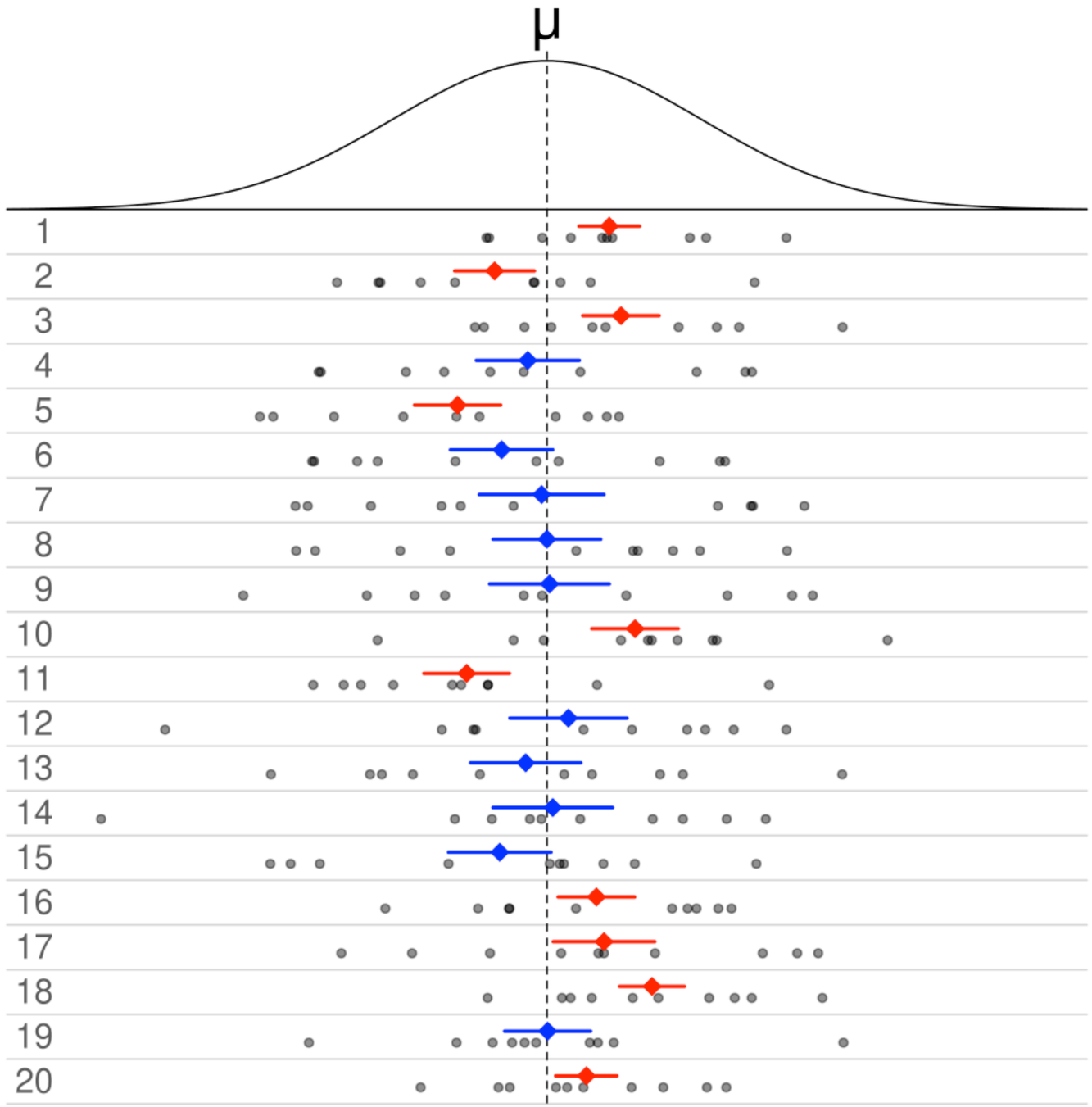
A range of values that are designed to include the true value of the parameter with some minimum probability (e.g. 95%). This is calculated based on the data and is a random variable as it depends on the data (viewed as random).

NB! This does not mean that there is a 95% probability that the true parameter lies within the interval. It does or does not in the frequentist interpretation.

Credibl ntervals

Frequentists: c

A range of e of the
parameter s
calculated t depends
on the datc



NB! This do the true
parameter frequentist
interpretati

Credible intervals vs confidence intervals

The Bayesian way - credible intervals:

An interval within which the true parameter value falls with a given (say 95%) probability.

Most of the time, that is what you want! But there are some subtleties - see next page!

For more details, Jake VanderPlas's write-up is interesting (and opinionated!):

https://jakevdp.github.io/blog/2014/06/12/frequentism-and-bayesianism-3-confidence-credibility/

And a (much) older consideration comes from E. T. Jaynes:

https://bayes.wustl.edu/etj/articles/confidence.pdf

Credible intervals - a subtle point:

It is not always clear what it means - two ways to define it:

1. Choose the narrowest interval containing a particular probability. This is the Highest Posterior Density Region
2. Choose the interval that has equal probability below and above the interval. This can be called the equal-tailed interval, or the Central Credible Region.

Credible intervals - a subtle point:

Highest Posterior Density Region

$$1 - \alpha = \int_{\theta : p(\theta|D) \geq p^*} p(\theta \,|\, D) d\theta$$

So find a p* so that the summed probability above this is equal to the probability we want.

Central Credible Region

$$C_\alpha(D) = (l, u) : P(l \leq \theta \leq u \,|\, D) = 1 - \alpha$$

Usually we want this to have equal probability mass below l and above u.
In which case this is an equal tail central credible region.

See this old Stackoverflow discussion on the topic which also has code: https://stackoverflow.com/questions/22284502/highest-posterior-density-region-and-central-credible-region

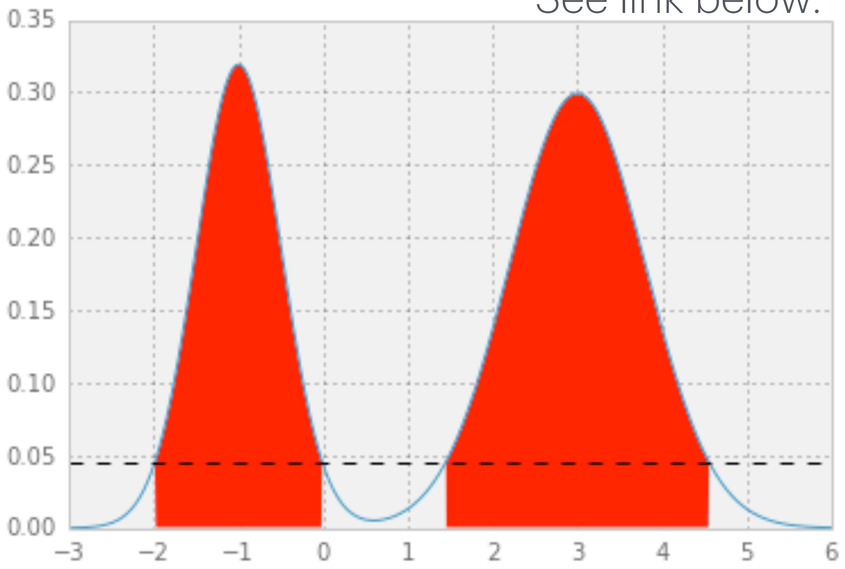
Credible intervals - a subtle point:

Highest Posterior Density Region

$$1 - \alpha = \int_{\theta : p(\theta|D) \geq p^*} p(\theta \,|\, D) d\theta$$

So find a p* so that the summed probability above this is equal to the probability we want.

See link below:



Central Credible Region

$$C_\alpha(D) = (l, u) : P(l \leq \theta \leq u \,|\, D) = 1 - \alpha$$

Usually we want this to have equal probability mass below l and above u.
In which case this is an equal tail central credible region.

See this old Stackoverflow discussion on the topic which also has code: https://stackoverflow.com/questions/22284502/highest-posterior-density-region-and-central-credible-region

Trying out MCMC

51 Peg b - the first confirmed exo-planet around a normal star

We will model radial velocities. Here we will consider a circular orbit with period P, inclination i, and the mass of the planet, M_P, and the star, $M_\star$. In this case the amplitude of the variation in the radial velocity, v_R, is given by

$$k = \left(\frac{2\pi G}{P}\right)^{1/3} \frac{M_P \sin i}{\left(M_P + M_\star\right)^{2/3}}$$

So we want P and k which we will estimate from the measured radial velocities using the following model:

$$v_R = k \sin\left(\frac{2\pi(t - t_0)}{P}\right) + v_0 = k \sin\left(2\pi(f + f_0)\right) + v_0$$

Thus in total we have four parameters: $\theta = (P, k, f_0, v_0)$

Trying out MCMC

1. Get the file 51Peg mayorqueloz95.dat.

The file provides the original dataset from 1995 which was used to find the exo-planet. The final consists of three columns of data. The first column gives the Julian date of the observations relative to some reference date. The second column gives v_R measured in meters per second and the third column gives the estimated uncertainty on v_R.

2. Get the notebook - Starting 51 Peg b MCMC - skeleton - from the Github site and run this.

Use this as a starting point for estimating the parameters - if you have time also try to get the planet mass.

A brief peek outside academia - Kaggle

Kaggle - data science competitions

But also a community - supposedly the largest data scientist community on the internet (>10^6 users)

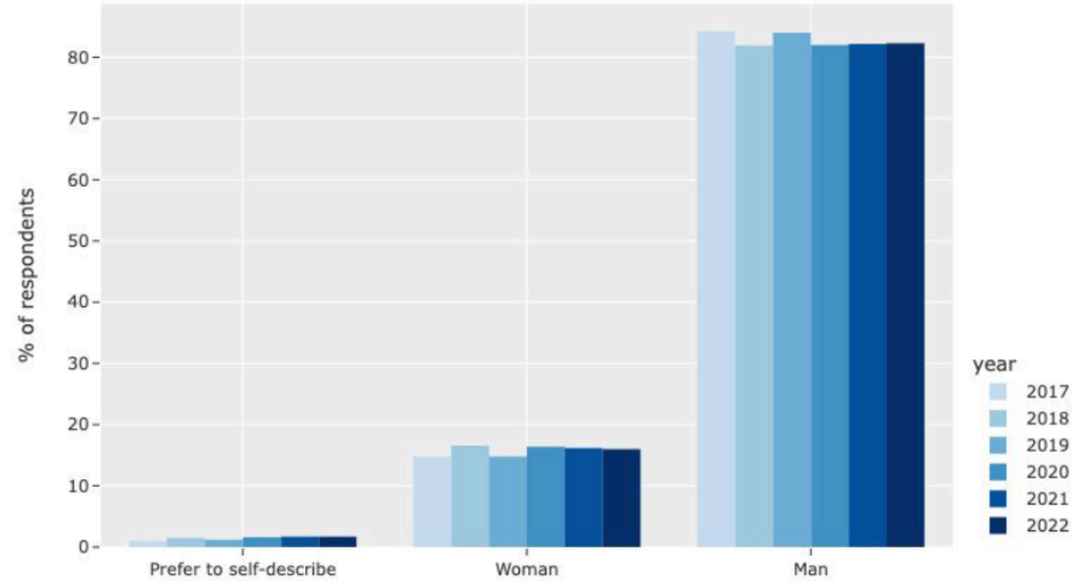
Lots of data sets (>296 000)

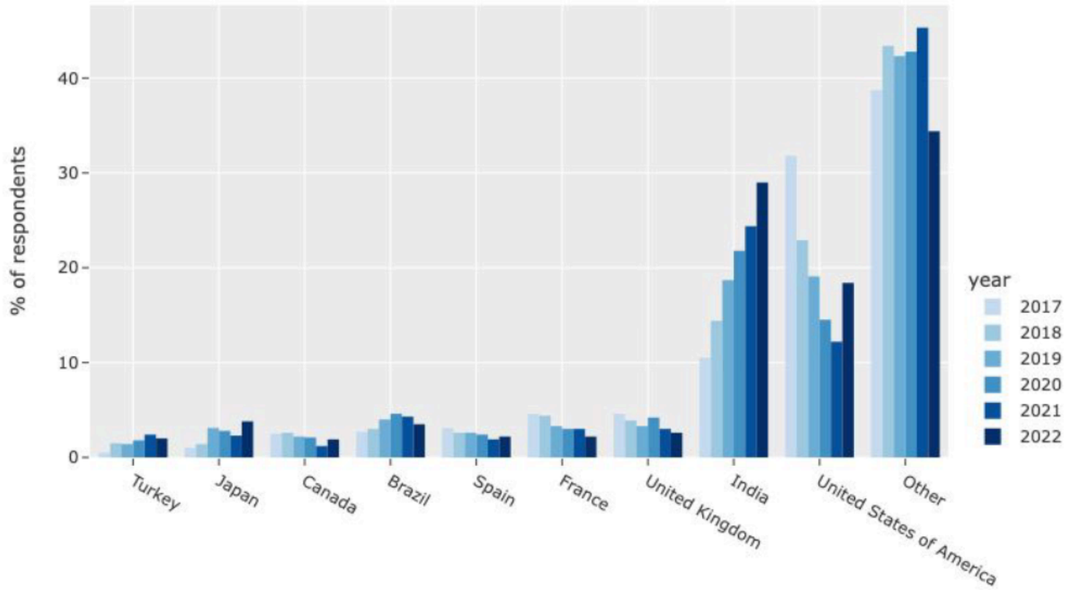
Lots of notebooks (>968 000)

Even if you do not want to compete, might be a good place to consult.

kaggle.com - note, owned by Google.

Kaggle's user survey (also freely available):



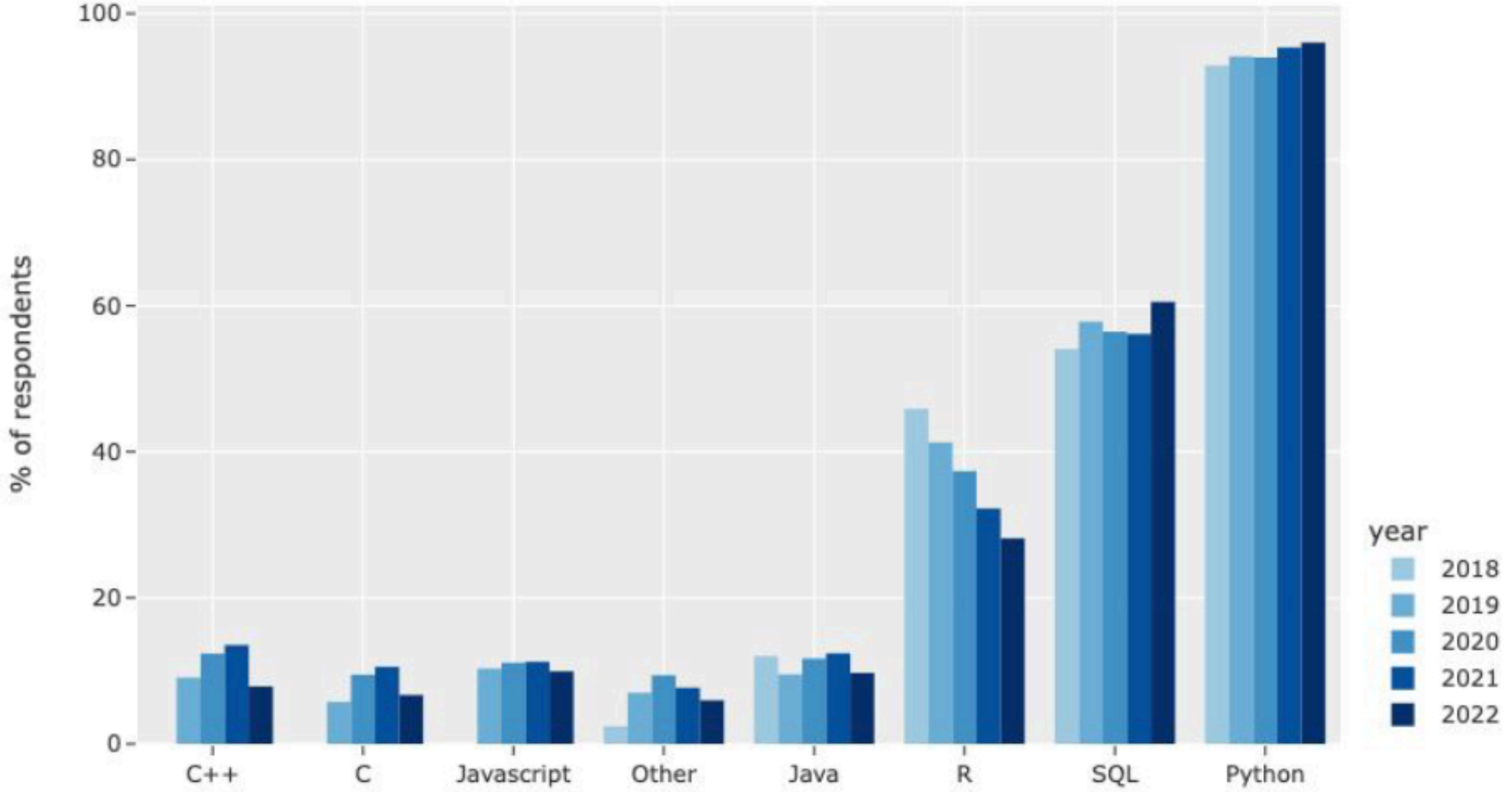


Are you surprised?

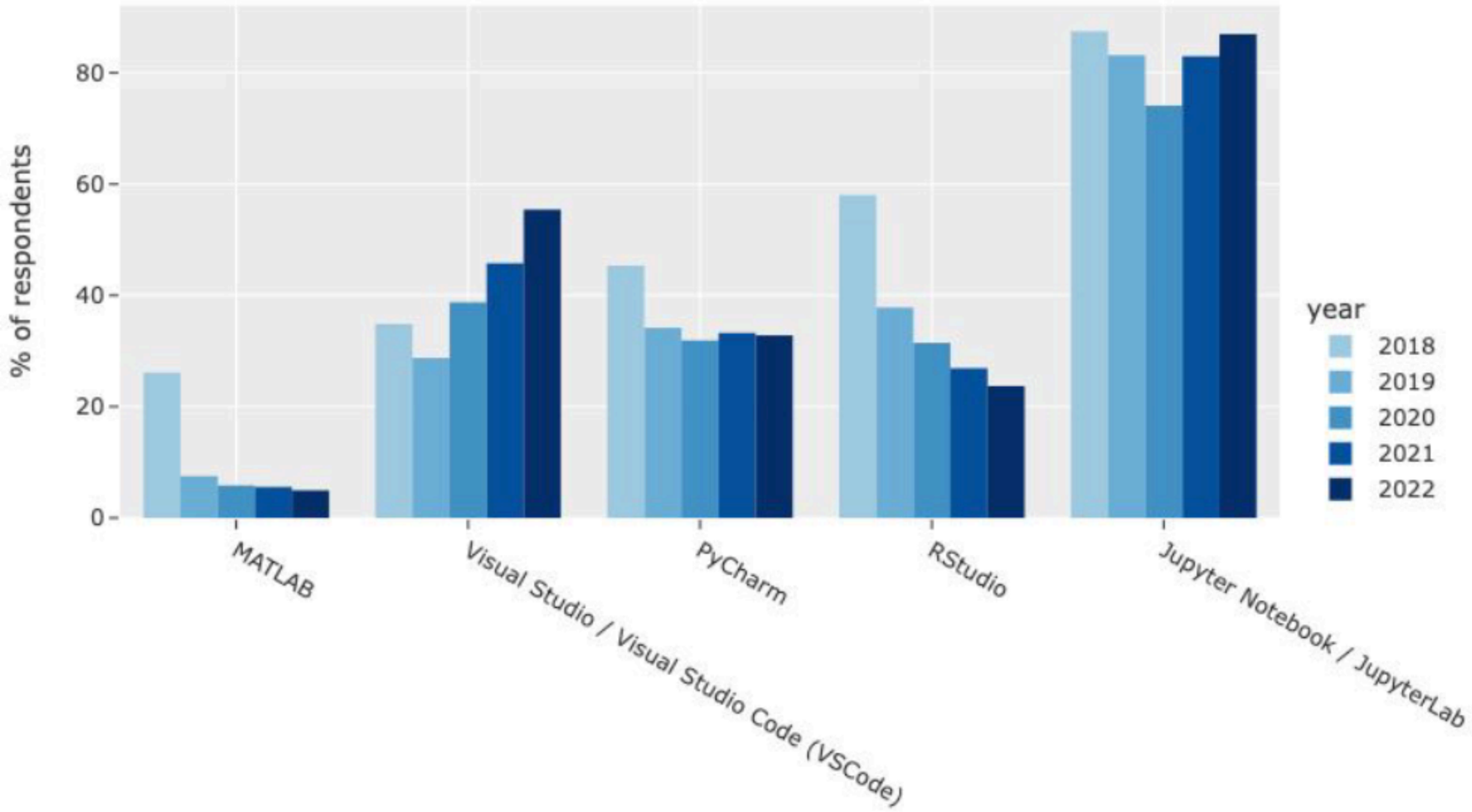
87 = 0.36%

from Portugal

Kaggle user's skills

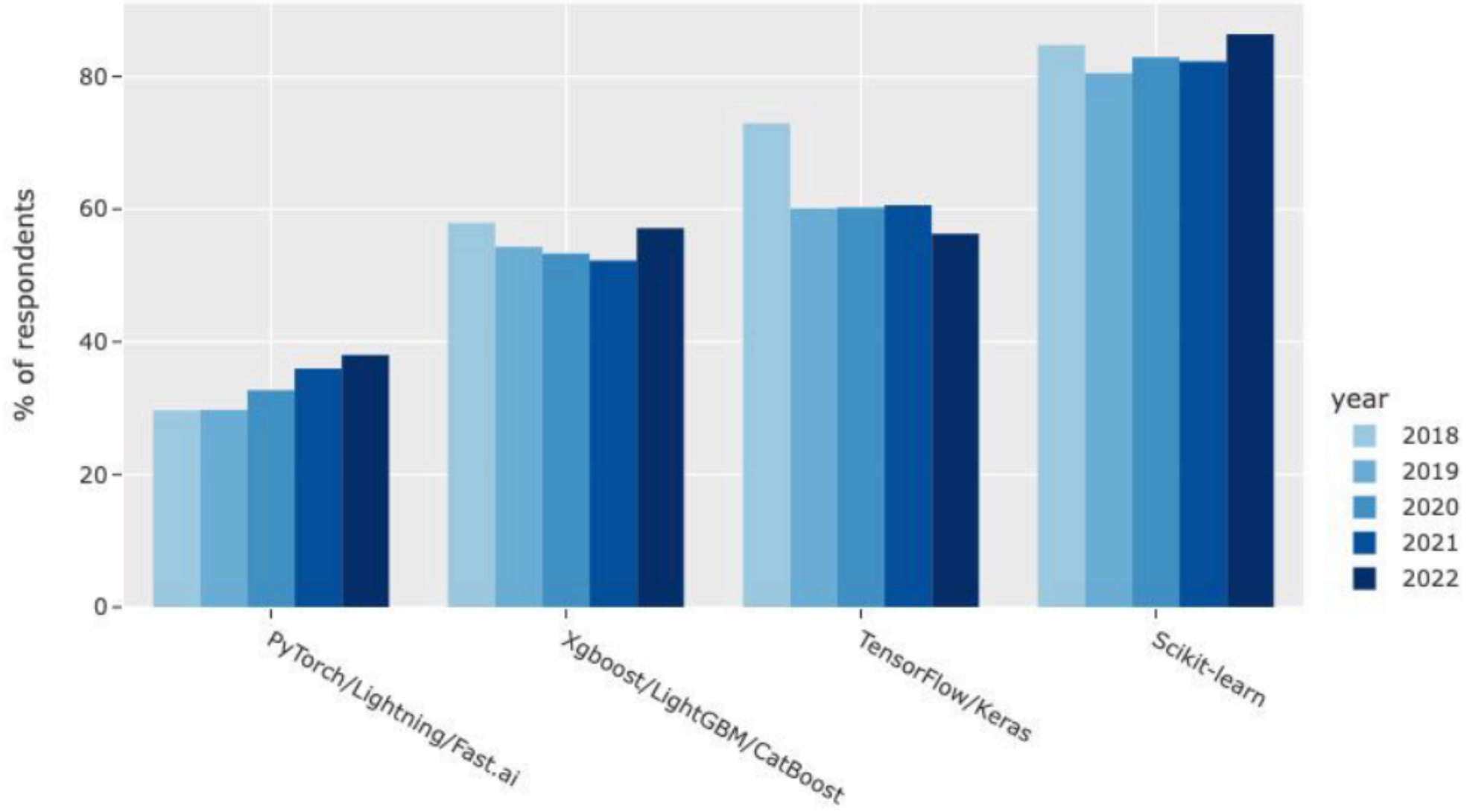


IDEs used by Kaggle data scientists



But note that only ~23,000 out of 10^6 answered the survey

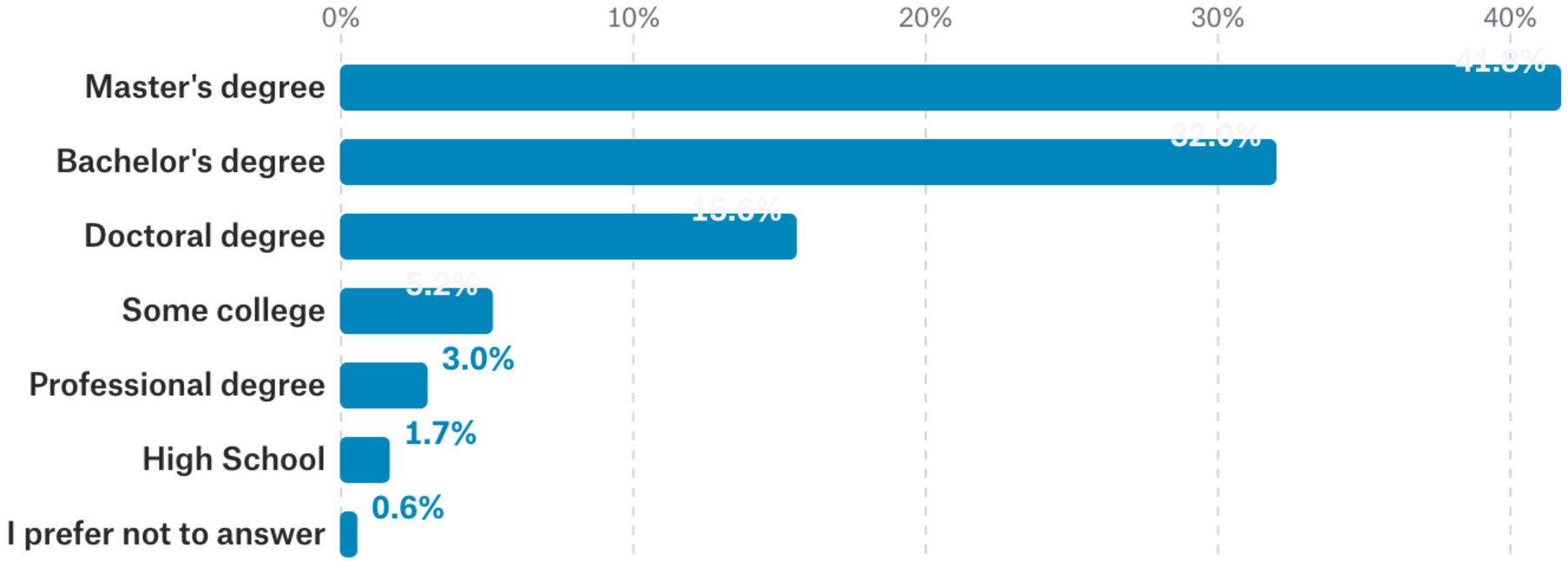
Frameworks used



Some fun survey results

From Kaggle's 2017 user survey (their first):

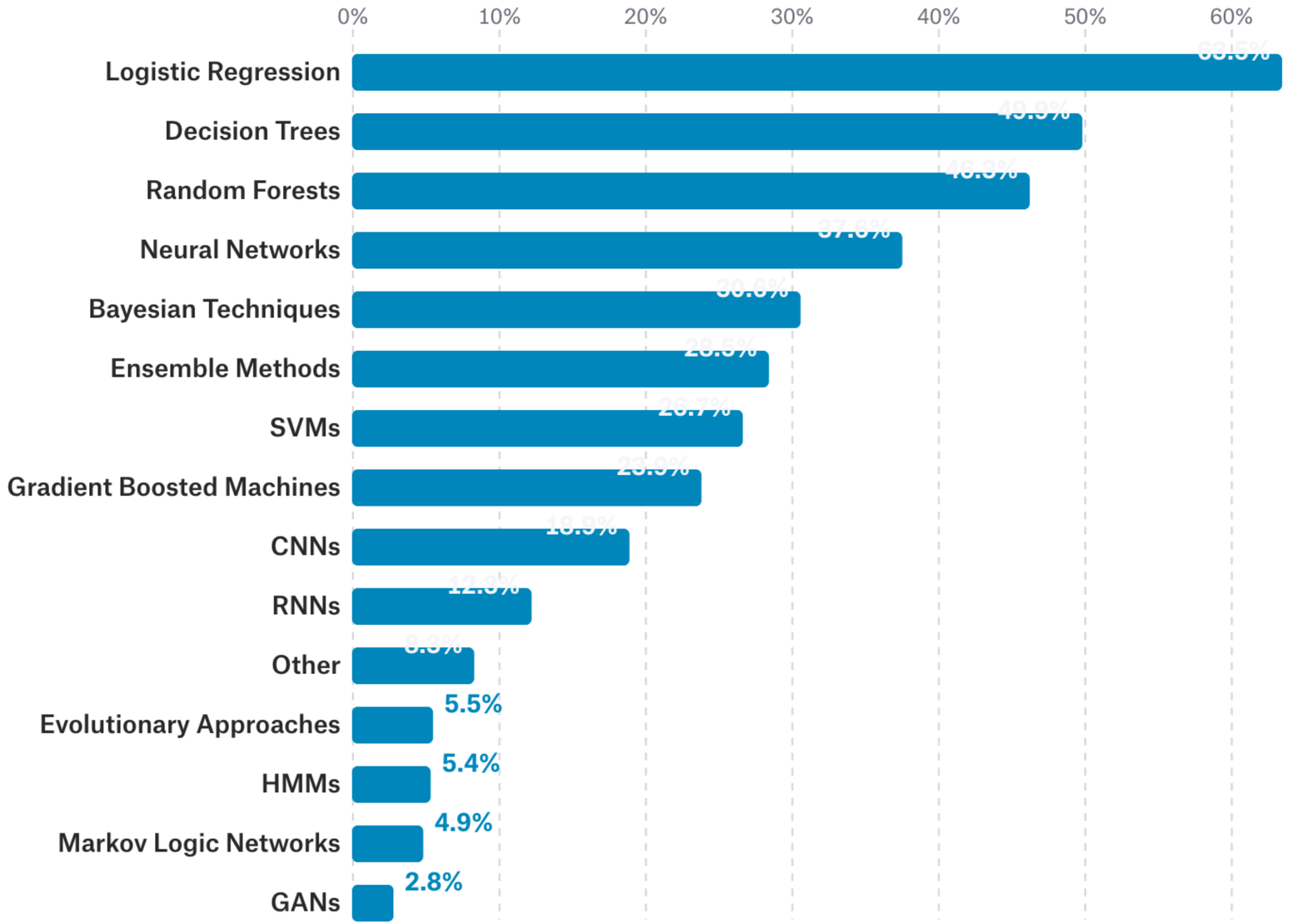
Participant's highest education level:



15,015 responses

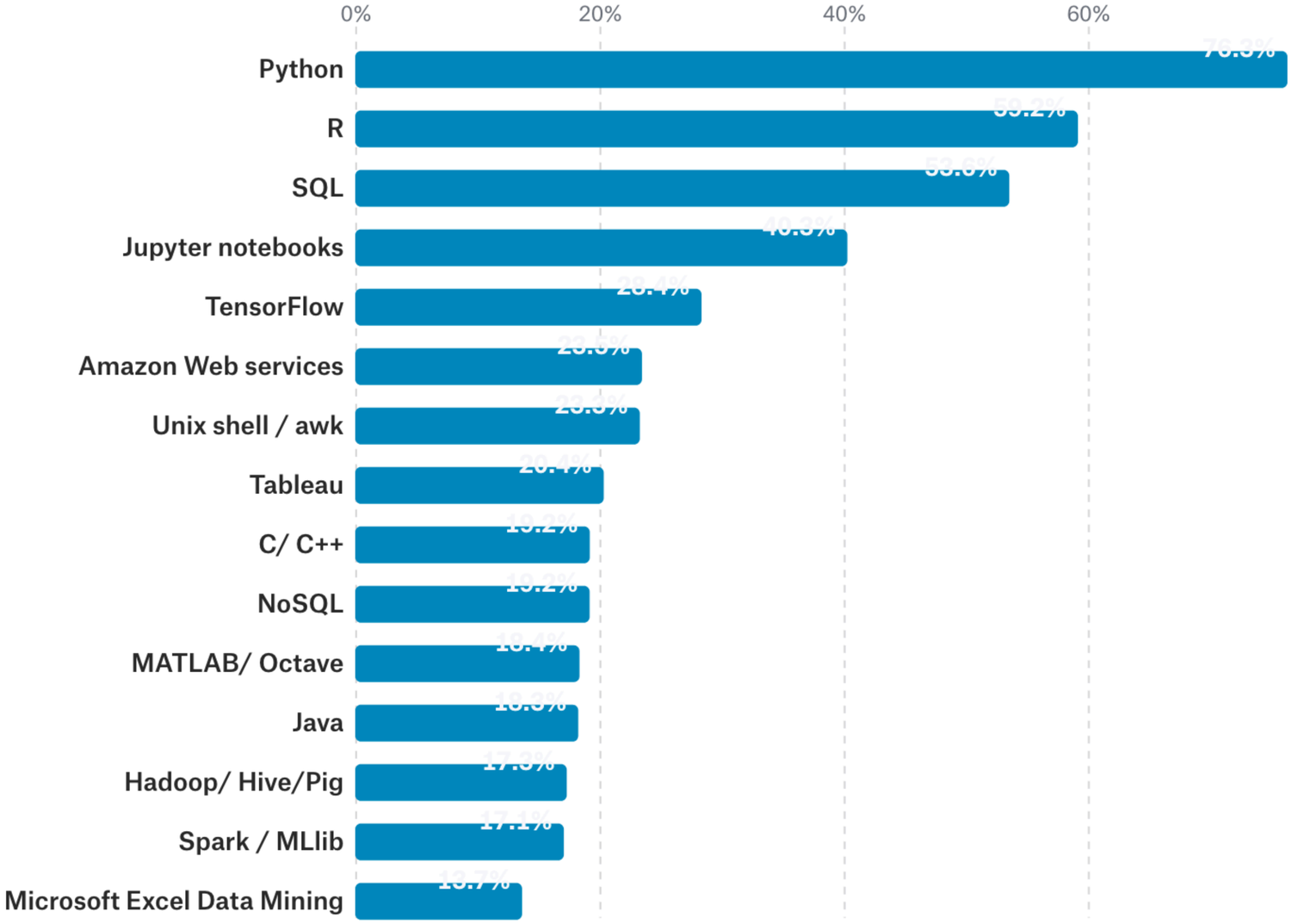
View code in Kaggle Kernels

Techniques used at work (lots we have not discussed!):



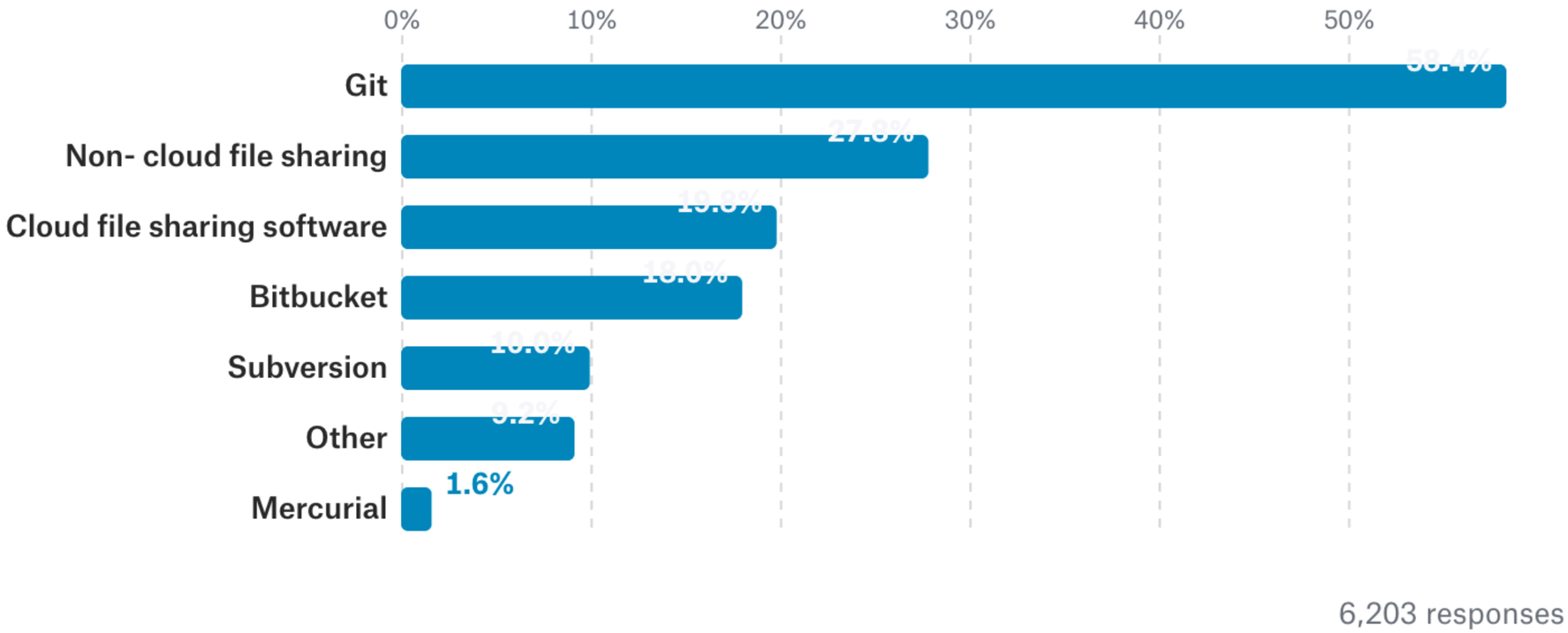
7,301 responses

Tools used at work:



7,955 responses

Code sharing used at work:



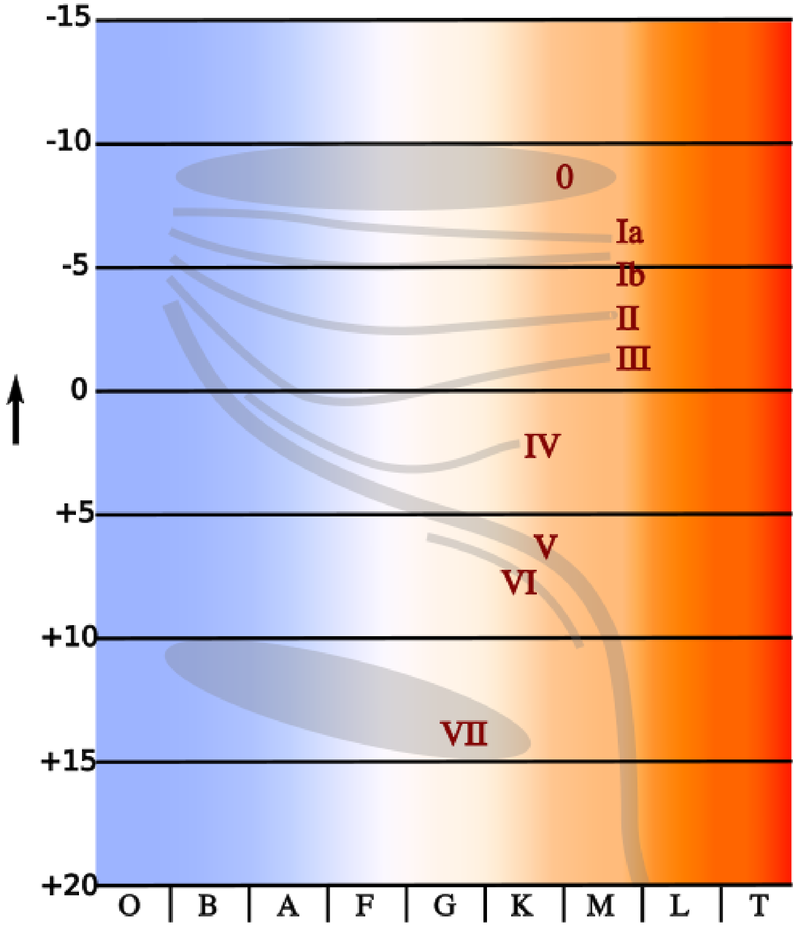
See more at:

https://www.kaggle.com/code/mhajabri/what-do-kagglers-say-about-data-science

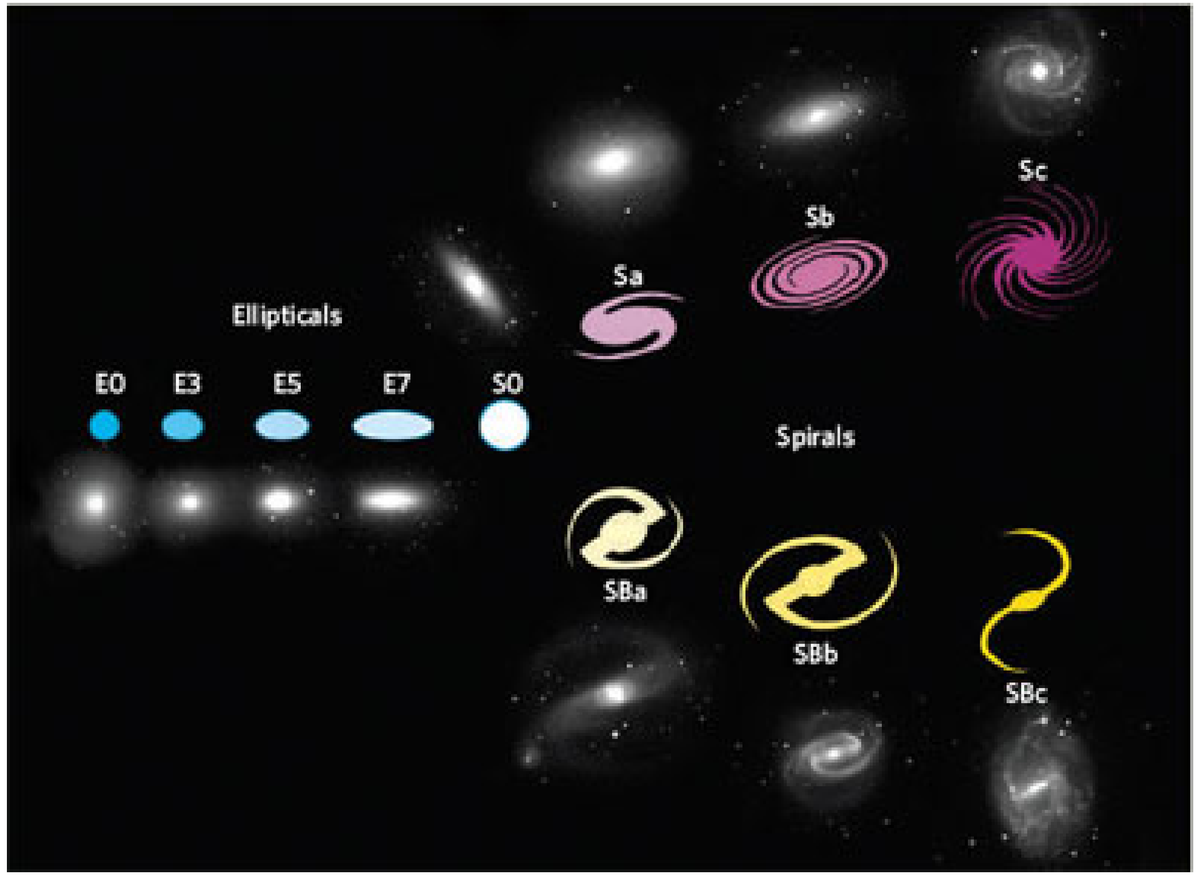
Classification

Classifying data

Astronomers very often classify stuff - this is one classical data-mining task that is very common in astronomy.



Wikipedia



Kennicutt (2006), Nature

The meaning of classification

I. Combination into known classes:

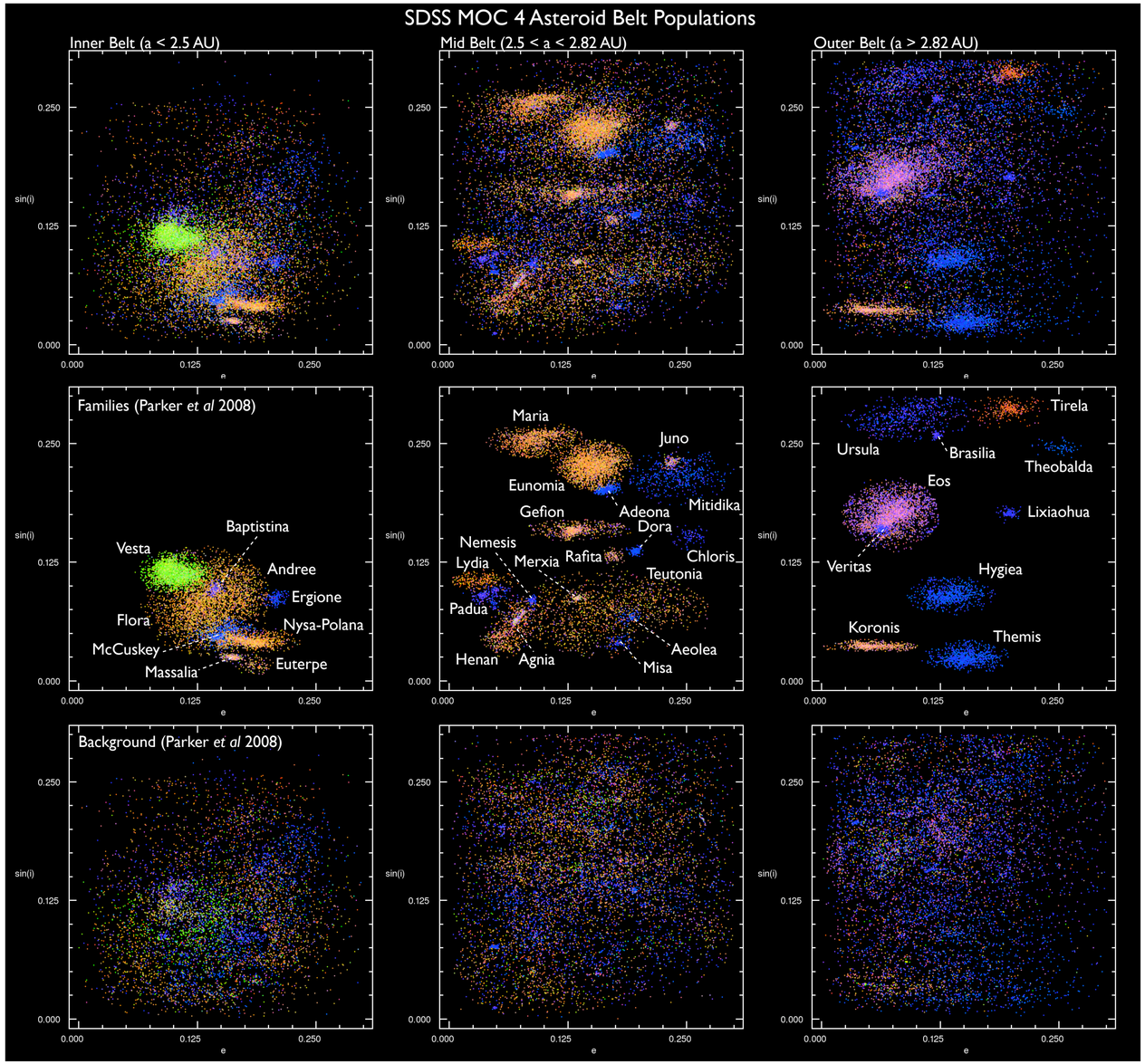
E.g GAIA:

Have: Photometric observations of N objects and low-resolution spectroscopic observations of M objects.

Want: Classification of objects into stars, galaxies, quasars etc. For the stars we also want T_{eff}, log g, [Fe/H] etc. [this is actually at least two different problems!]

So how do you do that?

II. Group objects to explore their nature



Finding and classifying asteroids

Parker et al (2008, Icarus 198,138-155)

Supervised & unsupervised classification

Supervised classification:

Given: Objects with “features” $\{x_i\}$, and known class ω_j

Needed: For an object with features $\{y_i\}$, assign a class ω

It is supervised, because we know of the existence of the classes before and we have some objects that have been reliably classified in advance.

Classification of stellar spectra and galaxy images can fall into this category.

Supervised & unsupervised classification

Unsupervised classification:

Given: Objects with “features” $\{x_i\}$
Needed: Assign these to N (which can be known or given) classes.
Then use this to assign classes to new data.

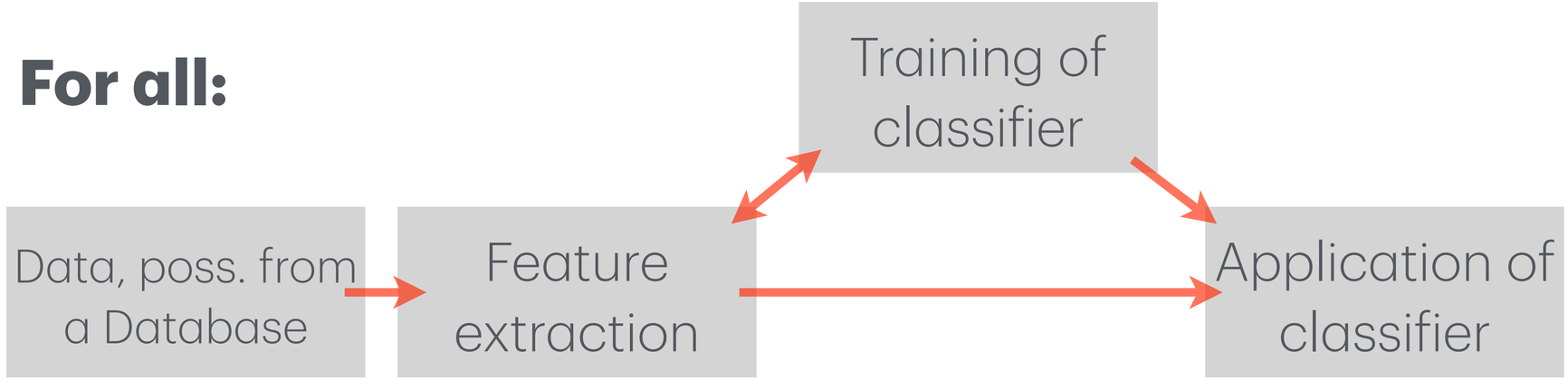
In this case we know nothing, or very little about the classification of the data. Thus this is often exploratory and the results can be a lot harder to interpret.

Finding asteroid families fall in this category.

Techniques for classification

- Bayesian decision criteria
- Nearest neighbour methods & friends.
- Linear models.
- Artificial Neural Networks.
- And many more (cladistics, hierarchical trees, **support vector machines** etc etc.)

For all:



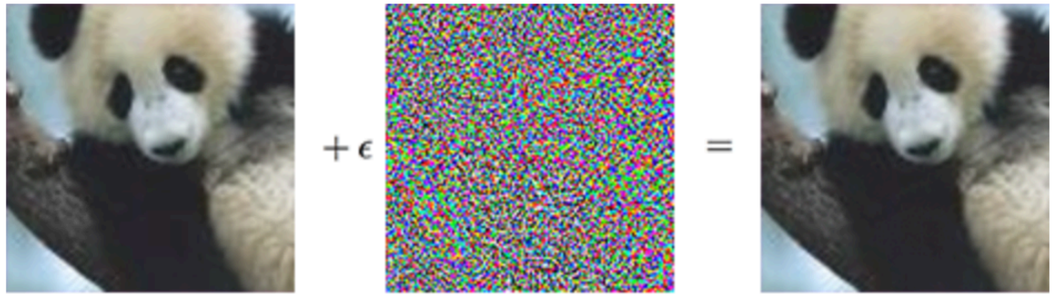
What is feature extraction?

- A crucial part of data classification/data mining - it is how you go about choosing the input data to the training algorithm.
- Depends strongly on the scientific question under consideration.
- For spectra it could be emission lines and absorption lines.
- For an image magnitudes, colours, light concentration, light profiles etc.
- Often it can be advisable to use something like principal component analysis (later) to guide your choice.
- It could also be the full image/full spectrum - this is often the way taken by deep learning algorithms (but with some modifications).

Classification - and training

Machine learning typically relies on you having good training data covering the possible results, but even so it sometimes fails:

https://arxiv.org/abs/1412.6572



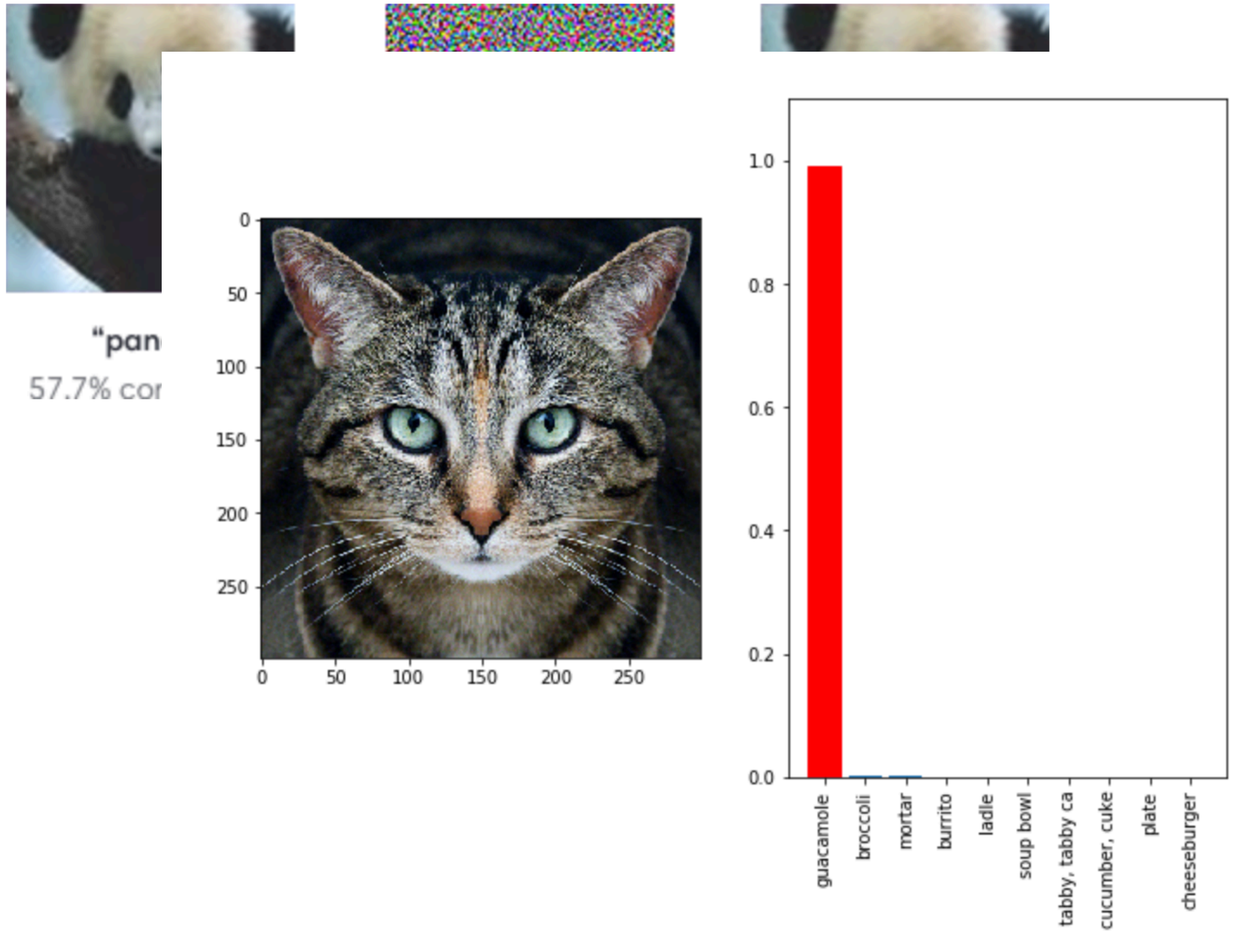
"panda"
57.7% confidence

"gibbon"
99.3% confidence

Classification - and training

Machine learning typically relies on you having good training data covering the possible results, but even so it sometimes fails:

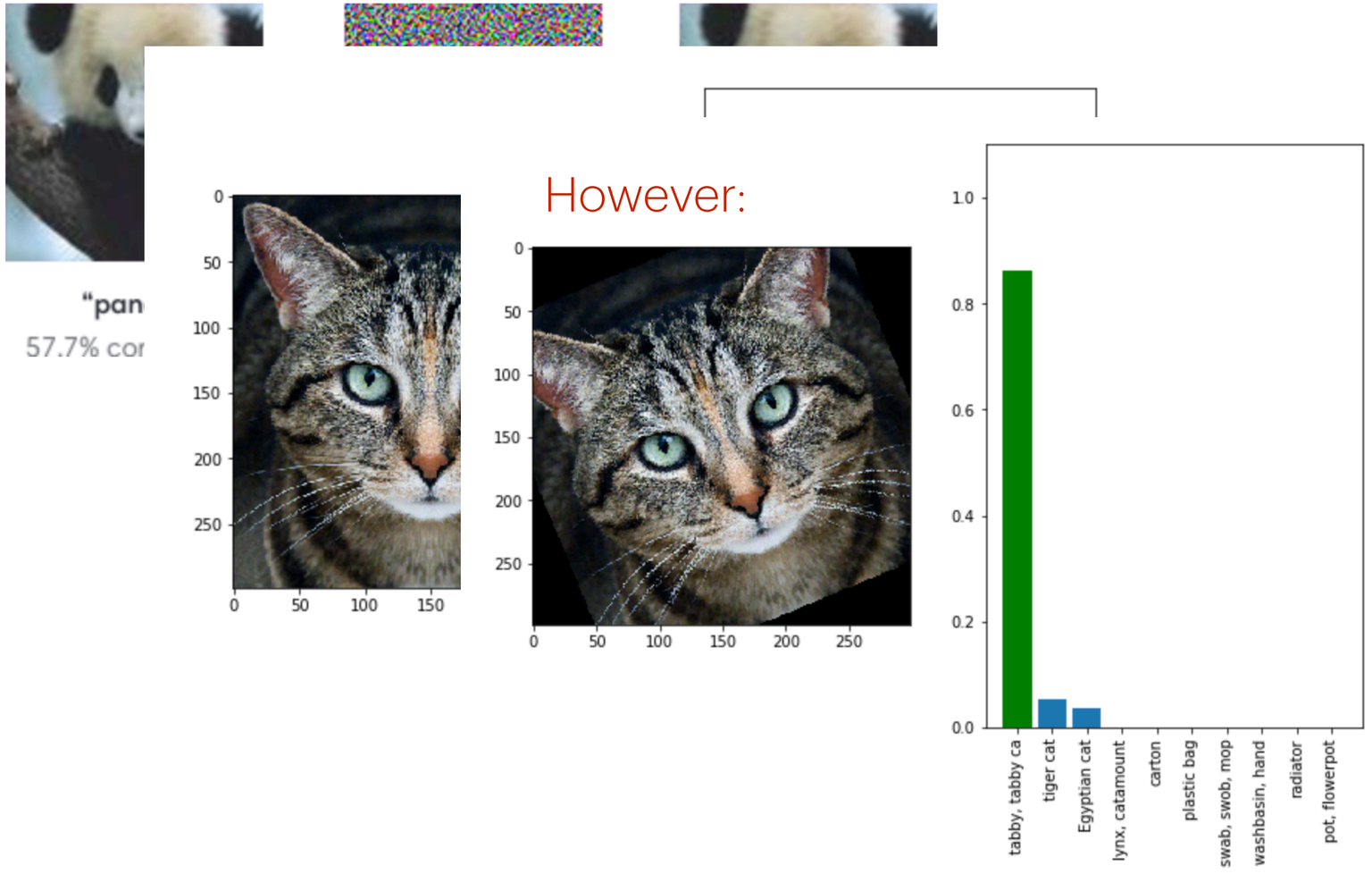
https://arxiv.org/abs/1412.6572



Classification - and training

Machine learning typically relies on you having good training data covering the possible results, but even so it sometimes fails:

https://arxiv.org/abs/1412.6572



An example: Bayesian classification of galaxies

Starting point: Morphological classifications by Nair & Abraham (2010), downloaded from Vizir & matched to SDSS absolute magnitudes I calculated separately.

The task: Design a classifier that given a galaxy colour will return the morphological class.

Notation:

ω_0 Elliptical

ω_1 Spiral

$\vec{x}$ Observables/Features - here: g-r colour

Bayes' theorem & optimal class assignment

Bayes' theorem for our case:

$$p(\omega_i|\vec{x}) = \frac{p(\vec{x}|\omega_i)p(\omega_i)}{p(\vec{x})}$$

$p(\omega_i|\vec{x})$ is the **posterior** probability

$p(\vec{x}|\omega_i)$ is the **likelihood**

$p(\omega_i)$ is the **prior** probability of a class

$$p(\vec{x}) = \sum_i p(\vec{x}|\omega_i)p(\omega_i)$$

ensures the probabilities are normalised but as it is independent of the classes we can ignore it here.

Bayes' theorem & optimal class assignment

Bayes' theorem for our case:

$$p(\omega_i|\vec{x}) = \frac{p(\vec{x}|\omega_i)p(\omega_i)}{p(\vec{x})}$$

Classification is then very straightforward:

$$p(\omega_0|\vec{x}) > p(\omega_1|\vec{x}) \Rightarrow \text{Class } 0 \text{ (elliptical)}$$
$$p(\omega_0|\vec{x}) < p(\omega_1|\vec{x}) \Rightarrow \text{Class } 1 \text{ (spiral)}$$

Bayes' theorem & optimal class assignment

Bayes' theorem for our case:

$$p(\omega_i|\vec{x}) = \frac{p(\vec{x}|\omega_i)p(\omega_i)}{p(\vec{x})}$$

Classification is then very straightforward:

$$p(\omega_0|\vec{x}) > p(\omega_1|\vec{x}) \Rightarrow \text{Class } 0 \text{ (elliptical)}$$
$$p(\omega_0|\vec{x}) < p(\omega_1|\vec{x}) \Rightarrow \text{Class } 1 \text{ (spiral)}$$

But this supposes we know $p(\vec{x} \mid \omega_i)$ and $p(\omega_i)$

Bayes' theorem & optimal class assignment

Bayes' theorem for our case:

$$p(\omega_i|\vec{x}) = \frac{p(\vec{x}|\omega_i)p(\omega_i)}{p(\vec{x})}$$

Classification is then very straightforward:

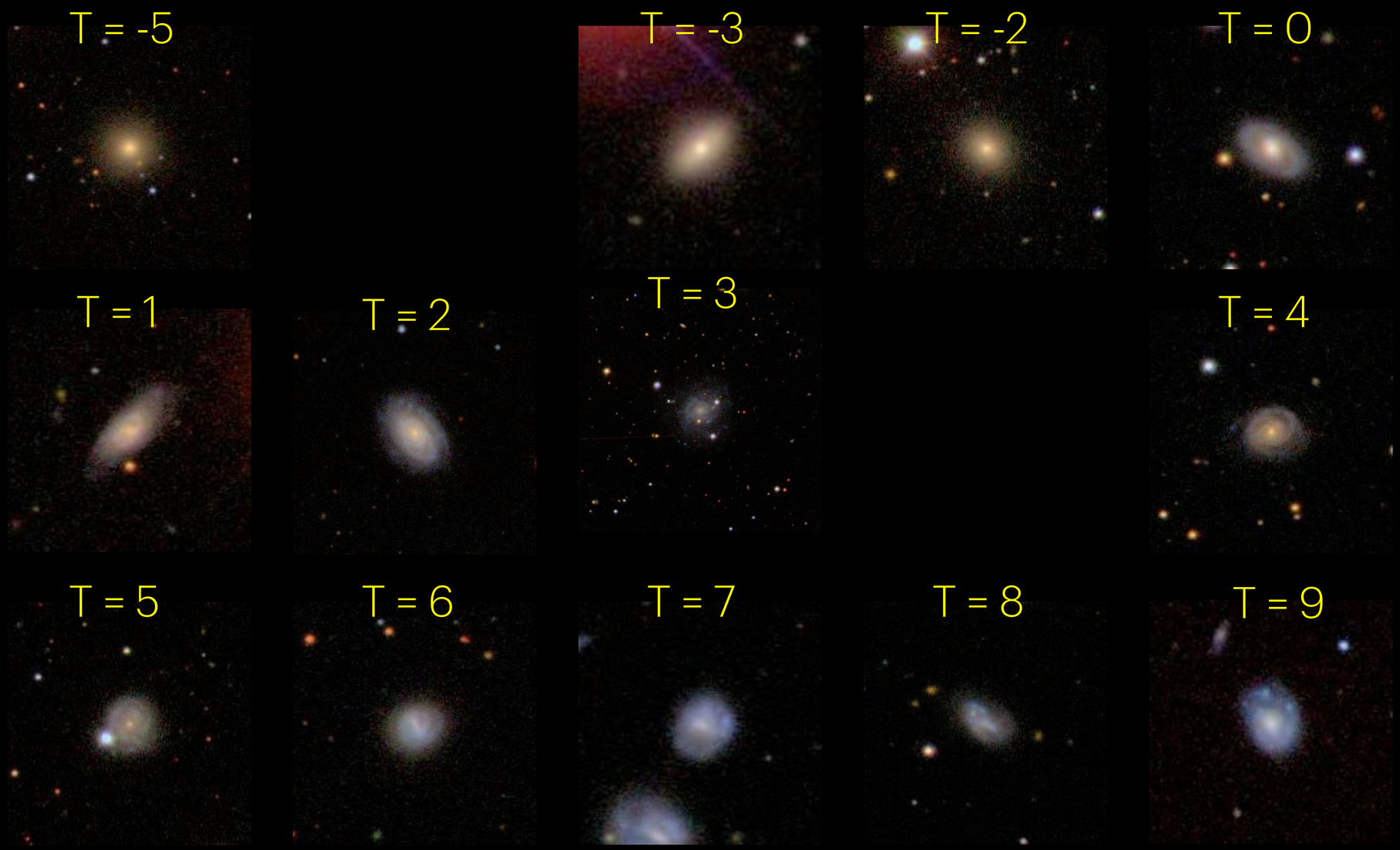
$$p(\omega_0|\vec{x}) > p(\omega_1|\vec{x}) \Rightarrow \text{Class 0 (elliptical)}$$
$$p(\omega_0|\vec{x}) < p(\omega_1|\vec{x}) \Rightarrow \text{Class 1 (spiral)}$$

But this supposes we know $p(\vec{x} \mid \omega_i)$ and $p(\omega_i)$

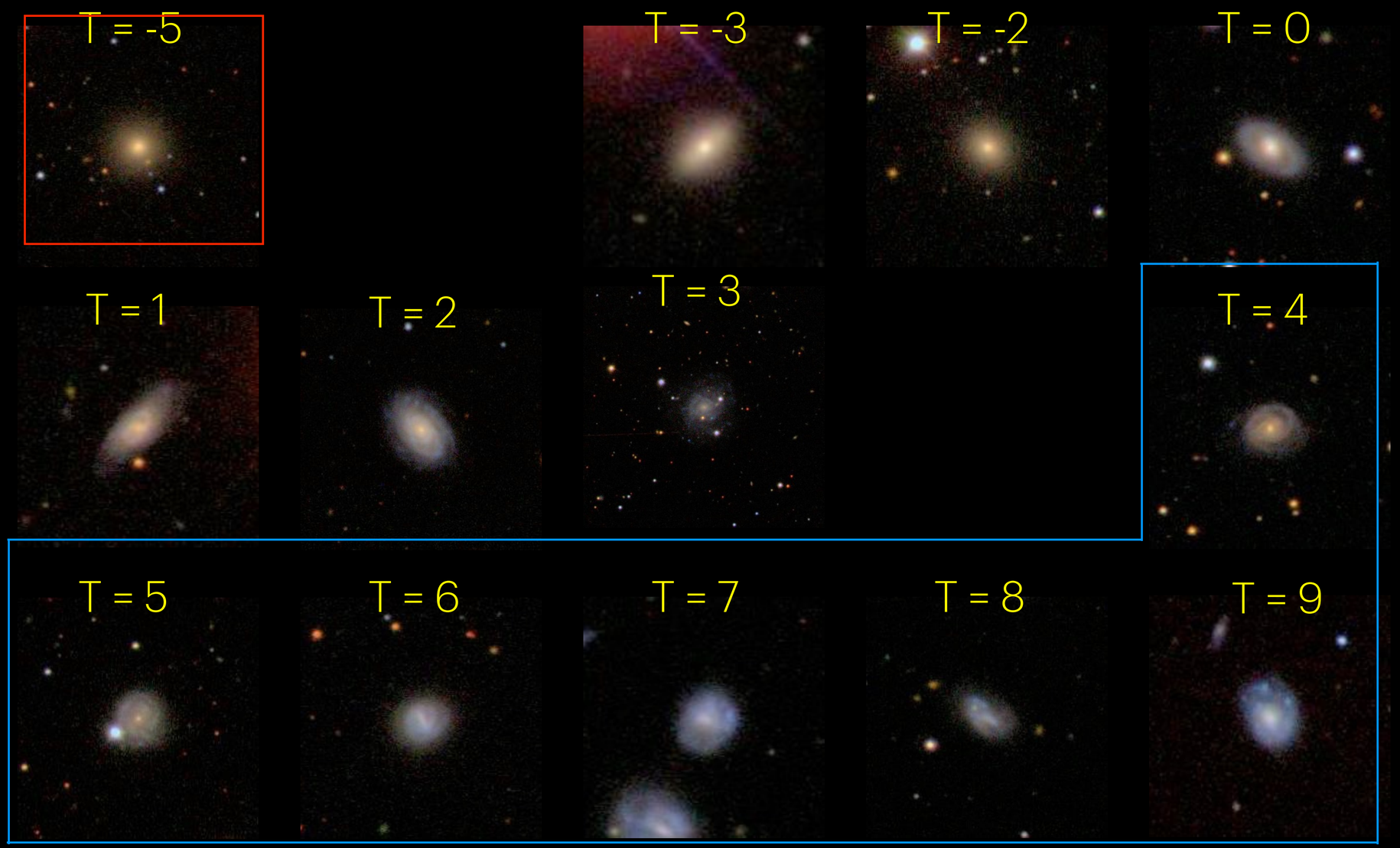
If we assume all classes are equally probable, we just need $p(\vec{x} \mid \omega_i)$ and for that we can use the tools from yesterday:

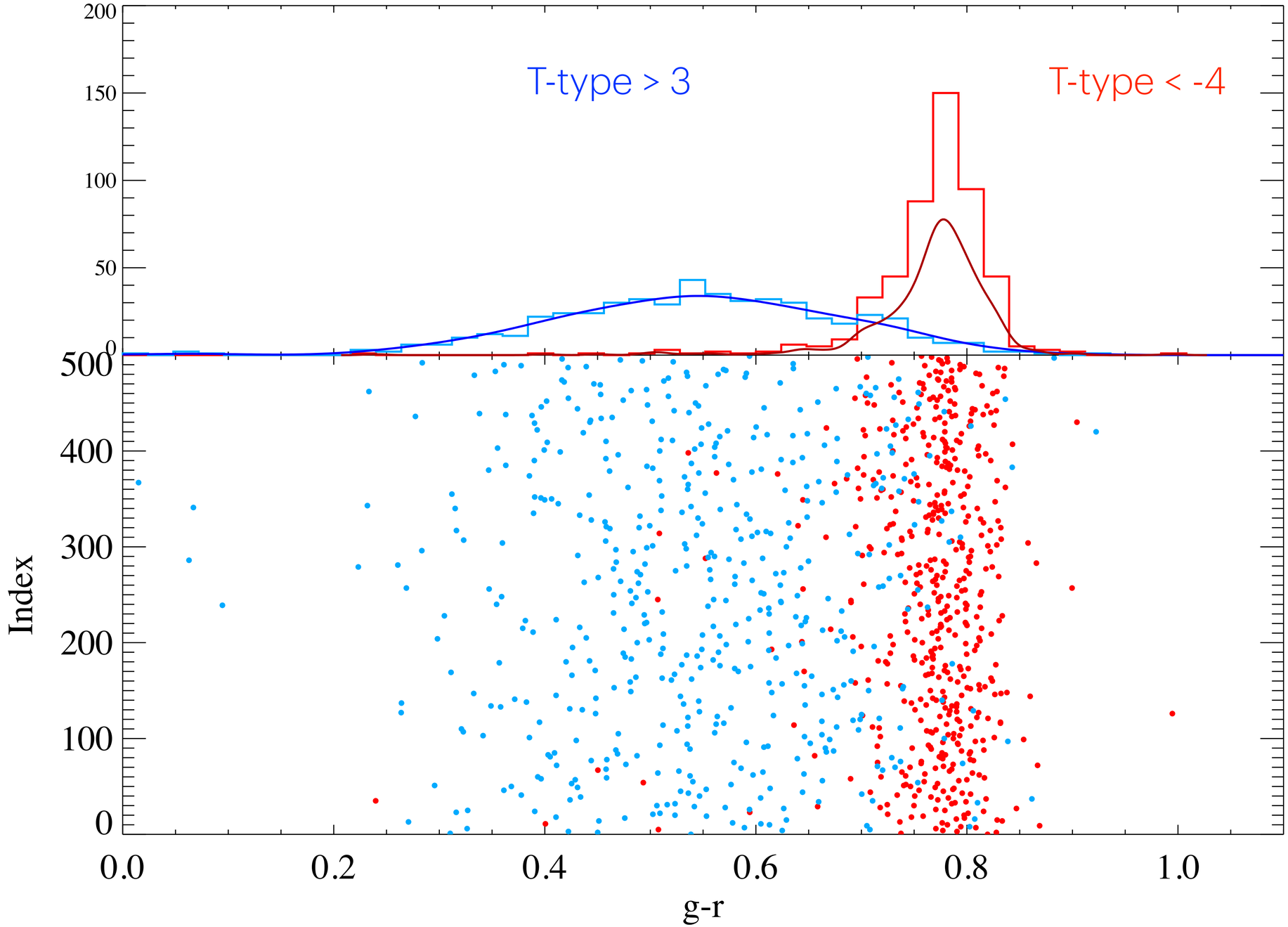
Histograms, kernel estimates or estimating the mean and standard deviation of a Gaussian.

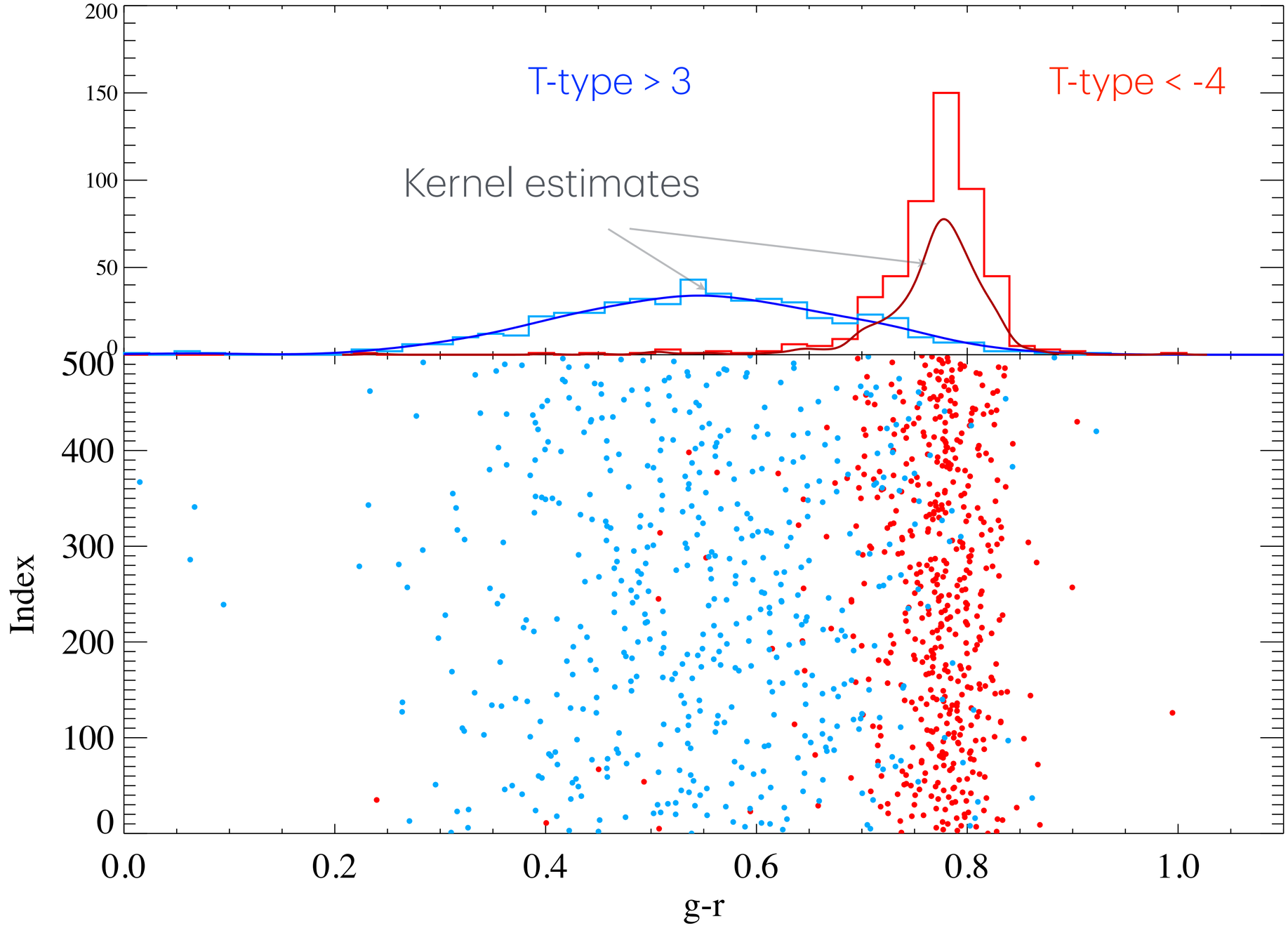
Interlude - Galaxy T-types

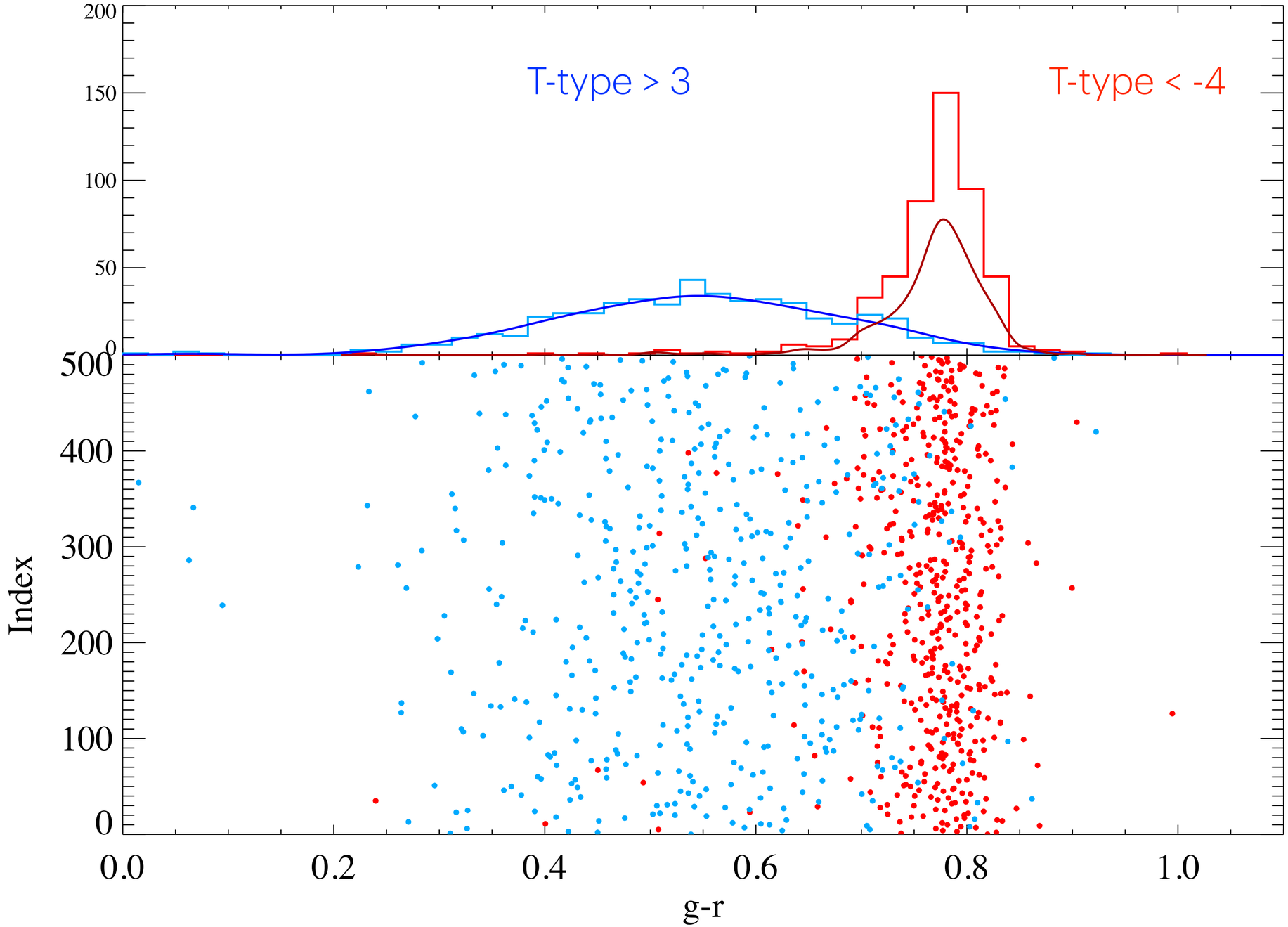


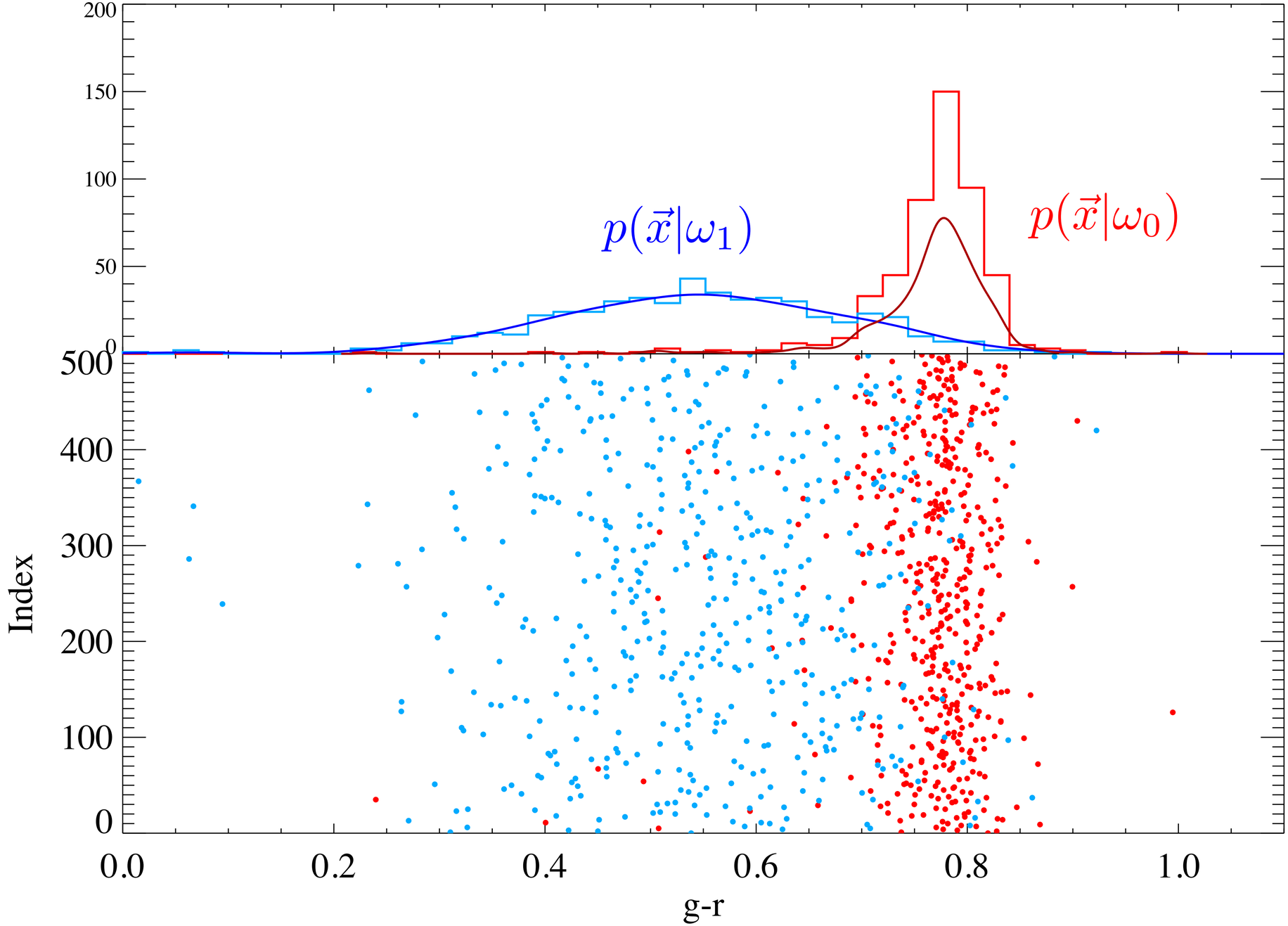
Interlude - Galaxy T-types



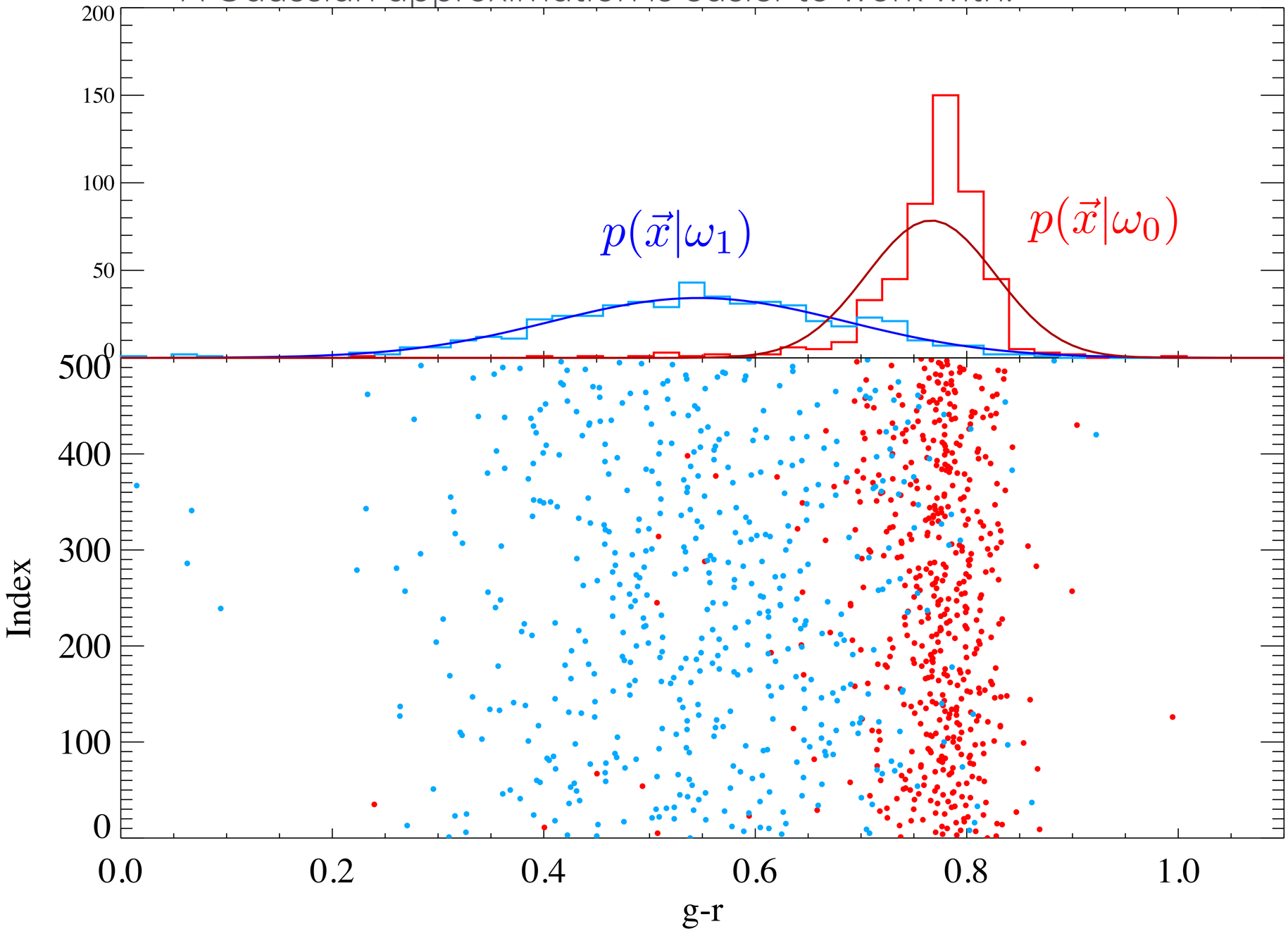

T-type > 3
T-type < -4
200
150
100
50
0
500
400
300
200
100
0
Index
0.0
0.2
0.4
0.6
0.8
1.0
g-r


T-type > 3
T-type < -4
Kernel estimates
200
150
100
50
0
500
400
300
200
100
0
Index
0.0
0.2
0.4
0.6
0.8
1.0
g-r

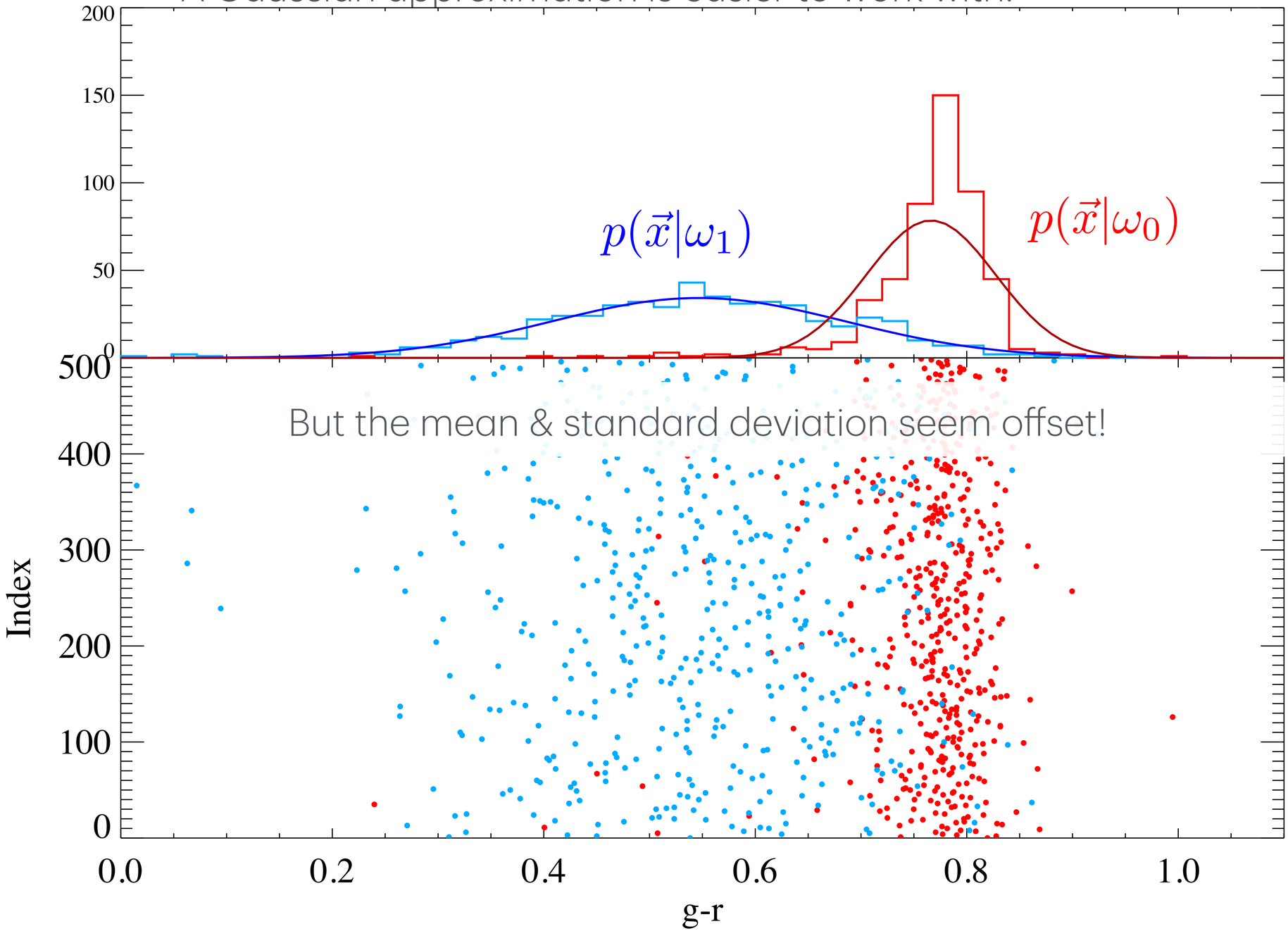

T-type > 3
T-type < -4
200
150
100
50
0
500
400
300
200
100
0
Index
0.0
0.2
0.4
0.6
0.8
1.0
g-r


200
150
100
50
0
$p(\vec{x}|\omega_1)$
$p(\vec{x}|\omega_0)$
500
400
300
200
100
0
Index
0.0
0.2
0.4
0.6
0.8
1.0
g-r

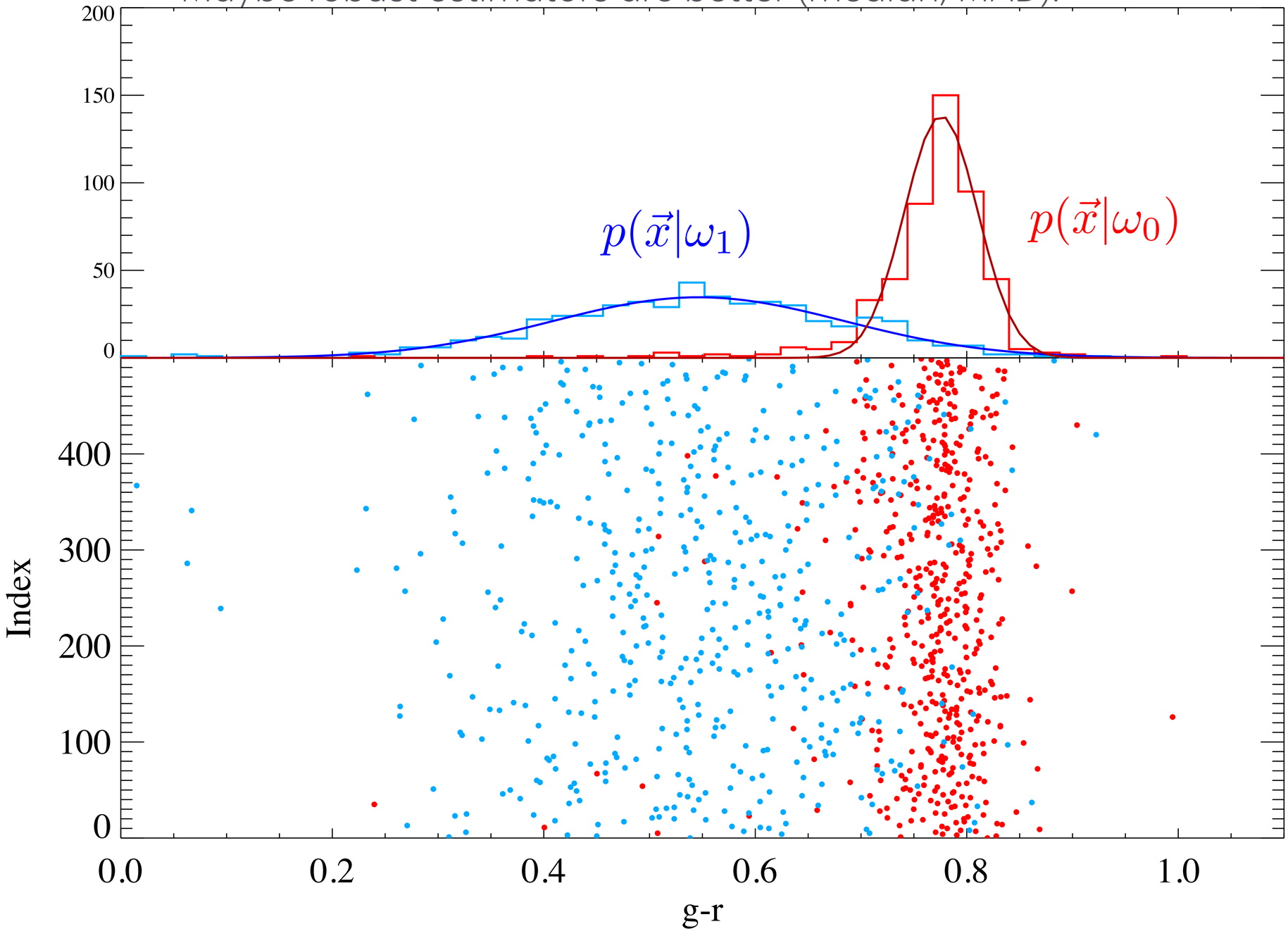
A Gaussian approximation is easier to work with:

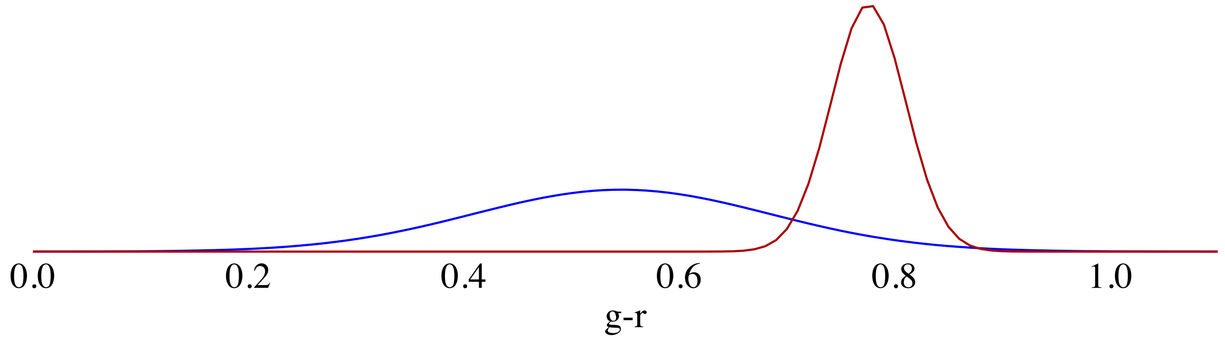


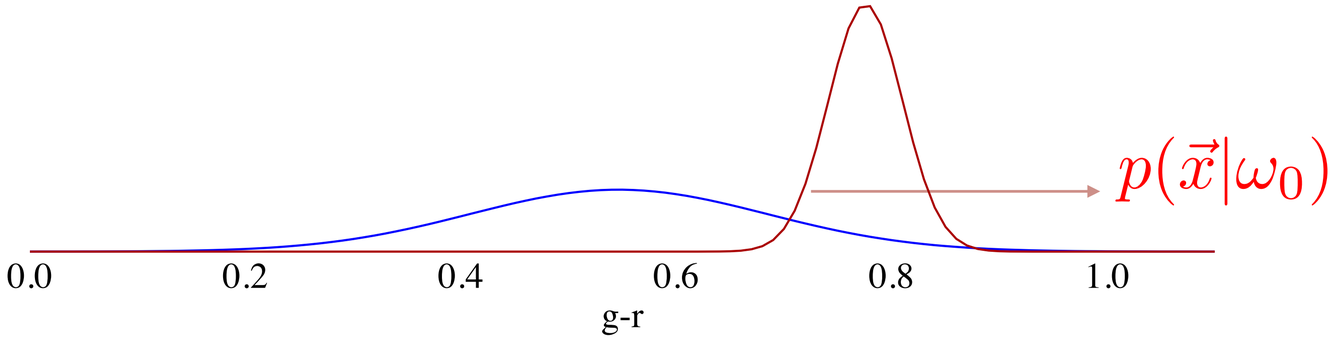
A Gaussian approximation is easier to work with:



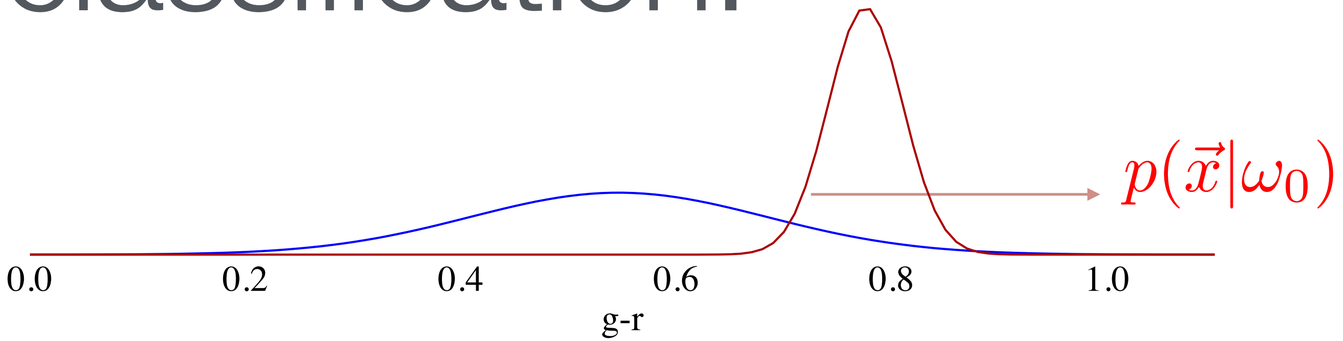
Maybe robust estimators are better (median, MAD):



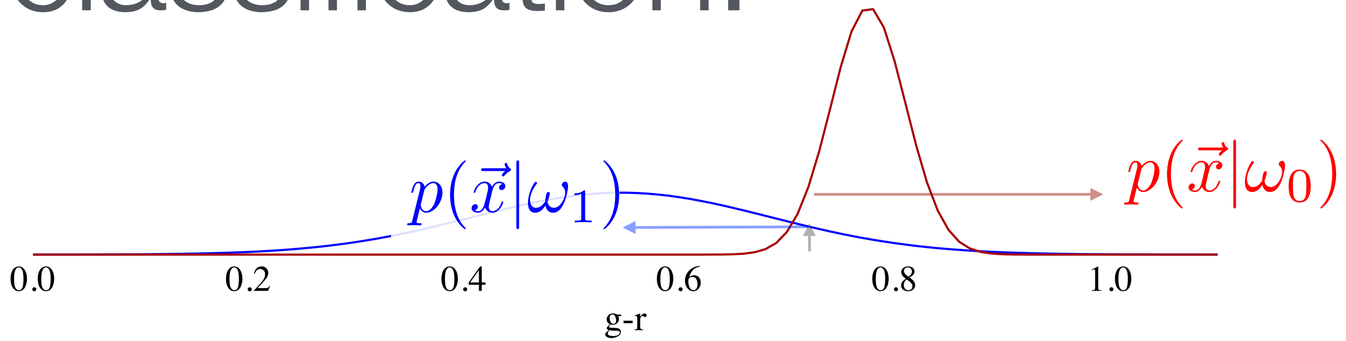

0.0
0.2
0.4
0.6
0.8
1.0
g-r


$p(\vec{x}|\omega_0)$
0.0
0.2
0.4
0.6
0.8
1.0
g-r

Bayesian classification:

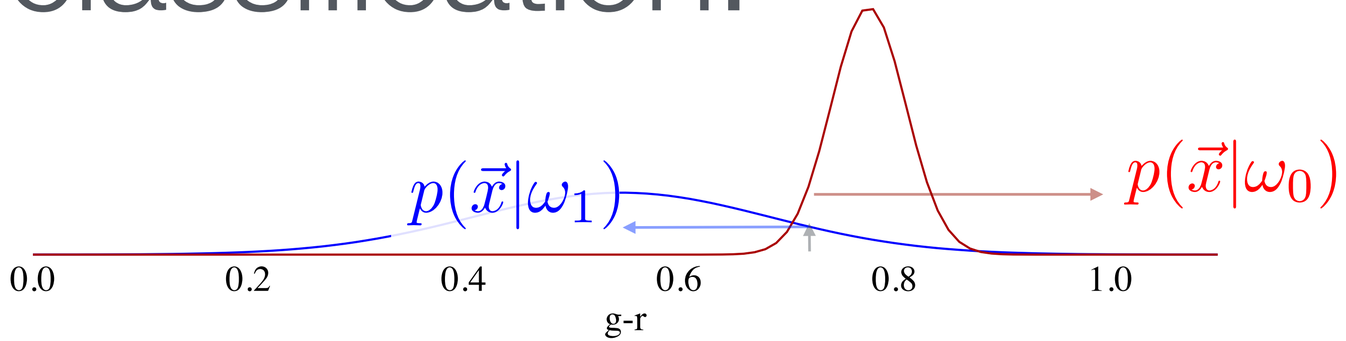


Bayesian classification:

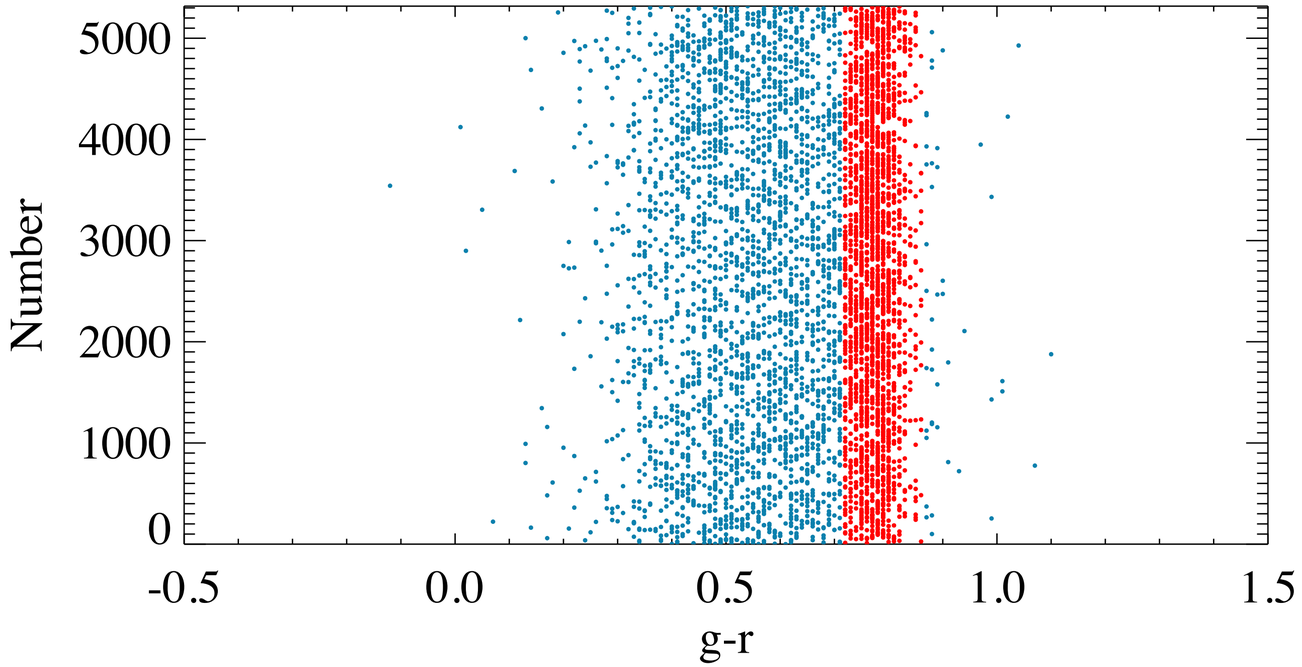


So in this case we assign a class Elliptical and we can also give the likelihood relative to the spiral class. We do this for all objects:

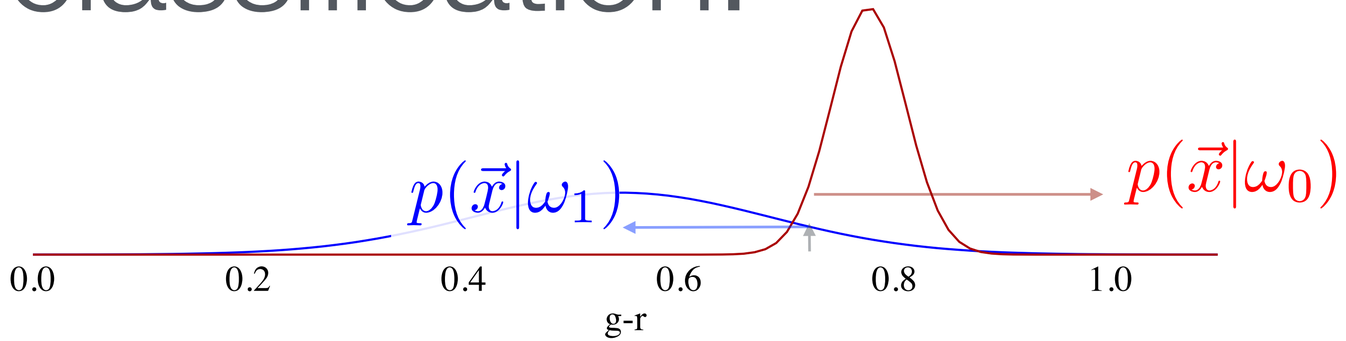
Bayesian classification:



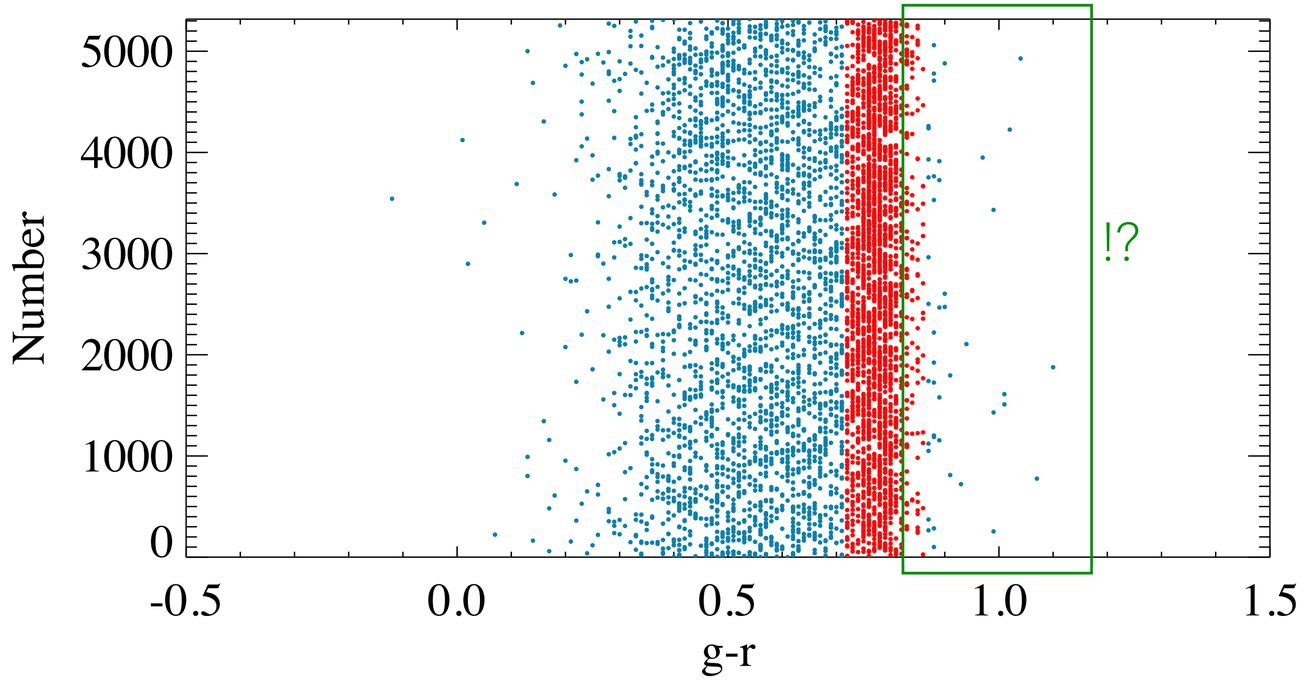
So in this case we assign a class Elliptical and we can also give the likelihood relative to the spiral class. We do this for all objects:



Bayesian classification:



So in this case we assign a class Elliptical and we can also give the likelihood relative to the spiral class. We do this for all objects:



Bayesian classification - decision surfaces

We can write our class separation criterion as:

$$p(\omega_0|\vec{x}) = p(\omega_1|\vec{x})$$

or

$$\ln p(\vec{x}|\omega_0)p(\omega_0) = \ln p(\vec{x}|\omega_1)p(\omega_1)$$

This will define decision regions, sometimes a surface

Taking logarithms and assuming a Gaussian PDF we have:

$$\left(\frac{x-\mu_0}{2\sigma_0}\right)^2 - \left(\frac{x-\mu_1}{2\sigma_1}\right)^2 = \ln\frac{\sigma_1}{\sigma_0}$$

$$p(\vec{x}|\omega_0) = \frac{1}{\sqrt{2\pi}\sigma_0}e^{-(x-\mu_0)^2/2\sigma_0^2}$$

$$p(\vec{x}|\omega_1) = \frac{1}{\sqrt{2\pi}\sigma_1}e^{-(x-\mu_1)^2/2\sigma_1^2}$$

So generally you get two solutions!

Validation

Getting a solution is a start - now we need to test (validate) the solution.

For this we have another sample which has known class - we apply our classifier and compare the predicted class with the known class.

$$\frac{\#\text{ cases where predicted class != true class}}{\#\text{ of cases}}$$

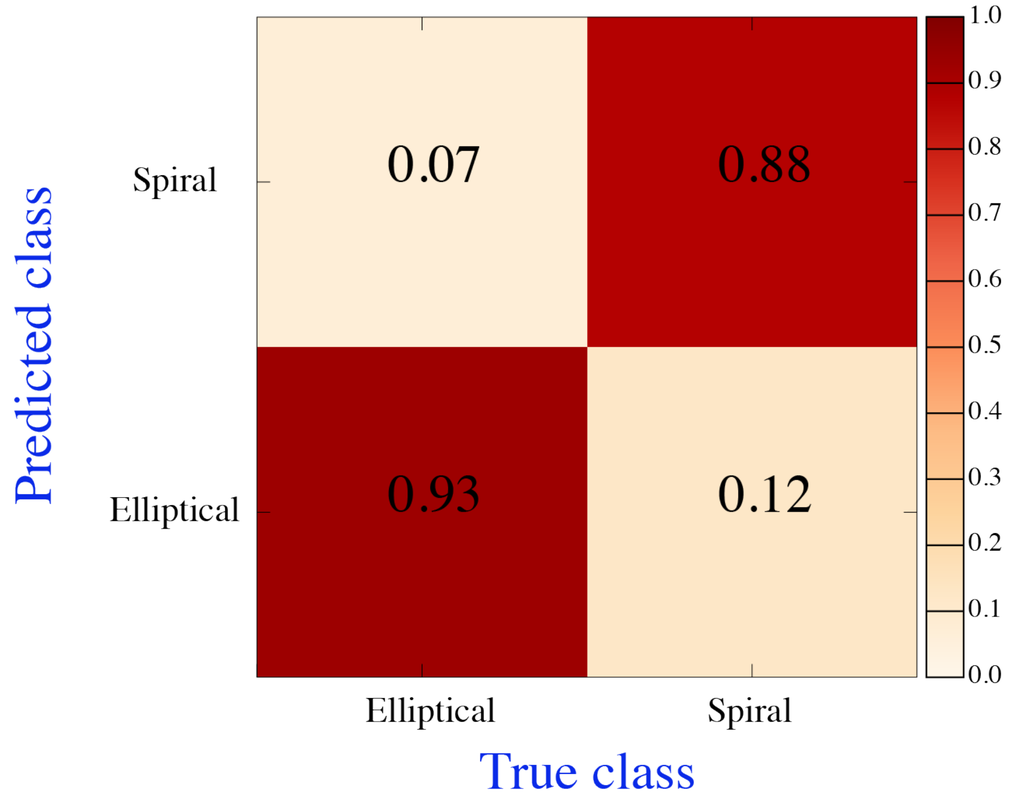
= Misclassification rate

In our case: 10% (also if we use kernel estimation)

But if we remove our gap in T-types - the misclassification rate goes up to 25%...

Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.



In python:

```
confusion_matrix in sklearn.metrics.
```

In Julia:

```
ConfusionMatrix in EvalMetrics.jl
```

In R:

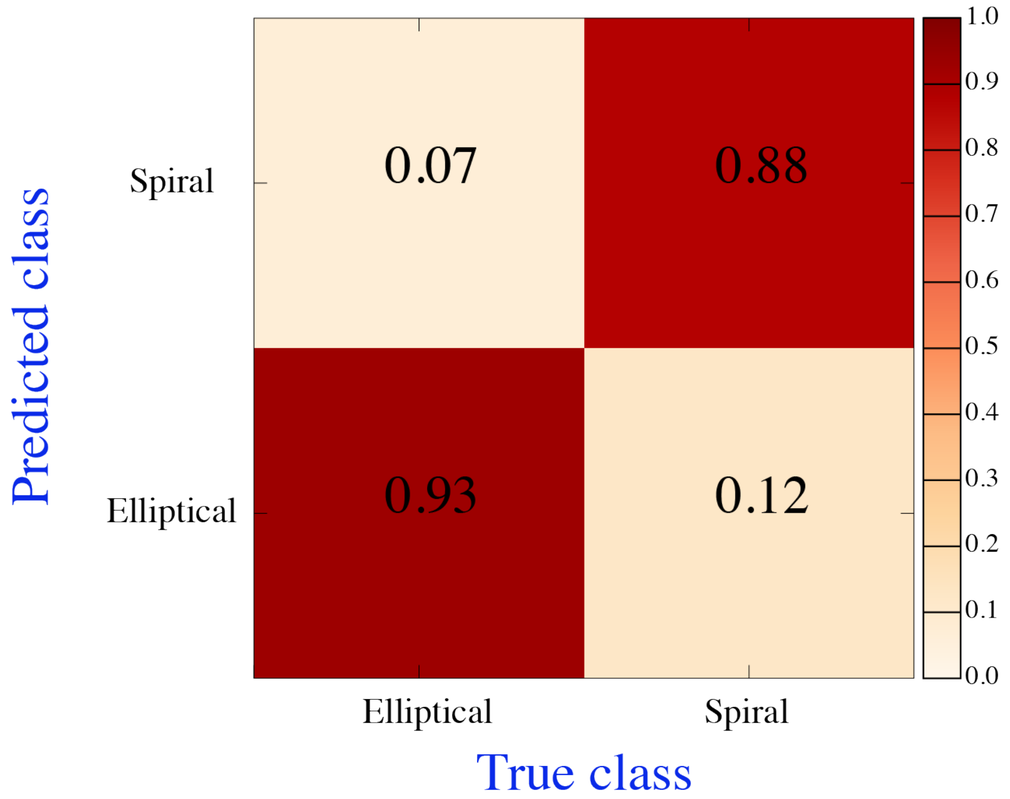
```
confusionMatrix in caret
```

Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.

Often you are only interested in one class - let us say elliptical. In that case it is usual to call this “Positive” and spiral “Negative”.

The resulting confusion matrix can then be summarised in many ways

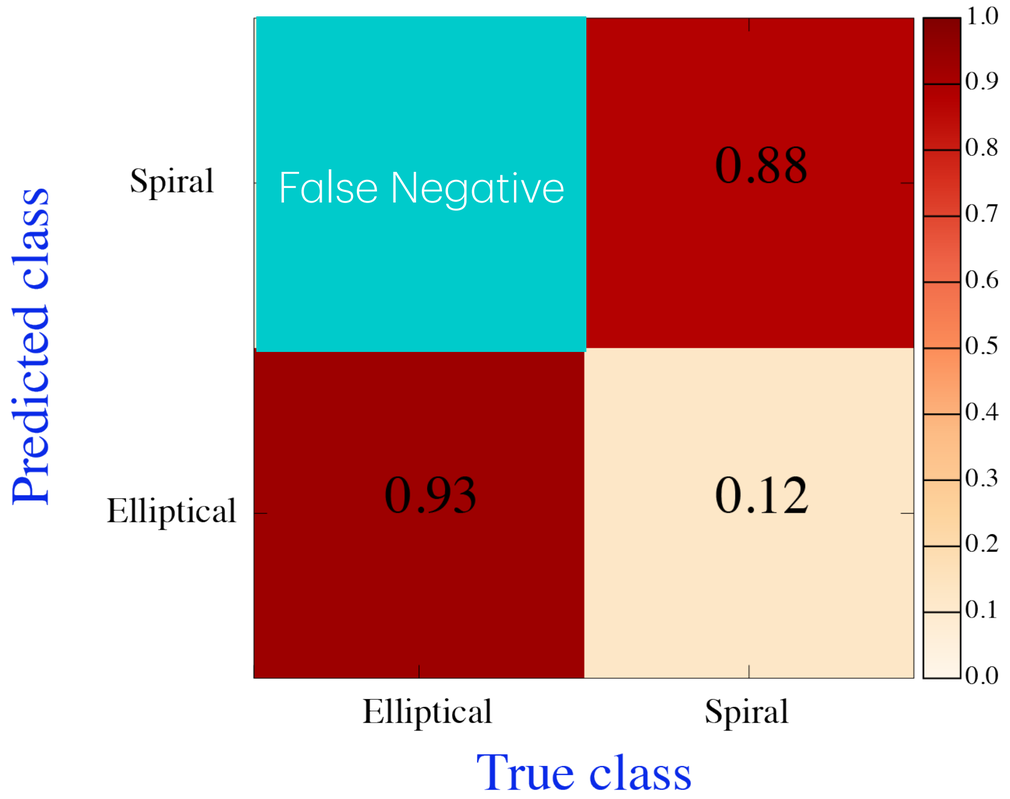


Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.

Often you are only interested in one class - let us say elliptical. In that case it is usual to call this “Positive” and spiral “Negative”.

The resulting confusion matrix can then be summarised in many ways

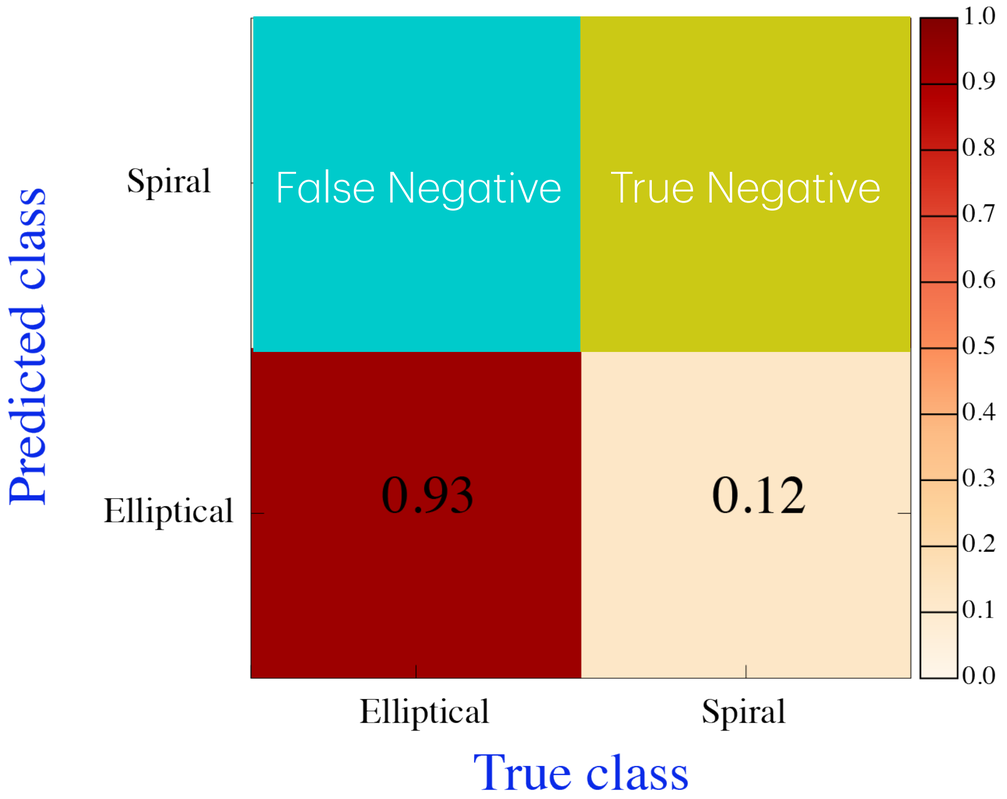


Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.

Often you are only interested in one class - let us say elliptical. In that case it is usual to call this “Positive” and spiral “Negative”.

The resulting confusion matrix can then be summarised in many ways

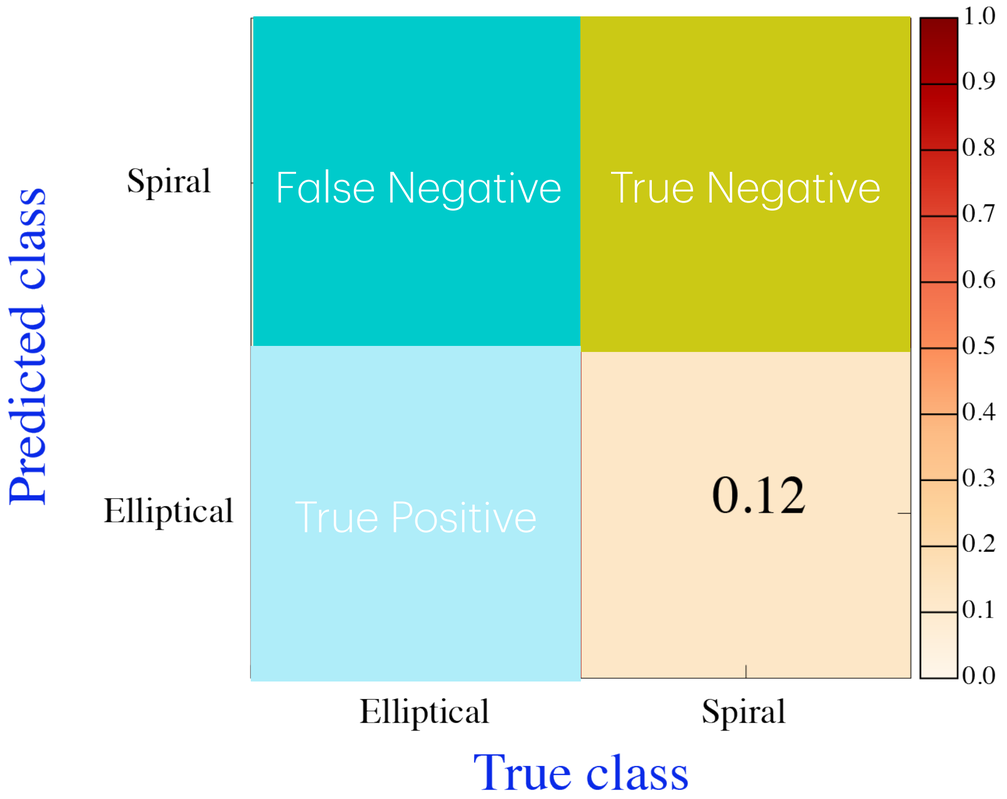


Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.

Often you are only interested in one class - let us say elliptical. In that case it is usual to call this “Positive” and spiral “Negative”.

The resulting confusion matrix can then be summarised in many ways

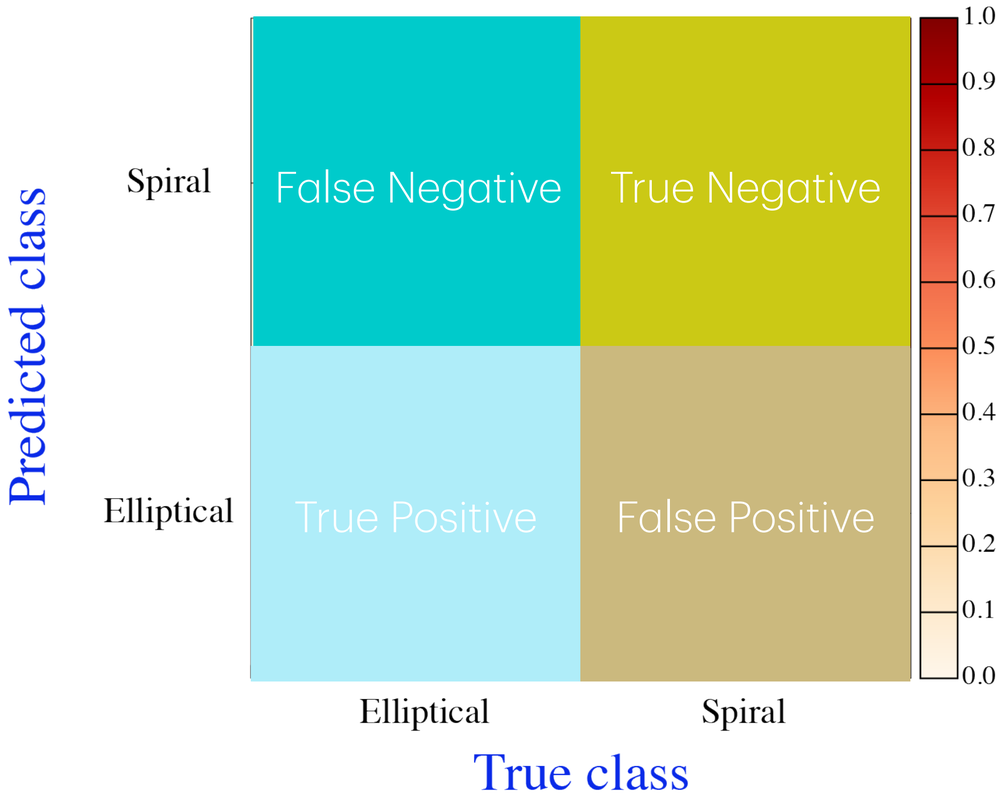


Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.

Often you are only interested in one class - let us say elliptical. In that case it is usual to call this “Positive” and spiral “Negative”.

The resulting confusion matrix can then be summarised in many ways



Validation - confusion matrix & misclassifications

False Negative = **FN**

True Negative = **TN**

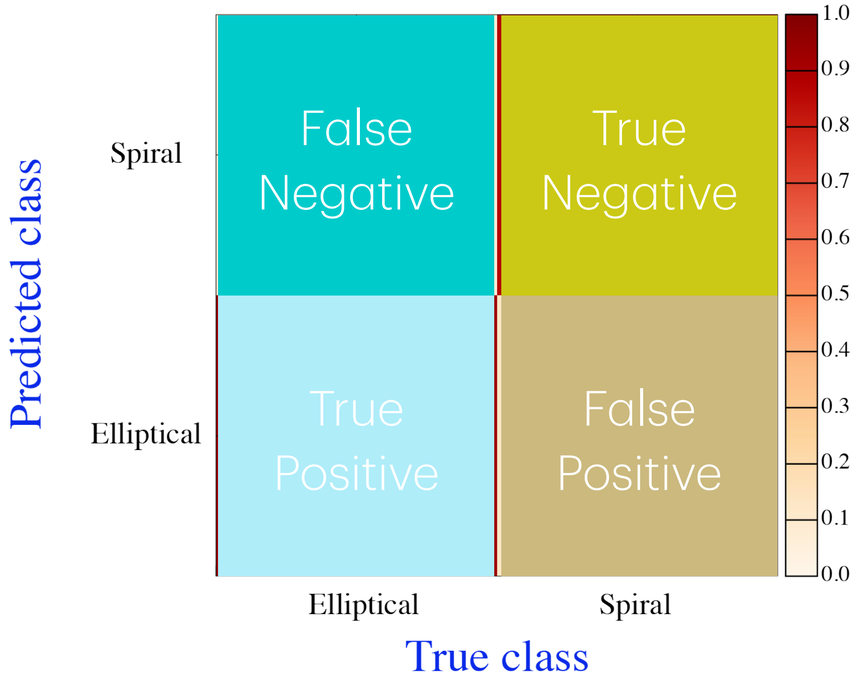
True Positive = **TP**

False Positive = **FP**

P = TP+FN

N = TN + FP

Some commonly seen terms:



Validation - confusion matrix & misclassifications

False Negative = **FN**

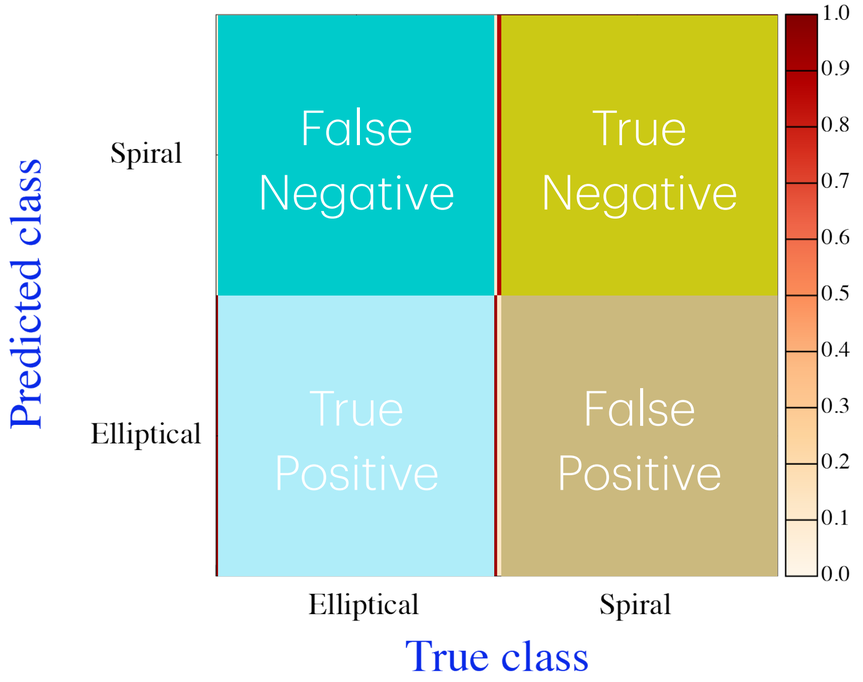
True Negative = **TN**

True Positive = **TP**

False Positive = **FP**

P = TP+FN

N = TN + FP



Some commonly seen terms:

Recall, sensitivity, true positive rate:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

Validation - confusion matrix & misclassifications

False Negative = **FN**

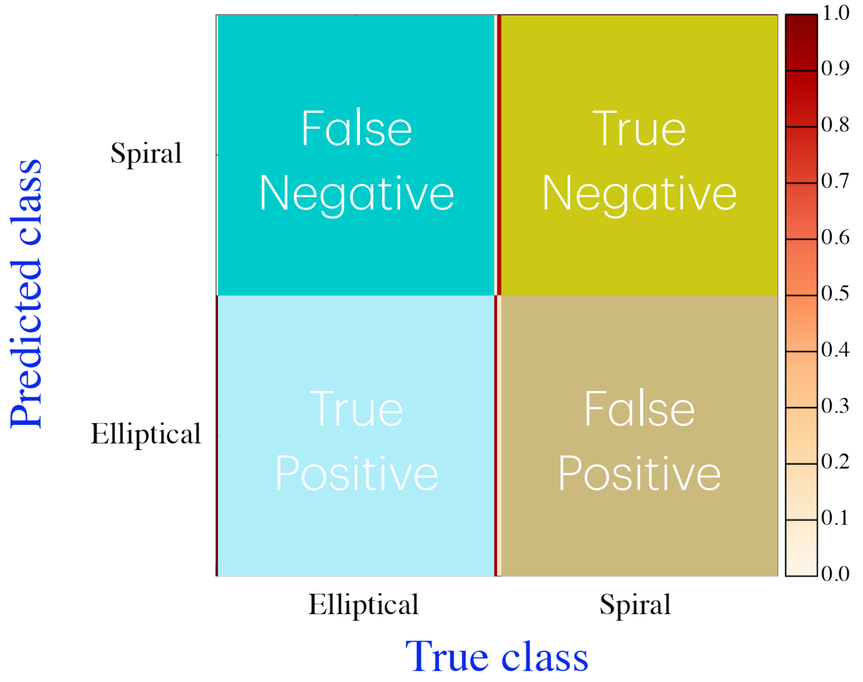
True Negative = **TN**

True Positive = **TP**

False Positive = **FP**

P = TP+FN

N = TN + FP



Some commonly seen terms:

Recall, sensitivity, true positive rate:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

Precision, positive predictive value:

$$PPV = \frac{TP}{TP + FP}$$

Validation - confusion matrix & misclassifications

False Negative = **FN**

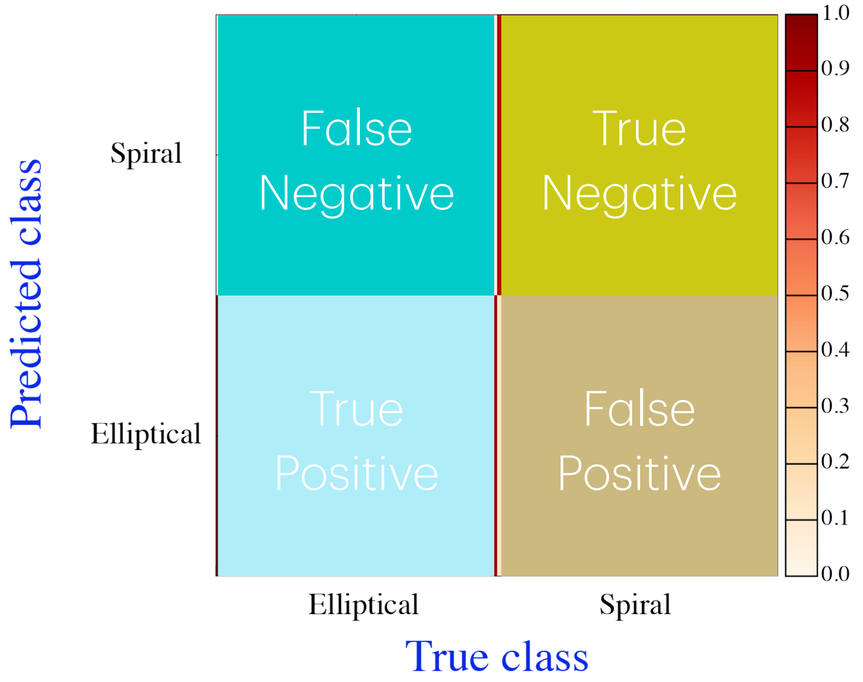
True Negative = **TN**

True Positive = **TP**

False Positive = **FP**

P = TP+FN

N = TN + FP



Some commonly seen terms:

Recall, sensitivity, true positive rate:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Precision, positive predictive value:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

False negative rate:

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Validation - confusion matrix & misclassifications

False Negative = **FN**

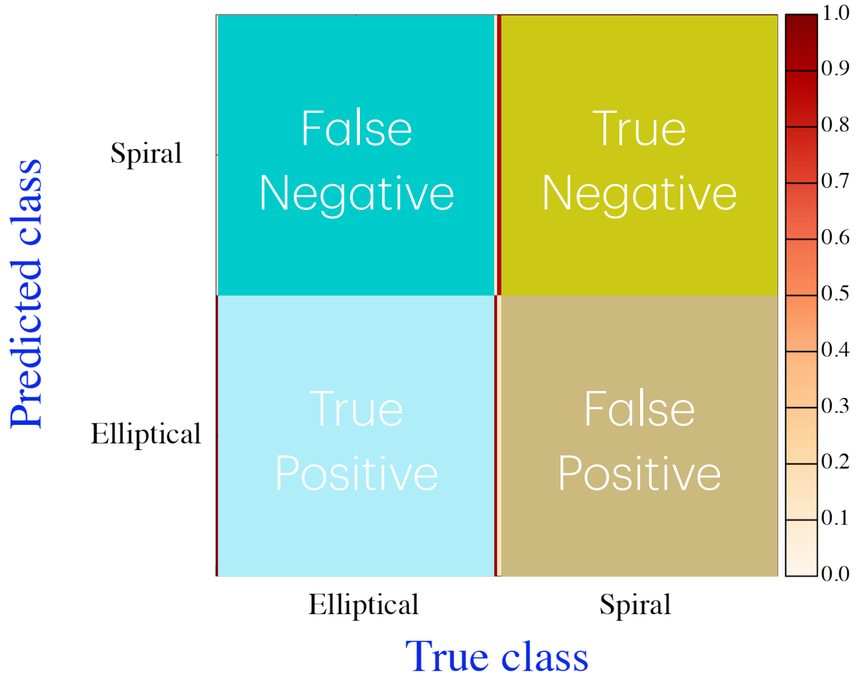
True Negative = **TN**

True Positive = **TP**

False Positive = **FP**

P = TP+FN

N = TN + FP



Some commonly seen terms:

Recall, sensitivity, true positive rate:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Precision, positive predictive value:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

False negative rate:

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Accuracy:

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}}$$

Validation - confusion matrix & misclassifications

False Negative = **FN**

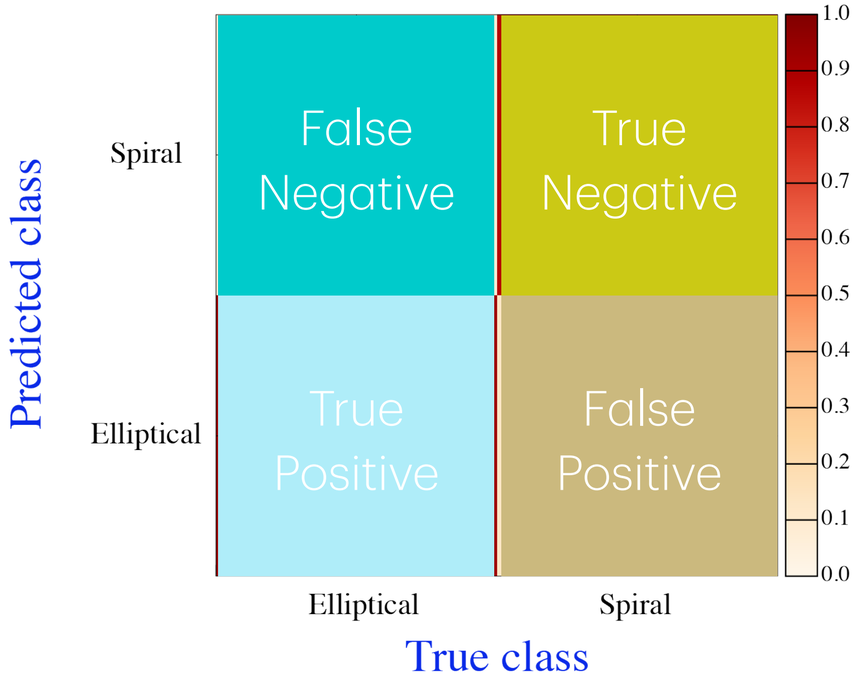
True Negative = **TN**

True Positive = **TP**

False Positive = **FP**

P = TP+FN

N = TN + FP



Some commonly seen terms:

Recall, sensitivity, true positive rate:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Precision, positive predictive value:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

False negative rate:

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Accuracy:

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}}$$

F1 score:

$$\mathrm{F}_1 = 2\frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}}$$

There are many more! See e.g. https://en.wikipedia.org/wiki/Confusion_matrix

Validation - confusion matrix & misclassifications

So what should you use?

Terminology and derivations from a confusion matrix

condition positive (P)
the number of real positive cases in the data
condition negative (N)
the number of real negative cases in the data

true positive (TP)
A test result that correctly indicates the presence of a condition or characteristic
true negative (TN)
A test result that correctly indicates the absence of a condition or characteristic
false positive (FP)
A test result which wrongly indicates that a particular condition or attribute is present
false negative (FN)
A test result which wrongly indicates that a particular condition or attribute is absent

sensitivity, recall, hit rate, or true positive rate (TPR)
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$
specificity, selectivity or true negative rate (TNR)
$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$
precision or positive predictive value (PPV)
$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR}$$
negative predictive value (NPV)
$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}} = 1 - \mathrm{FOR}$$
miss rate or false negative rate (FNR)
$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}$$
fall-out or false positive rate (FPR)
$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$
false discovery rate (FDR)
$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TP}} = 1 - \mathrm{PPV}$$
false omission rate (FOR)
$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TN}} = 1 - \mathrm{NPV}$$
Positive likelihood ratio (LR+)
$$\mathrm{LR+} = \frac{\mathrm{TPR}}{\mathrm{FPR}}$$
Negative likelihood ratio (LR-)
$$\mathrm{LR-} = \frac{\mathrm{FNR}}{\mathrm{TNR}}$$
prevalence threshold (PT)
$$\mathrm{PT} = \frac{\sqrt{\mathrm{FPR}}}{\sqrt{\mathrm{TPR}} + \sqrt{\mathrm{FPR}}}$$
threat score (TS) or critical success index (CSI)
$$\mathrm{TS} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN} + \mathrm{FP}}$$

Prevalence
$$\frac{\mathrm{P}}{\mathrm{P} + \mathrm{N}}$$
accuracy (ACC)
$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$
balanced accuracy (BA)
$$\mathrm{BA} = \frac{TPR + TNR}{2}$$
F1 score
is the harmonic mean of precision and sensitivity:
$$\mathrm{F}_1 = 2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$
phi coefficient (φ or r_φ) or Matthews correlation coefficient (MCC)
$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}}$$
Fowlkes–Mallows index (FM)
$$\mathrm{FM} = \sqrt{\frac{TP}{TP + FP} \times \frac{TP}{TP + FN}} = \sqrt{PPV \times TPR}$$
informedness or bookmaker informedness (BM)
$$\mathrm{BM} = \mathrm{TPR} + \mathrm{TNR} - 1$$
markedness (MK) or deltaP (Δp)
$$\mathrm{MK} = \mathrm{PPV} + \mathrm{NPV} - 1$$
Diagnostic odds ratio (DOR)
$$\mathrm{DOR} = \frac{\mathrm{LR+}}{\mathrm{LR-}}$$

Sources: Fawcett (2006),[1] Piryonesi and El-Diraby (2020),[2] Powers (2011),[3] Ting (2011),[4] CAWCR,[5] D. Chicco & G. Jurman (2020, 2021, 2023),[6][7][8] Tharwat (2018),[9] Balayla (2020)[10]

Validation - confusion matrix & misclassifications

So what should you use?

It depends on your question!

Terminology and derivations from a confusion matrix

condition positive (P)
the number of real positive cases in the data
condition negative (N)
the number of real negative cases in the data

true positive (TP)
A test result that correctly indicates the presence of a condition or characteristic
true negative (TN)
A test result that correctly indicates the absence of a condition or characteristic
false positive (FP)
A test result which wrongly indicates that a particular condition or attribute is present
false negative (FN)
A test result which wrongly indicates that a particular condition or attribute is absent

sensitivity, recall, hit rate, or true positive rate (TPR)
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$
specificity, selectivity or true negative rate (TNR)
$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$
precision or positive predictive value (PPV)
$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR}$$
negative predictive value (NPV)
$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}} = 1 - \mathrm{FOR}$$
miss rate or false negative rate (FNR)
$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}$$
fall-out or false positive rate (FPR)
$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$
false discovery rate (FDR)
$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TP}} = 1 - \mathrm{PPV}$$
false omission rate (FOR)
$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TN}} = 1 - \mathrm{NPV}$$
Positive likelihood ratio (LR+)
$$\mathrm{LR+} = \frac{\mathrm{TPR}}{\mathrm{FPR}}$$
Negative likelihood ratio (LR-)
$$\mathrm{LR-} = \frac{\mathrm{FNR}}{\mathrm{TNR}}$$
prevalence threshold (PT)
$$\mathrm{PT} = \frac{\sqrt{\mathrm{FPR}}}{\sqrt{\mathrm{TPR}} + \sqrt{\mathrm{FPR}}}$$
threat score (TS) or critical success index (CSI)
$$\mathrm{TS} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN} + \mathrm{FP}}$$

Prevalence
$$\frac{\mathrm{P}}{\mathrm{P} + \mathrm{N}}$$
accuracy (ACC)
$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$
balanced accuracy (BA)
$$\mathrm{BA} = \frac{TPR + TNR}{2}$$
F1 score
is the harmonic mean of precision and sensitivity:
$$\mathrm{F_1} = 2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$
phi coefficient (φ or r_φ) or Matthews correlation coefficient (MCC)
$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}}$$
Fowlkes–Mallows index (FM)
$$\mathrm{FM} = \sqrt{\frac{TP}{TP + FP} \times \frac{TP}{TP + FN}} = \sqrt{PPV \times TPR}$$
informedness or bookmaker informedness (BM)
$$\mathrm{BM} = \mathrm{TPR} + \mathrm{TNR} - 1$$
markedness (MK) or deltaP (Δp)
$$\mathrm{MK} = \mathrm{PPV} + \mathrm{NPV} - 1$$
Diagnostic odds ratio (DOR)
$$\mathrm{DOR} = \frac{\mathrm{LR+}}{\mathrm{LR-}}$$

Sources: Fawcett (2006),[1] Piryonesi and El-Diraby (2020),[2] Powers (2011),[3] Ting (2011),[4] CAWCR,[5] D. Chicco & G. Jurman (2020, 2021, 2023),[6][7][8] Tharwat (2018),[9] Balayla (2020)[10]

Validation - confusion matrix & misclassifications

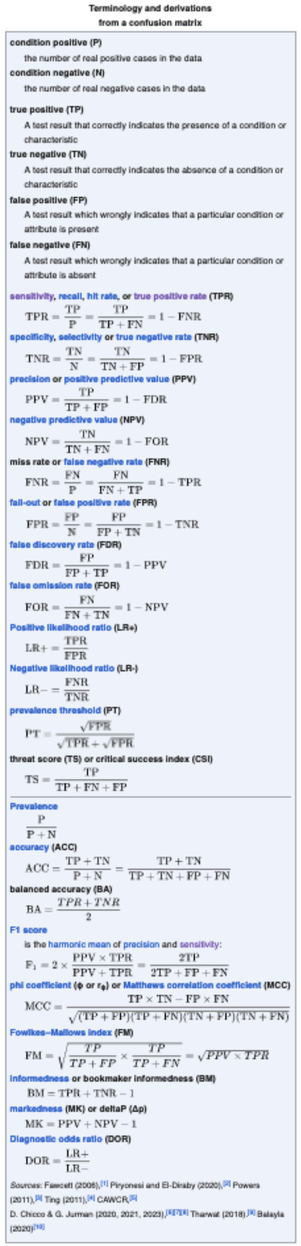
So what should you use?

It depends on your question!

If you are doing diagnosis of a deadly disease you are (hopefully) most interested in the false negatives.

If you want to create as pure a sample as possible, you probably want to focus on false positive rate.

If you are interested in a good overall metric, the F1 score is possibly your thing.



Terminology and derivations from a confusion matrix

condition positive (P)
the number of real positive cases in the data
condition negative (N)
the number of real negative cases in the data

true positive (TP)
A test result that correctly indicates the presence of a condition or characteristic
true negative (TN)
A test result that correctly indicates the absence of a condition or characteristic
false positive (FP)
A test result which wrongly indicates that a particular condition or attribute is present
false negative (FN)
A test result which wrongly indicates that a particular condition or attribute is absent

sensitivity, recall, hit rate, or true positive rate (TPR)
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = 1 - \mathrm{FNR}$$
specificity, selectivity or true negative rate (TNR)
$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}} = 1 - \mathrm{FPR}$$
precision or positive predictive value (PPV)
$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = 1 - \mathrm{FDR}$$
negative predictive value (NPV)
$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}} = 1 - \mathrm{FOR}$$
miss rate or false negative rate (FNR)
$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TP}} = 1 - \mathrm{TPR}$$
fall-out or false positive rate (FPR)
$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} = 1 - \mathrm{TNR}$$
false discovery rate (FDR)
$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TP}} = 1 - \mathrm{PPV}$$
false omission rate (FOR)
$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TN}} = 1 - \mathrm{NPV}$$
Positive likelihood ratio (LR+)
$$\mathrm{LR+} = \frac{\mathrm{TPR}}{\mathrm{FPR}}$$
Negative likelihood ratio (LR-)
$$\mathrm{LR-} = \frac{\mathrm{FNR}}{\mathrm{TNR}}$$
prevalence threshold (PT)
$$\mathrm{PT} = \frac{\sqrt{\mathrm{FPR}}}{\sqrt{\mathrm{TPR}}+\sqrt{\mathrm{FPR}}}$$
threat score (TS) or critical success index (CSI)
$$\mathrm{TS} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}+\mathrm{FP}}$$

Prevalence
$$\frac{\mathrm{P}}{\mathrm{P}+\mathrm{N}}$$
accuracy (ACC)
$$\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{P}+\mathrm{N}} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$$
balanced accuracy (BA)
$$\mathrm{BA} = \frac{TPR+TNR}{2}$$
F1 score
is the harmonic mean of precision and sensitivity:
$$\mathrm{F}_1 = 2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}$$
phi coefficient (φ or r_φ) or Matthews correlation coefficient (MCC)
$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$
Fowlkes–Mallows index (FM)
$$\mathrm{FM} = \sqrt{\frac{TP}{TP+FP} \times \frac{TP}{TP+FN}} = \sqrt{PPV \times TPR}$$
informedness or bookmaker informedness (BM)
$$\mathrm{BM} = \mathrm{TPR} + \mathrm{TNR} - 1$$
markedness (MK) or deltaP (Δp)
$$\mathrm{MK} = \mathrm{PPV} + \mathrm{NPV} - 1$$
Diagnostic odds ratio (DOR)
$$\mathrm{DOR} = \frac{\mathrm{LR+}}{\mathrm{LR-}}$$

Sources: Fawcett (2006),[1] Piryonesi and El-Diraby (2020),[2] Powers (2011),[3] Ting (2011),[4] CAWCR,[5] D. Chicco & G. Jurman (2020, 2021, 2023),[6][7][8] Tharwat (2018),[9] Balayla (2020)[10]

Validation - confusion matrix & misclassifications

So what should you use?

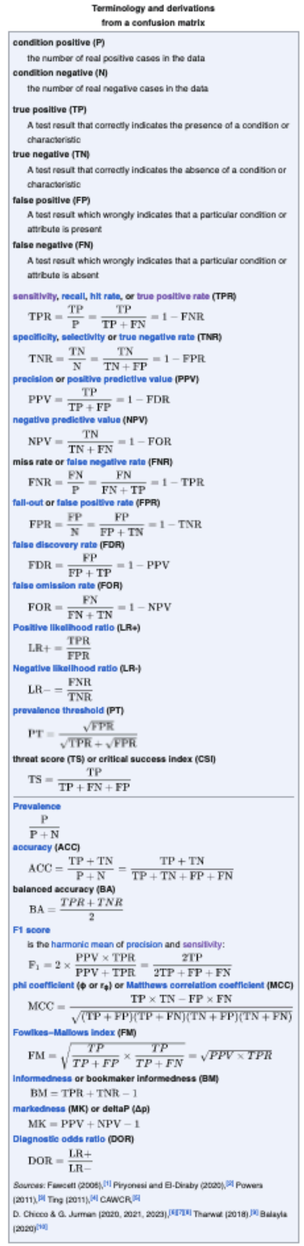
It depends on your question!

If you are doing diagnosis of a deadly disease you are (hopefully) most interested in the false negatives.

If you want to create as pure a sample as possible, you probably want to focus on false positive rate.

If you are interested in a good overall metric, the F1 score is possibly your thing.

The most important: think about what you need - the actual metric will come almost automatically.



Terminology and derivations from a confusion matrix

condition positive (P)
the number of real positive cases in the data
condition negative (N)
the number of real negative cases in the data

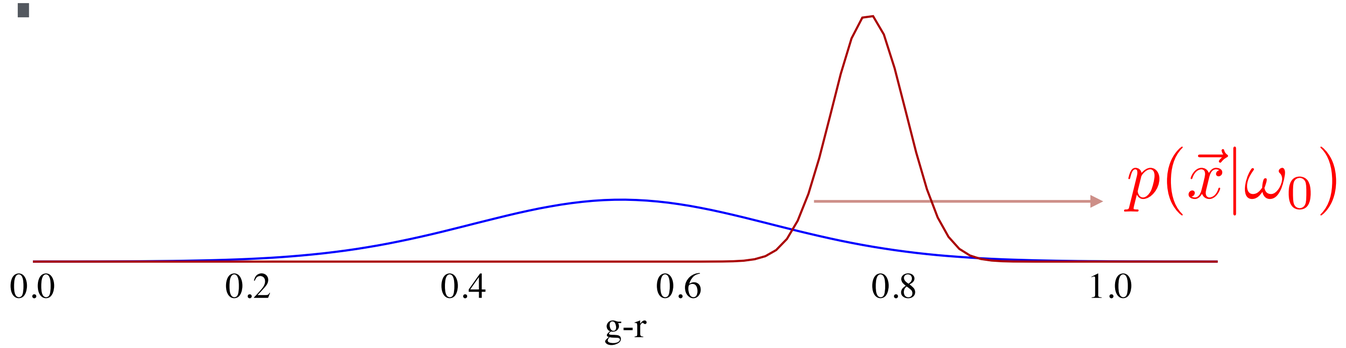
true positive (TP)
A test result that correctly indicates the presence of a condition or characteristic
true negative (TN)
A test result that correctly indicates the absence of a condition or characteristic
false positive (FP)
A test result which wrongly indicates that a particular condition or attribute is present
false negative (FN)
A test result which wrongly indicates that a particular condition or attribute is absent

sensitivity, recall, hit rate, or true positive rate (TPR)
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = 1 - \mathrm{FNR}$$
specificity, selectivity or true negative rate (TNR)
$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}} = 1 - \mathrm{FPR}$$
precision or positive predictive value (PPV)
$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = 1 - \mathrm{FDR}$$
negative predictive value (NPV)
$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}} = 1 - \mathrm{FOR}$$
miss rate or false negative rate (FNR)
$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TP}} = 1 - \mathrm{TPR}$$
fall-out or false positive rate (FPR)
$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} = 1 - \mathrm{TNR}$$
false discovery rate (FDR)
$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TP}} = 1 - \mathrm{PPV}$$
false omission rate (FOR)
$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TN}} = 1 - \mathrm{NPV}$$
Positive likelihood ratio (LR+)
$$\mathrm{LR+} = \frac{\mathrm{TPR}}{\mathrm{FPR}}$$
Negative likelihood ratio (LR-)
$$\mathrm{LR-} = \frac{\mathrm{FNR}}{\mathrm{TNR}}$$
prevalence threshold (PT)
$$\mathrm{PT} = \frac{\sqrt{\mathrm{FPR}}}{\sqrt{\mathrm{TPR}}+\sqrt{\mathrm{FPR}}}$$
threat score (TS) or critical success index (CSI)
$$\mathrm{TS} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}+\mathrm{FP}}$$

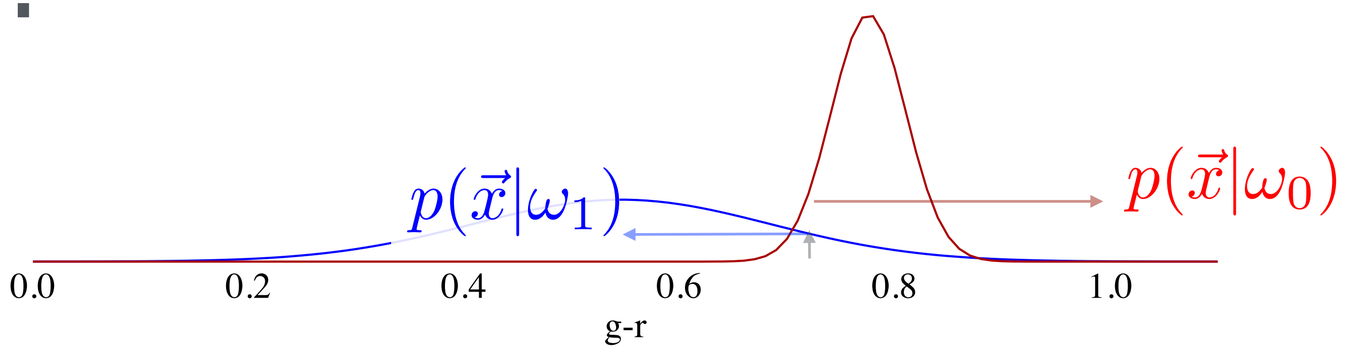
Prevalence
$$\frac{\mathrm{P}}{\mathrm{P}+\mathrm{N}}$$
accuracy (ACC)
$$\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{P}+\mathrm{N}} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$$
balanced accuracy (BA)
$$\mathrm{BA} = \frac{TPR+TNR}{2}$$
F1 score
is the harmonic mean of precision and sensitivity:
$$\mathrm{F_1} = 2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}$$
phi coefficient (φ or r_φ) or Matthews correlation coefficient (MCC)
$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$
Fowlkes–Mallows index (FM)
$$\mathrm{FM} = \sqrt{\frac{TP}{TP+FP} \times \frac{TP}{TP+FN}} = \sqrt{PPV \times TPR}$$
informedness or bookmaker informedness (BM)
$$\mathrm{BM} = \mathrm{TPR} + \mathrm{TNR} - 1$$
markedness (MK) or deltaP (Δp)
$$\mathrm{MK} = \mathrm{PPV} + \mathrm{NPV} - 1$$
Diagnostic odds ratio (DOR)
$$\mathrm{DOR} = \frac{\mathrm{LR+}}{\mathrm{LR-}}$$

Sources: Fawcett (2006),[1] Piryonesi and El-Diraby (2020),[2] Powers (2011),[3] Ting (2011),[4] CAWCR,[5] D. Chicco & G. Jurman (2020, 2021, 2023),[6][7][8] Tharwat (2018),[9] Balayla (2020)[10]

Reminder:

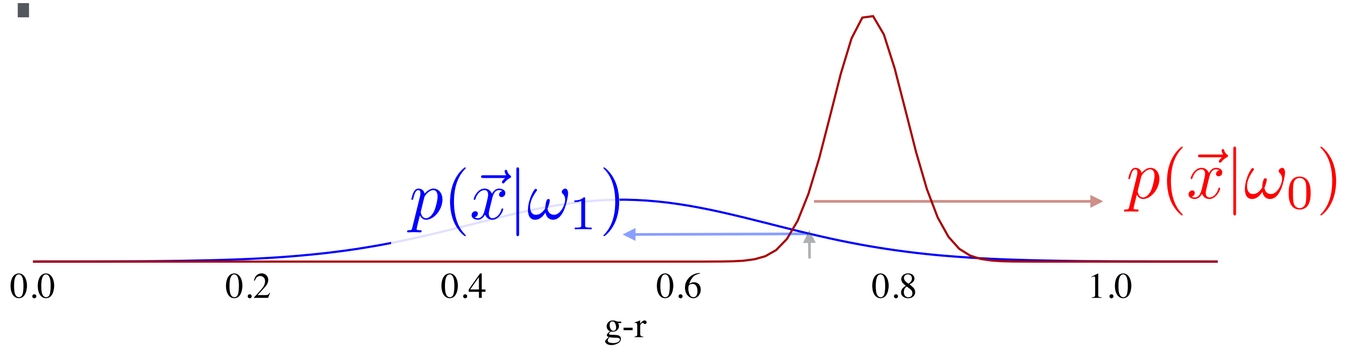


Reminder:

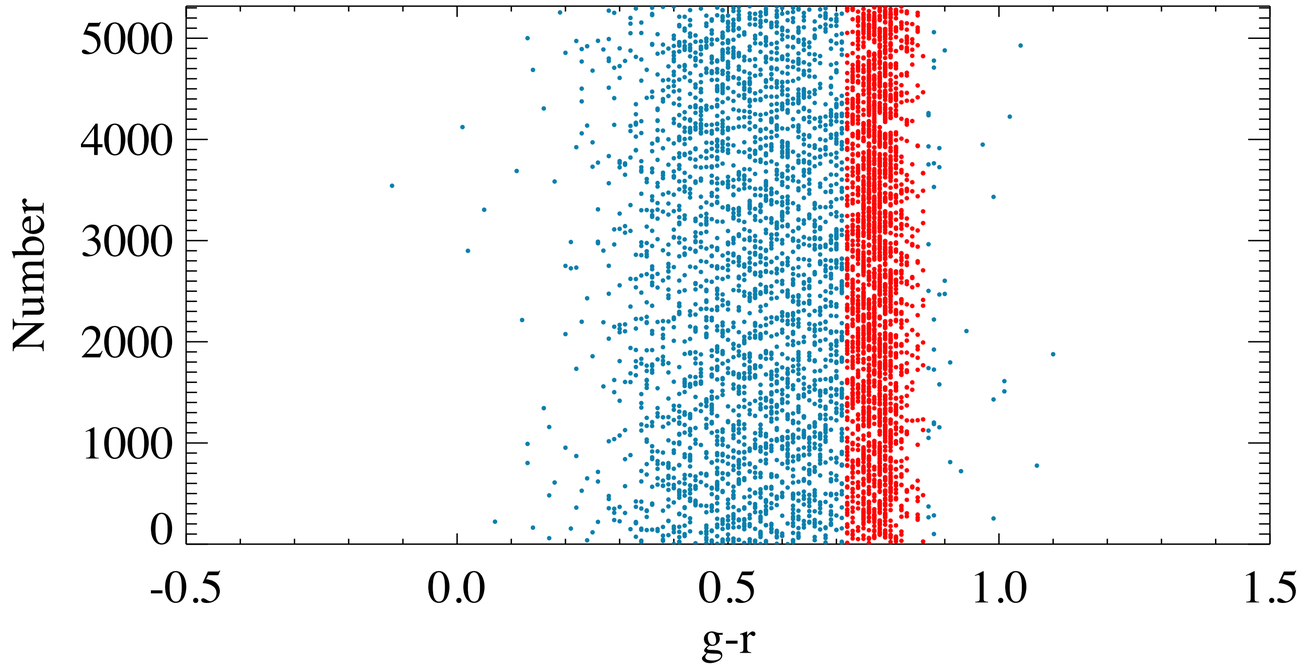


So in this case we assign a class Elliptical and we can also give the likelihood relative to the spiral class. We do this for all objects:

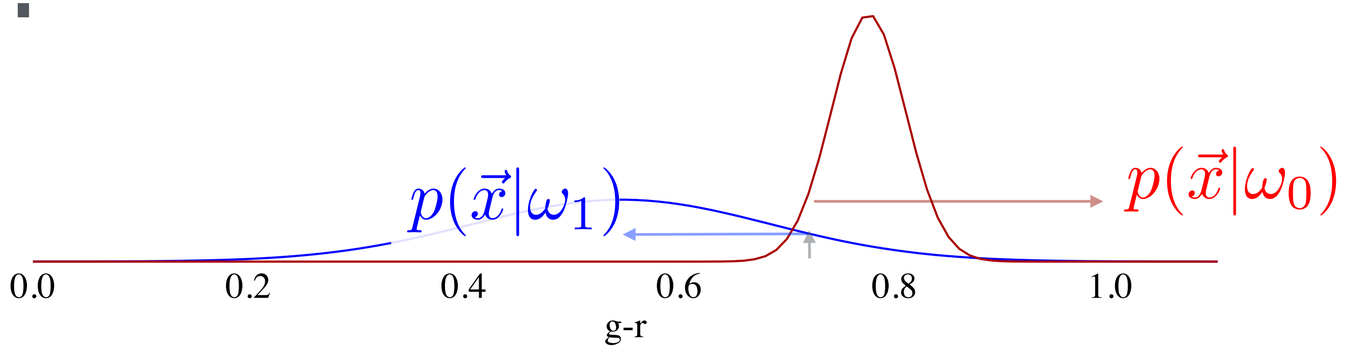
Reminder:



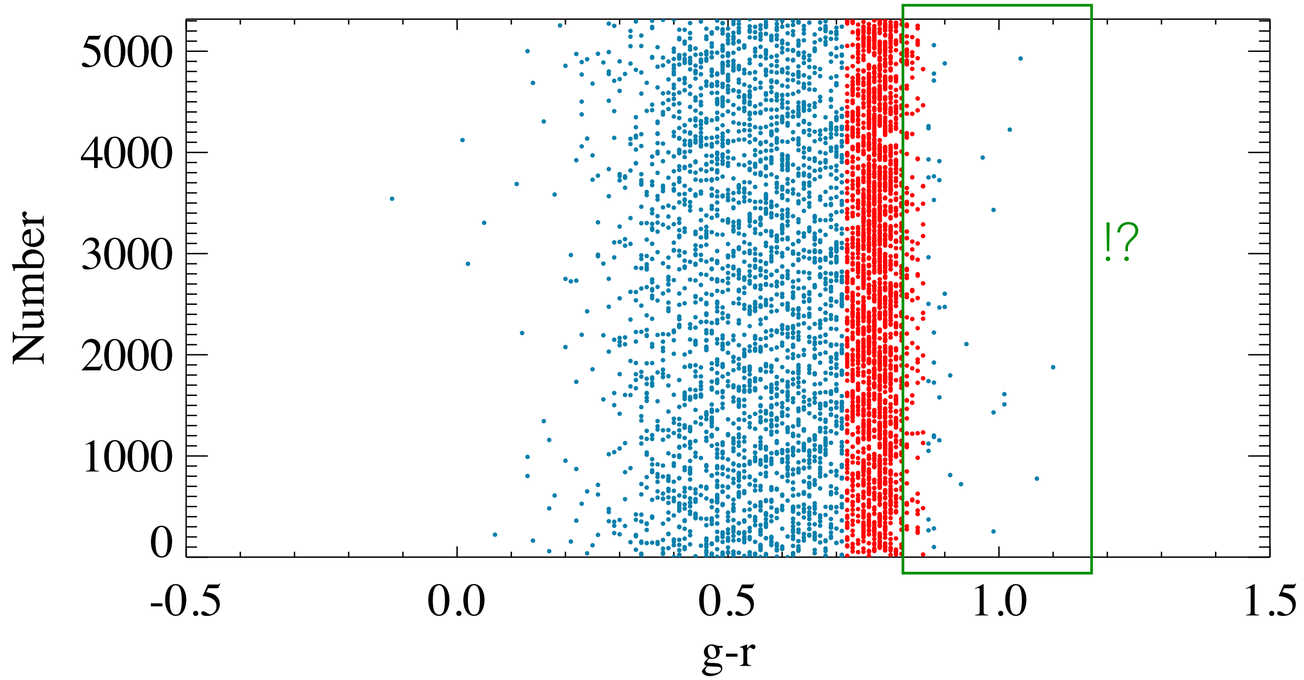
So in this case we assign a class Elliptical and we can also give the likelihood relative to the spiral class. We do this for all objects:



Reminder:



So in this case we assign a class Elliptical and we can also give the likelihood relative to the spiral class. We do this for all objects:



High dimensionality - Naïve Bayes & Bayesian networks

That appears to be a simple technique, but estimating $p(\vec{x} \mid \omega_i)$ is challenging in high dimensions - if you need N objects for a 1D PDF, you need N^d data points for d-dimensional PDF.

A simple way to reduce the requirements is to assume that all variables are independent, so that you have

$$p(\vec{x}|\omega_j) = \prod_i p(x_i|\omega_j)$$

Now you can just repeat what I showed for each 'feature' and multiply the results together - this is known as **Naïve Bayes**.

A bit more sophisticated is to partially factorise $p(\vec{x} \mid \omega_i)$. This leads to **Bayesian networks**. But the process is very similar to what we have seen.

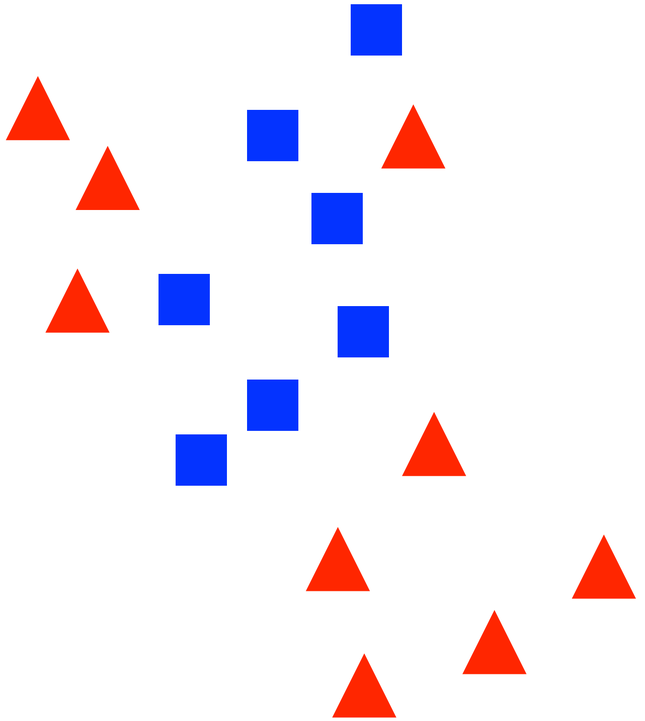
Other classification/grouping methods

In many situations a full application of Bayesian statistics is difficult to carry out, particularly in high dimension. There are therefore a range of alternative methods that are less “optimal” but can provide very flexible and useful techniques.

It is worth keeping in mind however, that the Bayesian technique not only is optimal (if you know the PDFs…), but it also assigns a probability for belonging to a particular class.

k-Nearest neighbours

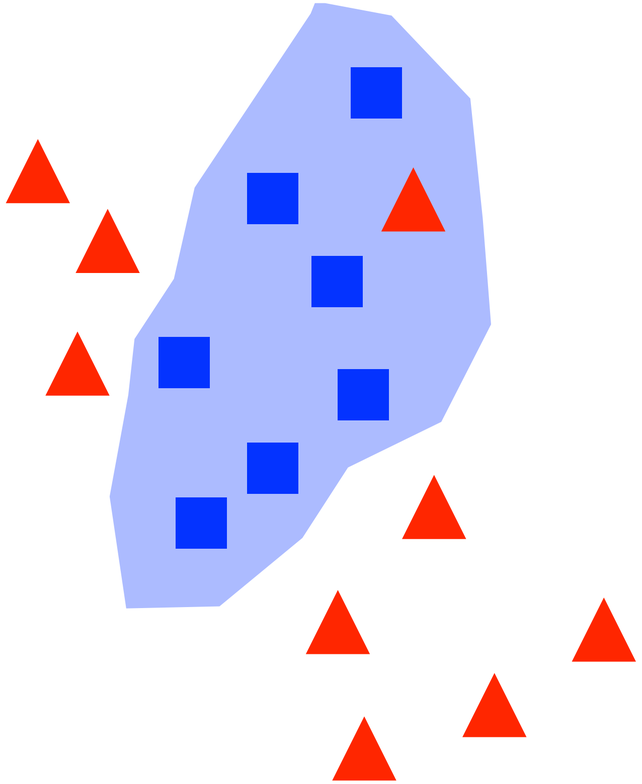
Training: For each object, find the nearest k objects - assign the class of the majority of the neighbours.



In python: `from sklearn.neighbors import KNeighborsClassifier`

k-Nearest neighbours

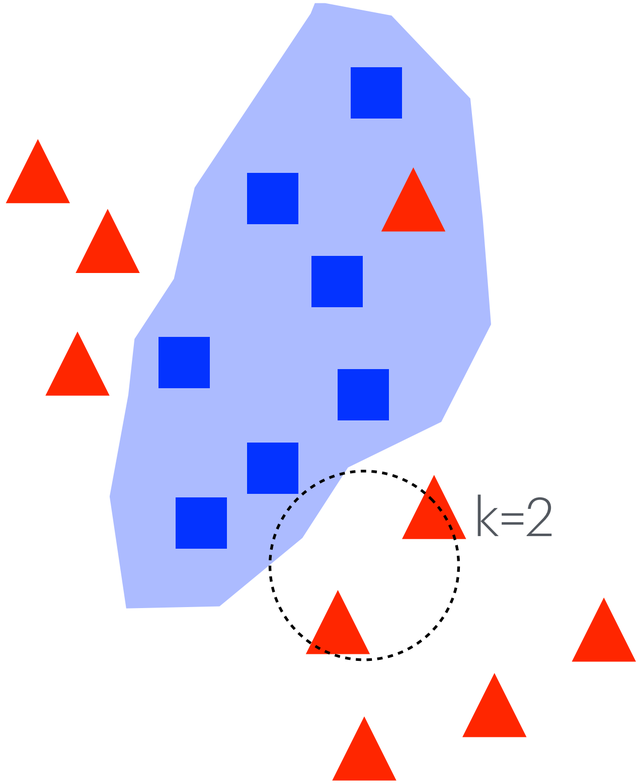
Training: For each object, find the nearest k objects - assign the class of the majority of the neighbours.



In python: `from sklearn.neighbors import KNeighborsClassifier`

k-Nearest neighbours

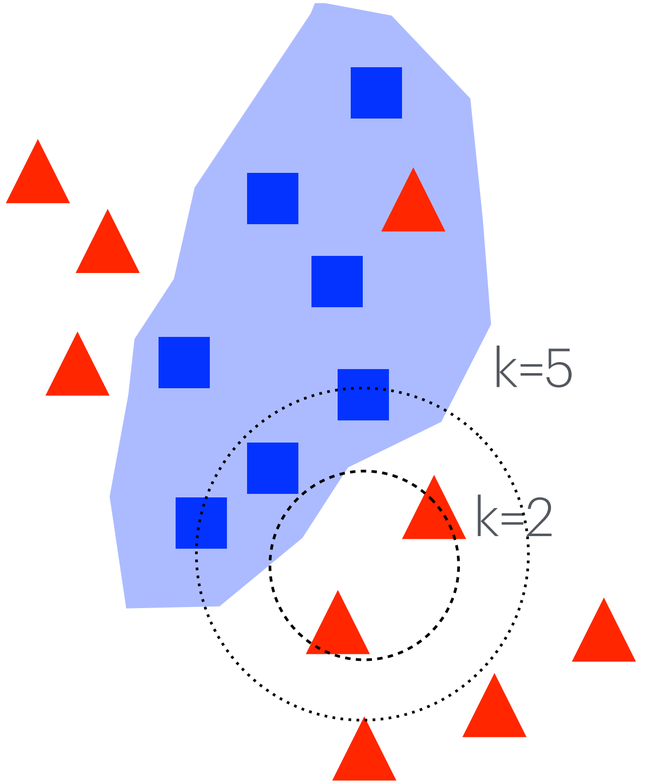
Training: For each object, find the nearest k objects - assign the class of the majority of the neighbours.



In python: `from sklearn.neighbors import KNeighborsClassifier`

k-Nearest neighbours

Training: For each object, find the nearest k objects - assign the class of the majority of the neighbours.

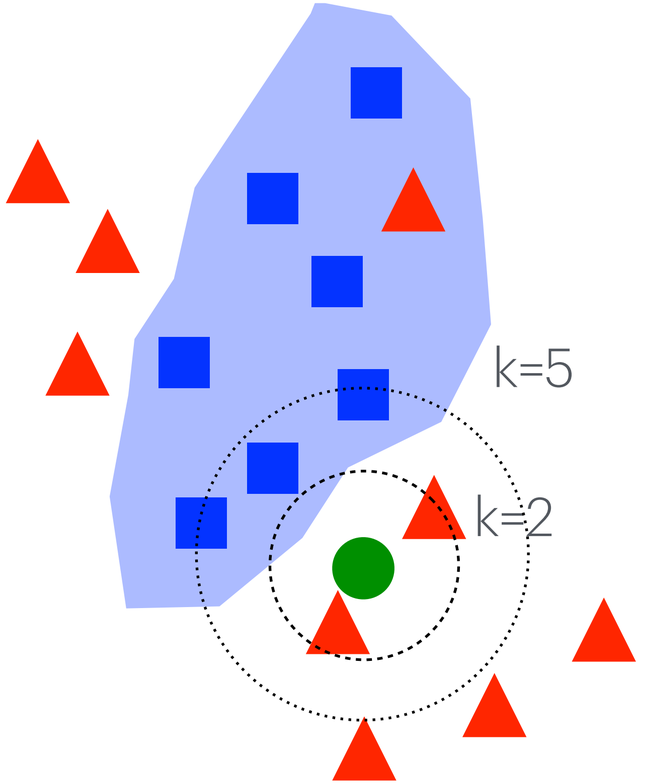


In python: `from sklearn.neighbors import KNeighborsClassifier`

k-Nearest neighbours

Training: For each object, find the nearest k objects - assign the class of the majority of the neighbours.

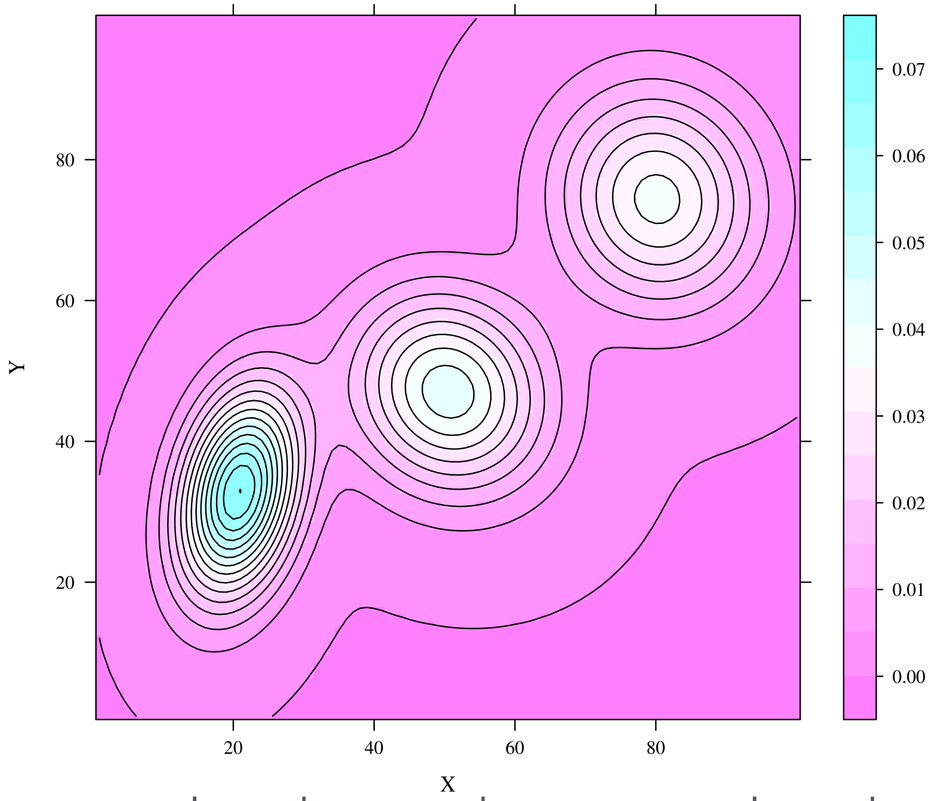
Application: Find the nearest k objects and assign the majority class.



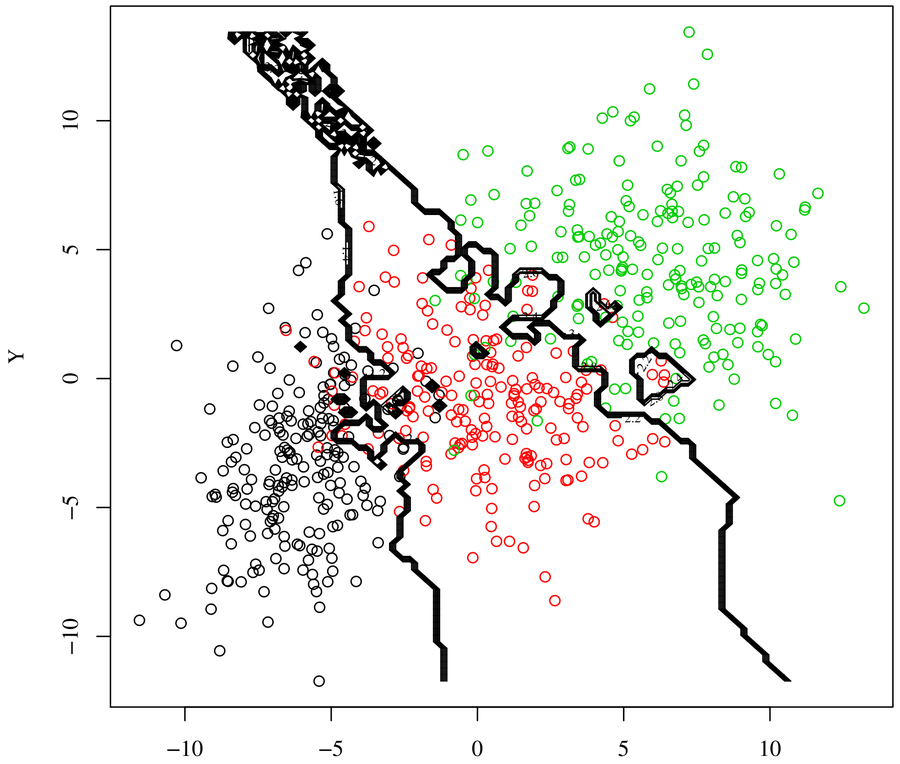
In python: `from sklearn.neighbors import KNeighborsClassifier`

k-Nearest neighbours

Underlying distributions



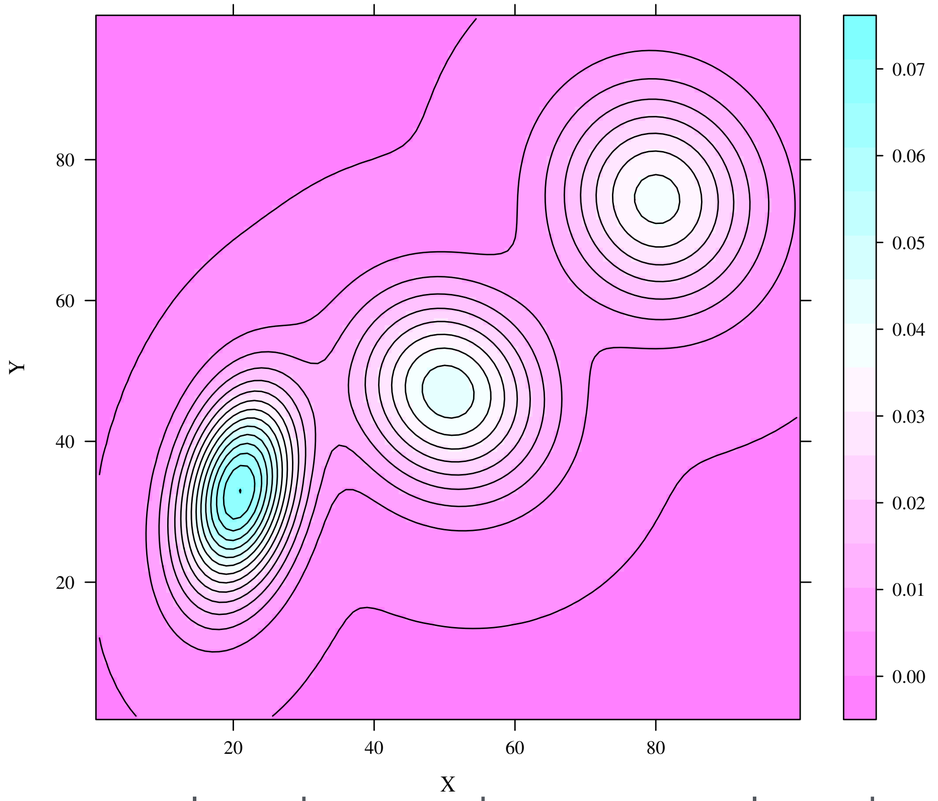
The result of applying knn



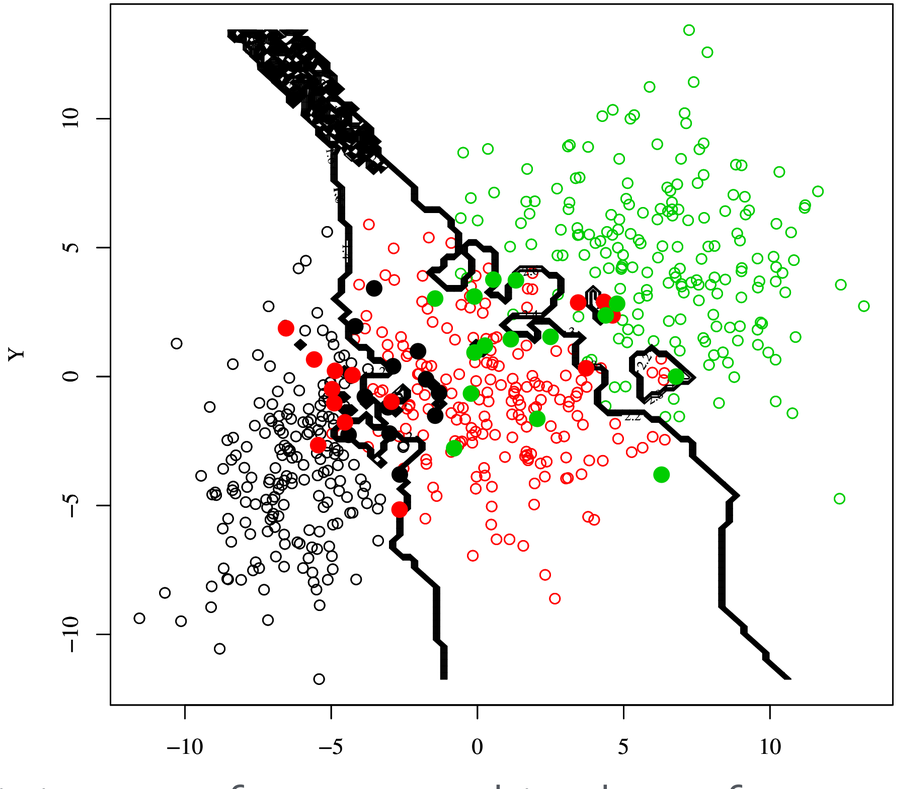
knn can lead to rather complex decision surfaces and is therefore not always optimal for **understanding** what is going on.

k-Nearest neighbours

Underlying distributions



The result of applying knn



knn can lead to rather complex decision surfaces and is therefore not always optimal for **understanding** what is going on.

Dimensional reduction

Fitting models & dimensionality

Assume you wish to fit a model to a high order polynomial:

$$y = w_0 + \sum_i w_i x_i + \sum_i \sum_j w_{ij} x_i x_j + \sum_i \sum_j \sum_k w_{ijk} x_i x_j x_k$$

The number of terms grows as d^M where d is the number of input variables and M is the polynomial order. So for higher order functions you need to constrain a large number of terms and need huge training datasets.

Fitting models & dimensionality

Assume you wish to fit a model to a high order polynomial:

$$y = w_0 + \sum_i w_i x_i + \sum_i \sum_j w_{ij} x_i x_j + \sum_i \sum_j \sum_k w_{ijk} x_i x_j x_k$$

The number of terms grows as d^M where d is the number of input variables and M is the polynomial order. So for higher order functions you need to constrain a large number of terms and need huge training datasets.

There is a theorem (Barron 1993) saying that for polynomials the error in an approximation goes as $O(1/M^{2/D})$ - D is the dimensionality. While for non-linear functions it goes as O(1/M). For large D you therefore need many more term for polynomial fits than for non-linear fits.

An aside: Higher dimensions - weirdities

You might think that the optimal way to sample spaces is in regular bins - and that the cube & the sphere cover similar volumes.

$$V_{\text{sphere}}(r) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)} r^D$$

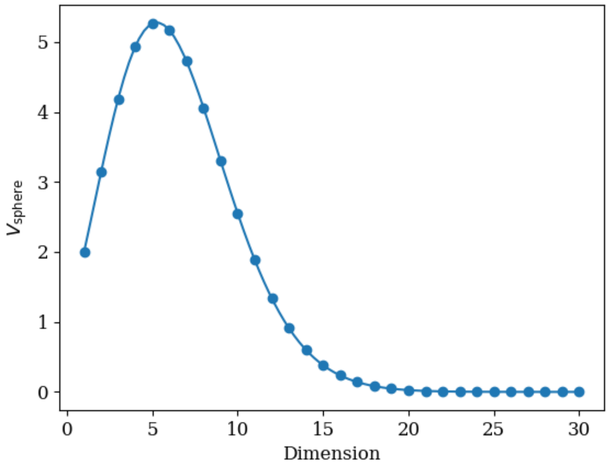
in D dimensions, and since:

$$\frac{V_{\text{sphere}}(1) - V_{\text{sphere}}(1-\epsilon)}{V_{\text{sphere}}(1)} = 1 - (1-\epsilon)^D$$

most of this volume is in a thin shell for high D. And indeed the volume of a sphere goes to zero when D goes to infinity!

Higher dimensions - weirdities

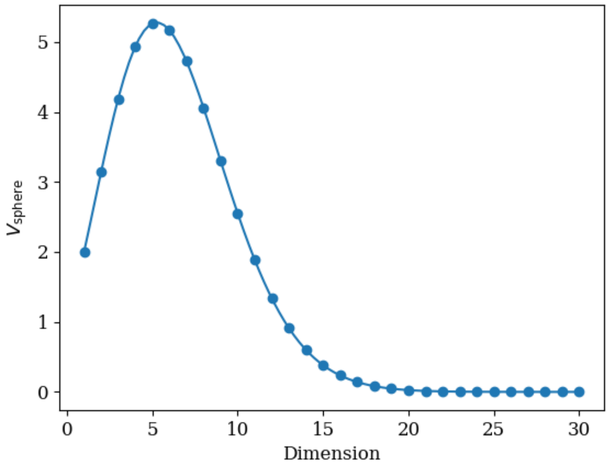
$$V_{\text{sphere}}(r) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)} r^D$$



We can also compare this to the volume of the cube, and we find:

Higher dimensions - weirdities

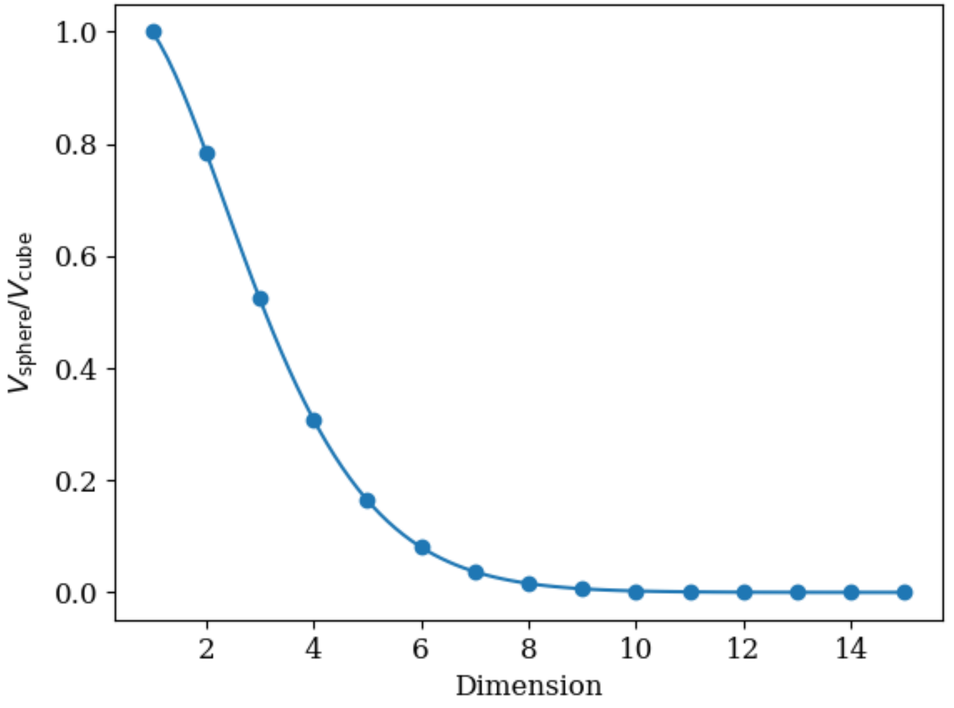
$$V_{\text{sphere}}(r) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)} r^D$$



We can also compare this to the volume of the cube, and we find:

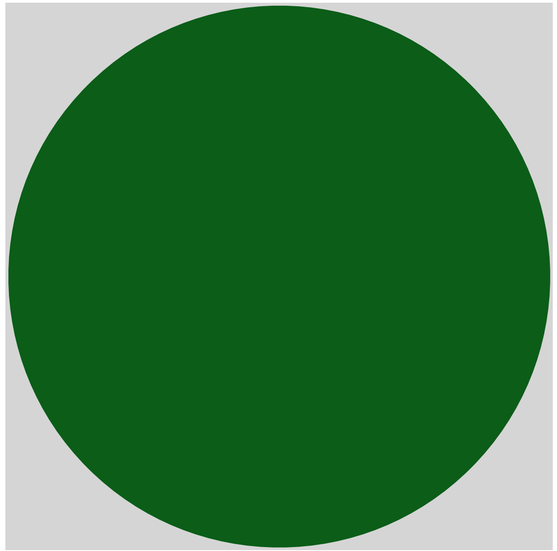
$$\frac{V_{\text{sphere}}}{V_{\text{cube}}} = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)}$$

Very few points lie within a unit radius from the origin!



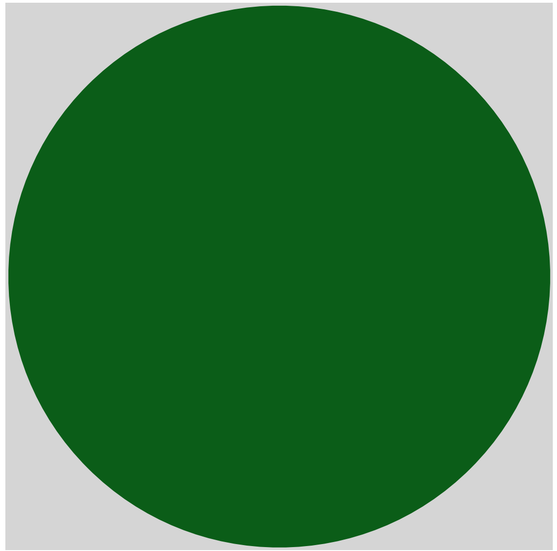
The curse of dimensions

So does this matter? It matters for instance in Monte Carlo techniques



The curse of dimensions

So does this matter? It matters for instance in Monte Carlo techniques



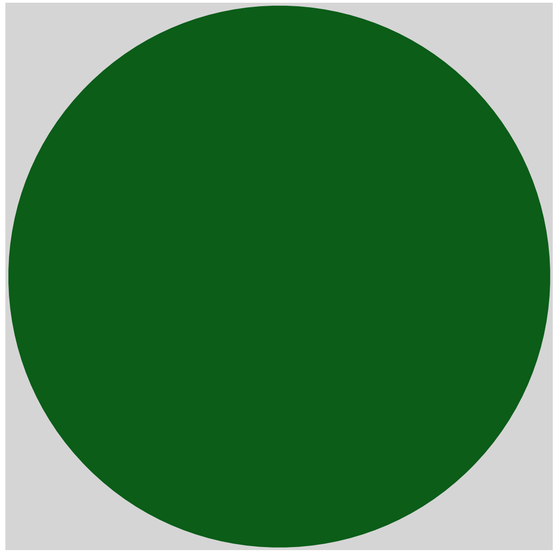
If you want to calculate

$$\int\int f(x, y)\, dxdy$$

over the green disk - you might opt for a Monte Carlo integration technique where you draw random numbers within the gray square and reject those that lie outside the green disk.

The curse of dimensions

So does this matter? It matters for instance in Monte Carlo techniques

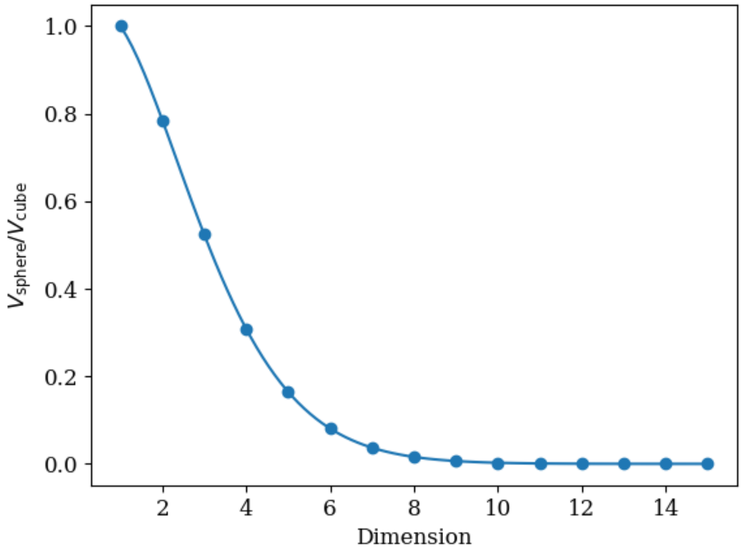


If you want to calculate

$$\int\int f(x,y)\,dxdy$$

over the green disk - you might opt for a Monte Carlo integration technique where you draw random numbers within the gray square and reject those that lie outside the green disk.

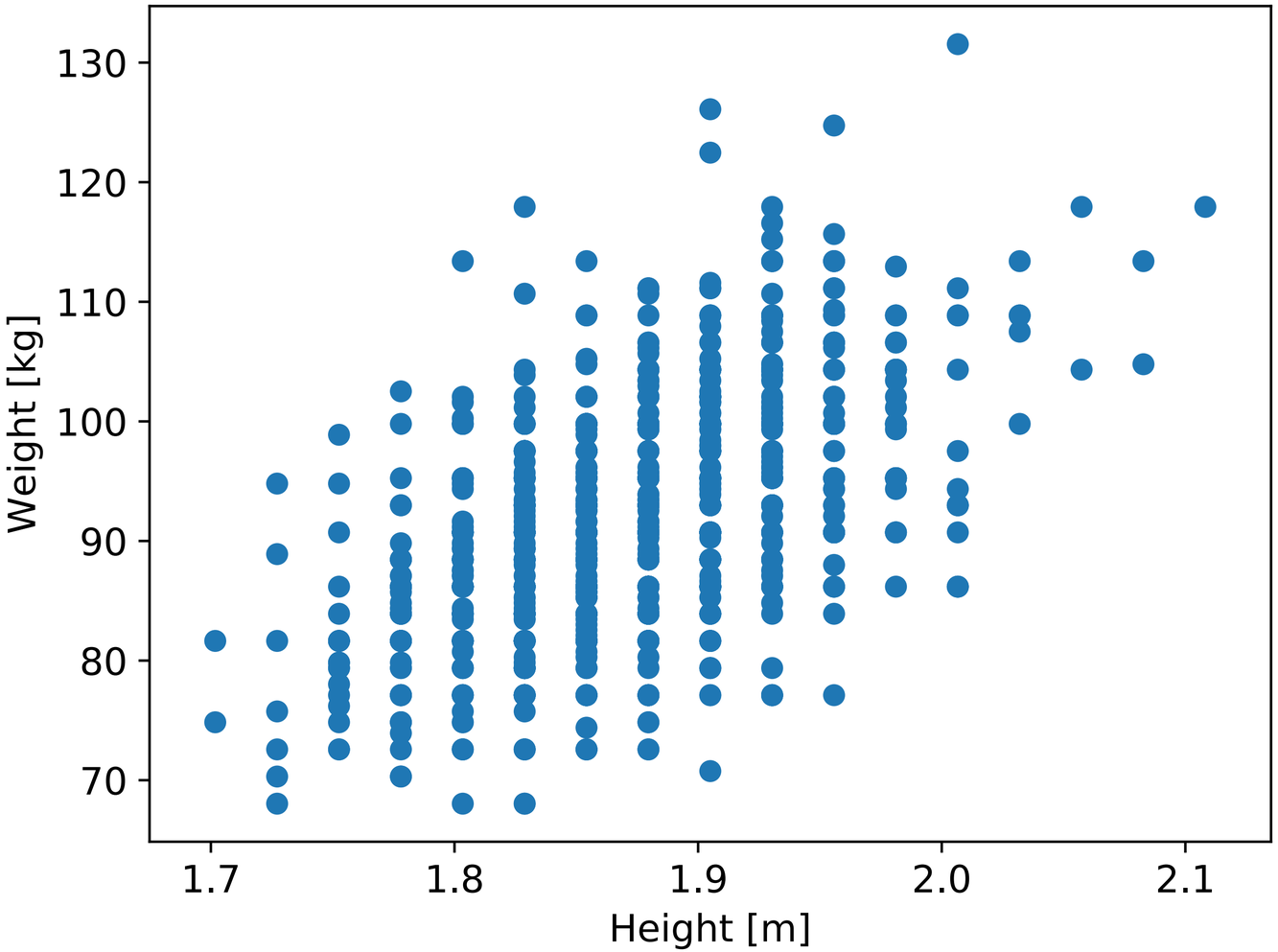
But in high-D, the chances of a point falling in the green sphere is very low!

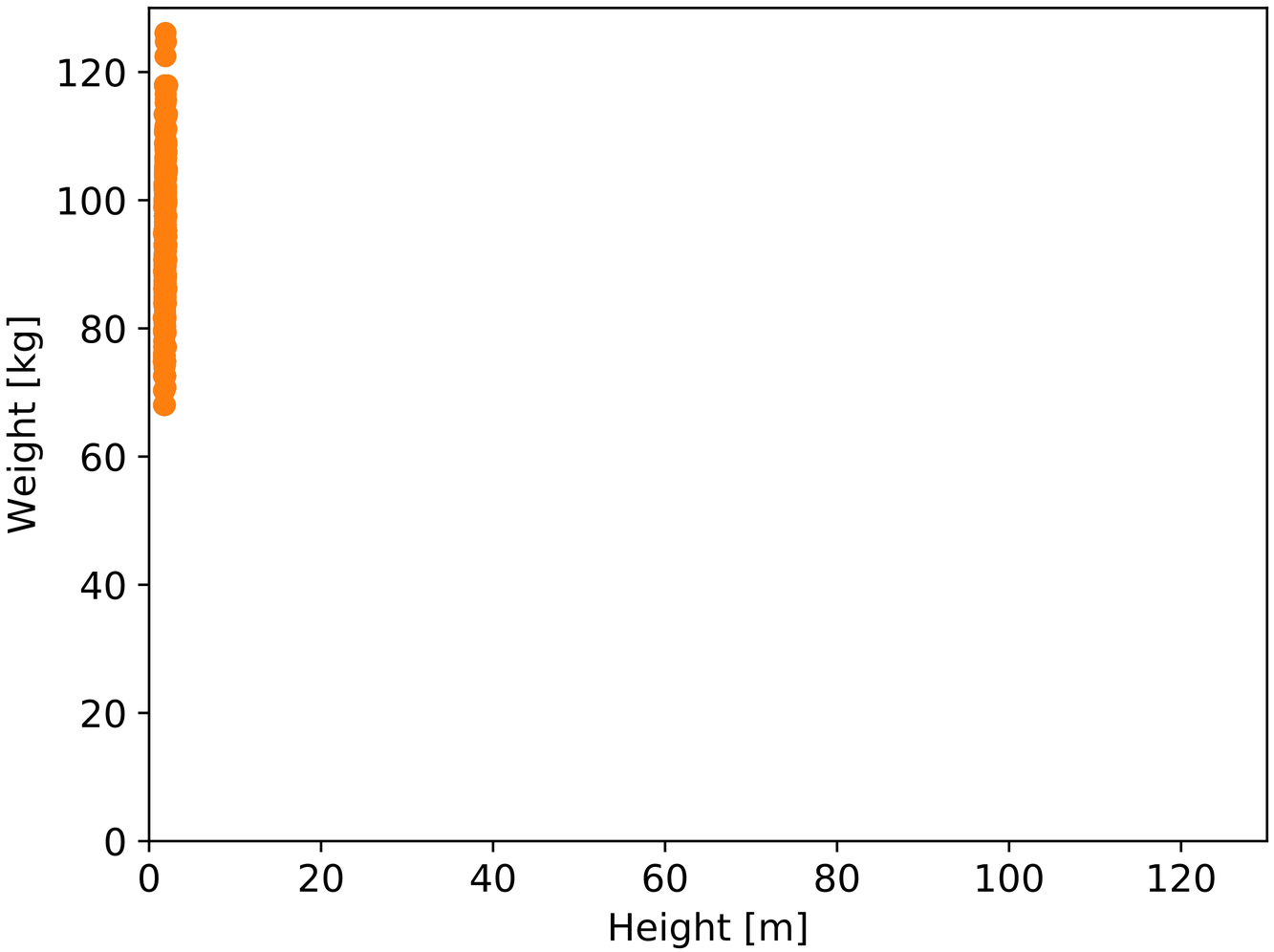


But in reality, life is better

Real physical objects do not fill “space” arbitrarily - thus we can look for ways to **reduce** the effective **dimension** of the problem.

But first - you might need to pre-process your data:


130
120
110
100
90
80
70
Weight [kg]
1.7
1.8
1.9
2.0
2.1
Height [m]


Weight [kg]
120
100
80
60
40
20
0
0
20
40
60
80
100
120
Height [m]

Rescaling data

In many algorithms it is highly desirable that the ranges probed in each dimension are comparable. To achieve that we **scale** our data.

Scale range to -1 to 1, or close to it:

$$x_i \to \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$$

min-max scaling

$$x_i \to \frac{x_i - \bar{x}}{\sigma_x}$$

standardizing

Often called whitening

Rescaling data

In many algorithms it is highly desirable that the ranges probed in each dimension are comparable. To achieve that we **scale** our data.

Scale range to -1 to 1, or close to it:

$$x_i \to \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$$

min-max scaling

$$x_i \to \frac{x_i - \bar{x}}{\sigma_x}$$

standardizing

Often called whitening

Approaches to standardizing

$$x_i \rightarrow \frac{x_i - \bar{x}}{\sigma_x}$$

1. Use sklearn convenience function:

```
from sklearn.preprocessing import StandardScaler

X_std = StandardScaler().fit_transform(X)
```

2. Calculate μ & σ yourself & scale the data manually

Approaches to standardizing

$$x_i \rightarrow \frac{x_i - \bar{x}}{\sigma_x}$$

1. Use sklearn convenience function:

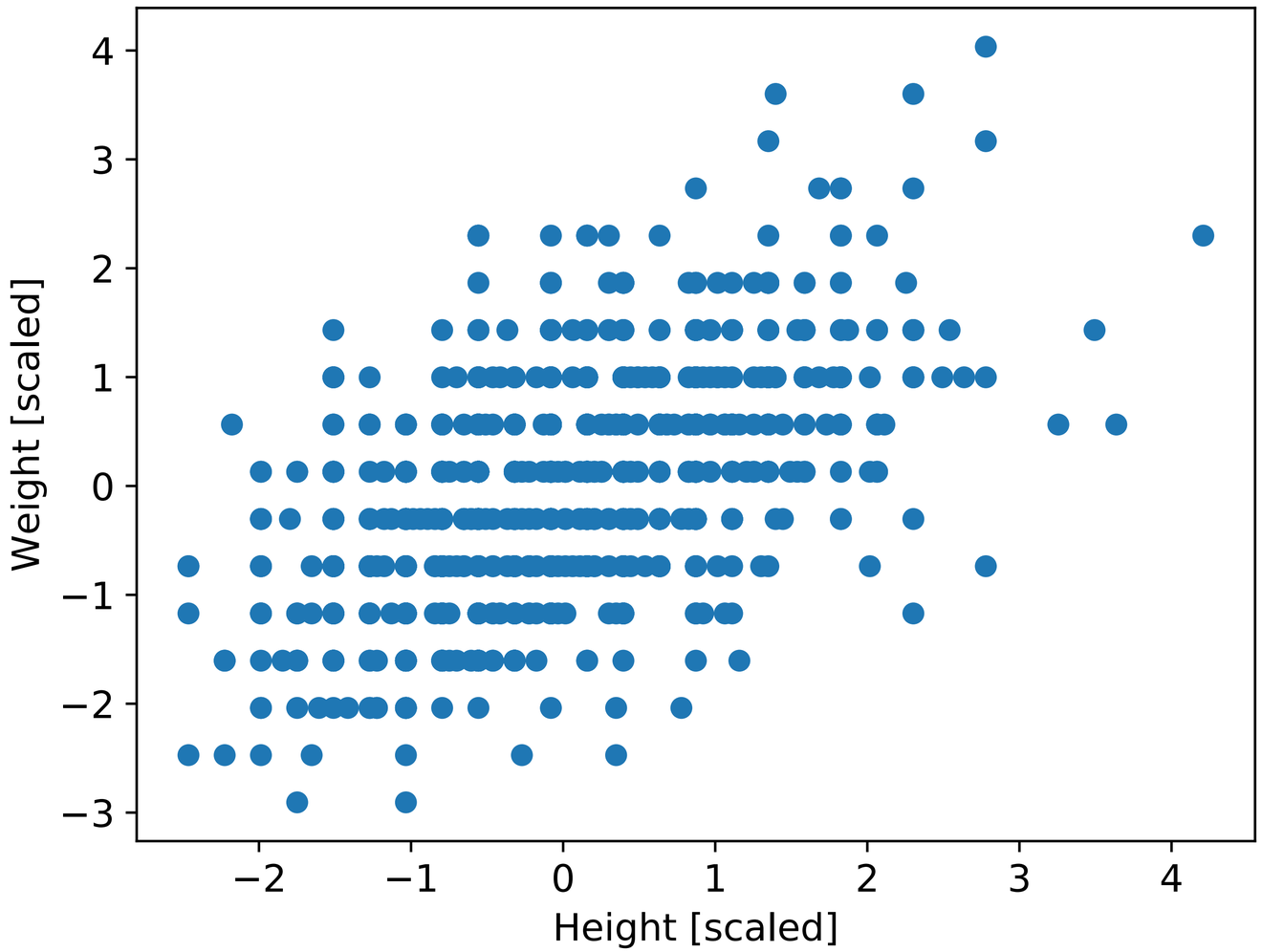
```
from sklearn.preprocessing import StandardScaler

X_std = StandardScaler().fit_transform(X)
```

2. Calculate μ & σ yourself & scale the data manually

1. has the advantage that it can be chained into **pipelines** in sklearn and keeps track of things for you. But if your data are badly affected by outliers it will not detect that for you.

2. has the advantage that you know what you are doing, and if you want to use a robust estimator for the mean or variance you can. The downside is that you need to keep track of the values and remember to apply them!

Whitened data

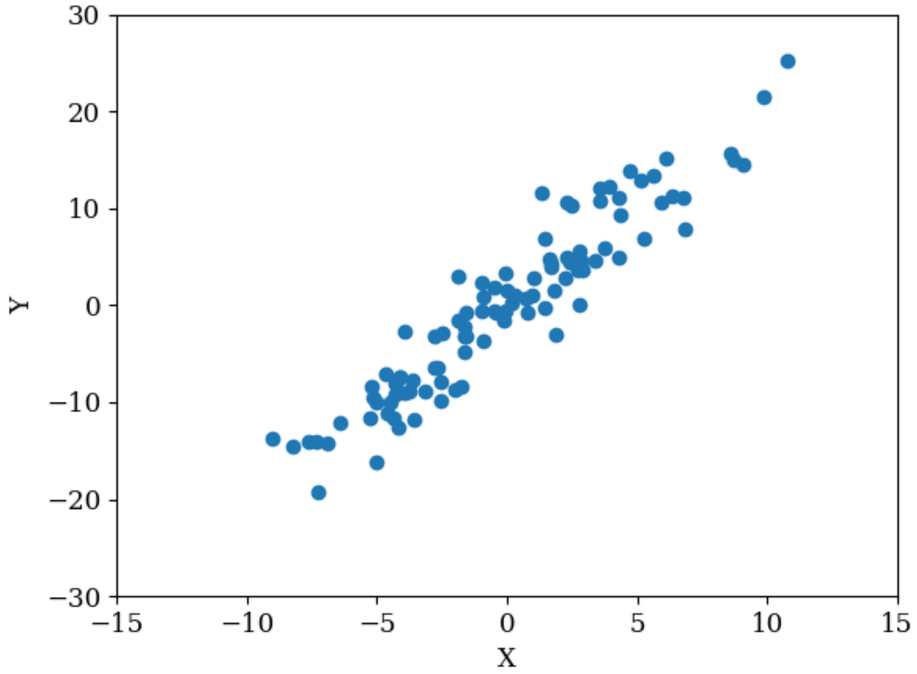


Principal Component Analysis (PCA)

To see the code for the simple example:

`MLD2025-08-Simple PCA example.ipynb`

An introductory example



Start with a dataset with x & y values.

Step 1: Subtract the means.

We now want to find the direction along which the data vary most.

What direction?
Look at: Covariance matrix

$$\mathrm{Cov}(i,j) = \frac{1}{N}\sum_{n=1}^{N}\left(v_{i,n} - \langle v \rangle\right)\left(v_{j,n} - \langle v_j \rangle\right)$$

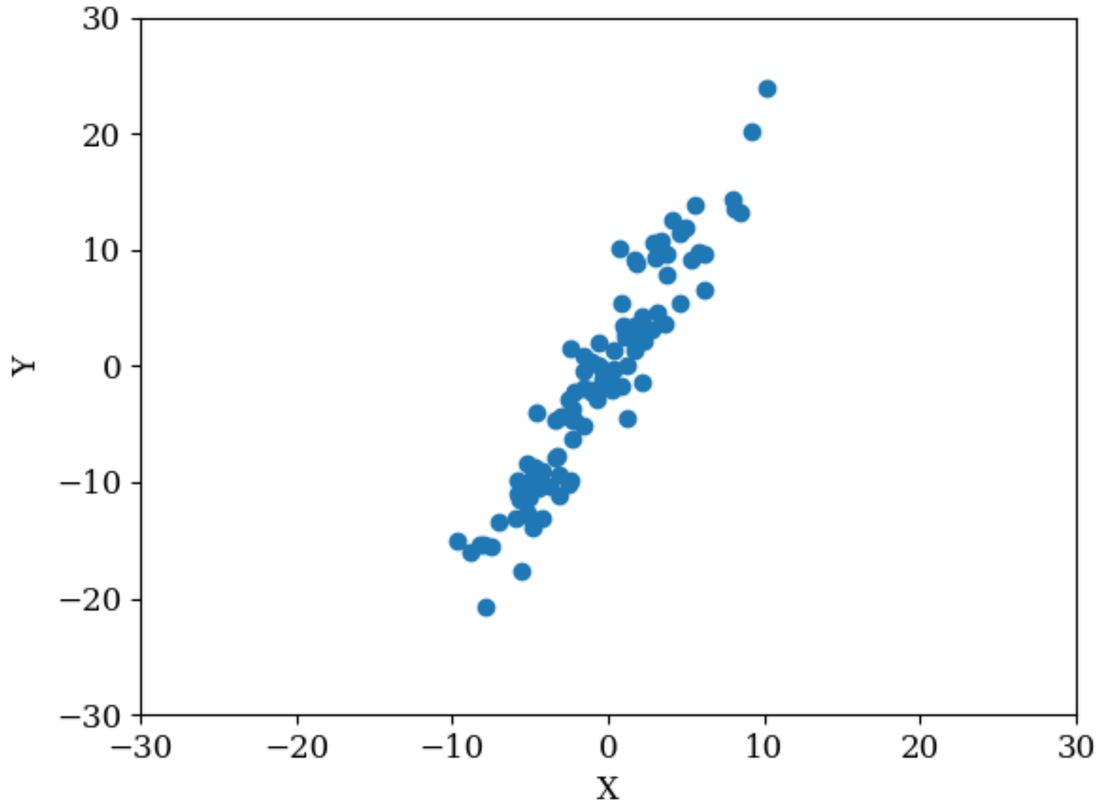
18.55	37.26
37.26	83.77

```
numpy: np.cov(x, y)
```

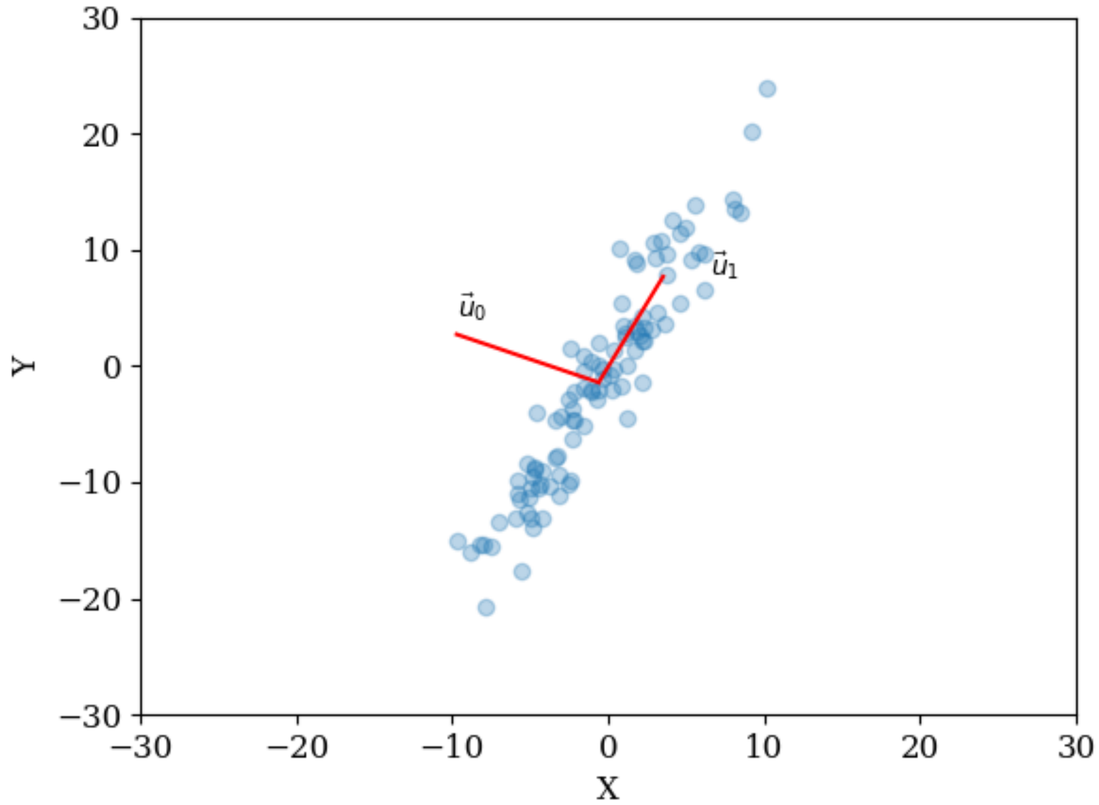
Then: We want to rotate this so that it becomes diagonal.

Eigenvectors & Eigenvalues

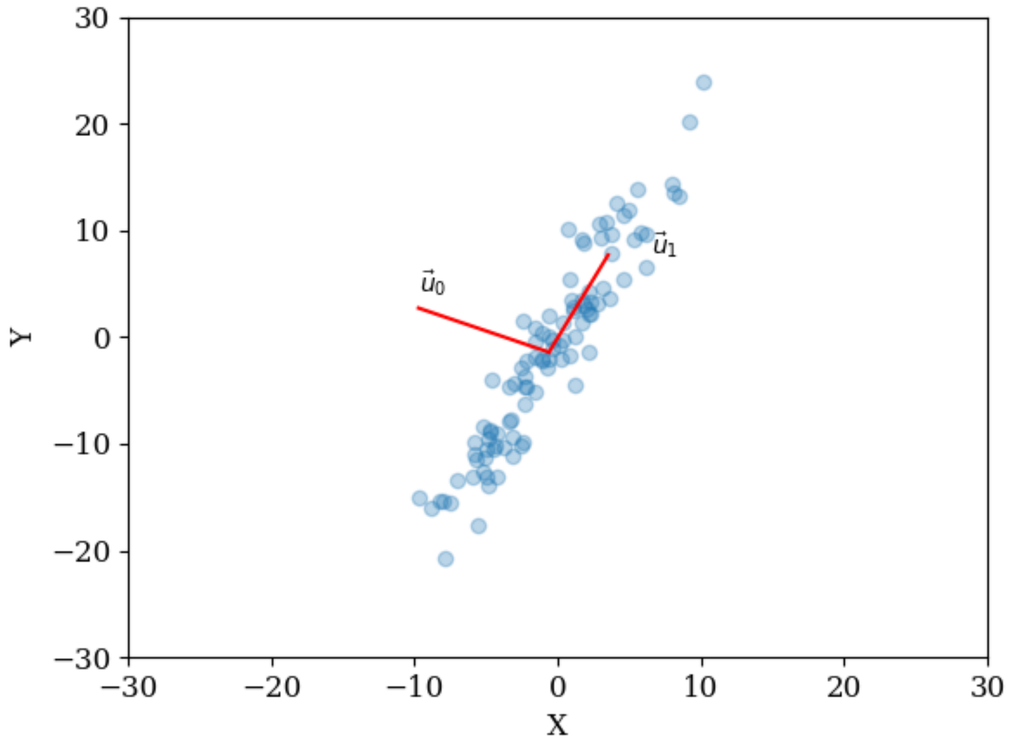
The two eigenvectors



The two eigenvectors



The two eigenvectors



A very standard calculation and all main programming languages support this easily.

Python:

```
import numpy as np
C = np.cov(x, y)
e_vals, e_vecs = np.linalg.eigh(C)
```

Julia:

```
using LinearAlgebra, Statistics
C = cov(hcat(x, y))
e_vals, e_vecs = eigen( C)
```

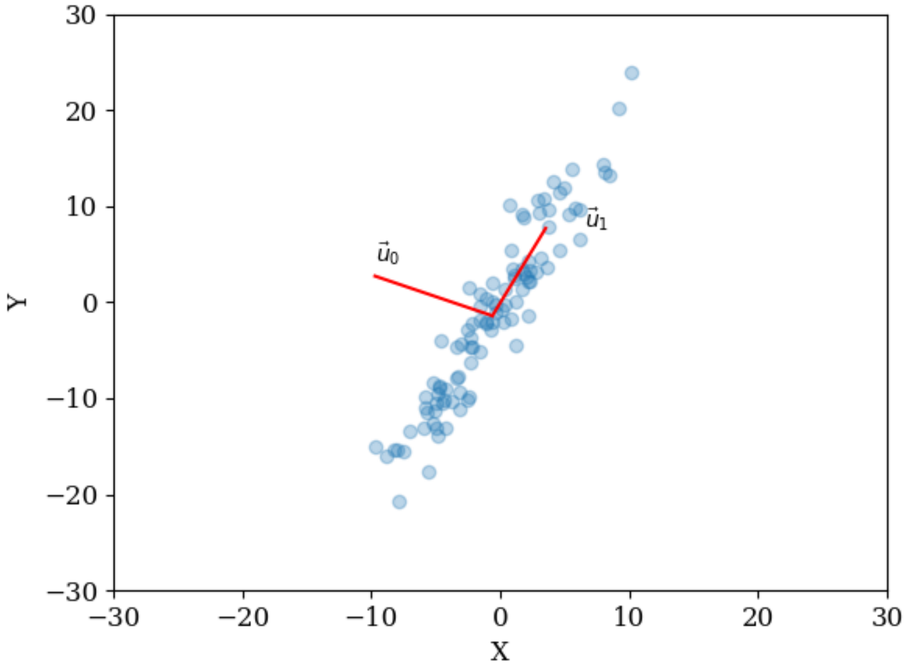
R:

```
C = cov(cbind(x, y))
e = eigen( C)
```

IDL:

```
M = transpose([[x], [y]])
C = correlate(M, /covar)
evals = eigenql(C, eigenvectors=evecs)
```

Interpreting the results



The eigenvector corresponding to the largest eigenvalue defines the direction of most variation in the data ($\mathbf{u}_1$).

So we could say: I do not care about the little scatter around this line, let me reduce the data to just 1D.

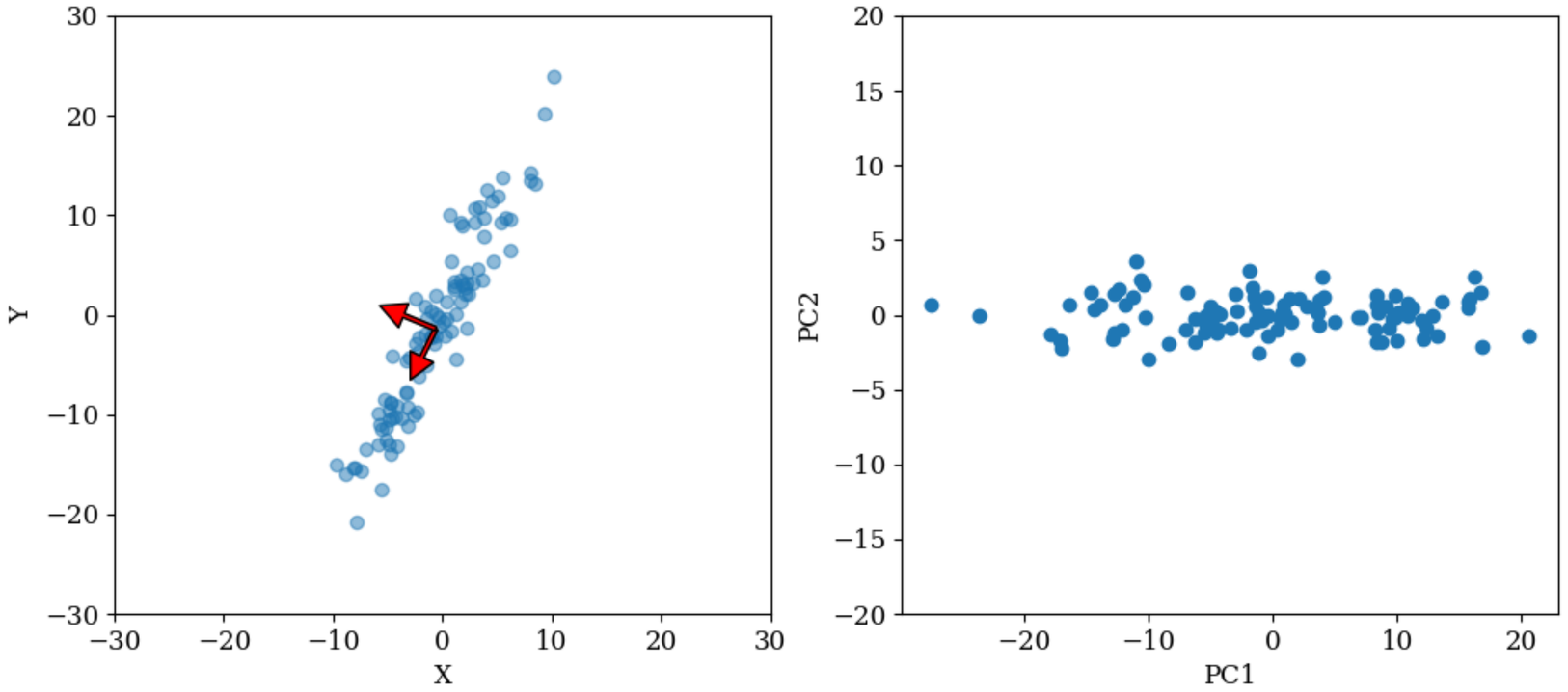
How do you do that?

Projecting the solution

We want to project:

$$(x, y) \rightarrow (u, v)$$ using the eigenvectors.

$$(u, v) = \mathsf{E}(x, y)^T$$ where **E** is the matrix of eigenvectors



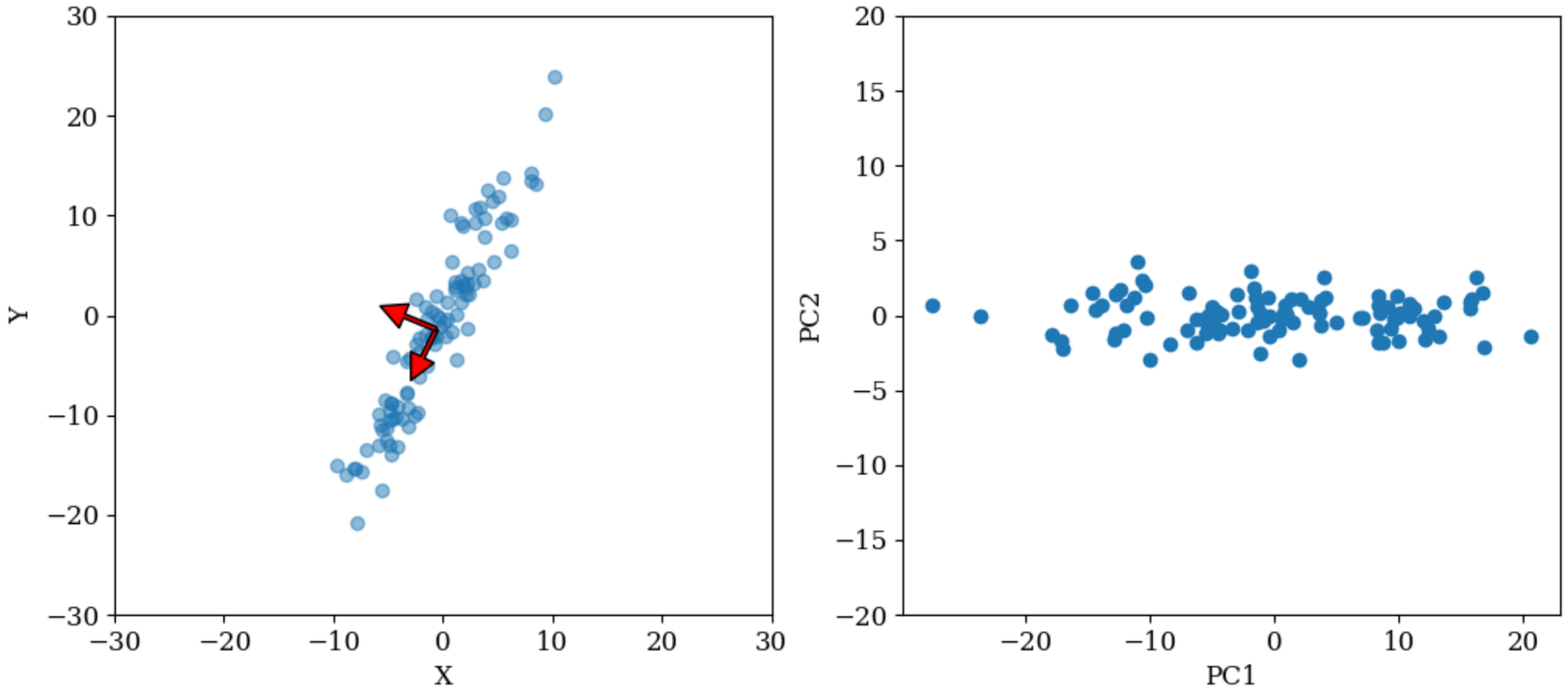
Projecting the solution

Note: The direction of the axes is arbitrary!

We want to project:

$$(x, y) \rightarrow (u, v)$$ using the eigenvectors.

$$(u, v) = \mathsf{E}(x, y)^T$$ where **E** is the matrix of eigenvectors



Dimension reduction

Until now we have considered a situation where we have 2 variables, x & y, and we recover two variables PC1 & PC2. We can also reduce the dimensionality of our problem by reconstructing the original data using only the first eigenvector.

First: Take the new variable u = PC1:

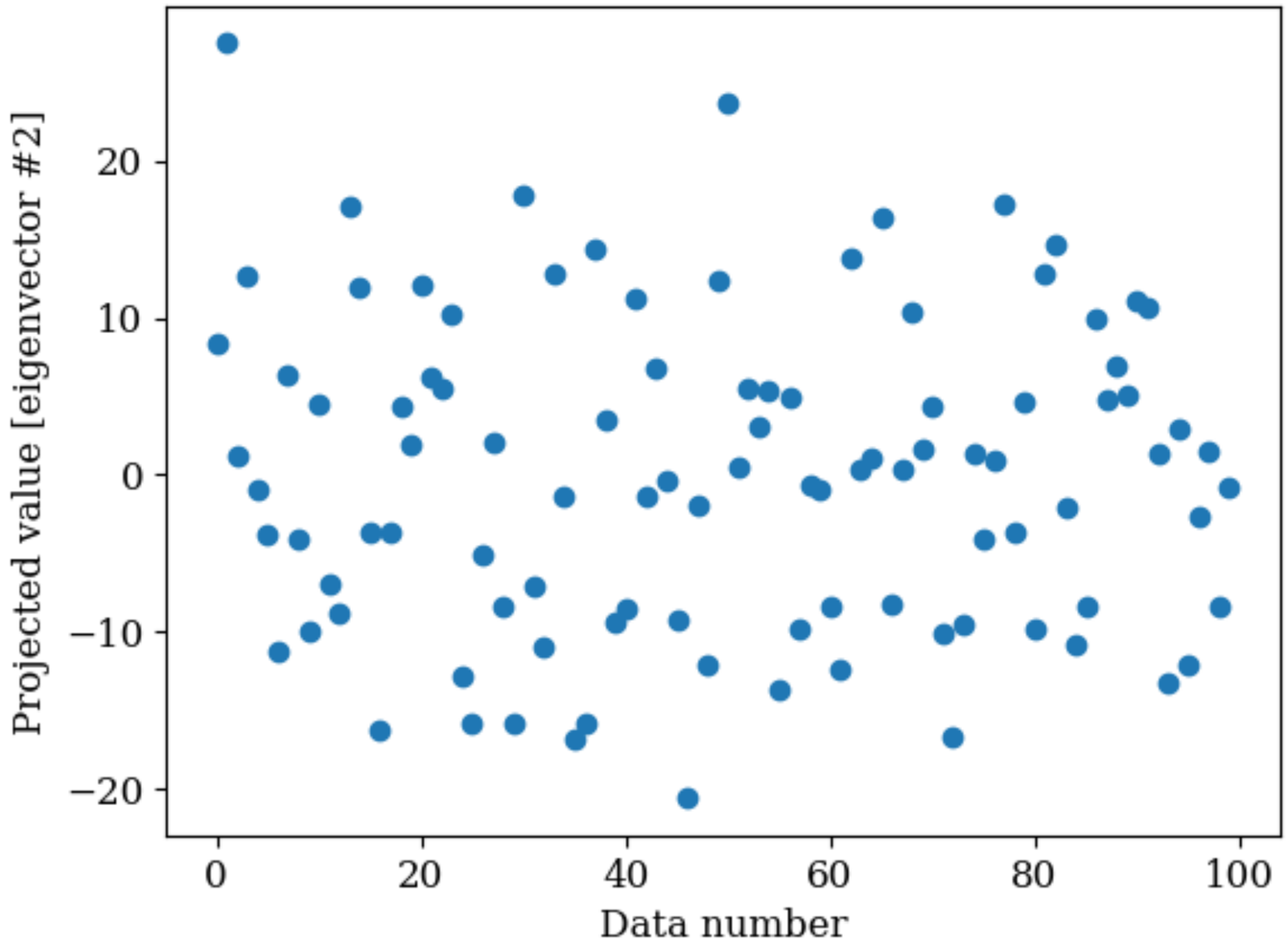
$$u_i = (x_i, y_i)\mathbf{x}\begin{pmatrix} e_{1,x} \\ e_{1,y} \end{pmatrix}$$

Then project back to the original space:

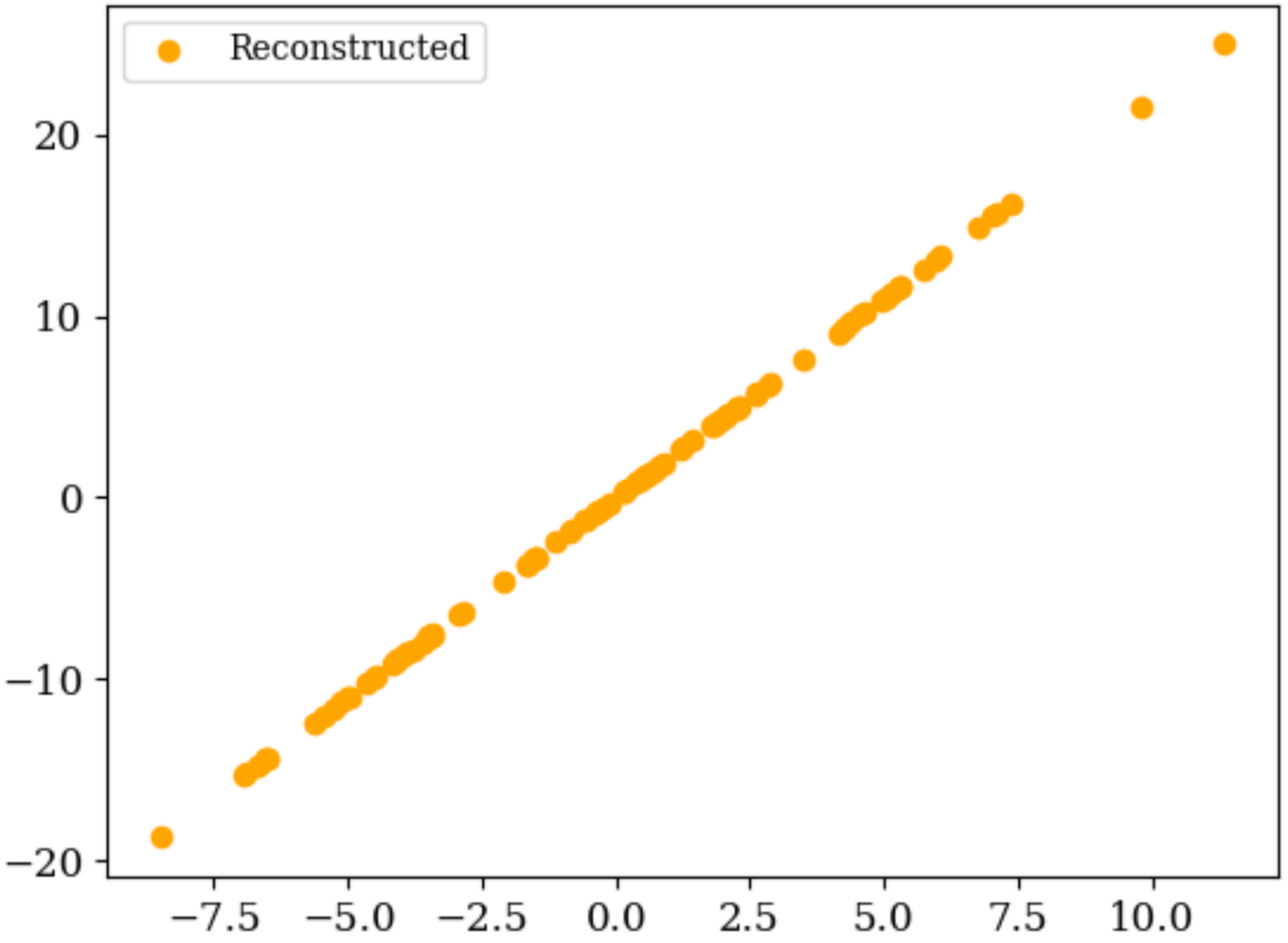
$$(\tilde{x_i}, \tilde{y_i}) = u_i(e_{1,x}, e_{1,y})$$

Dimension reduction

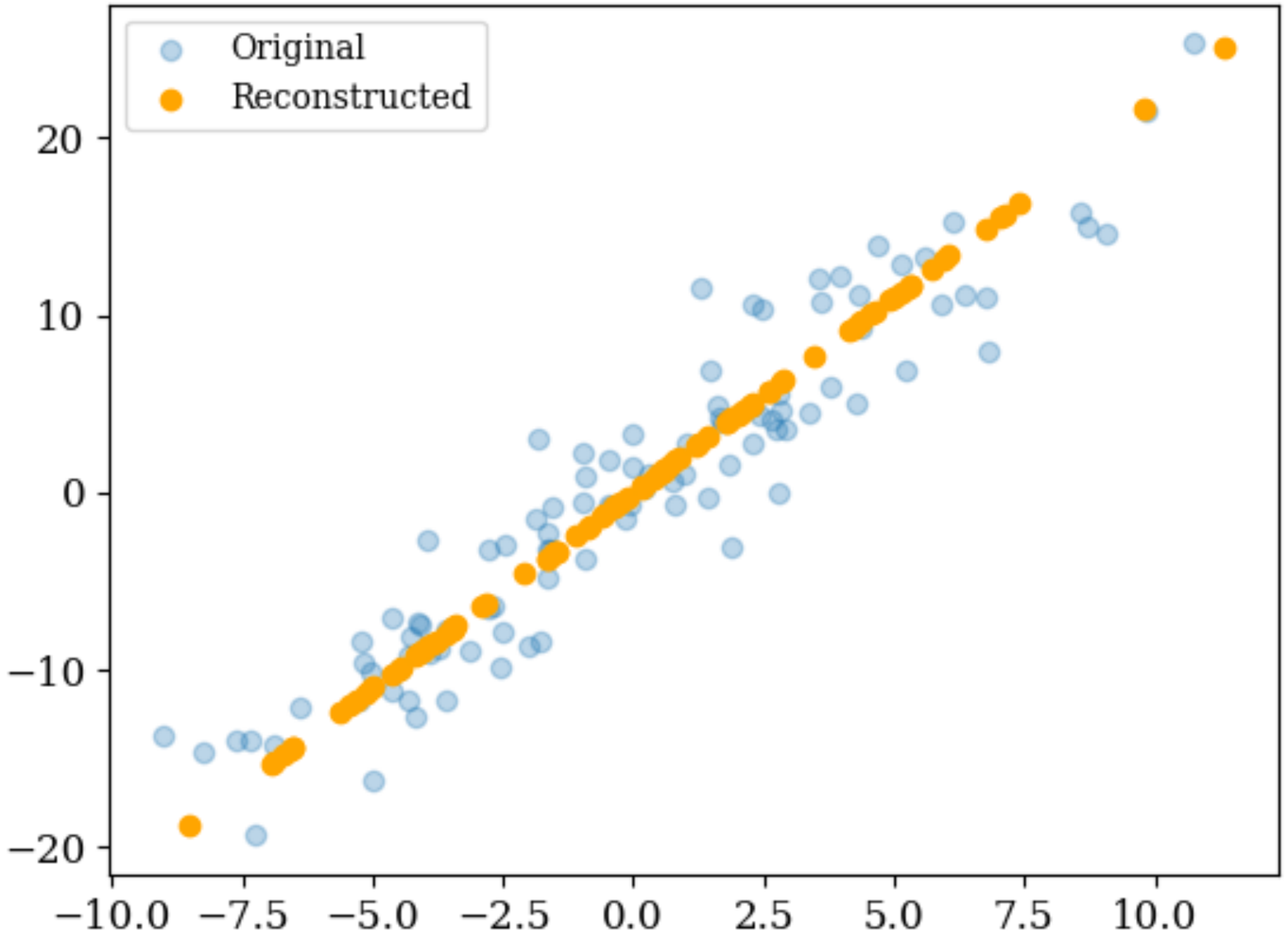
The result of projecting onto the first eigenvector



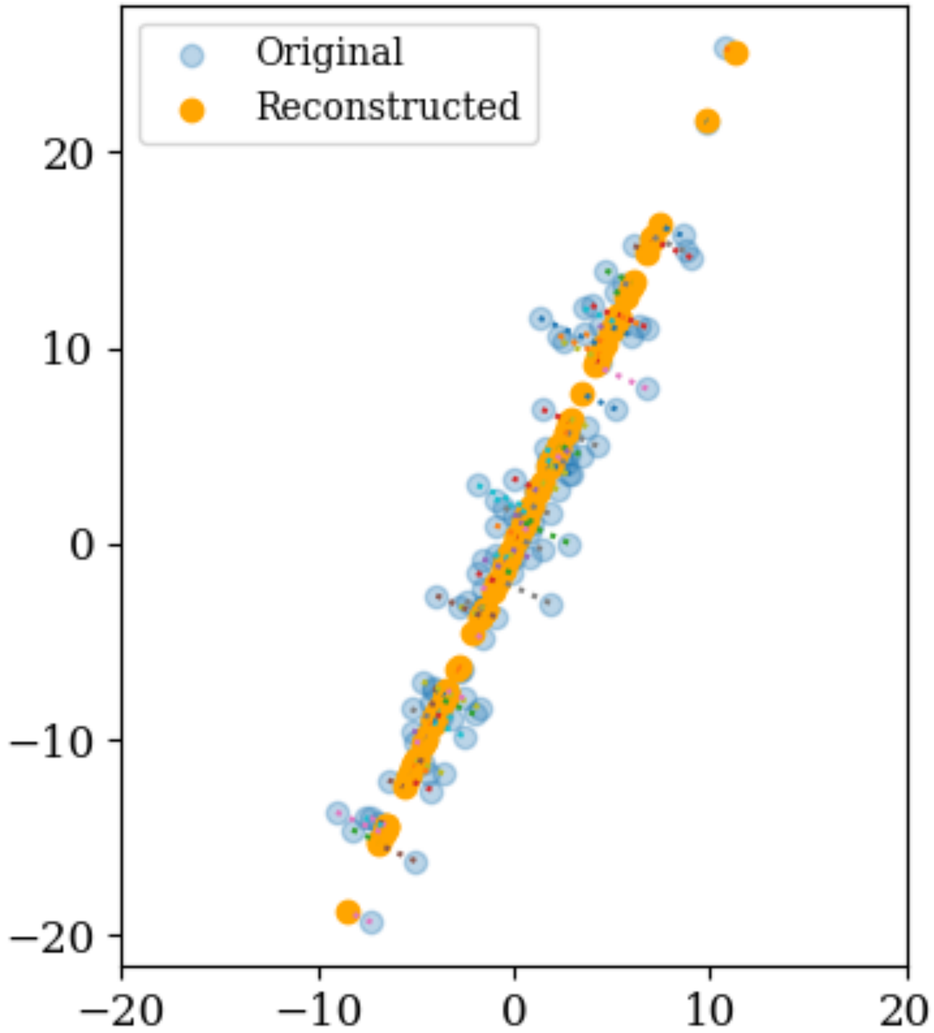
Dimension reduction



Dimension reduction



Dimension reduction - orthogonal



This shows fairly clearly that the projection is orthogonal.

While this is not a regression technique, it finds a best line treating x and y equally.

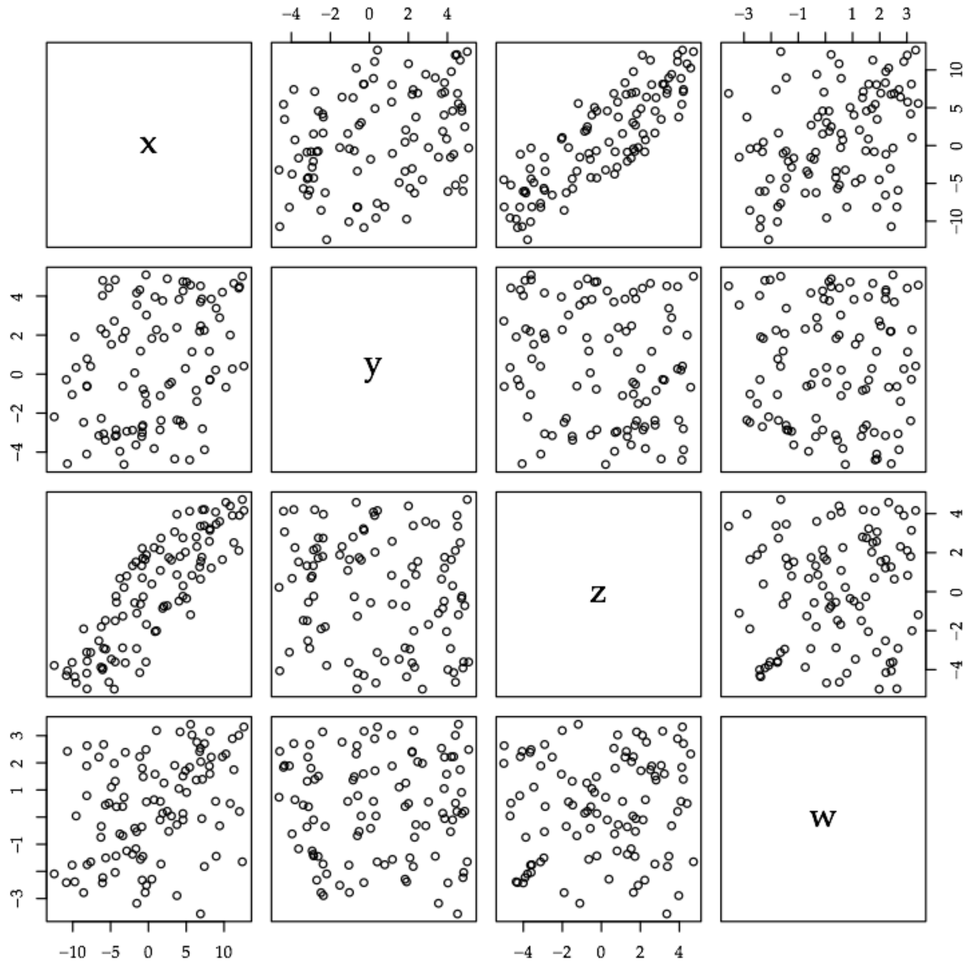
Looking at variances

So PCA gives us a way to find directions where the variance is large. It is therefore possible to ask how much of the variance is **explained** by particular eigenvectors.

The variance is given by the diagonal elements in the diagonalised covariance matrix (the sum of variances is independent of coordinate system).

Looking at variances

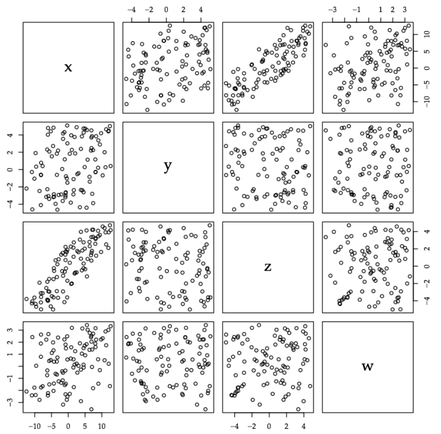
How much of the variance is **explained** by particular eigenvectors.



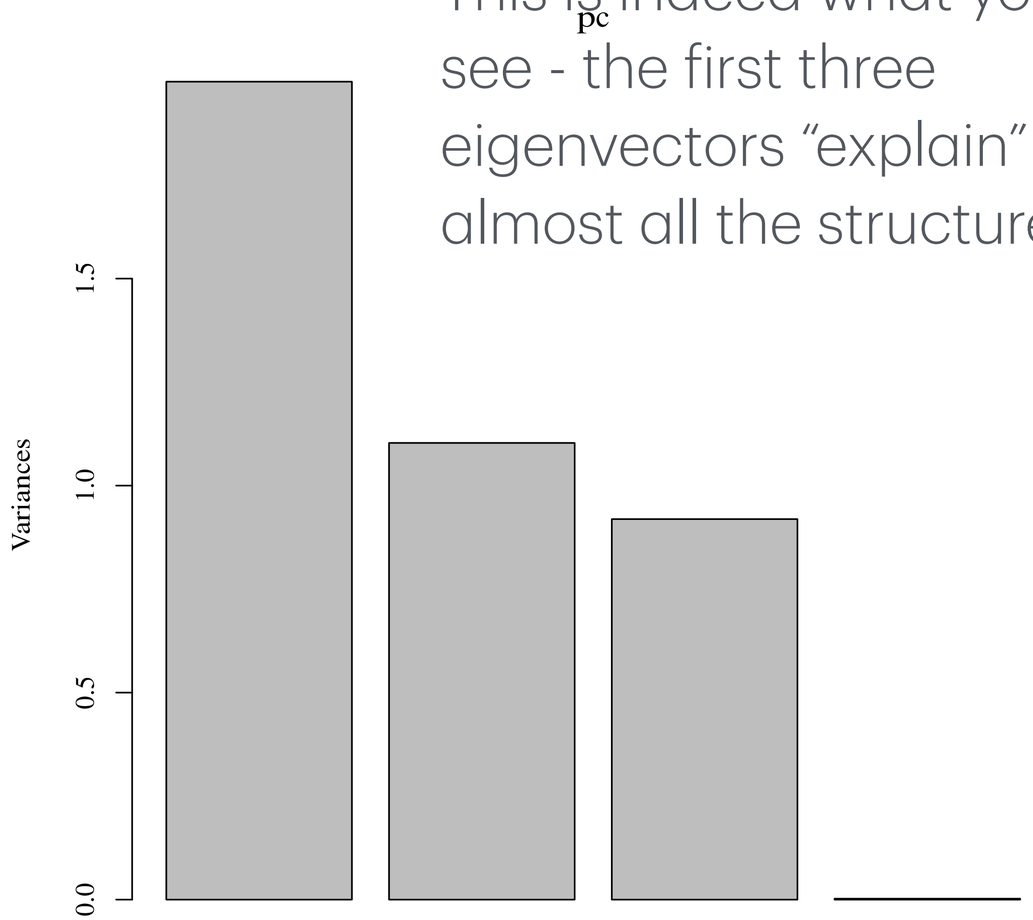
4D dataset but we see clearly a correlation between x & z so they probably add much the same information.

Looking at variances

How much of the variance is **explained** by particular eigenvectors.



4D dataset but we see clearly a correlation between x & z so they probably do not add a lot of information.



This is indeed what you see - the first three eigenvectors “explain” almost all the structure.

More Generally

We have a set of measurements: $\{x_n\}_{i=1}^{N}$

And let us now transform these measurements:

$$\mathbf{y}_n = \mathbf{M}\mathbf{x}_n$$

We would like these **y** to have nice properties. One reasonable requirement would be that they are (linearly) independent. We can formulate this in terms of their covariance matrix:

$$C_y = \frac{1}{N}\sum_i (y_i - \langle y \rangle)(y_i - \langle y \rangle)^T$$

Requiring the **y** values to be independent is equivalent to ensuring the covariance matrix being diagonal.

What does that mean?

Diagonalizing the covariance matrix

$$\mathrm{C}_{\mathbf{y}} = \frac{1}{N}\sum_{i}(\mathbf{y_i} - \langle \mathbf{y} \rangle)(\mathbf{y_i} - \langle \mathbf{y} \rangle)^T$$

Diagonalizing the covariance matrix

$$\mathrm{C}_{\mathbf{y}} = \frac{1}{N}\sum_{i}(\mathbf{y_i} - \langle \mathbf{y} \rangle)(\mathbf{y_i} - \langle \mathbf{y} \rangle)^T$$

$$\mathrm{C}_{\mathbf{y}} = \frac{1}{N}\sum_{i} M(\mathbf{x_i} - \langle \mathbf{x} \rangle)(M(\mathbf{x_i} - \langle \mathbf{x} \rangle))^T$$

Diagonalizing the covariance matrix

$$\mathrm{C}_{\mathbf{y}} = \frac{1}{N}\sum_i (\mathbf{y_i} - \langle \mathbf{y} \rangle)(\mathbf{y_i} - \langle \mathbf{y} \rangle)^T$$

$$\mathrm{C}_{\mathbf{y}} = \frac{1}{N}\sum_i M(\mathbf{x_i} - \langle \mathbf{x} \rangle)(M(\mathbf{x_i} - \langle \mathbf{x} \rangle))^T$$

$$\mathrm{C}_{\mathbf{y}} = \frac{1}{N}\sum_i M(\mathbf{x_i} - \langle \mathbf{x} \rangle)(\mathbf{x_i} - \langle \mathbf{x} \rangle)^T M^T$$

Diagonalizing the covariance matrix

$$\mathrm{C}_{\mathbf{y}} = \frac{1}{N}\sum_i (\mathbf{y_i} - \langle \mathbf{y} \rangle)(\mathbf{y_i} - \langle \mathbf{y} \rangle)^T$$

$$\mathrm{C}_{\mathbf{y}} = \frac{1}{N}\sum_i M(\mathbf{x_i} - \langle \mathbf{x} \rangle)(M(\mathbf{x_i} - \langle \mathbf{x} \rangle))^T$$

$$\mathrm{C}_{\mathbf{y}} = \frac{1}{N}\sum_i M(\mathbf{x_i} - \langle \mathbf{x} \rangle)(\mathbf{x_i} - \langle \mathbf{x} \rangle)^T M^T$$

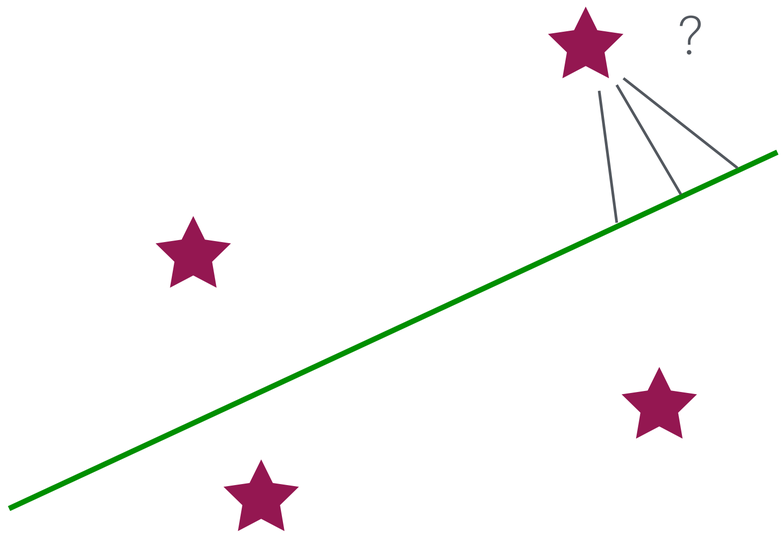
$$C_{\mathbf{y}} = M C_{\mathbf{x}} M^T$$

So by finding the matrix that diagonalizes the covariance matrix of the data, we find a new set of uncorrelated variables.

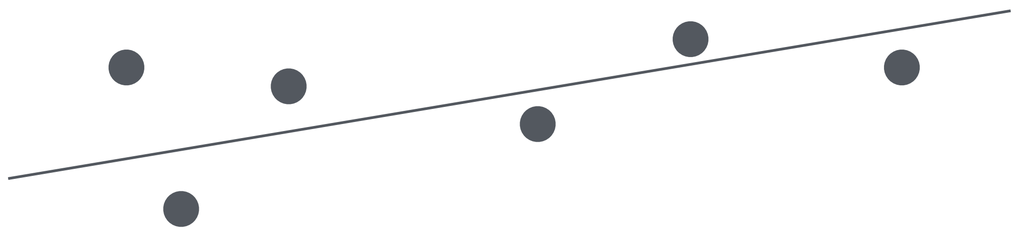
A different view

The data, $\mathbf{x}_n$, is D-dimensional, where D can be a large number (several thousand easily for images).

Let us now ask - what is the “best” way to project it down to a lower-dimensional space with M dimensions?



What is the “best” way to project D-dimensional data to a lower-dimensional space with M dimensions?

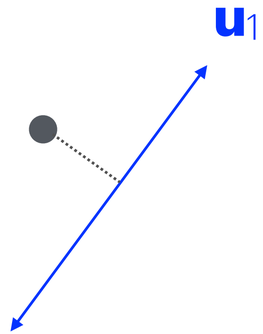


Going from 2D to 1D it would be along this line because it “explains” the data in the simplest way. Reformulating it we say that we want the direction where the data has maximal **variance**.

So from D dimensions to 1 dimension:

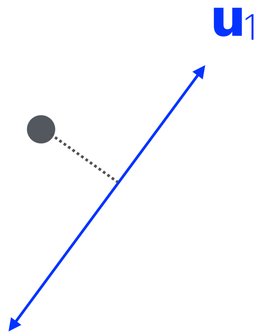
We project $\mathbf{x}_n$ onto a vector $\mathbf{u}_1$:

$$\mathbf{y}_n = \mathbf{u}_1^T \mathbf{x}_n$$
$$\mathbf{u}_1^T \mathbf{u}_1 = 1$$



So from D dimensions to 1 dimension:

We project $\mathbf{x}_n$ onto a vector $\mathbf{u}_1$:



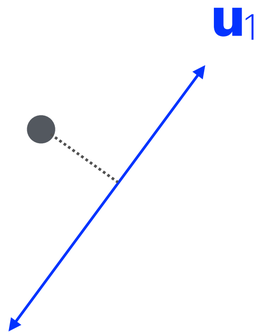
$$\mathbf{y}_n = \mathbf{u}_1^T \mathbf{x}_n$$
$$\mathbf{u}_1^T \mathbf{u}_1 = 1$$

So we want to maximise the variance of $\mathbf{y}$:

$$\mathrm{Var}[\mathbf{y}] = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{y}_n - \langle \mathbf{y} \rangle)^2$$

So from D dimensions to 1 dimension:

We project $\mathbf{x}_n$ onto a vector $\mathbf{u}_1$:



$$\mathbf{y}_n = \mathbf{u}_1^T \mathbf{x}_n$$

$$\mathbf{u}_1^T \mathbf{u}_1 = 1$$

So we want to maximise the variance of $\mathbf{y}$:

$$\mathrm{Var}[\mathbf{y}] = \frac{1}{N} \sum_{n=1}^{N} \left(\mathbf{y}_n - \langle \mathbf{y} \rangle\right)^2$$

Which can be rewritten:

$$\begin{aligned} (\mathsf{A} - \mathsf{B})^2 &= (\mathsf{A} - \mathsf{B})(\mathsf{A} - \mathsf{B})^T \\ (\mathsf{A}\mathbf{x})^T &= \mathbf{x}^T \mathsf{A}^T \end{aligned}$$

$$\mathrm{Var}[\mathbf{y}] = \mathbf{u}_1^T \mathsf{S} \mathbf{u}_1 = \mathbf{u}_1^T \mathsf{C}_{\mathbf{x}} \mathbf{u}_1$$

With

$$\mathsf{S} = \mathsf{C}_{\mathbf{x}} = \left(\mathbf{x} - \langle \mathbf{x} \rangle\right) \left(\mathbf{x} - \langle \mathbf{x} \rangle\right)^T$$

$$\mathrm{Var}[\mathbf{y}] = \mathbf{u}_1^T \mathsf{S} \mathbf{u}_1 = \mathbf{u}_1^T \mathsf{C}_{\mathbf{x}} \mathbf{u}_1$$

$$\mathrm{Var}[\mathbf{y}] = \mathbf{u}_1^T \mathsf{S} \mathbf{u}_1 = \mathbf{u}_1^T \mathsf{C}_{\mathbf{x}} \mathbf{u}_1$$

So to maximise this with the constraint $\mathbf{u}_1^T \mathbf{u}_1 = 1$:

$$V = \mathbf{u}_1^T \mathsf{C}_{\mathbf{x}} \mathbf{u}_1 + \lambda_1 \left(1 - \mathbf{u}_1^T \mathbf{u}_1\right)$$

$$\mathrm{Var}[\mathbf{y}] = \mathbf{u}_1^T \mathsf{S} \mathbf{u}_1 = \mathbf{u}_1^T \mathsf{C}_\mathbf{x} \mathbf{u}_1$$

So to maximise this with the constraint $\mathbf{u}_1^T \mathbf{u}_1 = 1$:

$$V = \mathbf{u}_1^T \mathsf{C}_\mathbf{x} \mathbf{u}_1 + \lambda_1 \left(1 - \mathbf{u}_1^T \mathbf{u}_1\right)$$

Set $\partial V / \partial \mathbf{u}_1 = 0$

$$\mathsf{C}_\mathbf{x} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1 \Rightarrow \mathbf{u}_1^T \mathsf{C}_\mathbf{x} \mathbf{u}_1 = \lambda_1$$

$$\mathrm{Var}[\mathbf{y}] = \mathbf{u}_1^T \mathsf{S} \mathbf{u}_1 = \mathbf{u}_1^T \mathsf{C}_\mathbf{x} \mathbf{u}_1$$

So to maximise this with the constraint $\mathbf{u}_1^T \mathbf{u}_1 = 1$:

$$V = \mathbf{u}_1^T \mathsf{C}_\mathbf{x} \mathbf{u}_1 + \lambda_1 \left(1 - \mathbf{u}_1^T \mathbf{u}_1\right)$$

Set $\partial V / \partial \mathbf{u}_1 = 0$

$$\mathsf{C}_\mathbf{x} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1 \Rightarrow \mathbf{u}_1^T \mathsf{C}_\mathbf{x} \mathbf{u}_1 = \lambda_1$$

Hence the variance is maximised in 1D if $\mathbf{u}_1$ is set to the largest eigenvector of the covariance matrix.

Projecting

So, now we have all the eigenvectors & eigenvalues, how do we calculate the new coordinates?

a) Create a “feature vector”/transform matrix by placing the eigenvectors in a matrix where each row is an eigenvector. $\mathbf{M}$

b) Transform the data to the new coordinates using this matrix

$$\mathbf{u} = \mathbf{M}\mathbf{x}$$

c) Transform back using the transpose of **M**:

$$\mathbf{r} = \mathbf{u}\mathbf{M}^T$$ (+ offsets & scalings applied to **r**)

Step-by-step

Python: sklearn.decomposition.PCA

R: prcomp

Julia: MultivariateStats.PCA

IDL: pcomp

Simple example: Lectures/Lecture 4/Notebooks/Simple PCA example.ipynb
More advanced: Lectures/Lecture 4/Notebooks/PCA of Pickles.ipynb

Some general issues

- The sign (direction) of an eigenvector is arbitrary.
- A covariance matrix depends on the scaling of the variables. This can be really important!
- New variables should be defined using normalised eigenvectors (not always done by computer packages, e.g. IDL)
- PCA is not a magic bullet. You must think carefully when using it!

SCALING

If we look at one element in the covariance matrix:

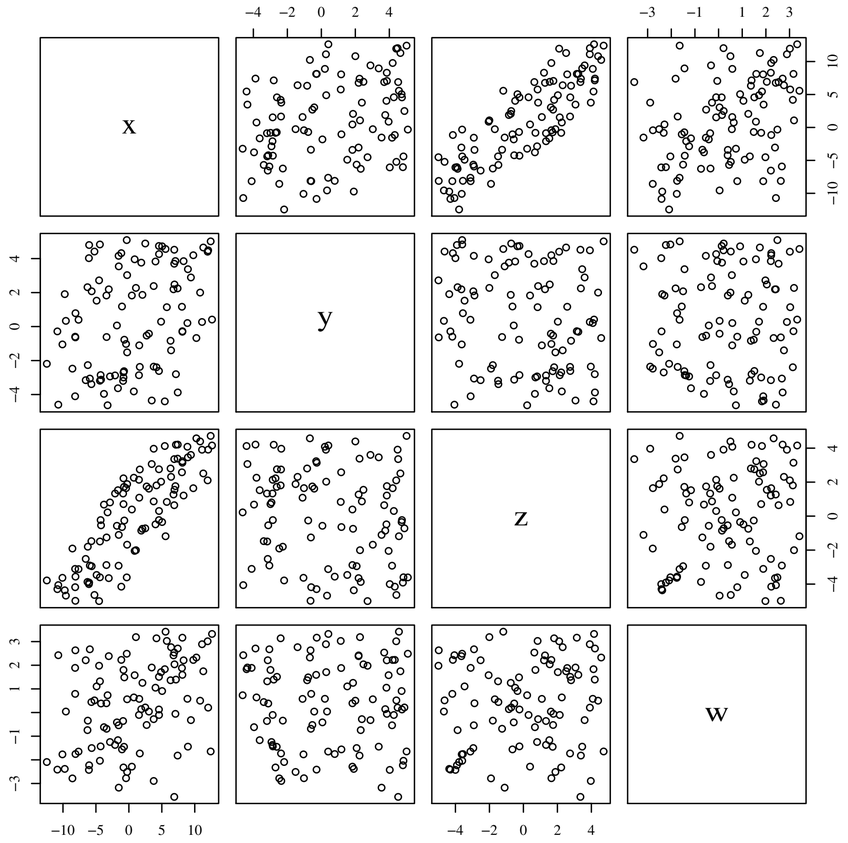
Clearly if we scale all y by the same factor we can make Cov(i, j) arbitrarily large or small. This means that the *eigenvectors* also can be arbitrarily changed!

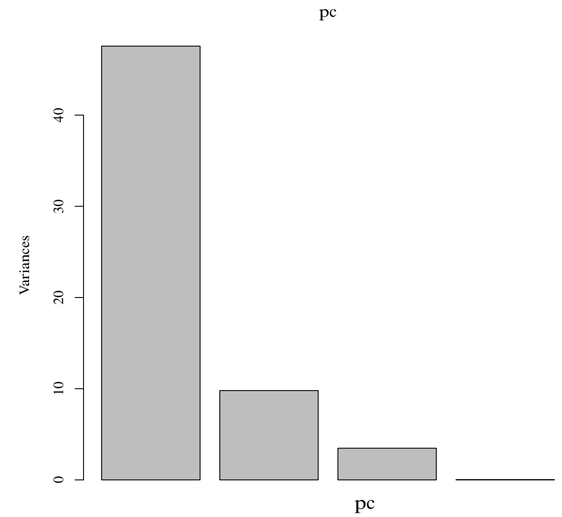
$$\mathrm{Cov}(i,j) = \frac{1}{N}\sum_{n=1}^{N}\left(v_{i,n} - \langle v_i \rangle\right)\left(v_{j,n} - \langle v_j \rangle\right)$$

Note also that the mean is subtracted off - it is usually a good idea to do that ourselves. Sometimes people recommend also scaling by their empirical variance (this can be be bad if you are careless). This is standardizing we saw earlier. For PCA this might be a bad idea but it depends on the question.

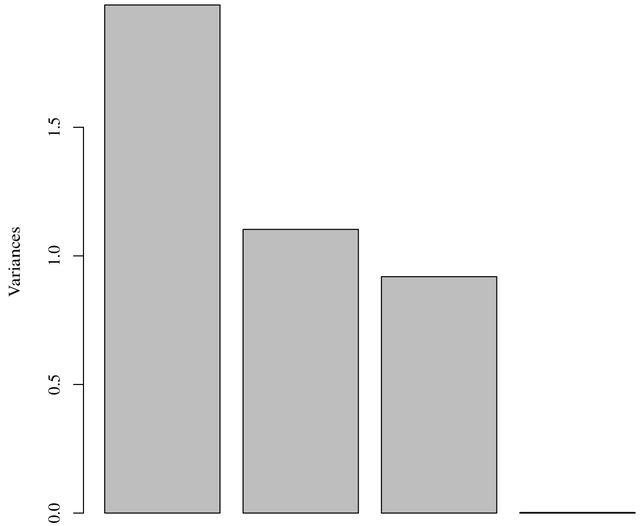
SCALING

But sometimes scaling is crucial to take out trivial features in the data!





No scaling



With scaling

SCALING

The physical meaning of this:

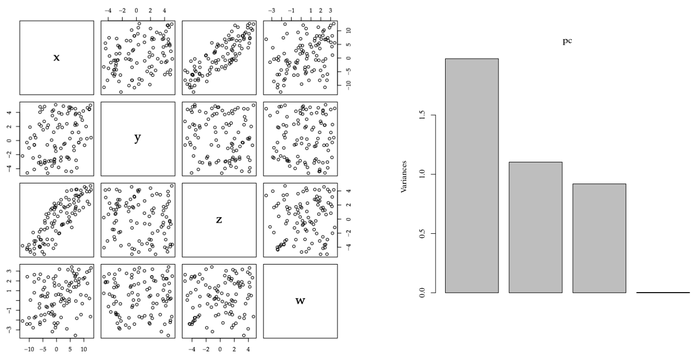
We might have a system where we know that x & y are correlated, say y increases a bit when x increases. This produces a non-zero covariance between the two variables. However we might know that this is an unimportant physical correlation (y is mostly governed by something else).

What would normalising by the spread in x & y mean?

Often a good alternative: Work with relative quantities/intrinsic quantities rather than absolute/extrinsic quantities.

Uses of PCA

I. Find the effective dimensionality of your data:



2. Identify useful features of your data (images, spectra, etc.) - both for feature extraction and science.

3. Compress your high-dimensional dataset to fewer variables to make fits to models easier. [pre-processing]

4. Find linear relations in your data while treating all variables on the same footing (no need to assume an independent variable)

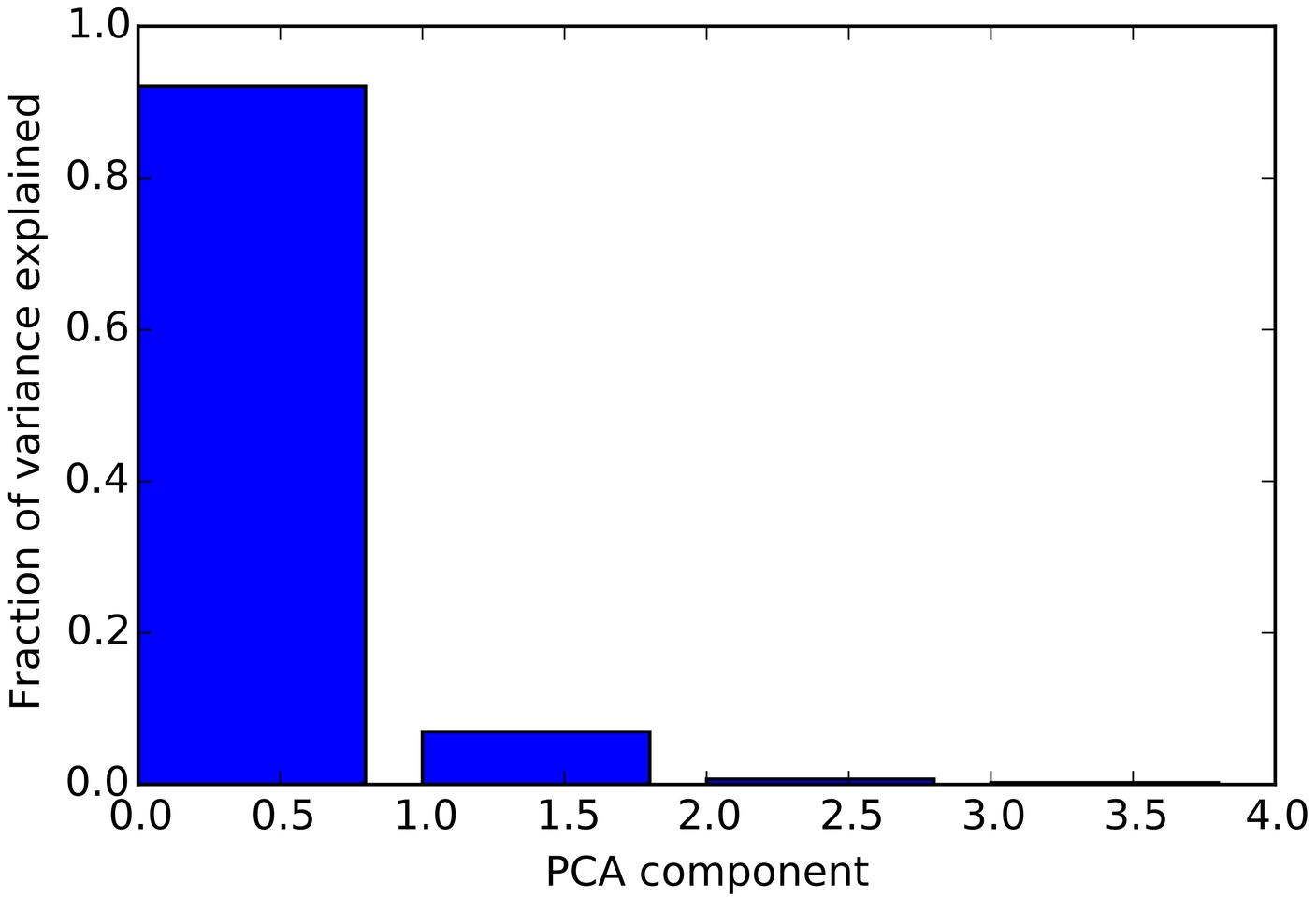
Trying it out in Python

```
from sklearn.decomposition import PCA
M, T = pickle_from_file('T-vs-colour-regression.pkl')
pca = PCA(whiten=True, n_components=4)
pca.fit(M)

pca.components_

plt.bar(np.arange(len(pca.explained_variance_)),
pca.explained_variance_ratio_)
```

From the PCA components I get that

1.84 (u-g) + 1.1 (g-r) + 0.39 (r-i) + 0.28 (i-z)

explains most of the variance.

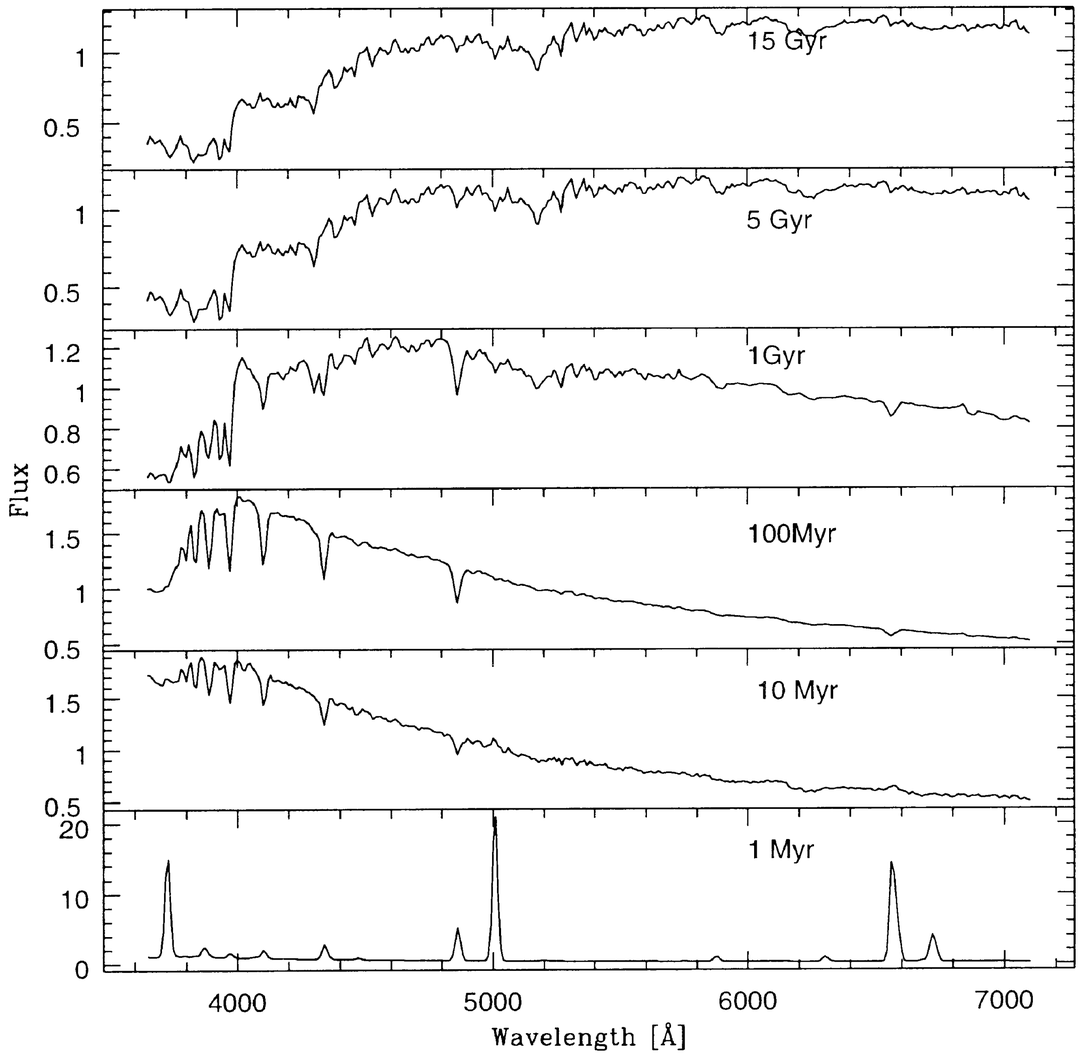



```
plt.bar(i_comp, pca.explained_variance_ratio_)
```

Some further applications of PCA

Classifying galaxy spectra using PCA

Ronen et al (1999)



Observed galaxy spectra show a wide range of structure.

They are composed of stars of different ages etc - can we disentangle them using PCA?

Classifying spectra using PCA

A spectrum is a function of wavelengths: $f(\lambda_i)$

But you could just as well think of this as a collection of numbers:

$$\vec{x} \quad \text{where} \quad x_i = f(\lambda_i)$$

Then your spectrum with N wavelength points become a vector in N dimensional space and you can use PCA if you have a set of spectra!

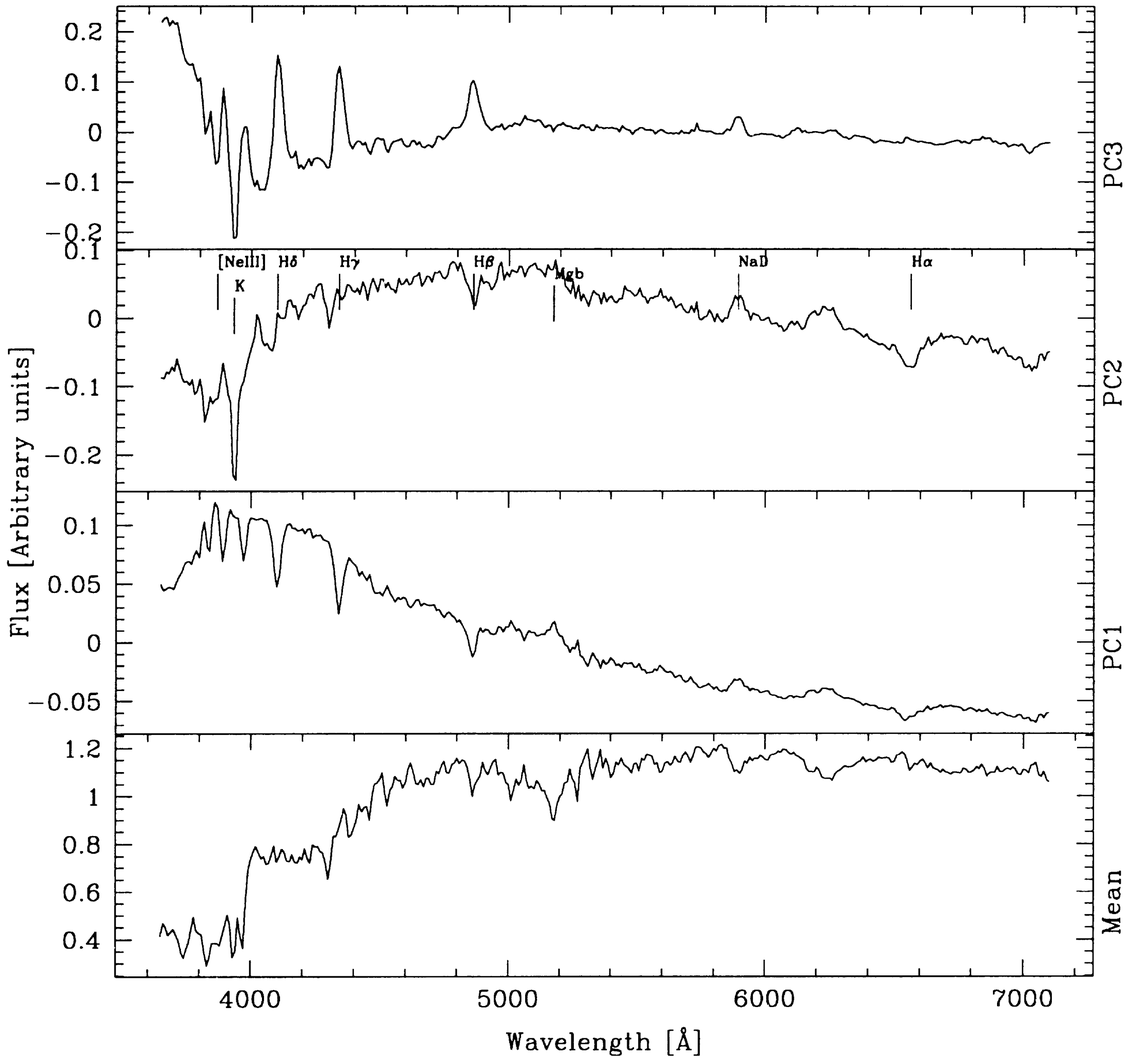
1. Calculate the covariance matrix.

2. Calculate eigenvalues & eigenvectors.

3. Determine the important eigenvectors.

Classifying galaxy spectra using PCA

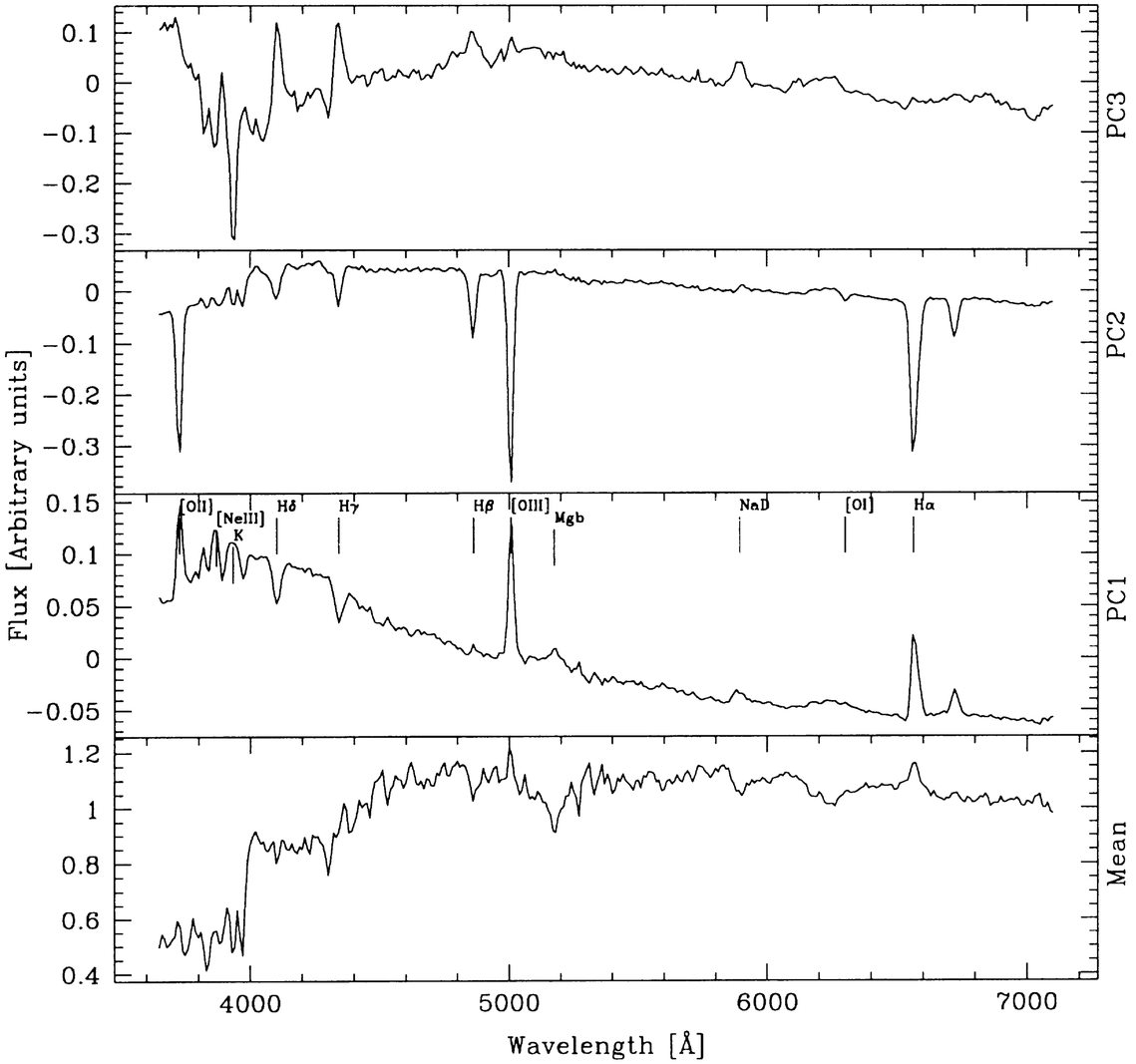
Ronen et al (1999)



By decomposing into PCAs you get a set of eigenspectra. These contain useful information but disentangling it can be hard.

Classifying galaxy spectra using PCA

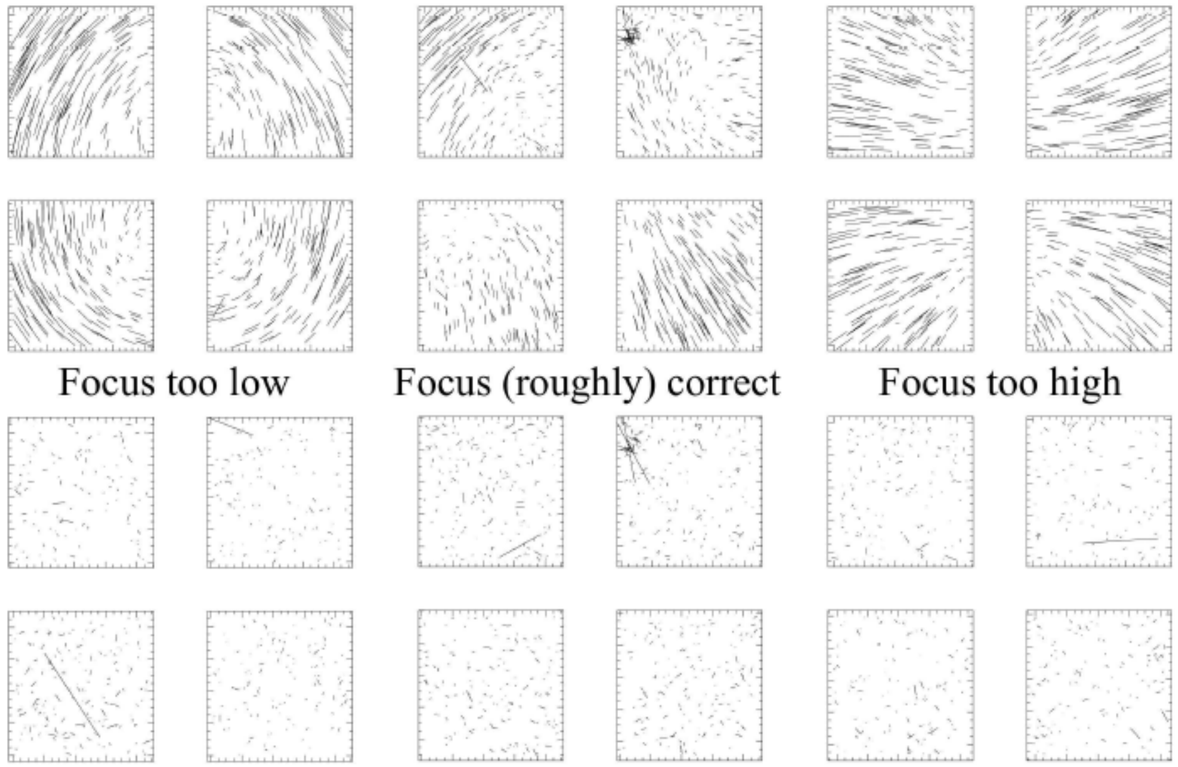
Ronen et al (1999)



A very young system

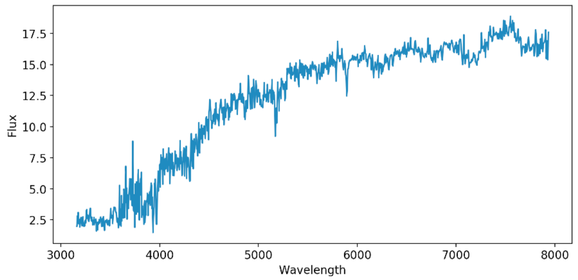
PCA for correcting point-spread functions

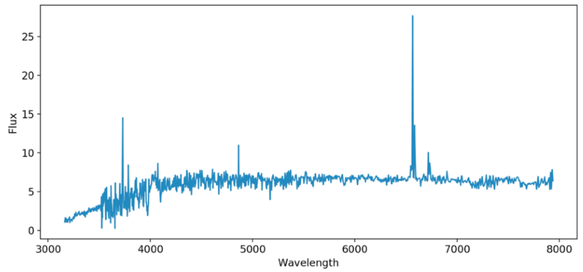
Weak lensing techniques for probing the structure of the Universe are very sensitive to the PSF. The true variation of the PSF can be very complex - PCA decomposition allows you to focus on the important features



Jarvis & Jain (2004), see also Schrabback et al (2010)

Handling of high-D data





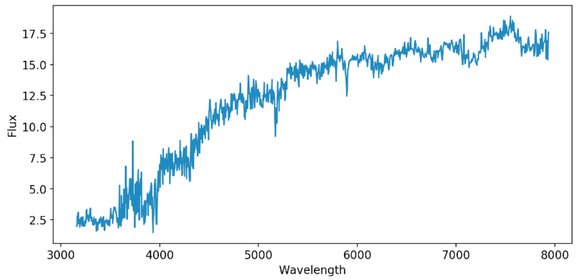
Say you want to create a grid of these spectra for fitting - how would you do that?

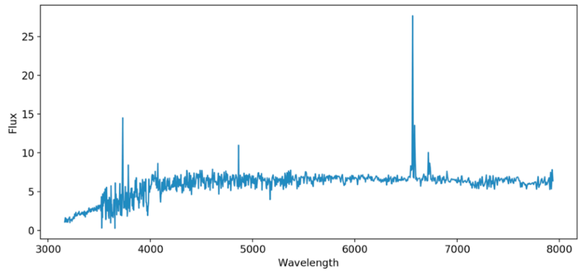
One possibility: PCA

$$f(\lambda_i) = \sum_i \omega_i(\theta)\Phi_i(\lambda)$$

One can then truncate this expansion and have a few (3-10) eigen components which can be put on a grid.

Handling of high-D data





Say you want to create a grid of these spectra for fitting - how would you do that?

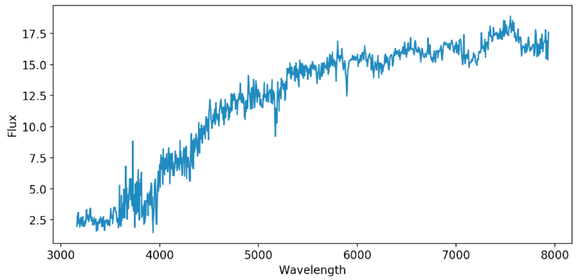
One possibility: PCA

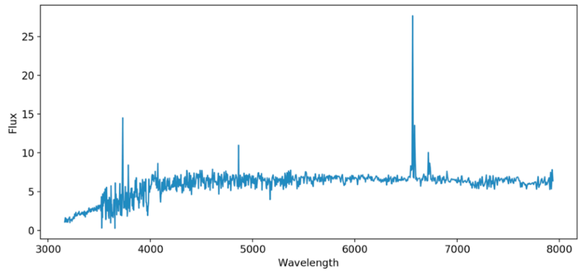
$$f(\lambda_i) = \sum_i \omega_i(\theta)\Phi_i(\lambda)$$

PCA weights

One can then truncate this expansion and have a few (3-10) eigen components which can be put on a grid.

Handling of high-D data





Say you want to create a grid of these spectra for fitting - how would you do that?

One possibility: PCA

Eigenvectors

$$f(\lambda_i) = \sum_i \omega_i(\theta)\Phi_i(\lambda)$$

PCA weights

One can then truncate this expansion and have a few (3-10) eigen components which can be put on a grid.

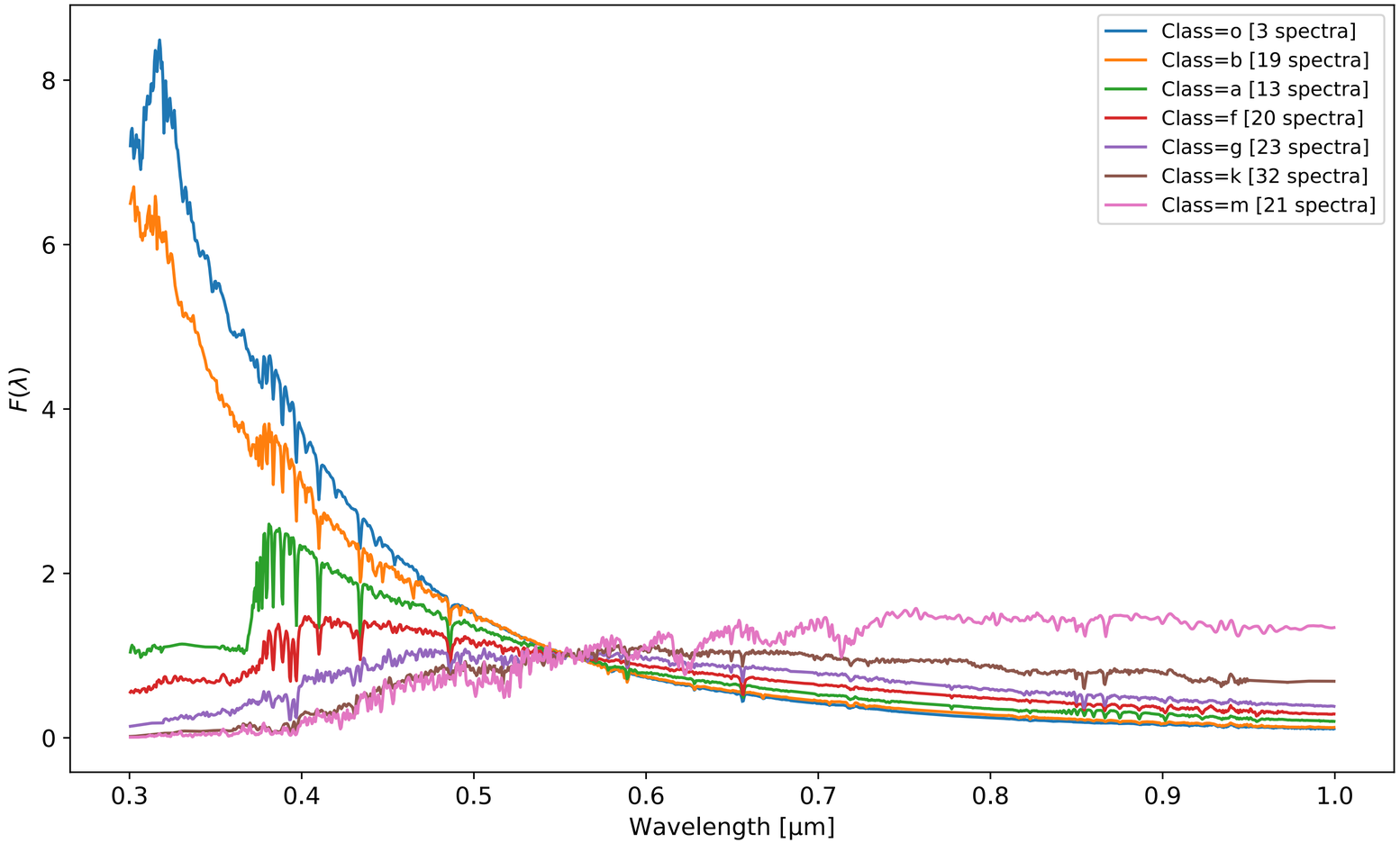
Let’s switch to Colab!

Notebook: MLD2025-09-PCA of Pickles

Direct link

PCA of stellar spectra

Starting point: Pickle's library of stars @ 5Å resolution



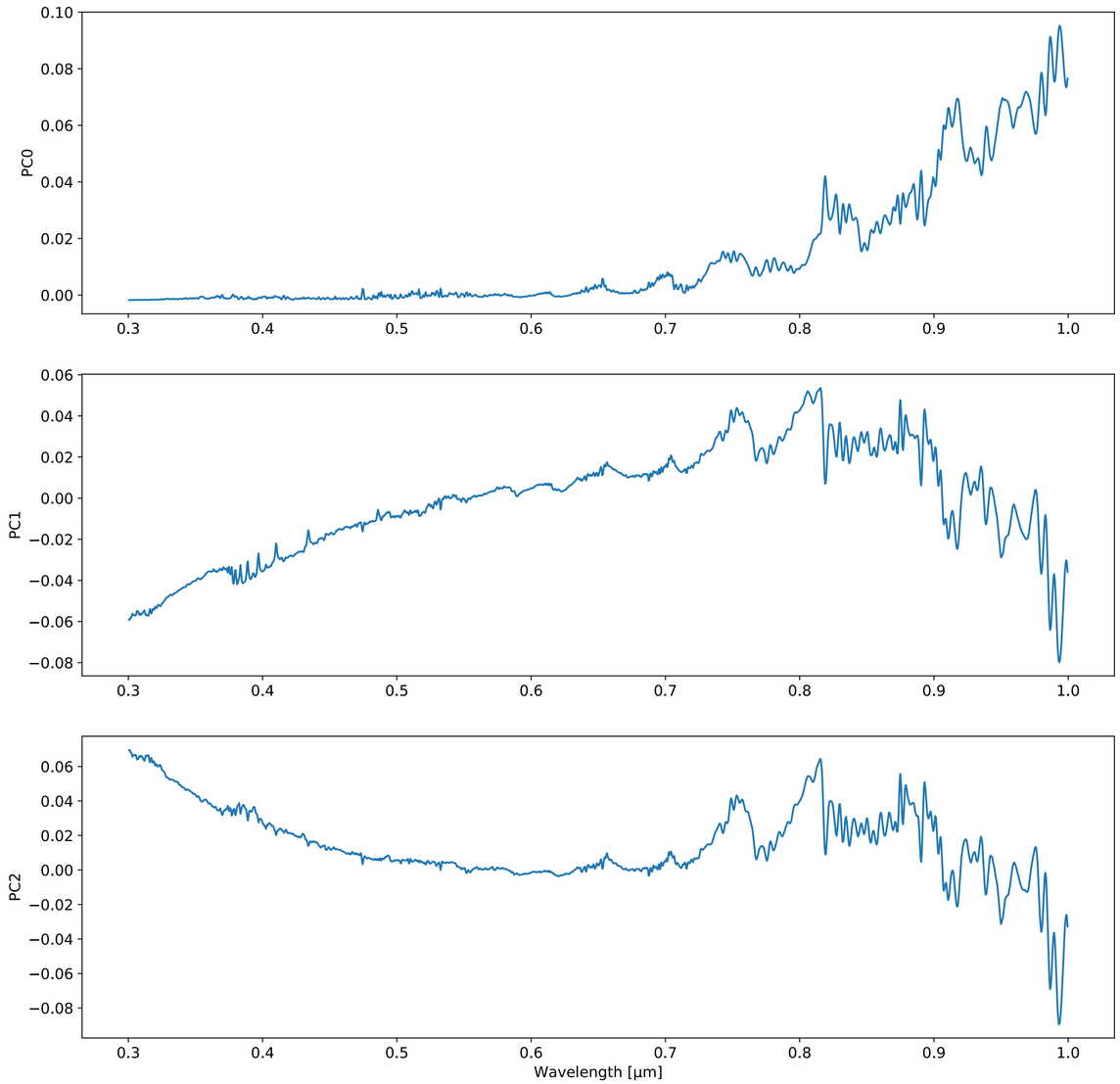
PCA of stellar spectra

X = fluxes of all stars stacked (131x4771)

So this has 131 examples in 4771-dimensional space.

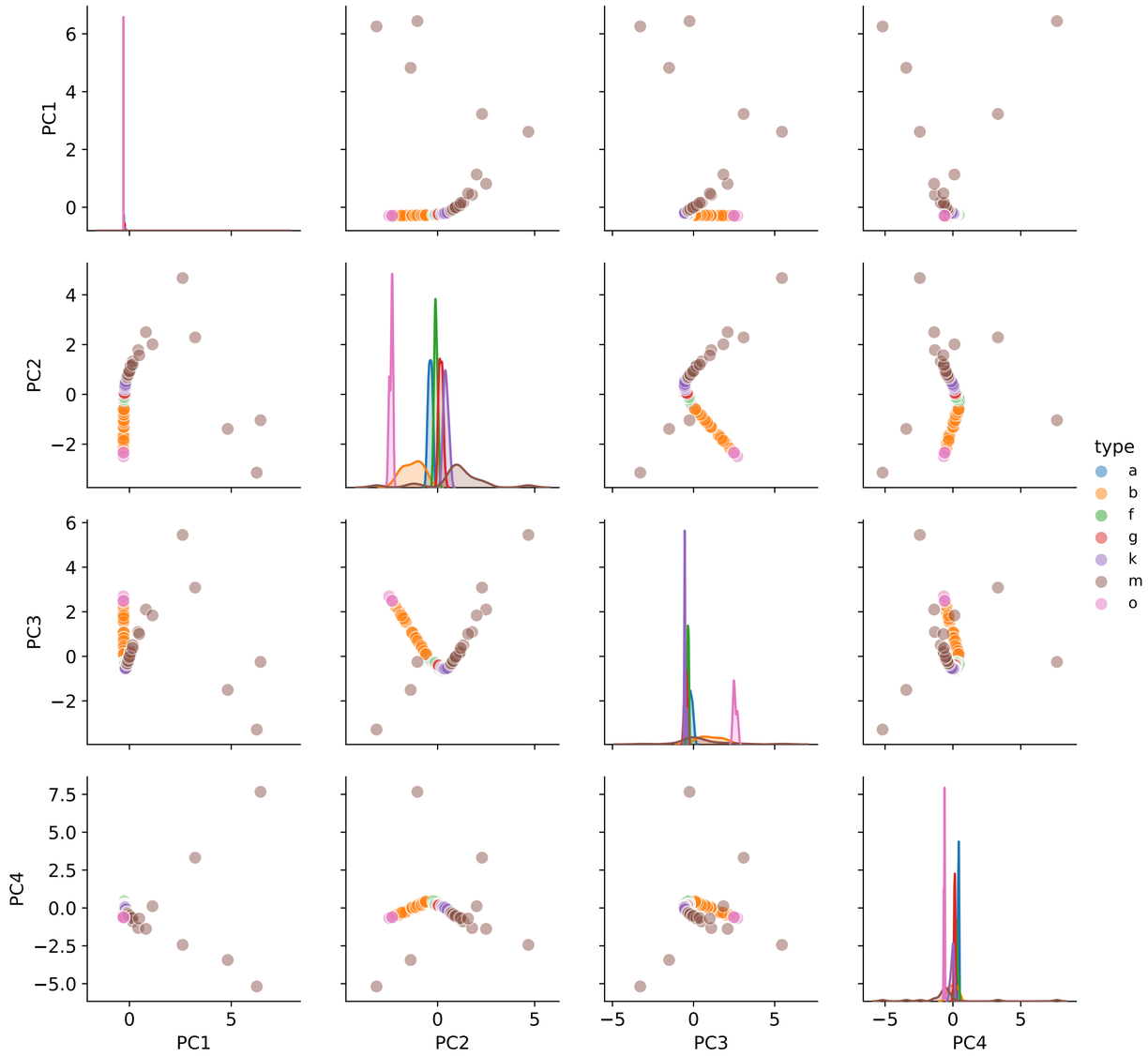
```
from sklearn.decomposition import PCA
pca = PCA(n_components=n_components, whiten=True)
pca.fit(X)
```

PCA of stellar spectra



The mean spectrum

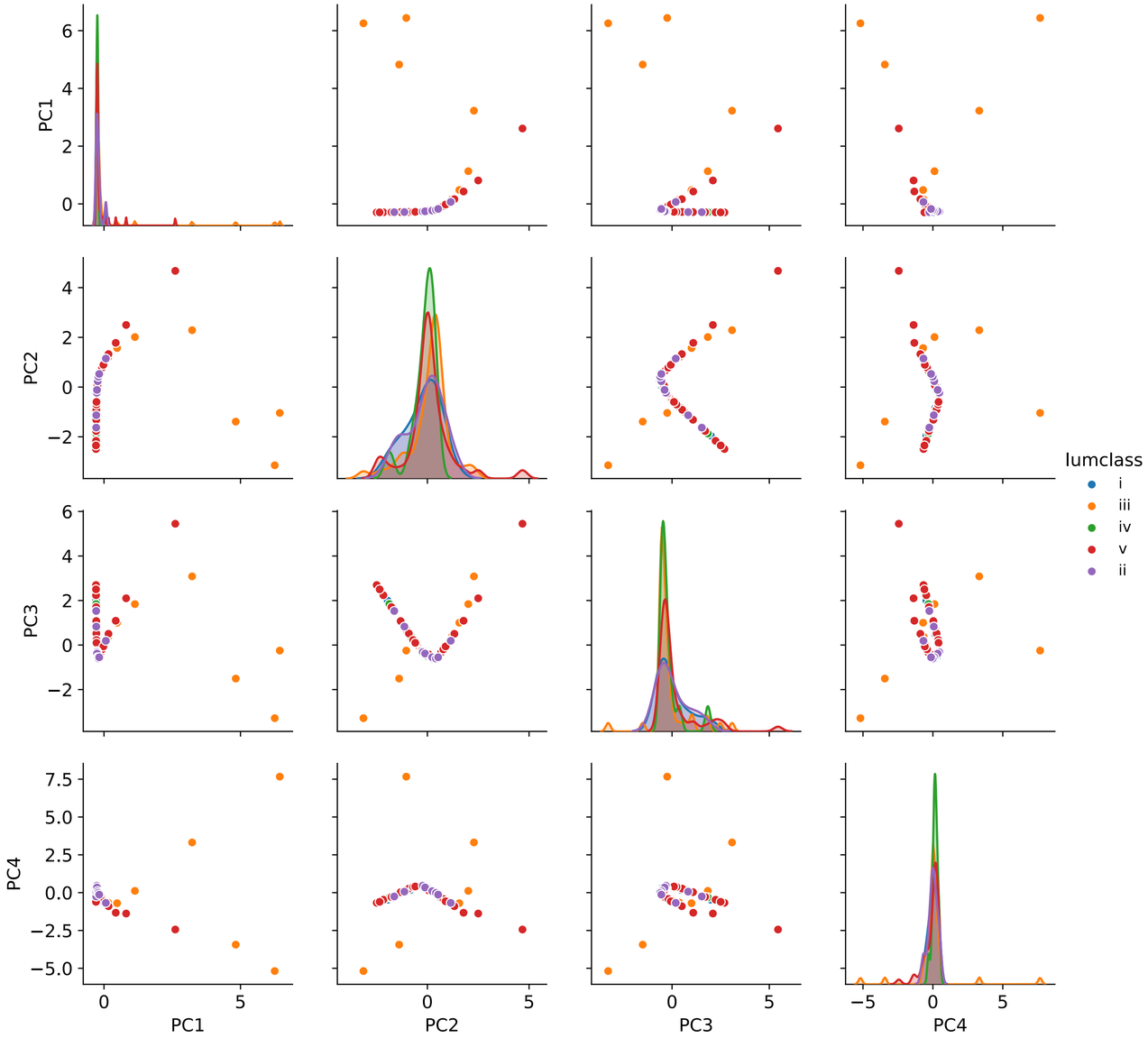
PCA of stellar spectra - the content of the PCs



Stellar types

seems to work fairly well

PCA of stellar spectra - the content of the PCs

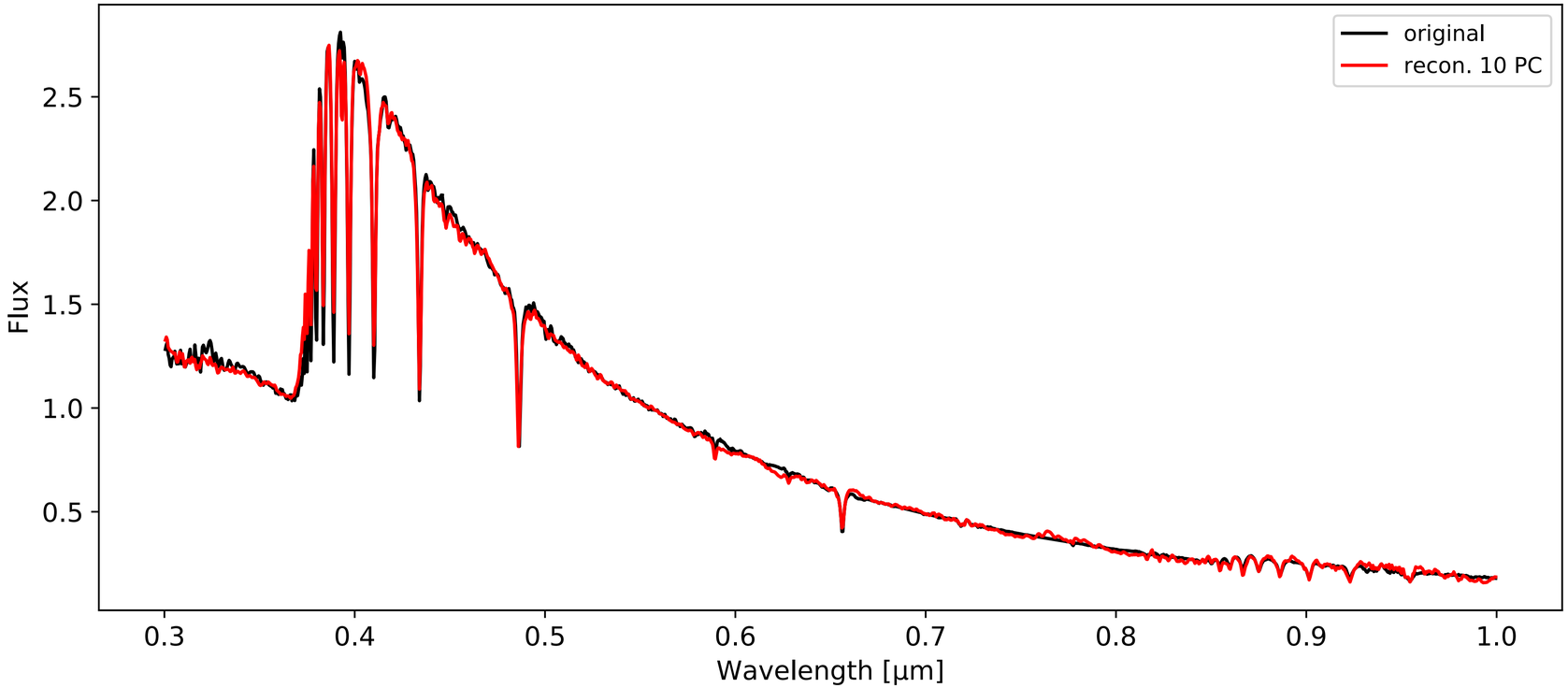


Luminosity class

Does not work at all

PCA of stellar spectra - reconstruction

```
pcs = pca.transform(X)
sp_depr = pca.inverse_transform(pcs)
```



Physical decomposition - positivity

One clear disadvantage of PCA when trying to interpret its results is that the eigencomponents can be negative (bad for spectra!)

An alternative approach is Non-negative Matrix Factorisation (NMF) which can enforce non-negative components.

This is used in astronomy - particularly in fitting galaxy spectra (Blanton & Roweis 2007), but has also seen use in exo-planet direct imaging (Gomez Gonzales et al 2017), sky subtraction of spectra (Zhang et al 2016), X-ray bursts (Degenaar et al 2016), and PAH spectra (Rosenberg et al 2011) to mention some.

One down-side is that you need to decide the number of components first and the data must be >0 (not surprisingly), so noisy data are not suitable.

Physical decomposition - positivity

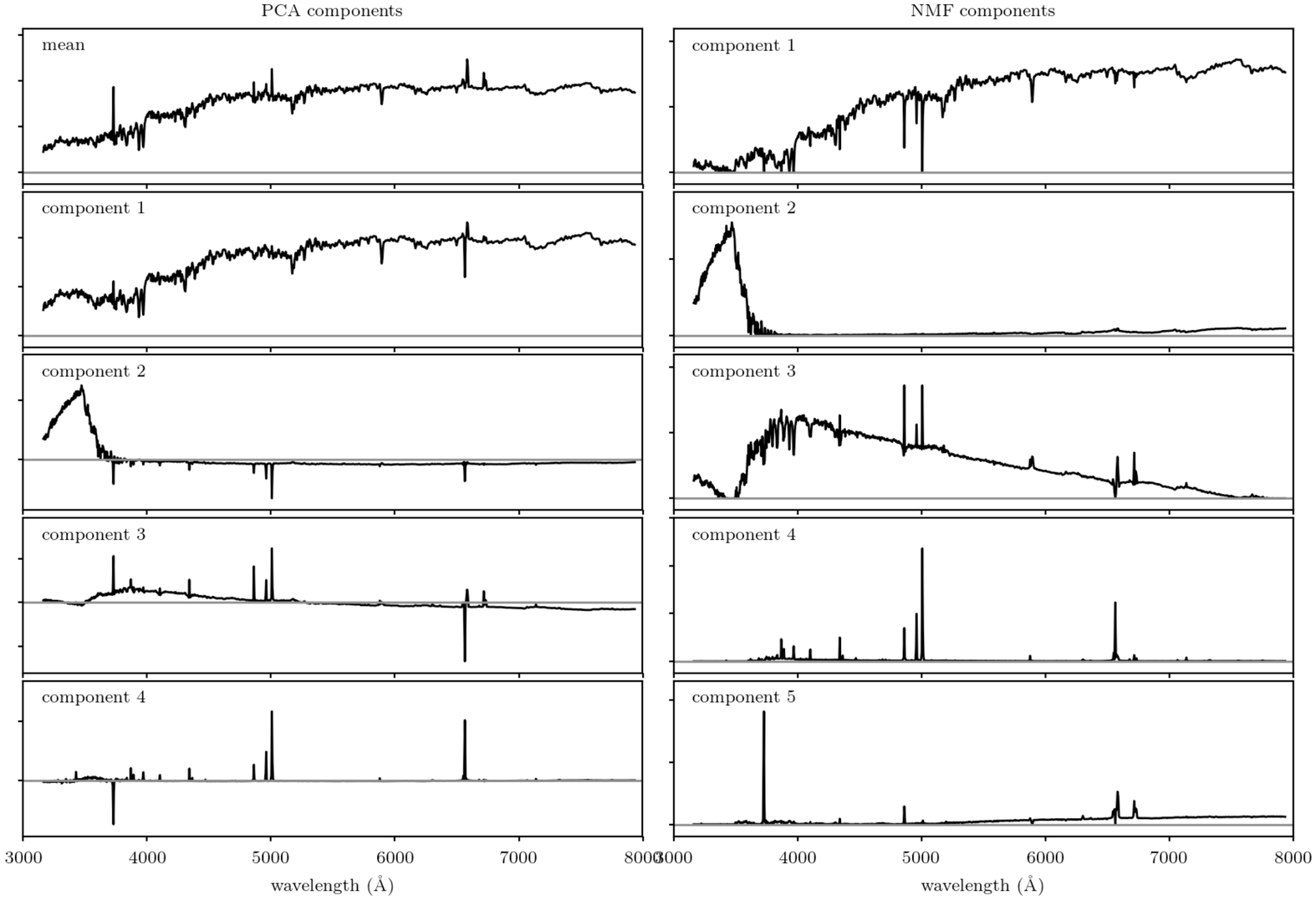
One clear disadvantage of PCA when trying to interpret its results is that the eigencomponents can be negative (bad for spectra!)

An alternative approach is Non-negative Matrix Factorisation (NMF) which can enforce non-negative components.

Physical decomposition - positivity

One clear di…nat the eigencompo…

An alternati…which can enforce …



NMF in Python

The use of NMF is nearly identical to that of PCA:

```
from sklearn.decomposition import NMF

nmf = NMF(n_components)
nmf.fit(spectra)
nmf_comp = nmf.components_
```

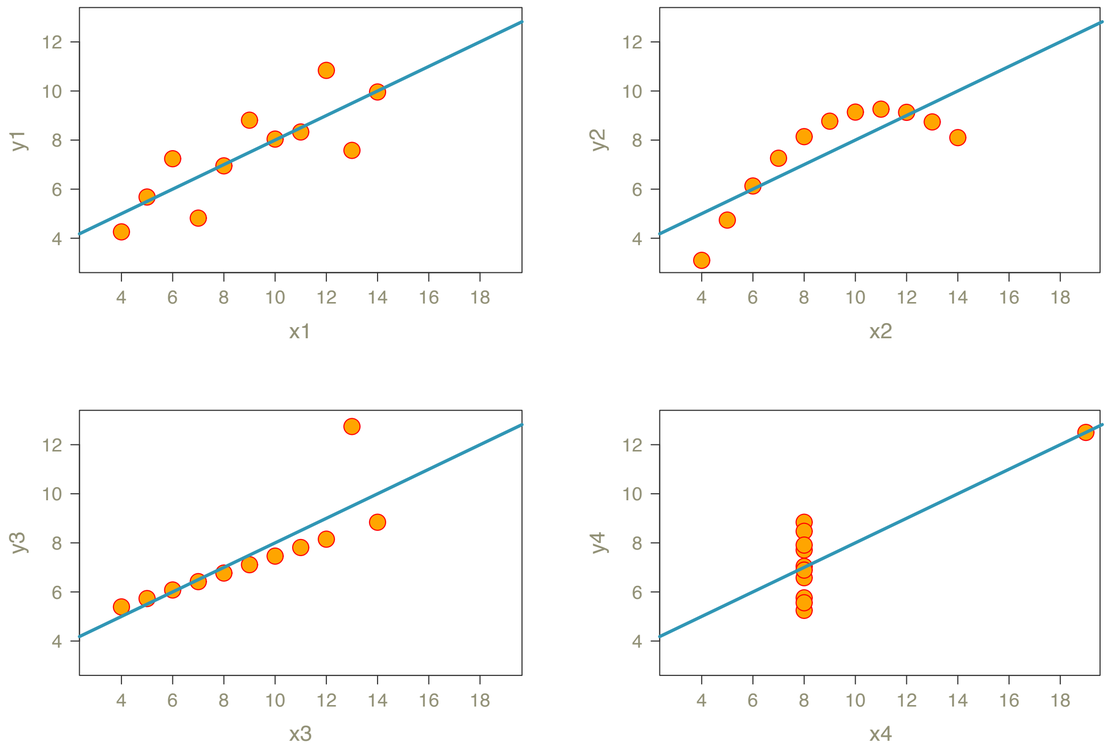
Manifold learning

Linear vs non-linear models

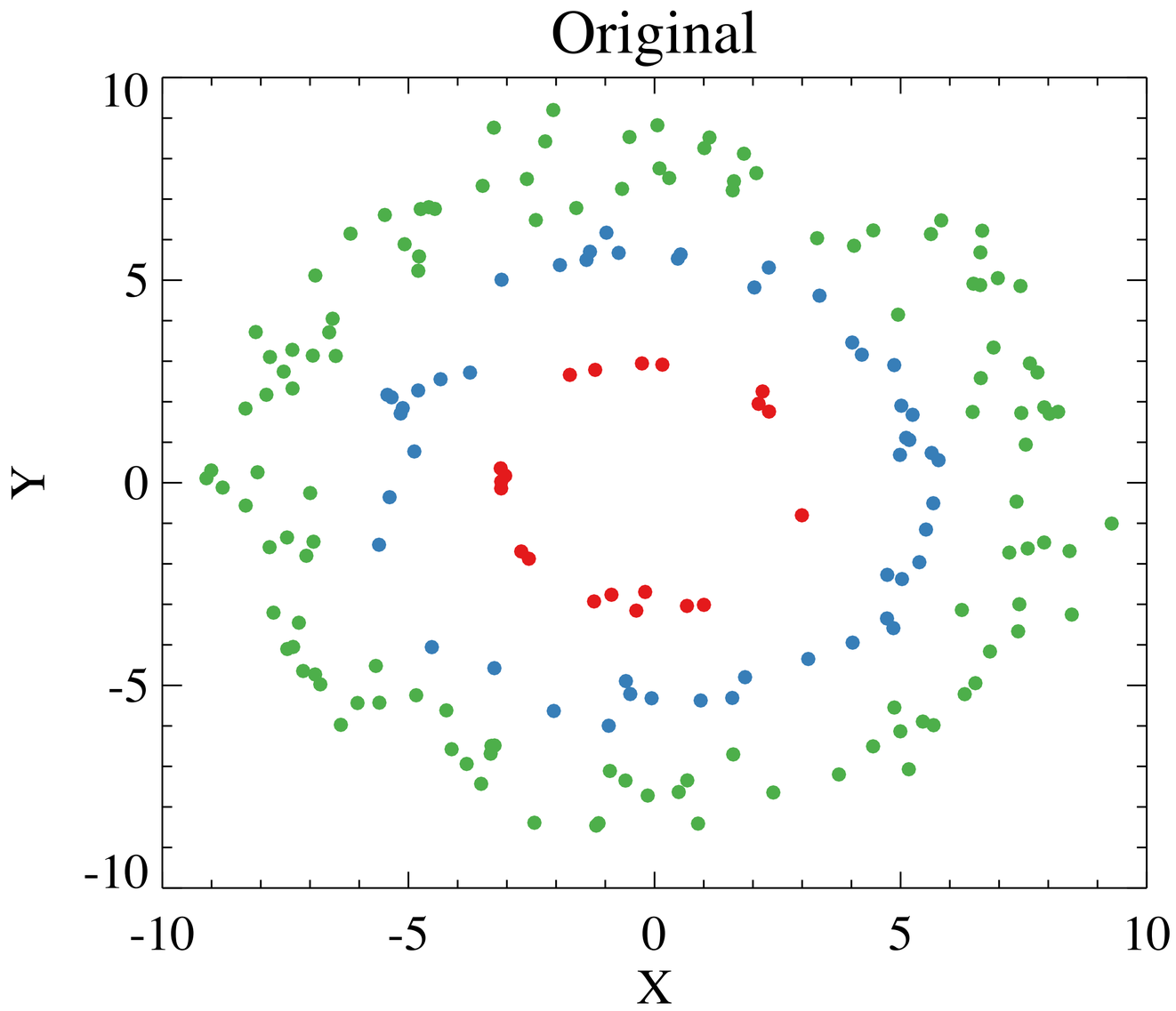
PCA is a linear method - it can find combinations of data vectors that make the resulting principal components linearly independent. Does that mean that PCs have to be independent?

No! (despite what people often write in their papers)

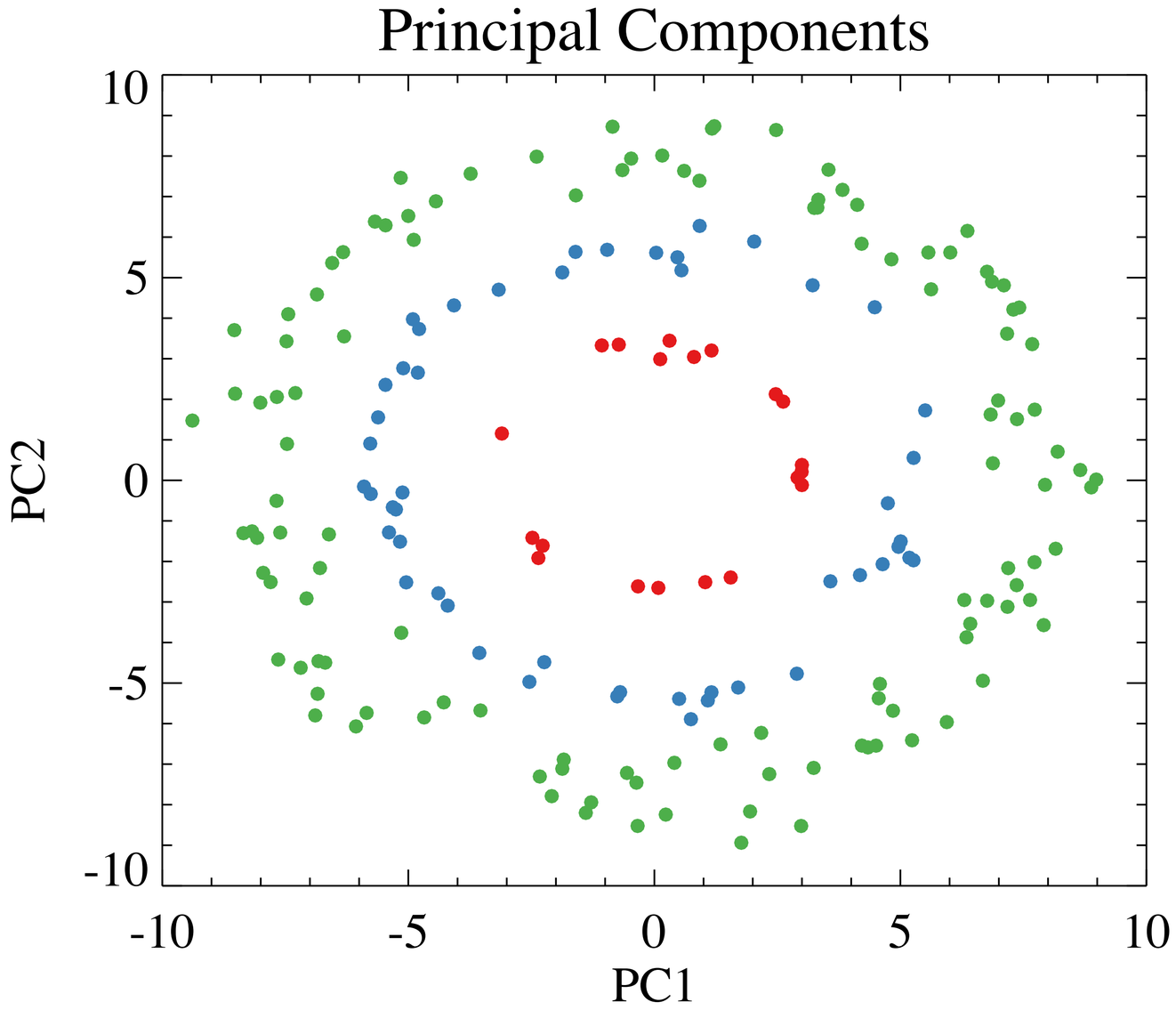
The **correlation coefficient** measures how closely a dataset can be fit by a **straight line** and the *correlation matrix* generalises this to higher dimensions.



Linear vs non-linear



Linear vs non-linear

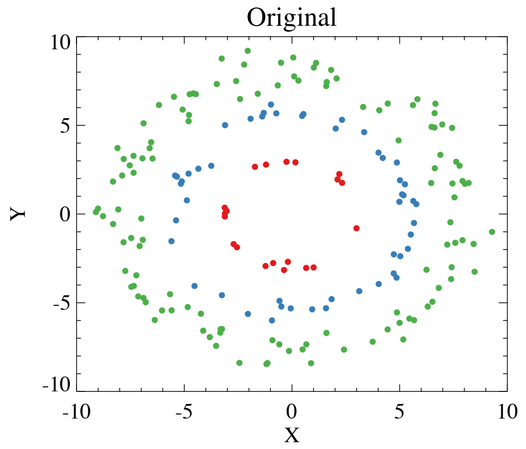


Linear vs non-linear

In this case we know what we should do: We should convert to polar coordinates:

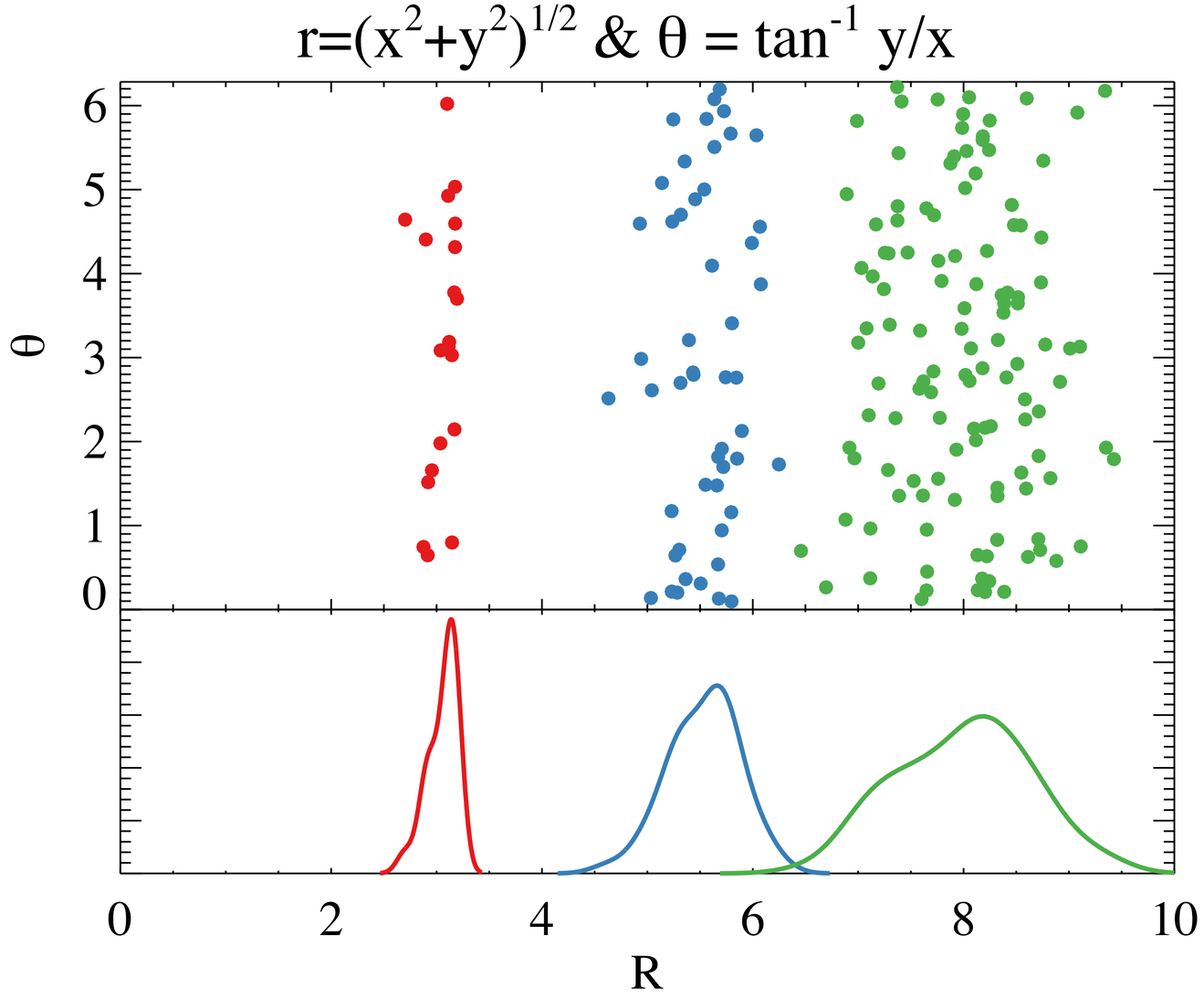
$$r = \sqrt{x^2 + y^2}$$

$$\theta = \tan^{-1} \frac{y}{x}$$

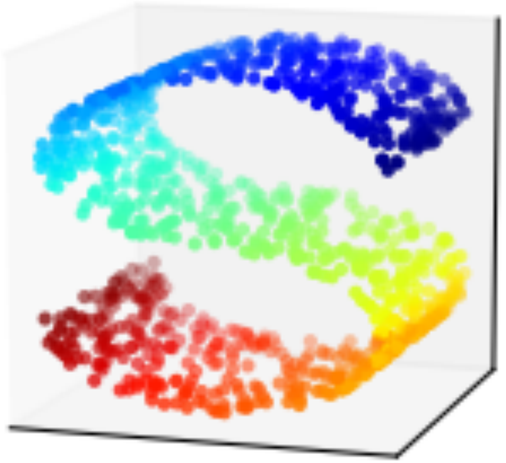


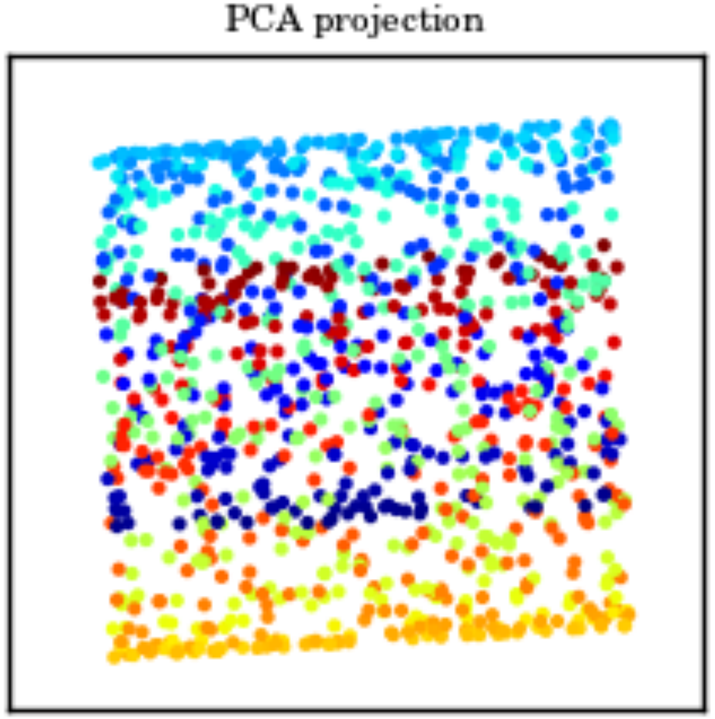
But this is a **non-linear** coordinate transformation so cannot be found by doing **PCA** which only can provide you with **linear transformations** of the input coordinates.

Linear vs non-linear



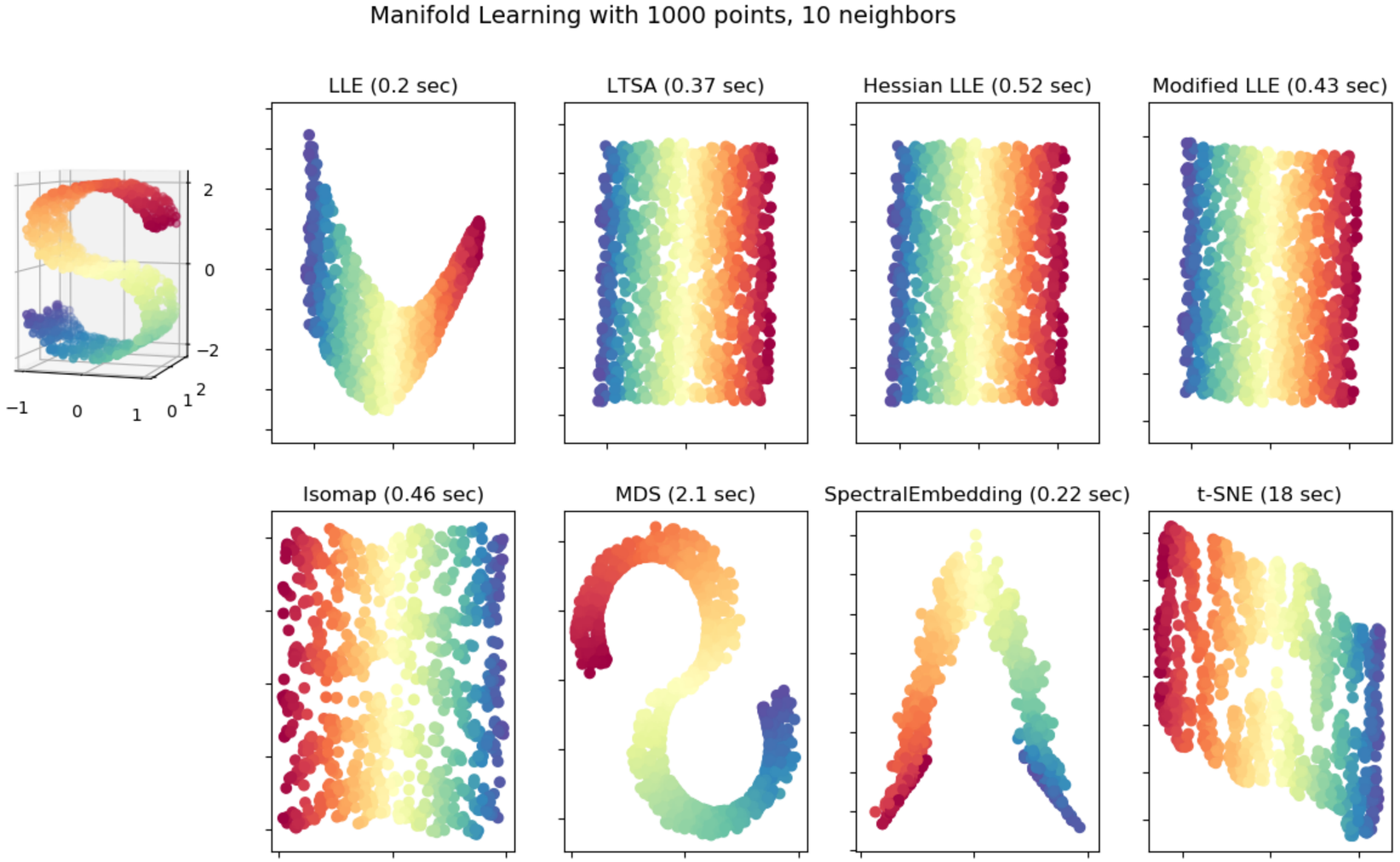
Linear versus non-linear





Enter manifold learning (or nonlinear dimensionality reduction)...

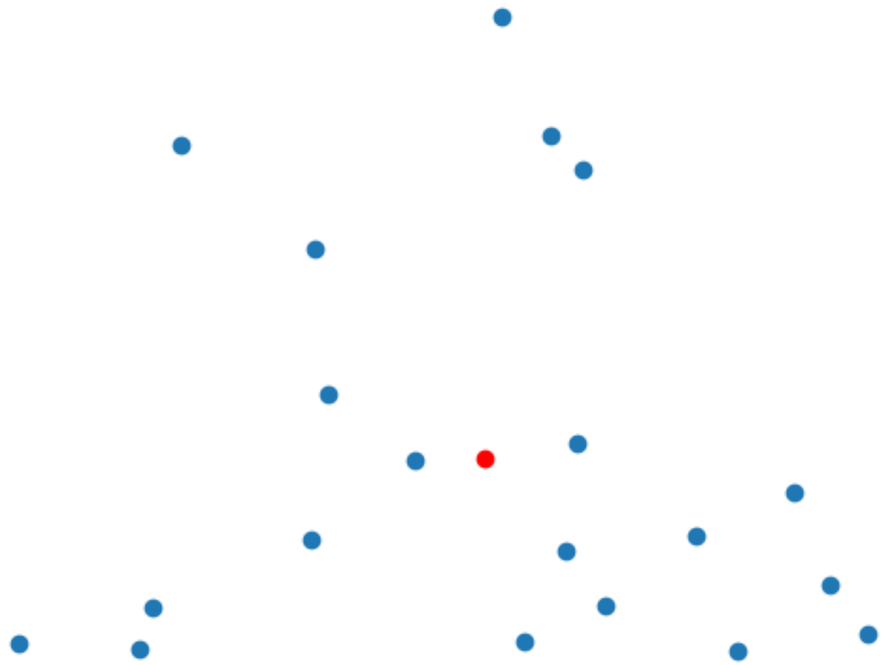
There is a zoo of these - here are a few:



See http://scikit-learn.org/stable/auto_examples/manifold/plot_compare_methods.html

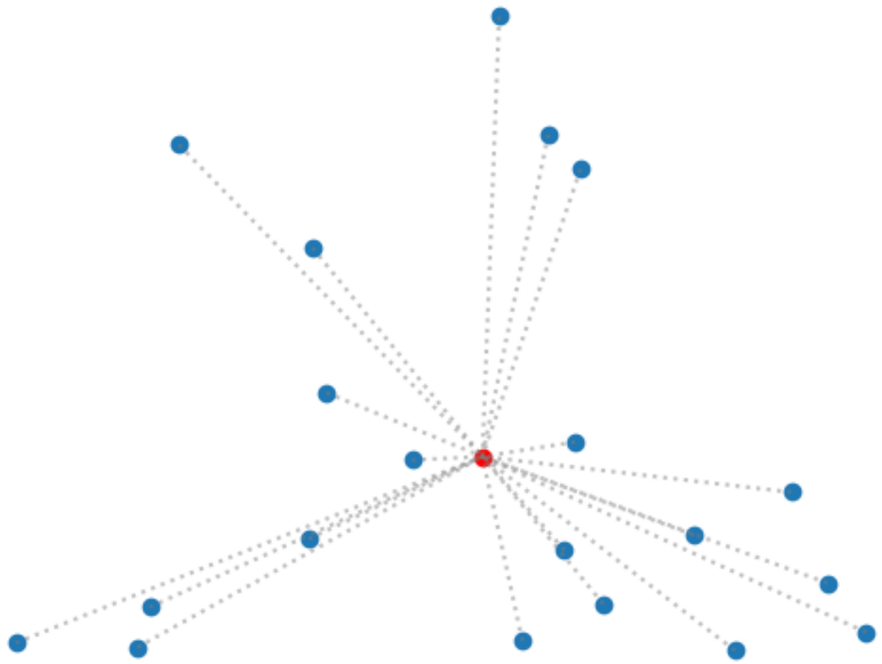
General features

As a rule, the various manifold learning methods use local structure to find good projections.



General features

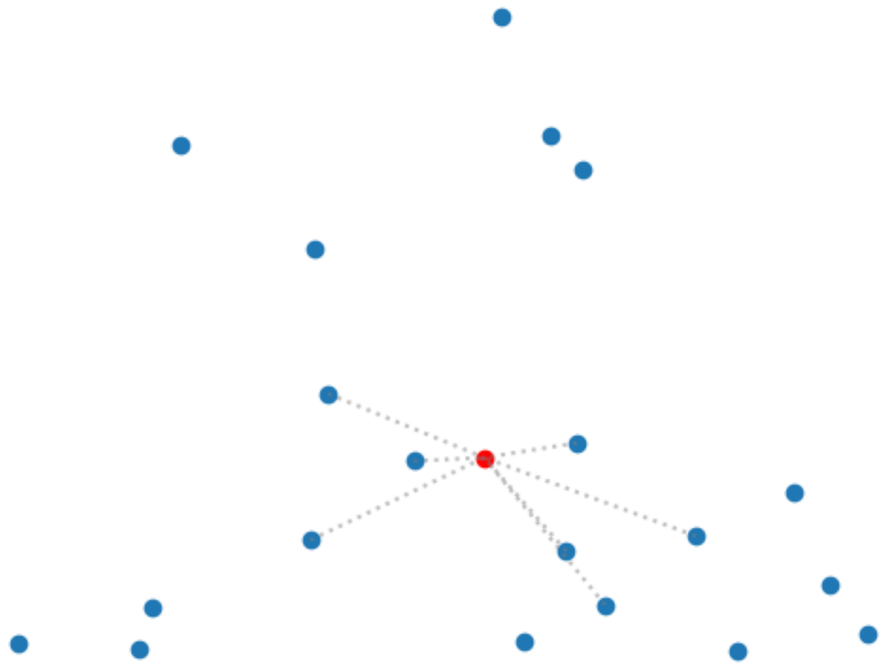
As a rule, the various manifold learning methods use local structure to find good projections.



Using global information to predict

General features

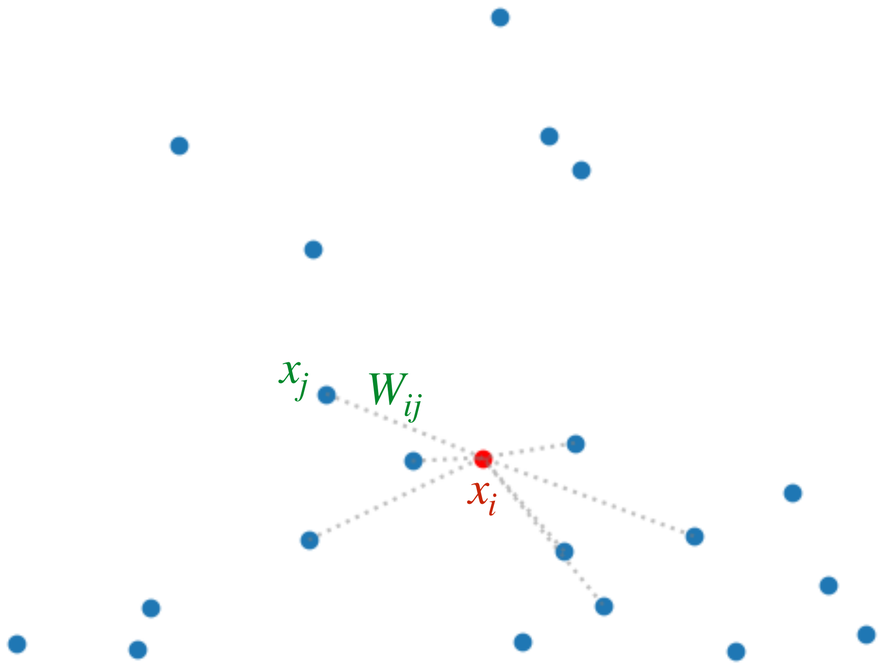
As a rule, the various manifold learning methods use local structure to find good projections.



Using local information to predict

General features

As a rule, the various manifold learning methods use local structure to find good projections.



Using local information to predict

General features

As a rule, the various manifold learning methods use local structure to find good projections.

Example: Locally Linear Embedding (LLE)

Here you try to fit small planes to the neighbourhood of each point and then match these together to project to lower dimensions.

So basically we want to minimise the reconstruction error:

$$\mathrm{Err} = |X - WX|^2$$

Locally Linear Embedding

So basically we want to minimise the reconstruction error of X based on its neighbours:

$$\text{Err} = |X - W X|^2$$

Writing this out we have

$$\text{Err}_x = \sum_{i=1}^{N} \left| x_i - \sum_{j=1}^{N} W_{ij} x_j \right|^2$$

In LLE the trick is to set $W_{ii}=0$ and W_{ij} to zero for all but the k nearest neighbours.

Then we minimise the error function to find W_{ij}

Locally Linear Embedding

That sorts things out in the high dimensional space, but now we want to project onto a lower dimensional space. To do this we minimise:

$$\mathrm{Err}_y = \sum_{i=1}^{N} \left| y_i - \sum_{j=1}^{N} W_{ij} y_j \right|^2$$

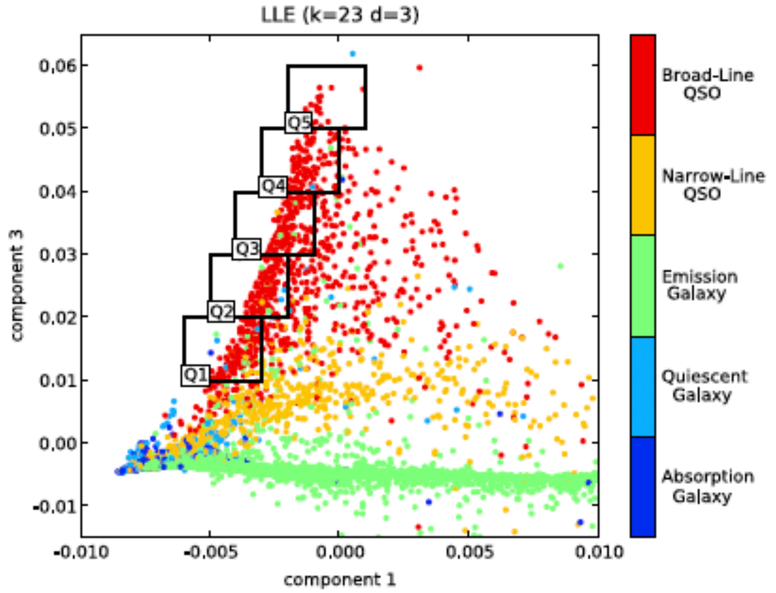
Here W_{ij} is fixed and the positions, $y_{i'}$ are modified. This is an eigenvector problem and you get some eigenvectors as results.

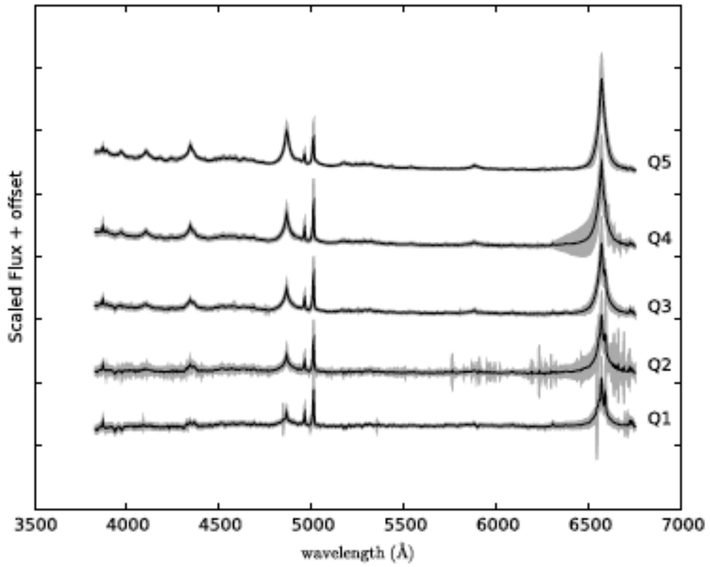
```
from sklearn.manifold import LocallyLinearEmbedding
lle = LocallyLinearEmbedding(k, n)
lle.fit(X)
proj = lle.transform(X)
```

Locally Linear Embedding in astronomy

Vanderplas & Connolly (2009):

LLEs can be used to classify galaxy spectra from the Sloan Digital Sky Survey. The data is here entire spectra and you look at some components of this.





t-Stochastic Neighbour Embedding (t-SNE)

This is an example of a method that is really aimed to project to 2-3 dimensions.

1. Measure the similarities ("closeness") of samples in the original high-D space

$$p_{j|i} = \frac{\exp\left(-\left|x_i - x_j\right|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\left|x_i - x_k\right|^2 / 2\sigma_i^2\right)}$$

2. Create an error function based on the separation in the low-dimensional space and optimise this:

$$q_{ij} = \frac{f\left(\left|x_i - x_j\right|\right)}{\sum_{k \neq i} f\left(\left|x_i - x_k\right|\right)}$$

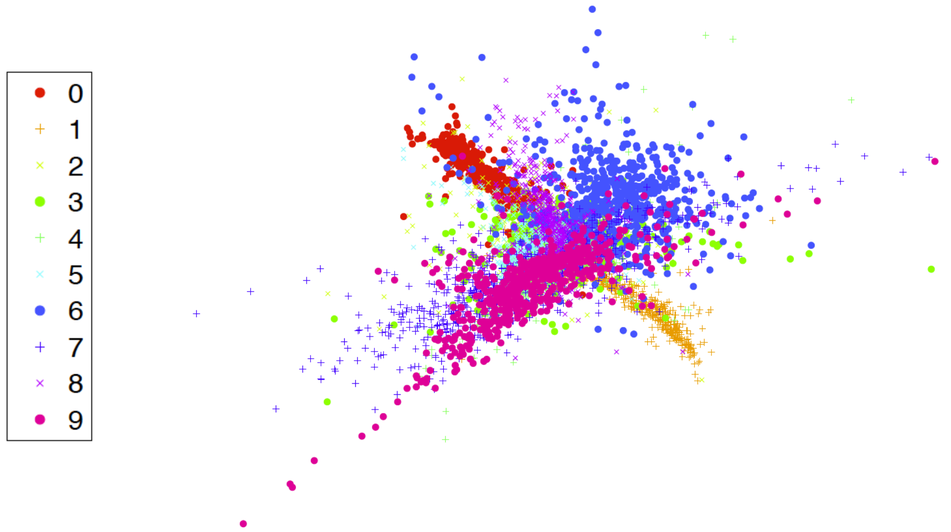
$$f(x) = \frac{1}{1 + x^2}$$

A Cauchy distribution

t-Stochastic Neighbour Embedding (t-SNE)

The MNIST hand-written digits dataset is a classic dataset for testing.

LLE:

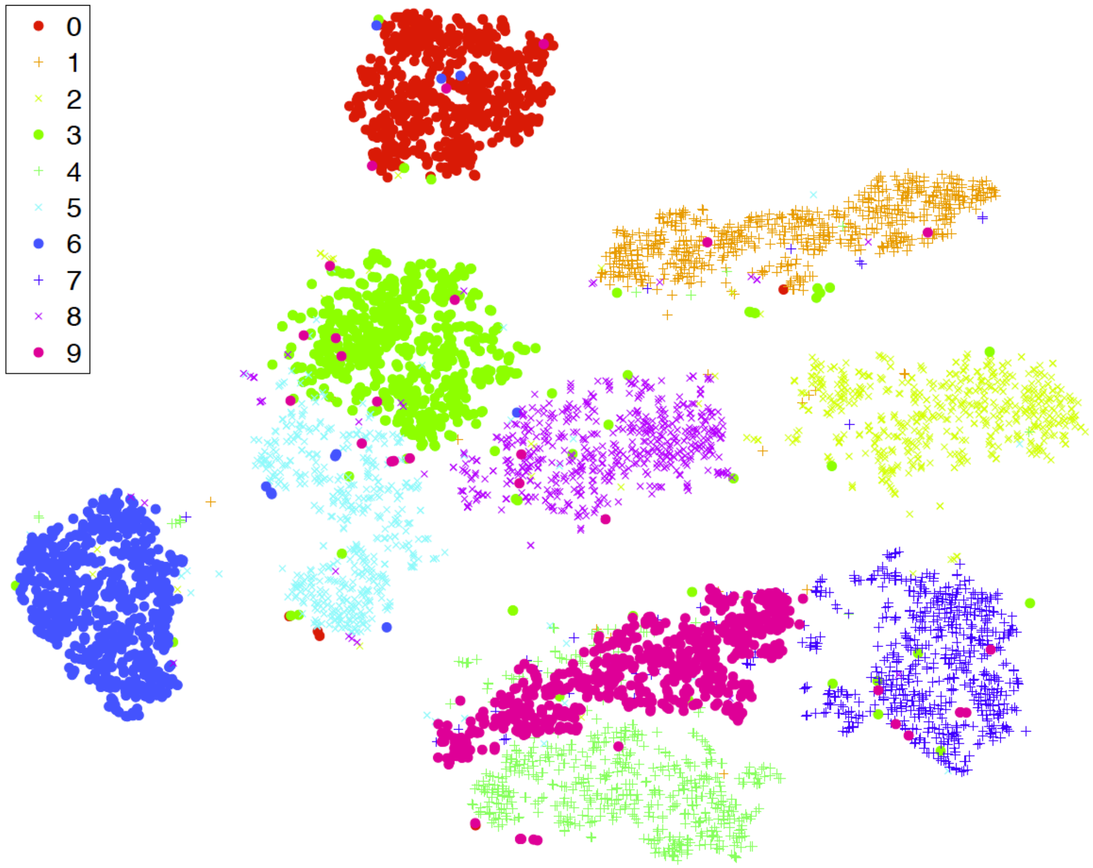


From van Maaten & Hinton (2008), Journal of Machine Learning Research 9 (2008) 2579-2605

t-Stochastic Neighbour Embedding (t-SNE)

The MNIST hand-written digits dataset is a classic dataset for testing.

t-SNE:



From van Maaten & Hinton (2008), Journal of Machine Learning Research 9 (2008) 2579-2605

t-Stochastic Neighbour Embedding (t-SNE)

see: https://www.oreilly.com/learning/an-illustrated-introduction-to-the-t-sne-algorithm

t-Stochastic Neighbour Embedding (t-SNE)

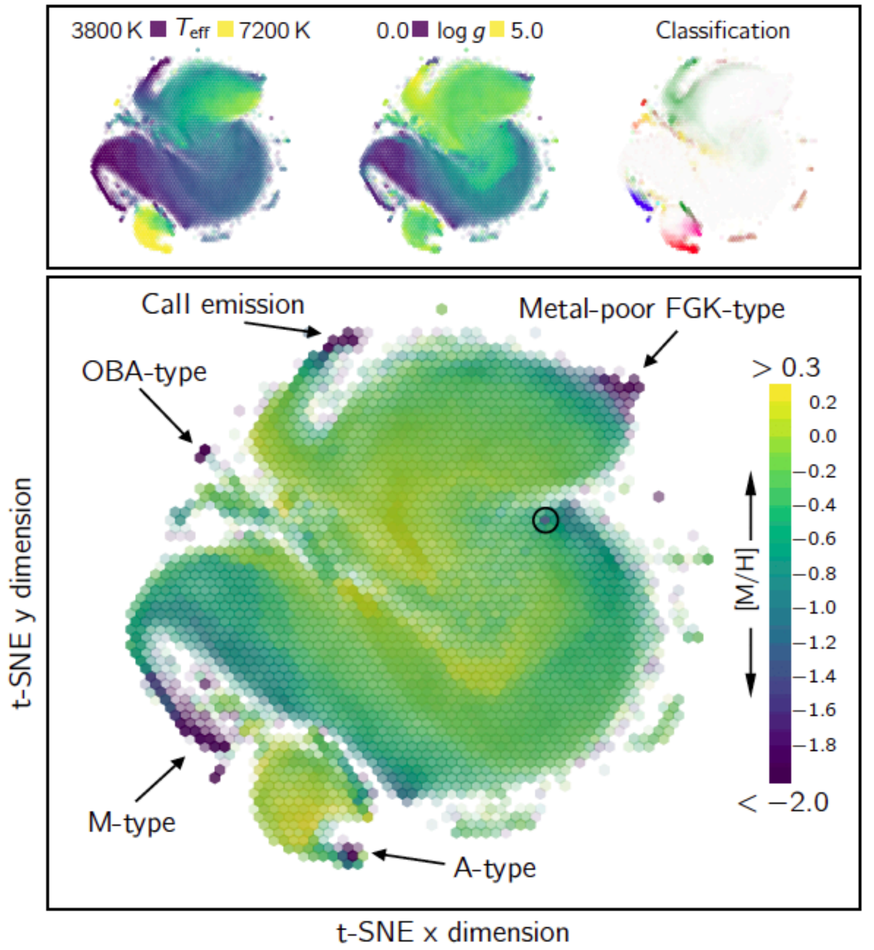
see: https://www.oreilly.com/learning/an-illustrated-introduction-to-the-t-sne-algorithm

t-SNE and others and astronomy

t-SNE is fairly slow but with an update using the Barnes-Hut N-body algorithm it is now useable for large-ish datasets.

Matijevic et al (2017):

Use t-SNE to analyse spectra of stars from the RAVE survey. This is used as a way to identify candidate metal-poor stars.

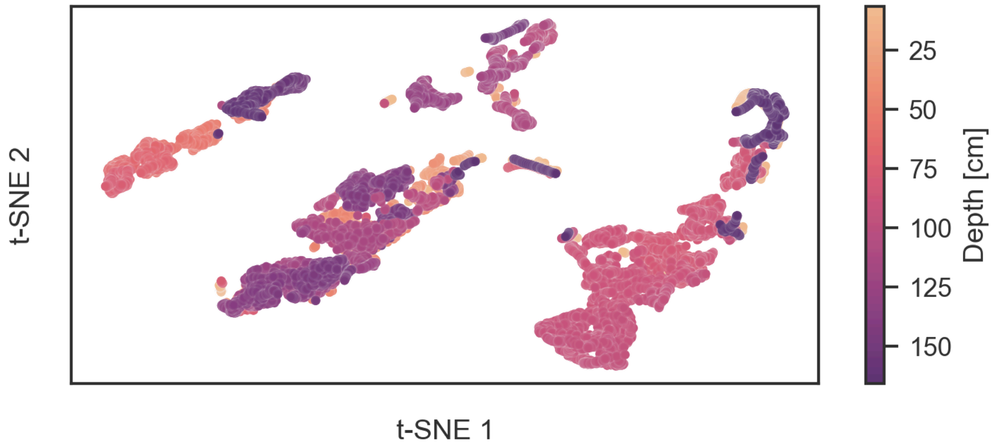


t-SNE and others and astronomy

t-SNE is fairly slow but with an update using the Barnes-Hut N-body algorithm it is now useable for large-ish datasets.

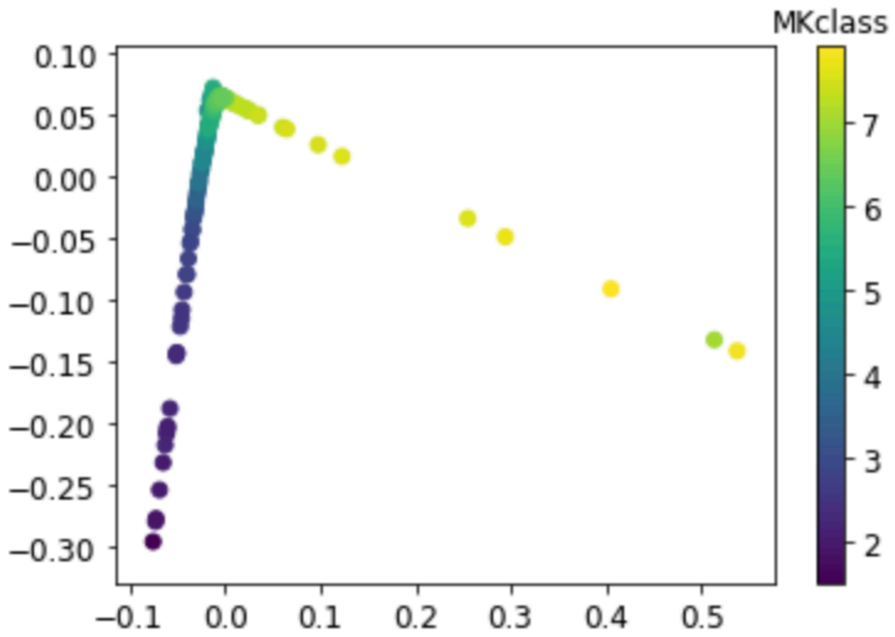
Rossi et al (2024):

Use t-SNE to analyse drilling (artificial!) telemetry data from the “Rosalind Franklin” Mars rover



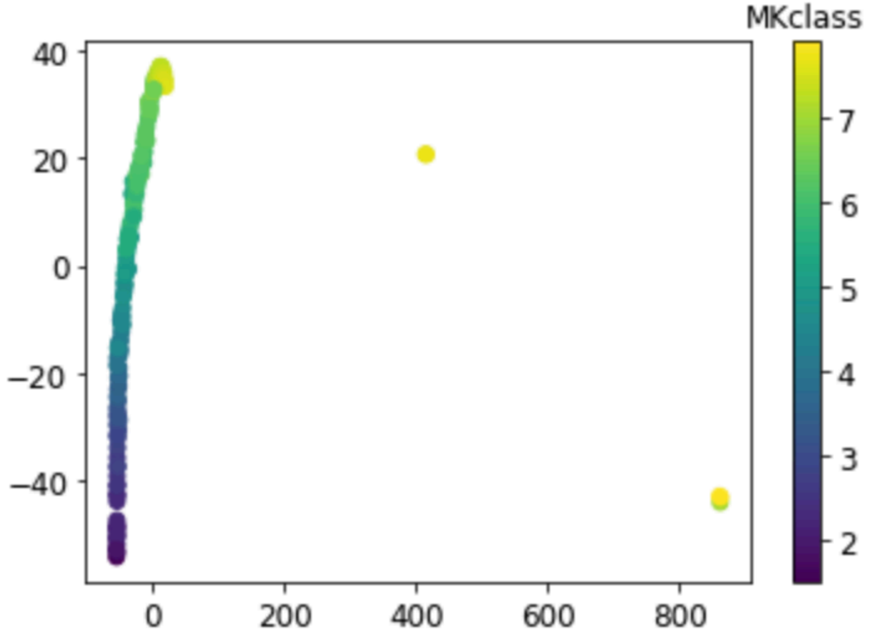
t-SNE & LLE on the stellar spectra

LLE



lle = LocallyLinearEmbedding(10, n_components=2)
proj = lle.fit_transform(X)

t-SNE



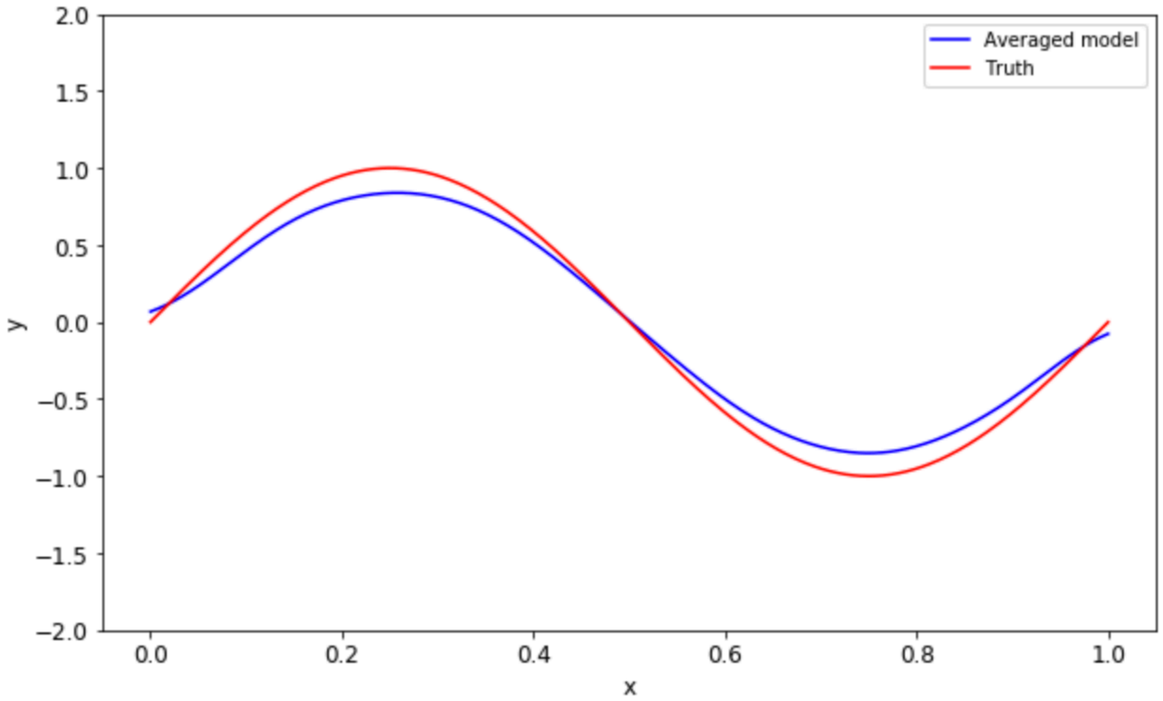
sne = TSNE(n_components=2, init='pca',
random_state=0, perplexity=10)
proj = sne.fit_transform(X)

The pros and cons of manifold learners

- Non-linear dimension reduction can extract structure out of data with non-linearities.
- Can lead to substantial dimensional reduction.
- They deal badly with noisy/gappy data and are (usually) sensitive to outliers.
- All have tuning parameters that you need to decide on.
- To project to lower dimensions you need the full data.
- Memory/CPU requirements can be substantial.

Bottom line: Try PCA first, or possibly NMF.

Combining it all



So averaging the results of the fits gives a better final result, less sensitive to overfitting.

Can this be generalised? We usually do not have N indepedent samples...

Ensemble methods

Bagging

- Draw bootstrap realisations of your data.
- Fit these data.
- Average the result.

This is also known as Bootstrap aggregating.

It typically can help reduce overfitting problems.

In the case of perfectly uncorrelated errors in prediction, the error goes down as

$$\frac{\sigma}{\sqrt{N_{\text{models}}}}$$

If the errors are completely correlated, the error stays the same - ie. bagging has no effect.

Decision trees - an intermezzo

These are tree structures where at each level you split the data according to some criterion.

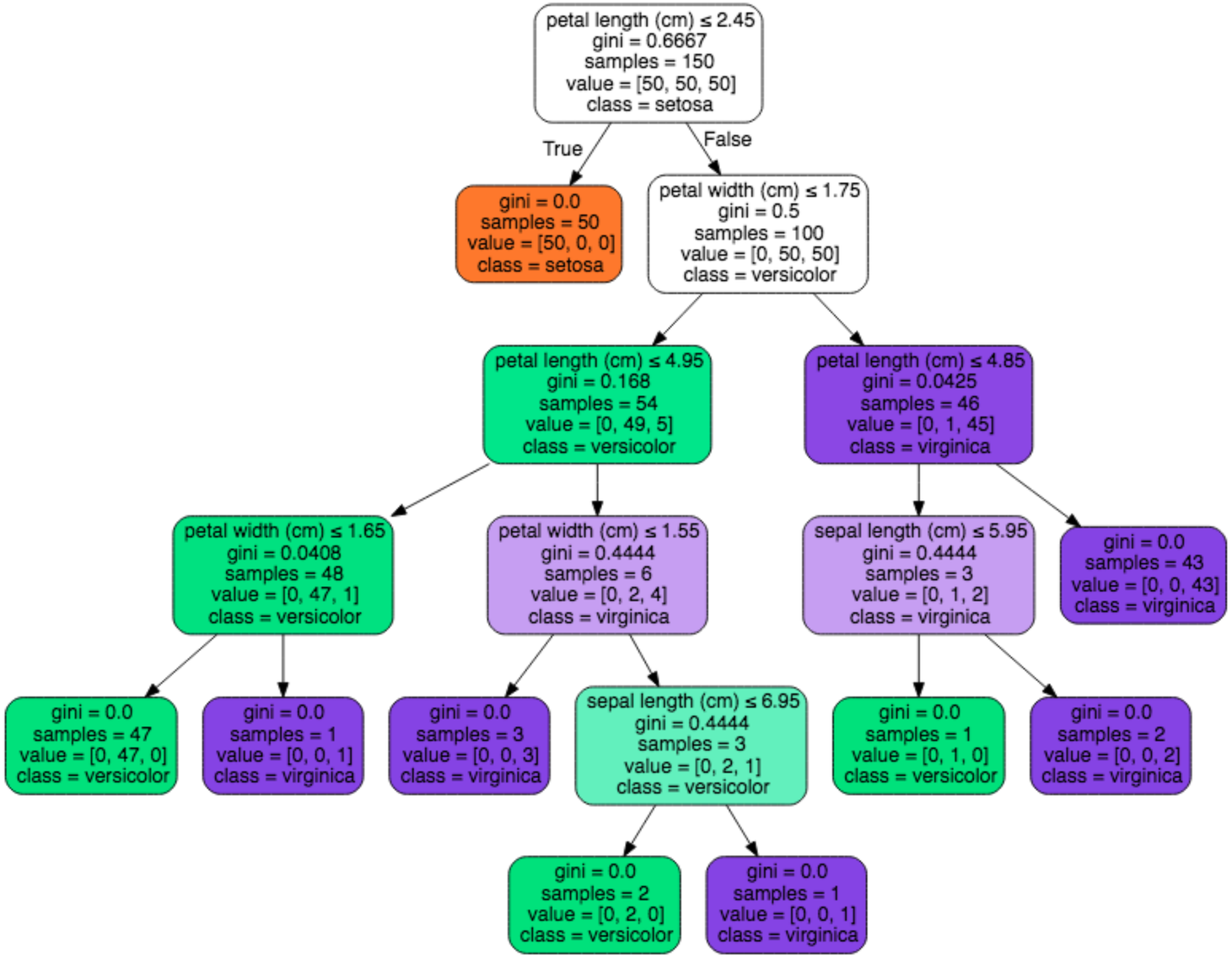
By making the tree deeper you can catch finer details in the data.

These are powerful techniques and can be inspected afterwards (good!) but do not usually have very high accuracy (not so good), and they have high variance (not ideal).

In sklearn:

```
from sklearn import tree
<get your data>
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)
```

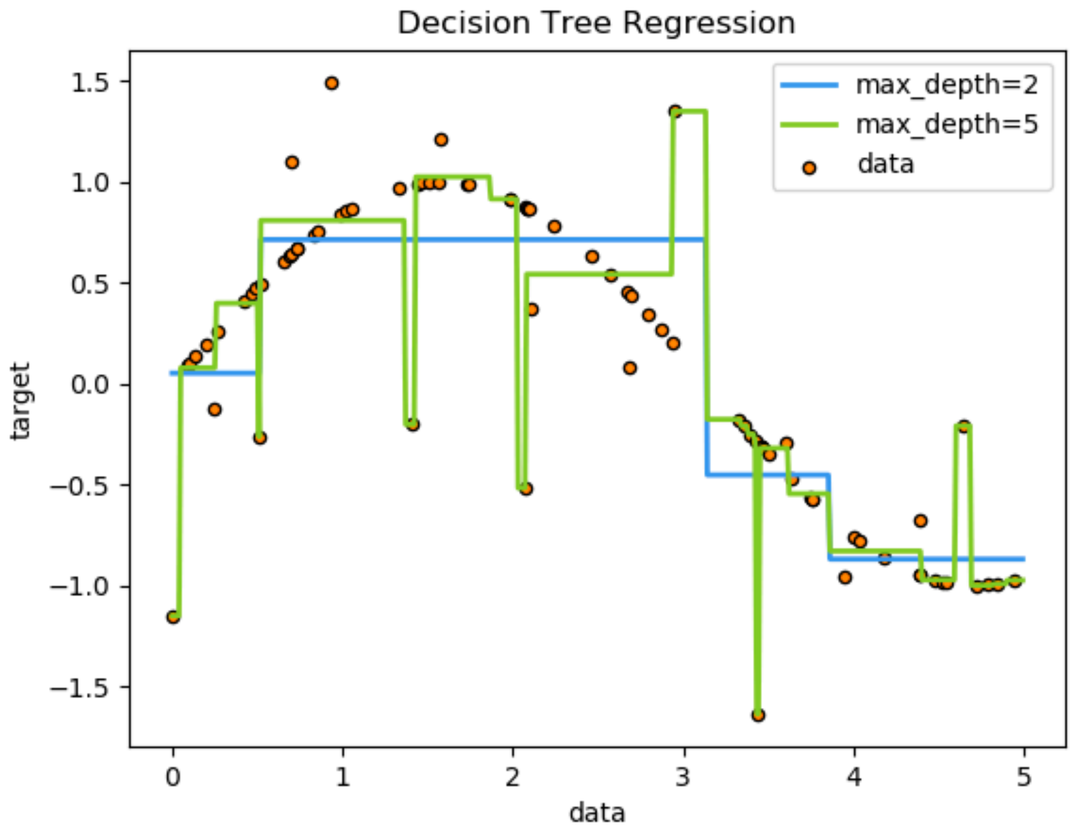
Decision trees - an intermezzo



http://scikit-learn.org/stable/modules/tree.html

Decision trees - an intermezzo

It can also be used for regression:



http://scikit-learn.org/stable/modules/tree.html

Random Forests

The bagging method can be applied to decision trees too.

However if you combine bagging with randomly choosing a subset of features at each level, you get **random forests** and these seem to perform particularly well.

Basic algorithm (https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

```
from sklearn.ensemble import RandomForestClassifier
```

For each tree:

1. Sample N cases at random - but with replacement, from the original data
2. At each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

The random subsets is to avoid correlation (recall the bagging error function)

Boosting

These are techniques that combine many very simple algorithms (“weak learners”) to create a powerful final algorithm.

A weak learner typically is chosen to have a low bias, so as a consequence has high variance. The aim of boosting is to combine these to get a low bias, low variance estimator.

Boosting

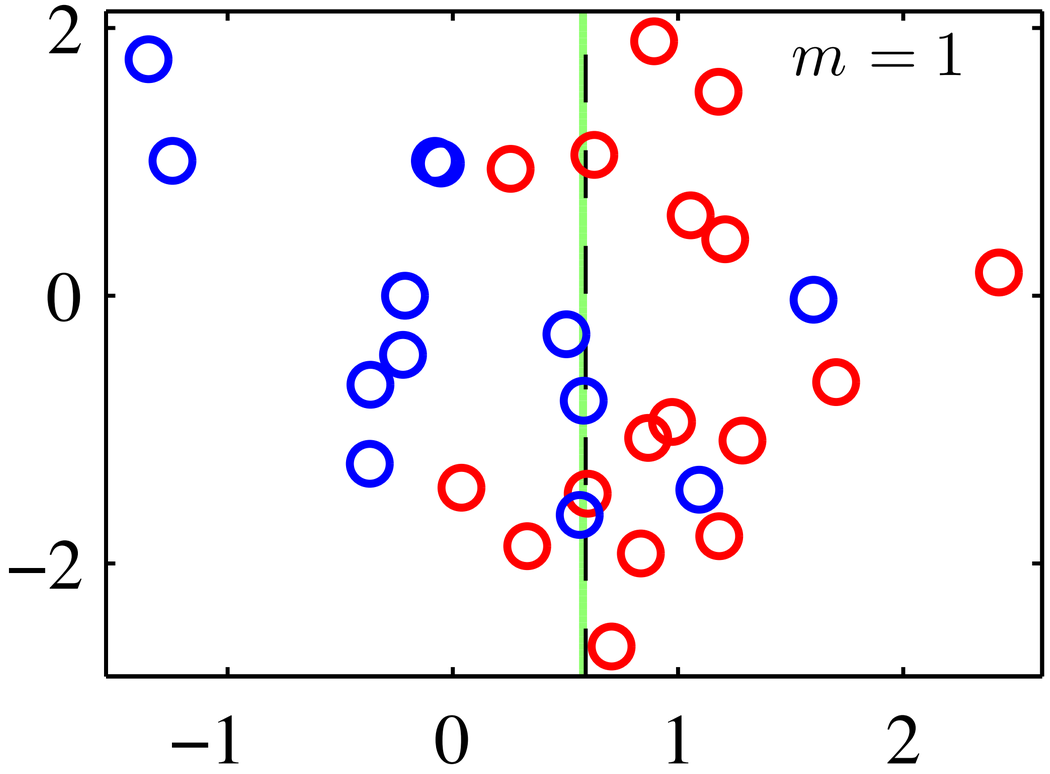
These are techniques that combine many very simple algorithms (“weak learners”) to create a powerful final algorithm.

The approach (AdaBoost):

1. Fit a learner (algorithm)
   Find how well this works and give high weight to those examples it did **not** fit.
   Repeat.
2. Average the results using weights estimated during the fitting procedure.

Boosting

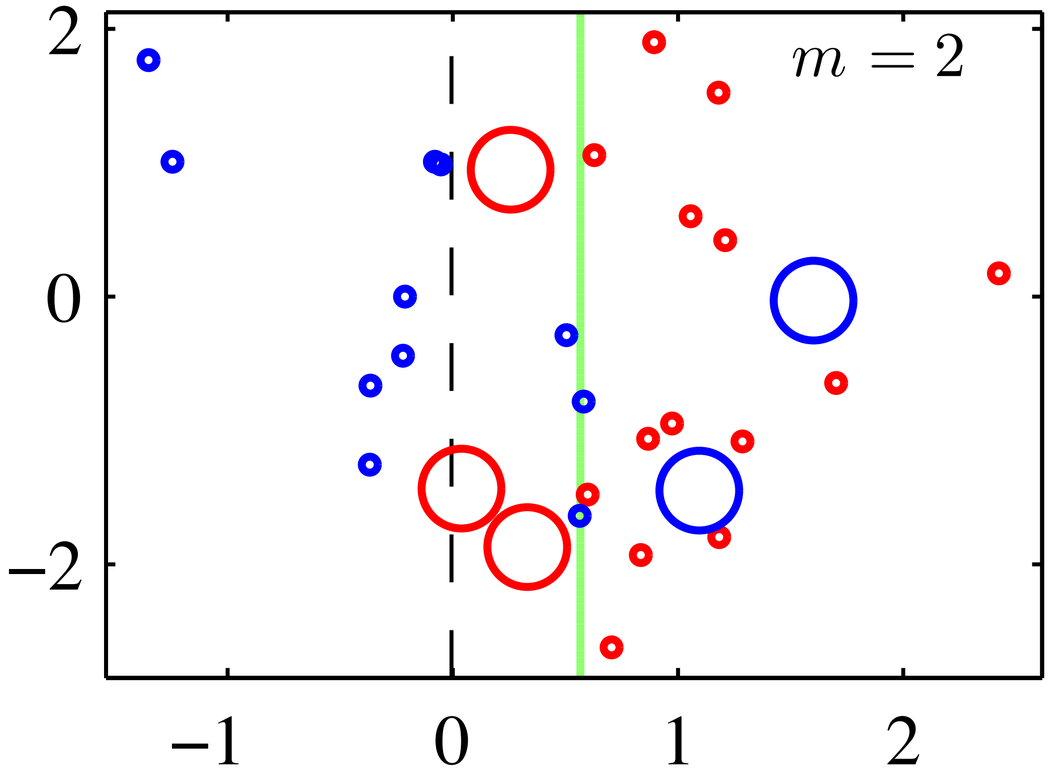
These are techniques that combine many very simple algorithms ("weak learners") to create a powerful final algorithm.



From Bishop "Pattern Recognition and Machine Learning", Figure 14.2

Boosting

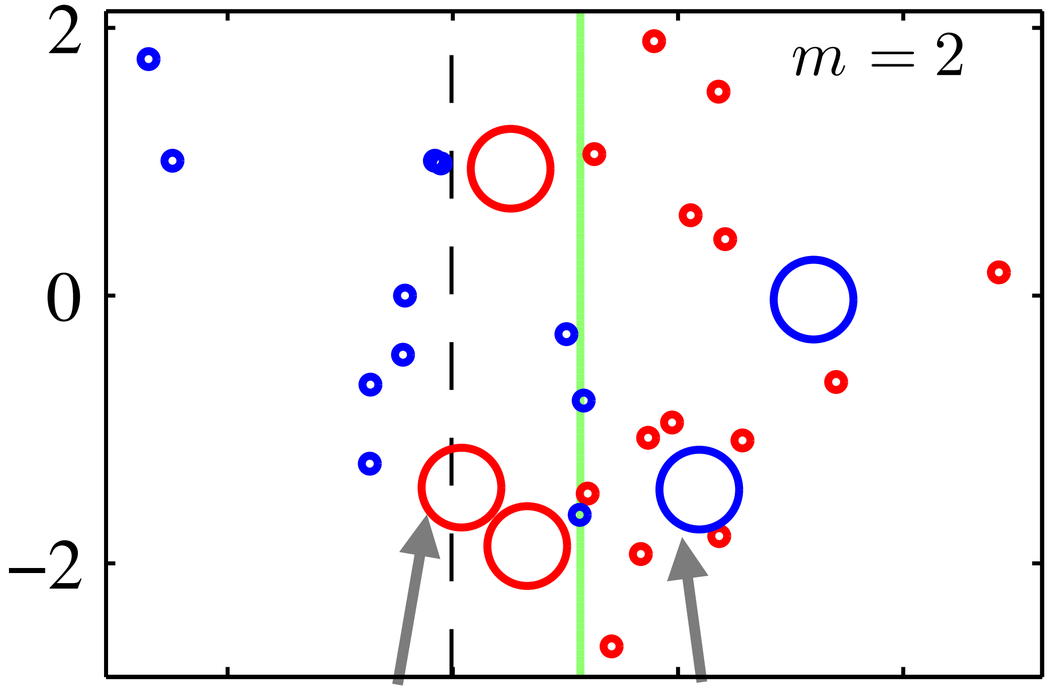
These are techniques that combine many very simple algorithms ("weak learners") to create a powerful final algorithm.



From Bishop "Pattern Recognition and Machine Learning", Figure 14.2

Boosting

These are techniques that combine many very simple algorithms ("weak learners") to create a powerful final algorithm.

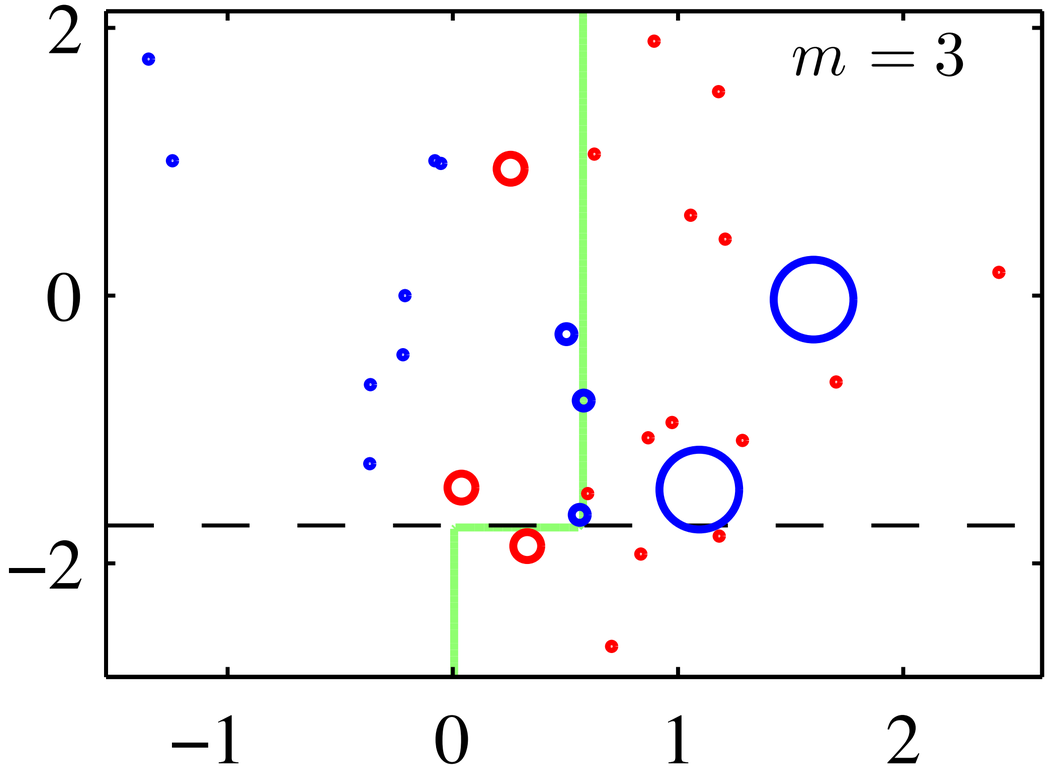


Higher weight to misclassified examples

From Bishop "Pattern Recognition and Machine Learning", Figure 14.2

Boosting

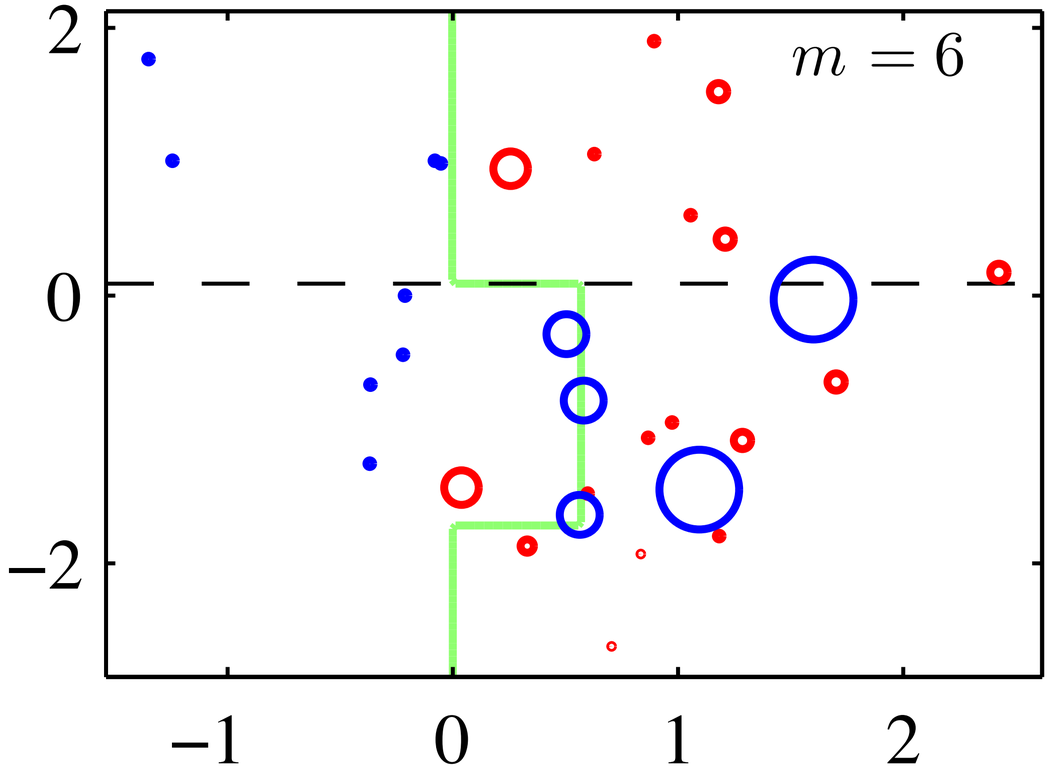
These are techniques that combine many very simple algorithms ("weak learners") to create a powerful final algorithm.



From Bishop "Pattern Recognition and Machine Learning", Figure 14.2

Boosting

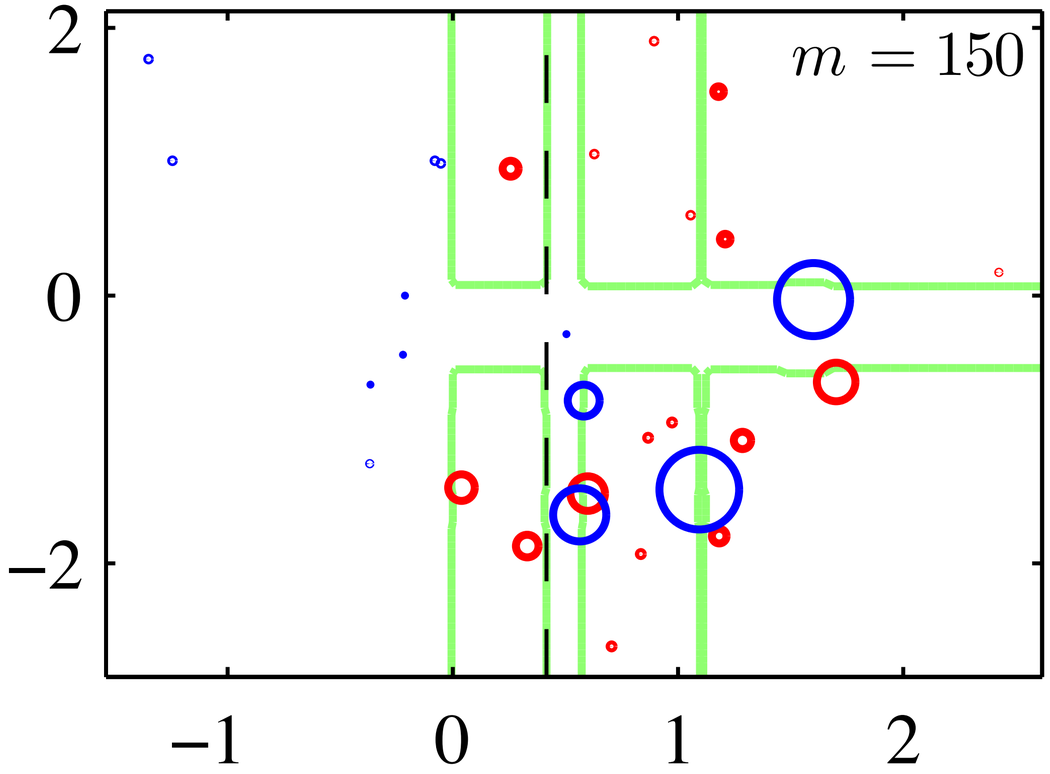
These are techniques that combine many very simple algorithms ("weak learners") to create a powerful final algorithm.



From Bishop "Pattern Recognition and Machine Learning", Figure 14.2

Boosting

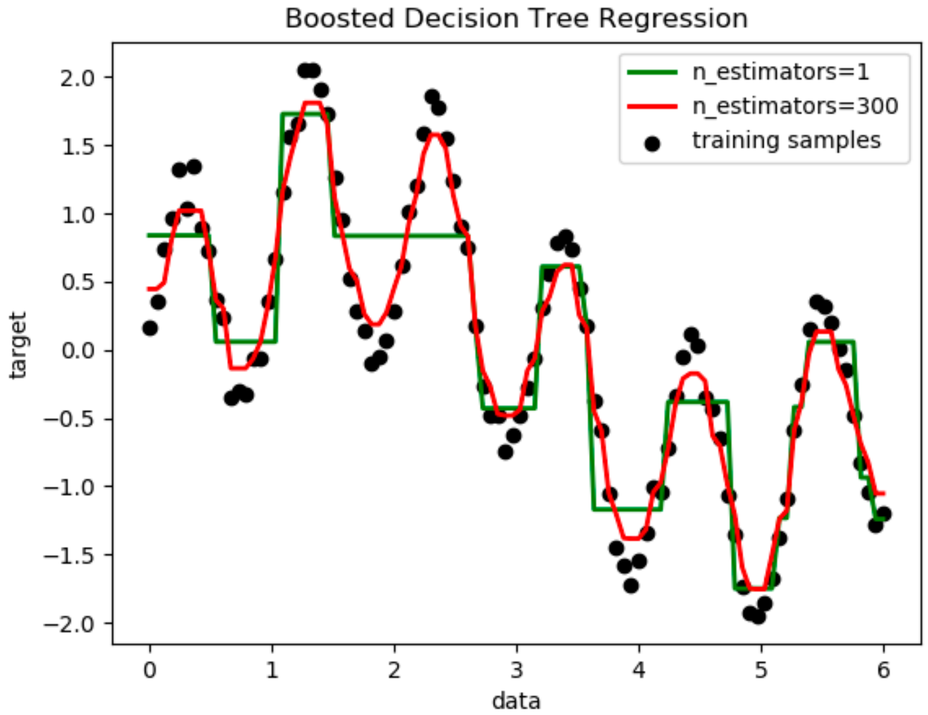
These are techniques that combine many very simple algorithms ("weak learners") to create a powerful final algorithm.



From Bishop "Pattern Recognition and Machine Learning", Figure 14.2

Boosting

These are techniques that combine many very simple algorithms (“weak learners”) to create a powerful final algorithm.



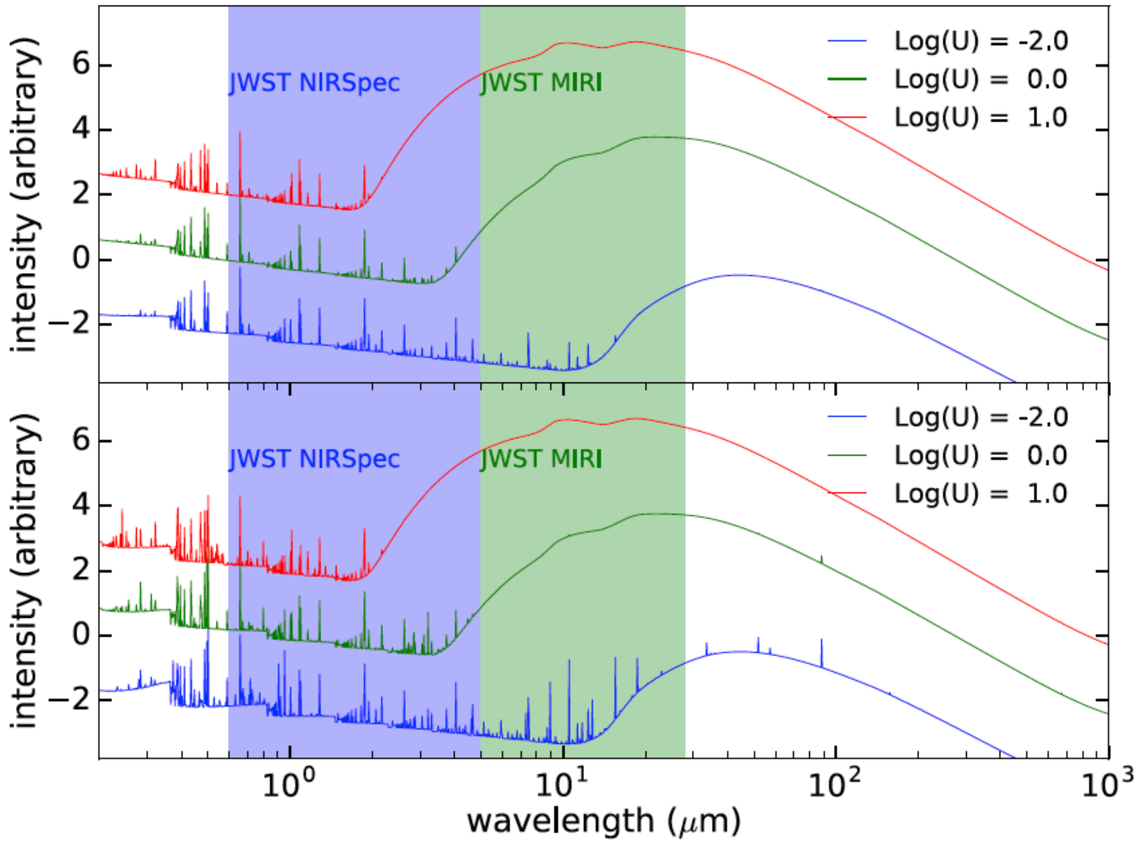

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
<…>
regr = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),
                         n_estimators=300)
regr.fit(X, y)
y_2 = regr.predict(X)
```

http://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_regression.htm

Use in astronomy

AdaBoost: Ucci et al (2017): Fitting emission line spectra (code GAME)

Input library to compare against:



Use in astronomy

AdaBoost:

Ucci et al (2017): Fitting emission line spectra.

Xin et al (2017): Finding impact craters on Mars.

Zitlau et al (2016): Photometric redshifts for SDSS.

++

Random Forests:

Kuntzer & Courbin (2017): Detecting binary stars.

García-Varela et al (2017): Finding variable stars.

Jouvel et al (2017): Photometric redshifts of galaxies.

Bastien et al (2017): Classification of radio galaxies

Torres et al (2019): White dwarfs in the MW from GAIA

Ulmer-Moll (2019): Exoplanet mass-radius relation

+++

Other boost options

dmlc XGBoost

XGBoost is somewhat similar to Random Forests but using boosting rather than bagging and also a more sophisticated combination method - very powerful and widely used.

Catboost is also boosting method - there are similarities to XGBoost but it is particularly suited for categorical data (non-numerical) which requires some pre-processing for XGBoost.