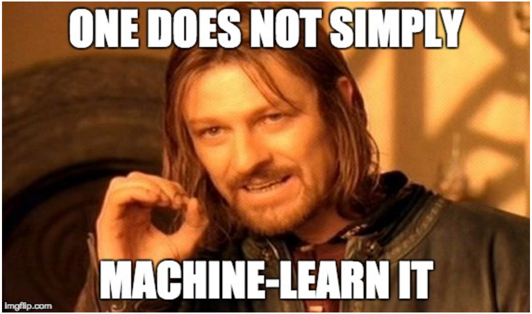
Data collection and preprocessing

Adriano Agnello

274th April 2020

Playing with multi-dimensional data

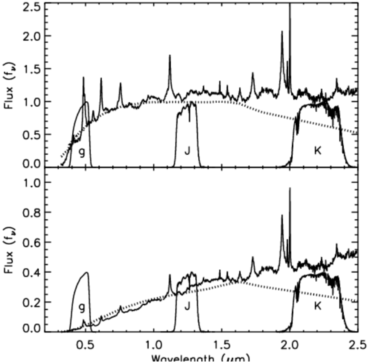
- Part 1: some real-life datasets, surveys and queries.
- Part 2: visualising and dimensionality reduction, PCA, kPCA



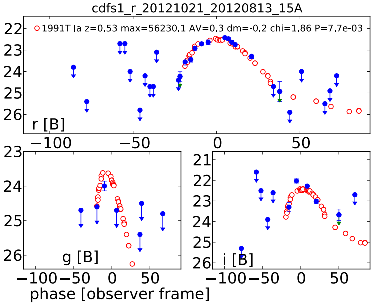
Part 1: surveys, databases, queries & thereabouts

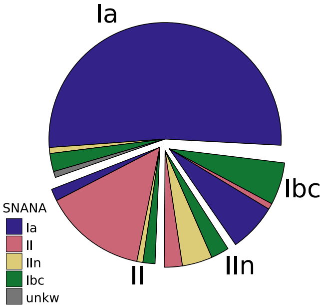
General problem: we have big heaps of data produced by surveys/experiments and need to make sense of them.

Example from astro: spectra, fluxes, colours.



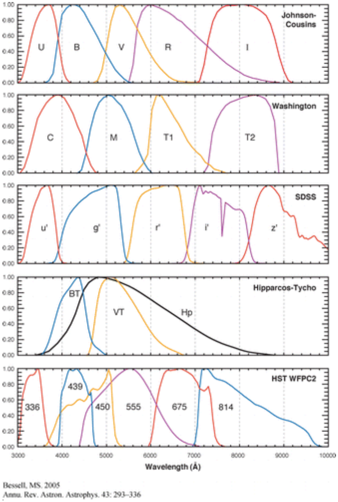
Spectrum: blueprint of an object (more or less).
Magnitudes: what we get most of the time.



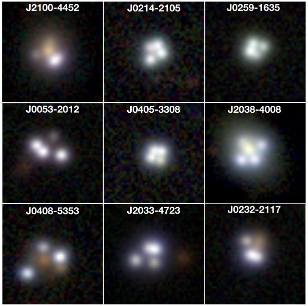


From big data to science: discover, classify, characterise.
NB: light-curve data (left) don't always have the same number of points!

Various magnitude systems for different uses[1].
Each magnitude has a *central wavelength* and a *width*.



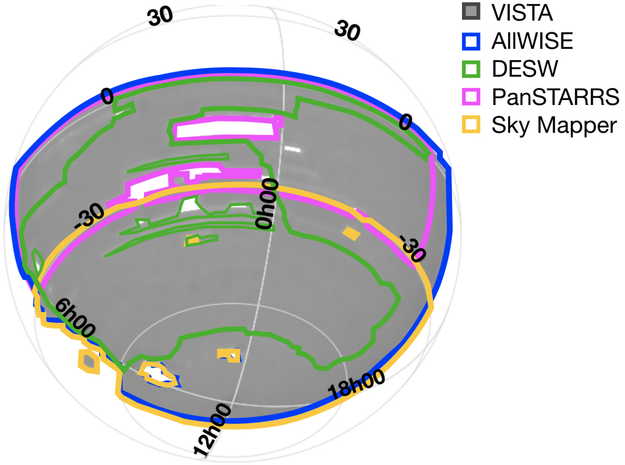
[1] If you're really, really curious: Bessel, M. S. 2005, ARA&A, 43, 293



From big data to science: finding rare objects/events.
(these ones are ***very*** rare)

OK, but where do we begin???

Different experiments//surveys gather different kinds of info.
We "just" need to grab it...



Telescope//experiment
(pipelines) $\mapsto$ data, various formats
(database) $\mapsto$ catalog tables

ID	ra	dec	class	subClass	z	zerr	b	lnLStar_i	umag	gmag	rmag	imag	zmag	W1	W2	psfgmag	psfrmag	psfima
1237678661968265435	16.878845	5.0594924	GALAXY	STARBURST	0.274081	1.216552E-05	-57.5784294922373	-1.180048	20.70									
1237678623308578947	17.145415	5.2240461	GALAXY	null	0.2789114	0.0001069236	-57.3832334335941	-351.6216	24.92443									
1237678623308644624	17.274179	5.1563299	GALAXY	null	0.2926412	3.324989E-05	-57.4343467194447	-2043.575	19.76747									
1237678622771773628	17.297309	4.7099285	GALAXY	null	0.2964781	6.580158E-05	-57.874095028543	-808.24	21.82316	20								
1237678661968396491	17.230792	4.9492185	GALAXY	null	0.3512432	5.516663E-05	-57.6452654554808	-239.4434	21.51004									
1237678661968331270	17.112495	4.895233	GALAXY	null	0.4003306	8.685785E-05	-57.7135771029507	-261.3694	20.7531	20								
1237678661968265425	16.932224	4.9781829	GALAXY	AGN	0.2789478	3.638213E-05	-57.6529419841515	-724.0213	24.58225	19								
1237678622771642595	16.988994	4.8418012	GALAXY	null	0.2760086	3.284616E-05	-57.7816144436961	-2073.651	21.62795									
1237678661968265378	16.918275	5.009444	GALAXY	null	0.2581182	3.791953E-05	-57.6235372847634	-1666.302	20.24423									
1237678622771642499	16.963018	4.7222274	QSO	BROADLINE	0.8016306	5.767976E-05	-57.9034313364451	-33.67056	19.65396									
1237678622771707950	17.151402	4.8186359	QSO	BROADLINE	0.634265	0.0001009258	-57.7847690821629	-5.105304	18.50608									
1237669702124241089	15.152699	7.2441582	QSO	BROADLINE	0.9044501	0.0001920112	-55.555840494391	-1.245545	19.21545									
1237669702124109952	14.874624	7.3149651	QSO	BROADLINE	2.622337	0.0002858732	-55.5014634997012	-4.972133	20.59216									
1237669702124175820	15.049748	7.194844	GALAXY	null	0.4973788	0.0003538404	-55.6113432040637	-11.59372	21.03806									

Queries

Sometimes you can do a bulk download of a catalog table, sometimes it’s unfeasible or unnecessary.

SQL: Structured Query Language. Basic syntax:
SELECT {fields} FROM {table} WHERE {conditions}

```
SELECT TOP 100
       objID, ra ,dec
FROM
       PhotoPrimary
WHERE
       ra > 185 and ra < 185.1
       AND dec > 15 and dec < 15.1
```

Queries

Sometimes you can do a bulk download of a catalog table, sometimes it's unfeasible or unnecessary.
SQL: Structured Query Language. Basic syntax:
SELECT {fields} FROM {table} WHERE {conditions}

```
SELECT TOP 100
      objID, ra ,dec
FROM
      PhotoPrimary
WHERE
      ra > 185 and ra < 185.1
      AND dec > 15 and dec < 15.1
```

Slightly more complicated:

```
SELECT D.coadd_object_id, W.cntr, D.alphawin_j2000 as
desra, D.deltawin_j2000 as desdec, D.mag_auto_i,
W.w1mpro, W.w2mpro
FROM des_dr1.main AS D
JOIN des_dr1.des_allwise AS W on
W.coadd_object_id=D.coadd_object_id
WHERE ( D.galactic_b<-20.0 AND D.mag_auto_i>8.0 AND
D.deltawin_j2000>-60.0 AND D.deltawin_j2000<-55.0 )
```

Q: how many differences can you spot with the simplest query?

Many examples here:
`http://skyserver.sdss.org/dr8/en/help/docs/realquery.asp`
Quote of the day:
"Most of the AI you may need is an SQL `SELECT` followed by an `ORDER BY` clause"

Slightly more complicated:

```
SELECT D.coadd_object_id, W.cntr, D.alphawin_j2000 as
desra, D.deltawin_j2000 as desdec, D.mag_auto_i,
W.w1mpro, W.w2mpro
FROM des_dr1.main AS D
JOIN des_dr1.des_allwise AS W on
W.coadd_object_id=D.coadd_object_id
WHERE ( D.galactic_b<-20.0 AND D.mag_auto_i>8.0 AND
D.deltawin_j2000>-60.0 AND D.deltawin_j2000<-55.0 )
```

Q: how many differences can you spot with the simplest query?

Many examples here:
http://skyserver.sdss.org/dr8/en/help/docs/realquery.asp
Quote of the day:
"Most of the AI you may need is an SQL `SELECT` followed by an `ORDER BY` clause"

Exercise

To familiarise with it a bit: Let's have a look at the *SDSS*

- Have a look at the *Schema Browser* for the PhotoObj and SpecPhoto tables.
- Query coordinates (`ra`, `dec`) and PSF magnitudes in $u-$, $g-$, $r-$, $i-$, $z-$bands, plus spectroscopic redshift, for ten thousand object with `CLASS=='QSO'`, ten thousand with `CLASS=='GALAXY'`, ten thousand with `CLASS=='STAR'`. You can use the web query page *here* .[2]
- **Q:** how well can you fit the redshift using only the magnitudes above? How well can you fit the class, given only the magnitudes?
- Repeat but also using magnitudes `w1mpro` and `w2mpro` from AllWISE.

Various examples of SDSS queries *here*

[2] To query and save heavier stuff, have a loot at *CasJobs*!

Can't we do it in python?

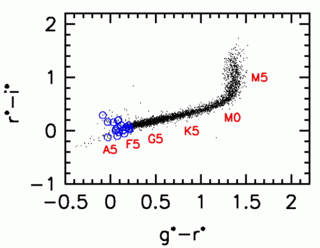
• For access to SQL servers, you can use `sqlite` (ask Carl!).

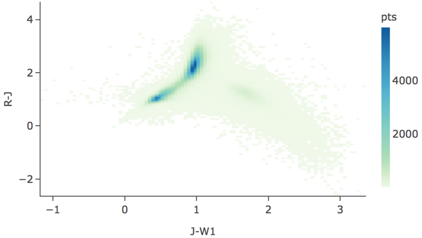
• For astronomical surveys, you can use `astroqueries`.
Some examples given in **ExampleQueries.txt** , courtesy of Zoe Ansari and Sofie H. Bruun (DARK-NBI).

Part 2: handling

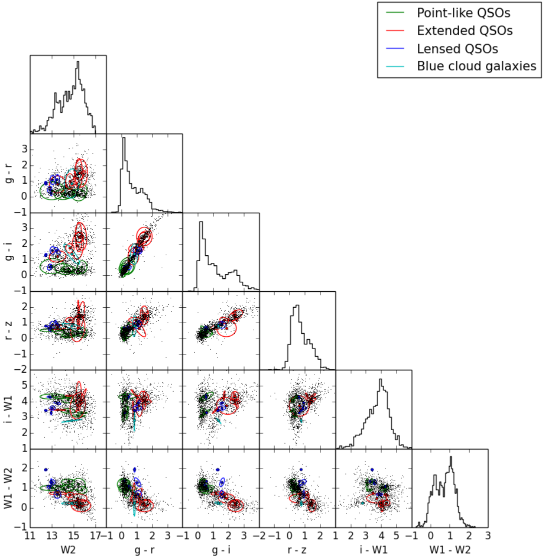
OK, I have my table: now what?

- First things first: look at it!
Do the entries make sense? Are there any missing entries? Are some lines redundant?
- Second: plot familiar (and unfamiliar) stuff.





Easy things first: plot feature vs feature:



Python tips and tricks: you should do it yourselves, but someone has already done it for you...

1. Pair plots (with `seaborn`)
`https://seaborn.pydata.org/generated/seaborn.pairplot.html`

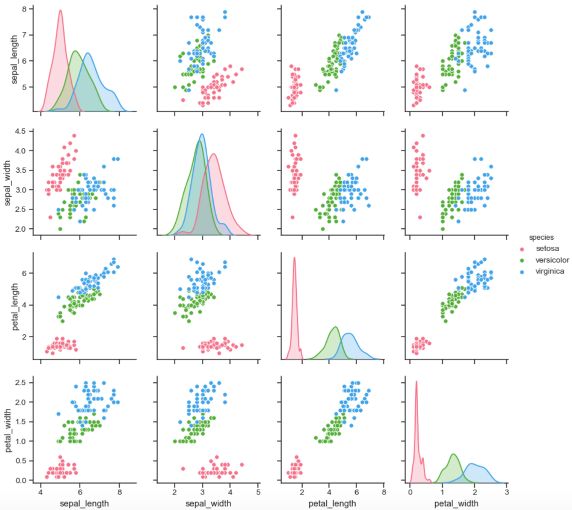
```
import seaborn as sns; sns.set(style="ticks", color_codes=True)
iris = sns.load_dataset("iris")
g = sns.pairplot(iris, hue="species", palette="husl")
```

2. Corner plots (with `corner`)
`https://corner.readthedocs.io/en/latest/pages/quickstart.html`

```
import corner
fig = corner.corner(samples, labels=["$m$", "$b$", "$\ln\,f$"])
fig.show()
```

seaborn 0.9.0 Gallery Tutorial API Site Page

```
>>> g = sns.pairplot(iris, hue="species", palette="husl")
```



But how do I decide which features are important?
Should I plot all of them?!

What if I'm dealing with collections of pictures instead of tables with some columns?

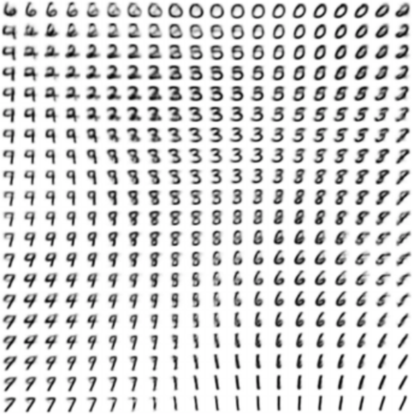
Common issue, 1: the dataset may be easier to crunch in a different coordinate system.
Common issue, 2: are there any combinations of features that maximize information?

But how do I decide which features are important?
Should I plot all of them?!

What if I'm dealing with collections of pictures instead of tables with some columns?

Common issue, 1: the dataset may be easier to crunch in a different coordinate system.
Common issue, 2: are there any combinations of features that maximize information?

Sometimes you ***don't*** need hundreds of features:



This is actually done with something more advanced (Kingma & Welling 2014), but still...

Linear: Principal Component Analysys (PCA)

The maths: we want to transform our feature vectors $\{\mathbf{x}_i \in \mathbb{R}^p\}_{i=1,\ldots,N}$ into others $\{\mathbf{f}_i \in \mathbb{R}^p\}_{i=1,\ldots,N}$ that are uncorrelated.
How to? Find eigenvectors of the covariance matrix:

$$C_{k,l} = \frac{1}{N}\sum_{i=1}^{N} x_{i,k}x_{i,l} \tag{1}$$

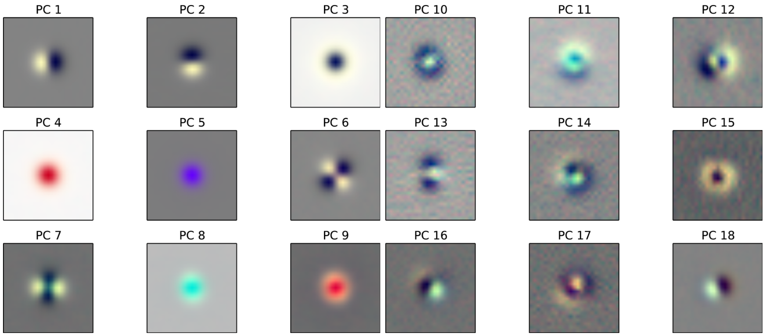
$$\mathbf{C}\,\mathbf{v}_k = \lambda_k \mathbf{v}_k \tag{2}$$

The eigenvectors are the *principal components*.
Fraction of explained variance:

$$\mathrm{var}_{(r)} := \frac{\sum_{k=1}^{r}\lambda_k}{\sum_{k=1}^{p}\lambda_k} \tag{3}$$

NB do you need to standardize your dataset?

Example on (simple stuff) images:[3]



[3] That's from an old paper of mine, you don't really need to know about it.

Example (from scikit-learn):[4]

```
>>> import numpy as np
>>> from sklearn.decomposition import PCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> pca = PCA(n_components=2)
>>> pca.fit(X)
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
>>> print(pca.explained_variance_ratio_)
[0.9924... 0.0075...]
>>> print(pca.singular_values_)
[6.30061... 0.54980...]
```

Methods

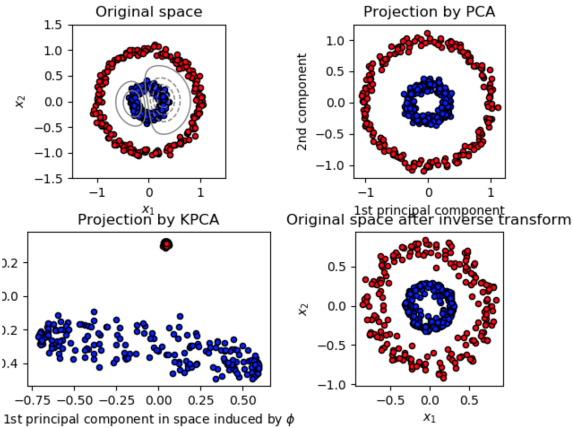
`fit` (X[, y])	Fit the model with X.
`fit_transform` (X[, y])	Fit the model with X and apply the dimensionality reduction on X.
`get_covariance` ()	Compute data covariance with the generative model.
`get_params` ([deep])	Get parameters for this estimator.
`get_precision` ()	Compute data precision matrix with the generative model.
`inverse_transform` (X)	Transform data back to its original space.
`score` (X[, y])	Return the average log-likelihood of all samples.
`score_samples` (X)	Return the log-likelihood of each sample.
`set_params` (**params)	Set the parameters of this estimator.
`transform` (X)	Apply dimensionality reduction to X.

Q: Run a PCA on the quark data table, see where the ‘1’ and ‘0’ subsamples lie.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Bonus track: kPCA

How it works:[5]



[5]You can find code for this example on the scikit-learn website.

How the 'kernel trick' works: map feature space $\Phi : \mathbb{R}^p \mapsto \mathcal{H}$ to very-high-dimensional space with its own scalar product $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$. Diagonalize a *big* matrix

$$K_{i,j} = (1/N) k(\mathbf{x}_i, \mathbf{x}_j) \tag{4}$$
$$K\mathbf{a} = \lambda \mathbf{a} \tag{5}$$

Then the components of a given feature vector $\Phi(\mathbf{f})$ in this space, relative to $r-$th component, are

$$t_r = \langle \mathbf{a}_r, \Phi(\mathbf{f}) \rangle \ = \sum_{i=1}^{N} a_{r,i} k(\mathbf{x}_i, \mathbf{f}) \tag{6}$$

Theorem: everything exists if $k(\bullet, \bullet)$ is semi-positive definite.
Q: Run a (k)PCA on the b-quark data table, try to separate the jets.
Q: Run a (k)PCA on the SDSS data table, try to separate the classes.

Summary

So to sum it up:

1. data are ugly.
2. know where your data come from!
3. inspect your data tables, plot stuff.
4. one method does not necessarily fit every purpose.
5. there is already technology to parse tables, if needed (SQL and thereabouts).
6. datasets can be very-high-dimensional
7. Linear: PCA; non-linear: kPCA (and tSNE, and UMAP...)

Summary

So to sum it up:

1. data are ugly.
2. know where your data come from!
3. inspect your data tables, plot stuff.
4. one method does not necessarily fit every purpose.
5. there is already technology to parse tables, if needed (SQL and thereabouts).
6. datasets can be very-high-dimensional
7. Linear: PCA; non-linear: kPCA (and tSNE, and UMAP...)

Summary

So to sum it up:

1. data are ugly.
2. know where your data come from!
3. inspect your data tables, plot stuff.
4. one method does not necessarily fit every purpose.
5. there is already technology to parse tables, if needed (SQL and thereabouts).
6. datasets can be very-high-dimensional
7. Linear: PCA; non-linear: kPCA (and tSNE, and UMAP...)

Summary

So to sum it up:

1. data are ugly.
2. know where your data come from!
3. inspect your data tables, plot stuff.
4. one method does not necessarily fit every purpose.
5. there is already technology to parse tables, if needed (SQL and thereabouts).
6. datasets can be very-high-dimensional
7. Linear: PCA; non-linear: kPCA (and tSNE, and UMAP...)

Summary

So to sum it up:

1. data are ugly.
2. know where your data come from!
3. inspect your data tables, plot stuff.
4. one method does not necessarily fit every purpose.
5. there is already technology to parse tables, if needed (SQL and thereabouts).
6. datasets can be very-high-dimensional
7. Linear: PCA; non-linear: kPCA (and tSNE, and UMAP...)

Summary

So to sum it up:

1. data are ugly.
2. know where your data come from!
3. inspect your data tables, plot stuff.
4. one method does not necessarily fit every purpose.
5. there is already technology to parse tables, if needed (SQL and thereabouts).
6. datasets can be very-high-dimensional
7. Linear: PCA; non-linear: kPCA (and tSNE, and UMAP...)