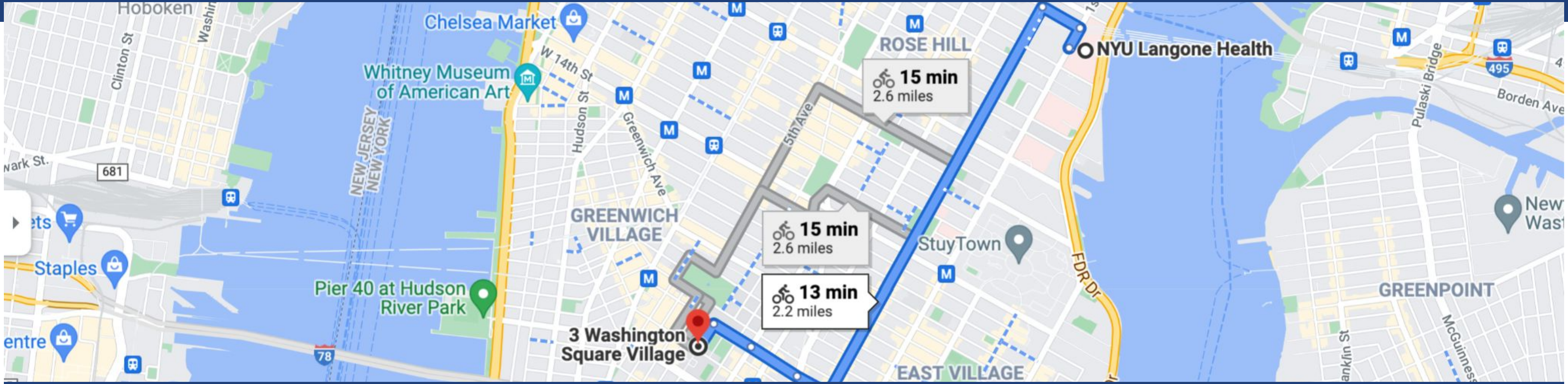


Distance metrics

Continuous variables

Minkowski family of distances

$$D(i,j) = \sqrt[1/p]{\sum_{k=1}^{N} |x_{ik} - x_{jk}|^p}$$

N features (dimensions)

properties

$$D(i,j) > 0$$

$$D(i,i) = 0$$

$$D(i,j) = D(j,i)$$

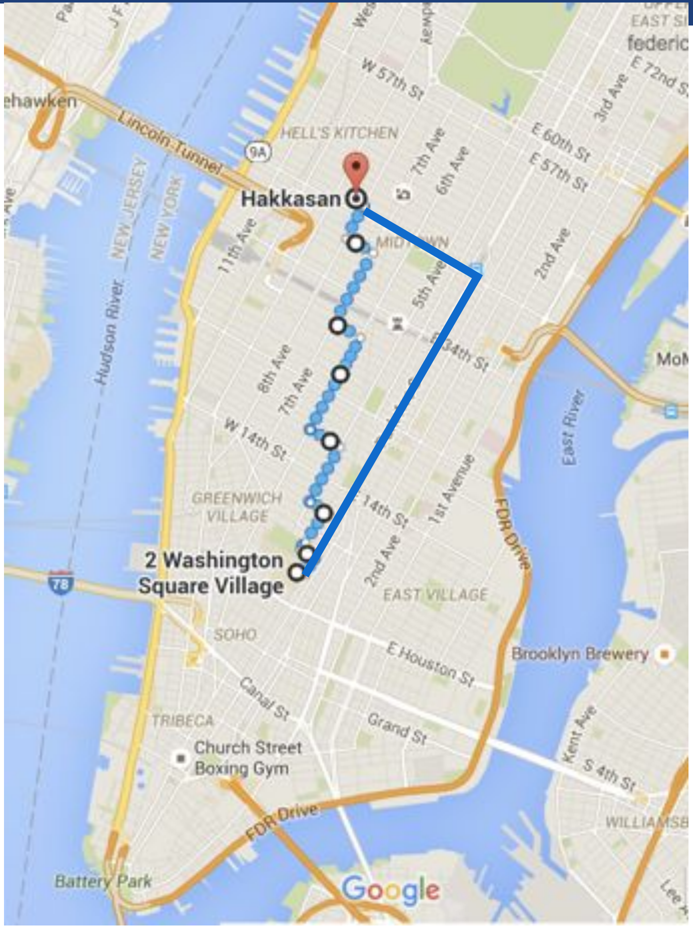
$$D(i,j) <= D(i,k) + D(k,j)$$

Continuous variables

Minkowski family of distances

Manhattan: p=1

$$D_{Man}(i,j) = \sum_{k=1}^{N} |x_{ik} - x_{jk}|$$

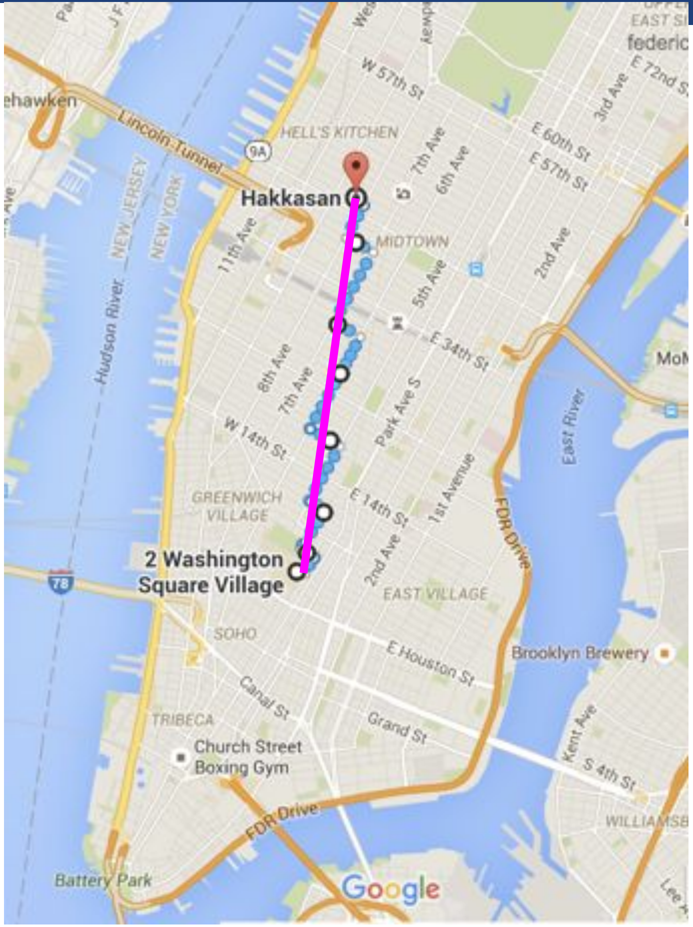


Continuous variables

Minkowski family of distances

Euclidean: p=2

$$D_{Euc}(i,j) = \sqrt{\sum_{k=1}^{N} |x_{ik} - x_{jk}|^2}$$



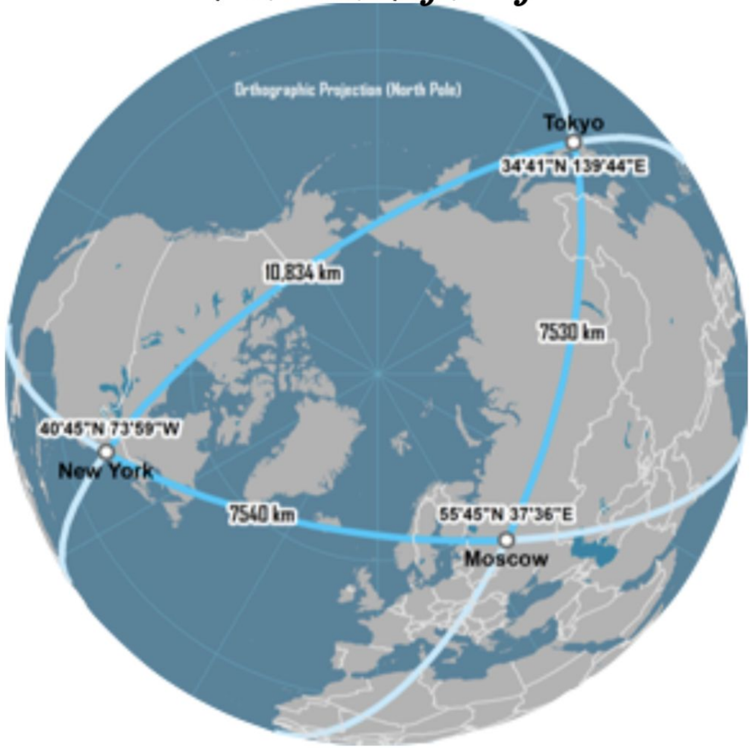
Continuous variables

Great Circle distance

features

latitude and longitude

$\phi_i, \lambda_i, \phi_j, \lambda_j$



$$D(i,j) = R \arccos\left(\sin\phi_i \cdot \sin\phi_j + \cos\phi_i \cdot \cos\phi_j \cdot \cos\Delta\lambda\right)$$

Categorical variables: binary

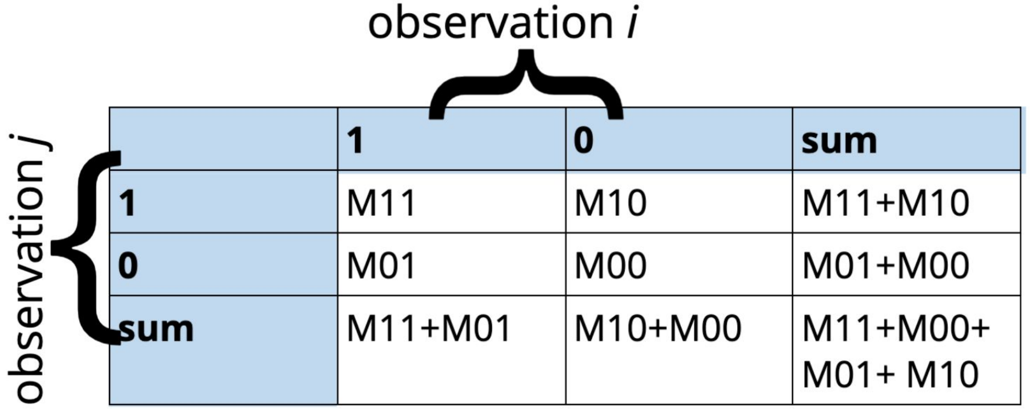
Uses presence/absence of features in data

$M_{i=0,j=0}$: number of features in neither

$M_{i=1,j=1}$: number of features in both

$M_{i=1,j=0}$: number of features in *i* but not *j*

$M_{i=0,j=1}$: number of features in *j* but not *i*



observation *i*

observation *j*

	1	**0**	**sum**
1	M11	M10	M11+M10
0	M01	M00	M01+M00
sum	M11+M01	M10+M00	M11+M00+ M01+ M10

Categorical variables: binary

Uses presence/absence of features in data

Simple Matching Coefficient or Rand similarity

$$SMC(i,j) = \frac{M_{i=0,j=0}+M_{i=1,j=1}}{M_{i=0,j=0}+M_{i=1,j=0}+M_{i=0,j=1}+M_{i=1,j=1}}$$

observation *i*

observation *j*

	1	**0**	**sum**
1	M11	M10	M11+M10
0	M01	M00	M01+M00
sum	M11+M01	M10+M00	M11+M00+ M01+ M10

Simple Matching Distance

$$SMD(i,j) = 1 - SMC(i,j)$$

$M_{i=0,j=0}$	: number of features in neither
$M_{i=1,j=1}$	: number of features in both
$M_{i=1,j=0}$	: number of features in *i* but not *j*
$M_{i=0,j=1}$	: number of features in *j* but not *i*

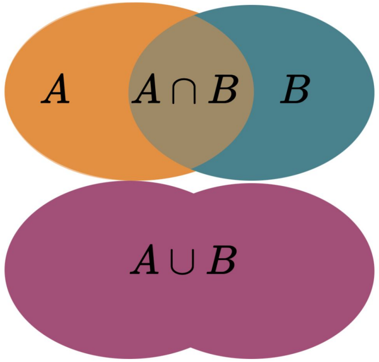
Categorical variables: binary

Jaccard similarity

$$J(i,j) = \frac{M_{i=1,j=1}}{M_{i=0,j=1}+M_{i=1,j=0}+M_{i=0,j=0}}$$

Jaccard distance

$$D(i,j) = 1 - J(i,j)$$



observation *i*

observation *j*

	1	0	sum
1	M11	M10	M11+M10
0	M01	M00	M01+M00
sum	M11+M01	M10+M00	M11+M00+ M01+ M10

$M_{i=0,j=0}$: number of features in neither

$M_{i=1,j=1}$: number of features in both

$M_{i=1,j=0}$: number of features in *i* but not *j*

$M_{i=0,j=1}$: number of features in *j* but not *i*

ETH zürich

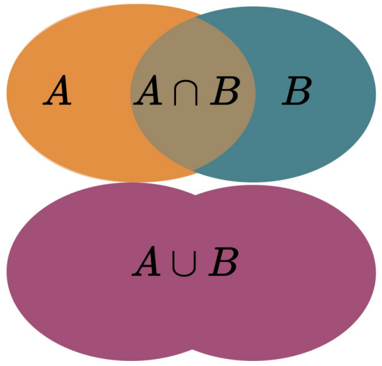
Categorical variables: binary

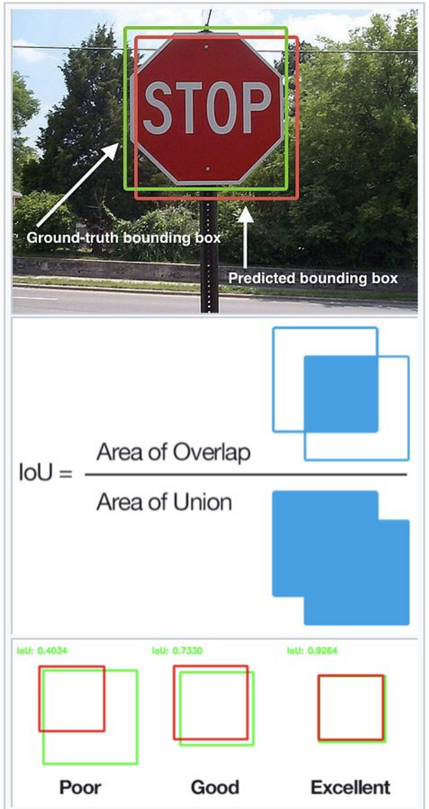
Jaccard index

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Application to Deep Learning for image recognition

Convolutional Neural Nets

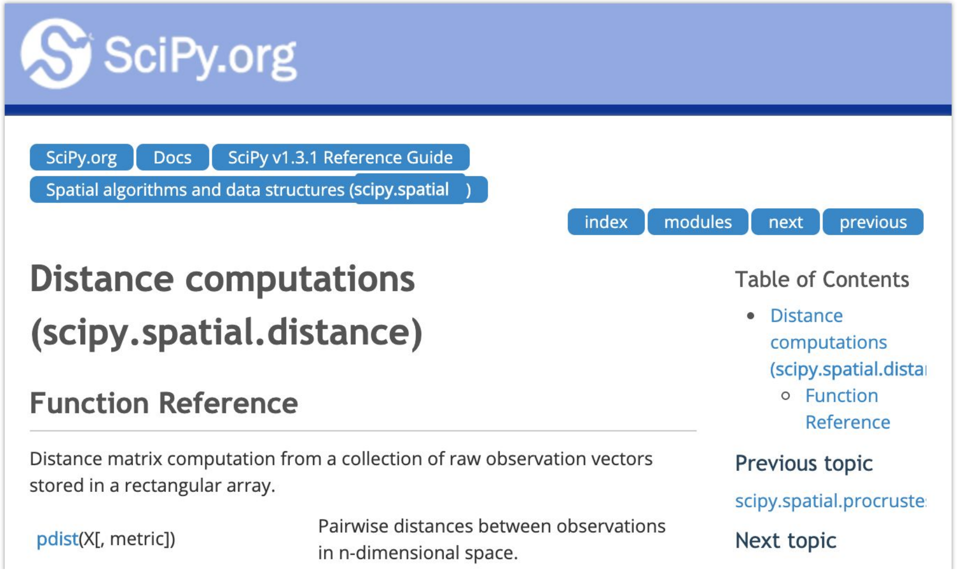




Intersection over Union as a similarity measure for object detection on images - an important task in computer vision.

ETH zürich

Another useful package for scientific Python: SciPy


SciPy.org
SciPy.org
Docs
SciPy v1.3.1 Reference Guide
Spatial algorithms and data structures (scipy.spatial)
index
modules
next
previous

Distance computations (scipy.spatial.distance)

Function Reference

Distance matrix computation from a collection of raw observation vectors stored in a rectangular array.

pdist(X[, metric])	Pairwise distances between observations in n-dimensional space.

Table of Contents

Previous topic

scipy.spatial.procruste

Next topic

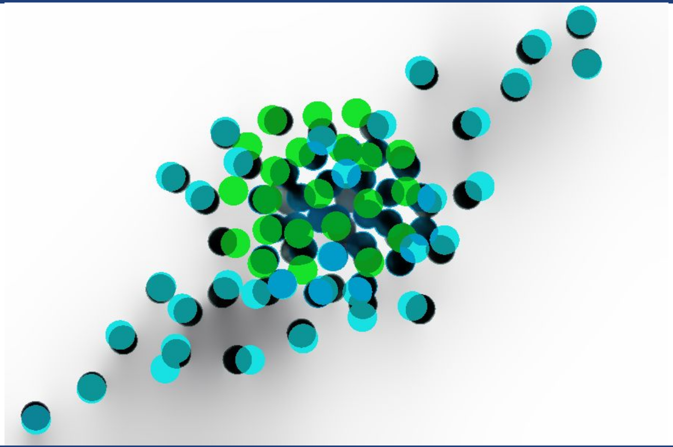
https://docs.scipy.org/doc/scipy/reference/spatial.distance.html

Distance functions between two boolean vectors (representing sets) `u` and `v`. As in the case of numerical vectors, `pdist` is more efficient for computing the distances between all pairs.

dice(u, v[, w])	Compute the Dice dissimilarity between two boolean 1-D arrays.
hamming(u, v[, w])	Compute the Hamming distance between two 1-D arrays.
jaccard(u, v[, w])	Compute the Jaccard-Needham dissimilarity between two boolean 1-D arrays.

Distance functions between two numeric vectors `u` and `v`. Computing distances over a large collection of vectors is inefficient for these functions. Use `pdist` for this purpose.

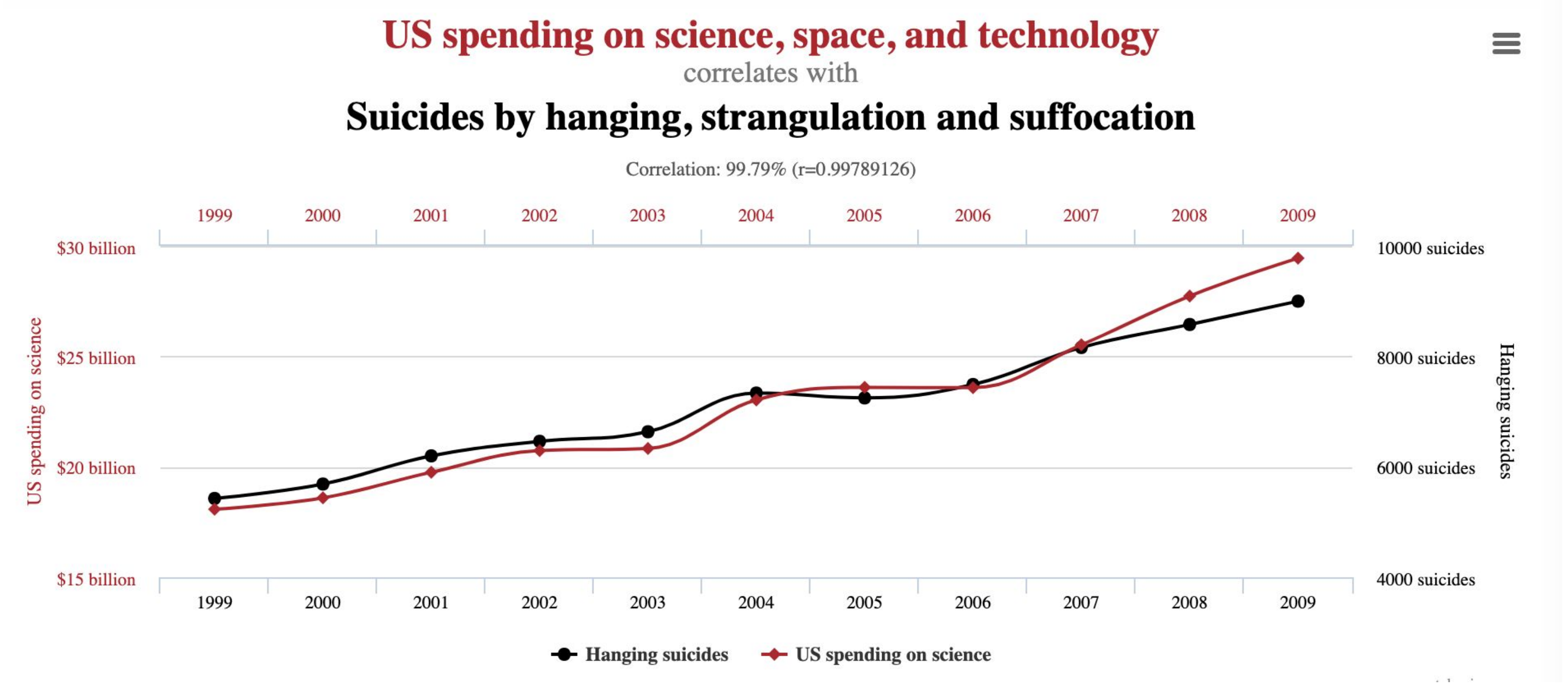
braycurtis(u, v[, w])	Compute the Bray-Curtis distance between two 1-D arrays.
canberra(u, v[, w])	Compute the Canberra distance between two 1-D arrays.
chebyshev(u, v[, w])	Compute the Chebyshev distance.
cityblock(u, v[, w])	Compute the City Block (Manhattan) distance.



Data whitening

ETH zürich

Data can have covariance (and it almost always does!)

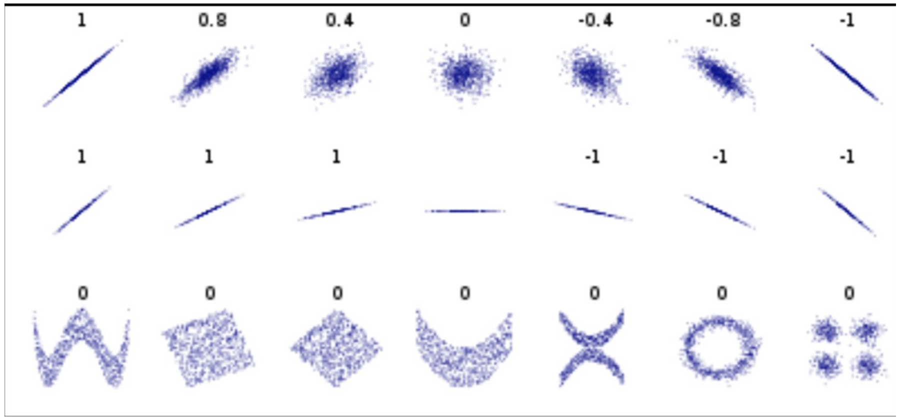


https://www.tylervigen.com/spurious-correlations

Data can have covariance (and it almost always does!)

Pearson's correlation (linear correlation)

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$



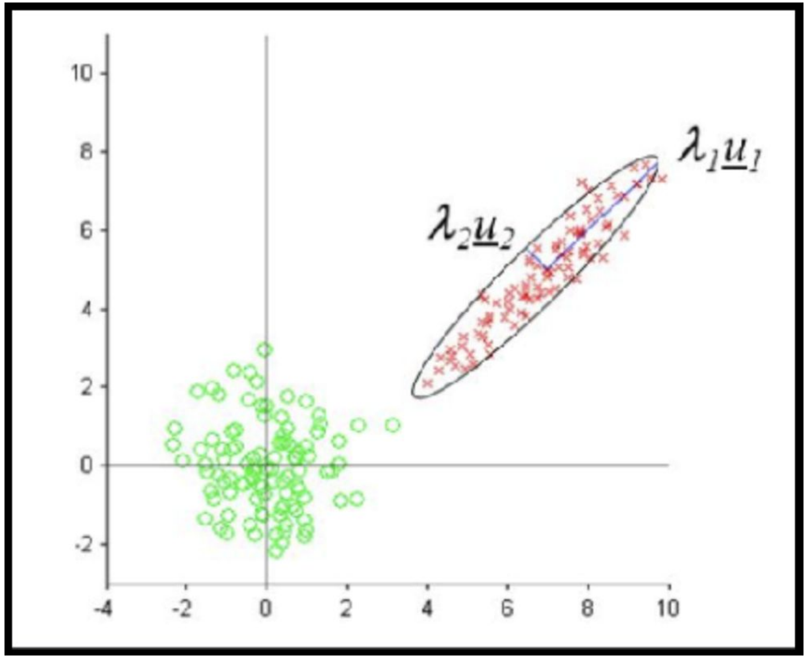
Pearson's correlation = covariance/ product of the two standard deviations

Generic preprocessing

Original data

Standardized data

Data that is not correlated appear as a sphere in the N-dimensional feature space



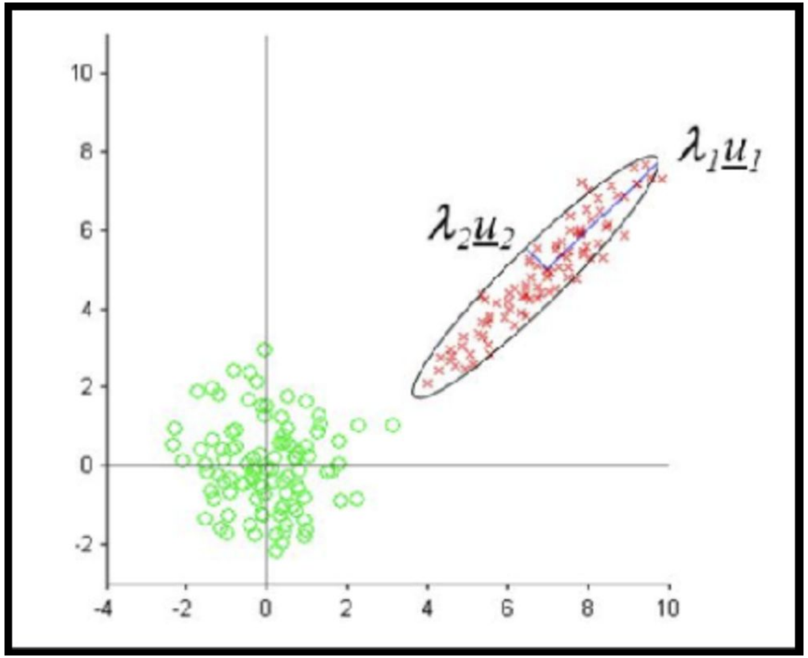
Generic preprocessing

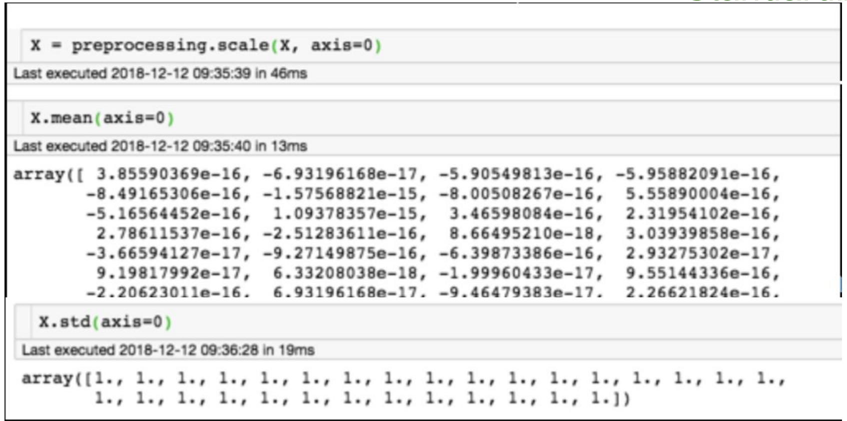
for each feature: divide by standard deviation and subtract mean

mean of each feature should be 0, standard deviation of each feature should be 1

Original data

Standardized data





```
X = preprocessing.scale(X, axis=0)
```
Last executed 2018-12-12 09:35:39 in 46ms

```
X.mean(axis=0)
```
Last executed 2018-12-12 09:35:40 in 13ms

```
array([ 3.85590369e-16, -6.93196168e-17, -5.90549813e-16, -5.95882091e-16,
       -8.49165306e-16, -1.57568821e-15, -8.00508267e-16,  5.55890004e-16,
       -5.16564452e-16,  1.09378357e-15,  3.46598084e-16,  2.31954102e-16,
        2.78611537e-16, -2.51283611e-16,  8.66495210e-18,  3.03939858e-16,
       -3.66594127e-17, -9.27149875e-16, -6.39873386e-16,  2.93275302e-17,
        9.19817992e-17,  6.33208038e-18, -1.99960433e-17,  9.55144336e-16,
       -2.20623011e-16,  6.93196168e-17, -9.46479383e-17,  2.26621824e-16,
```

```
X.std(axis=0)
```
Last executed 2018-12-12 09:36:28 in 19ms

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

ETH zürich

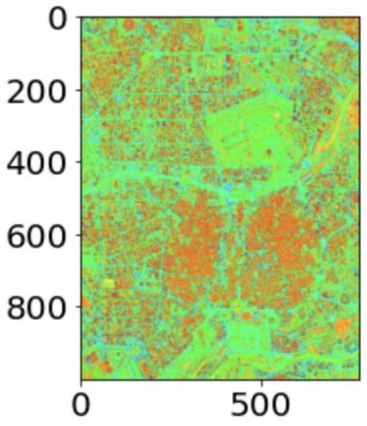
Generic preprocessing

For image processing (e.g. segmentation) often you need to min-max preprocess

```
from sklearn import preprocessing
Xopscaled = preprocessing.minmax_scale(op.reshape(op.shape[1] * op.shape[0], 3).astype(float), axis=1)
Xopscaled.reshape(op.shape)[200, 700]
```

```
pl.imshow(Xopscaled.reshape(op.shape));
```

```
Clipping input data to the valid range for imshow with RGB data
```



Clustering

Clustering is an unsupervised learning method

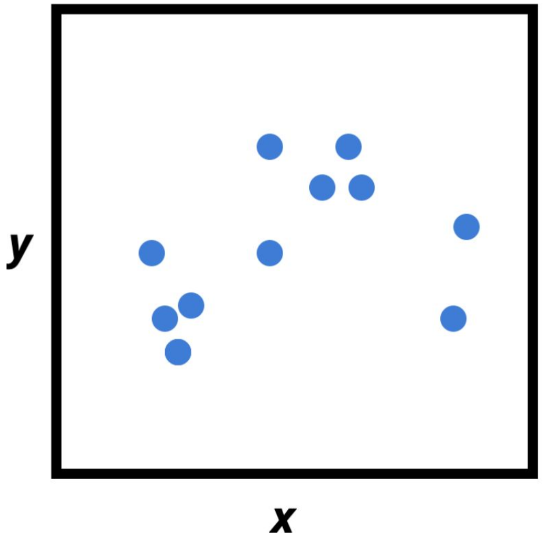
GOAL: partitioning data in maximally homogeneous,

maximally distinguished subsets.

observed

features:

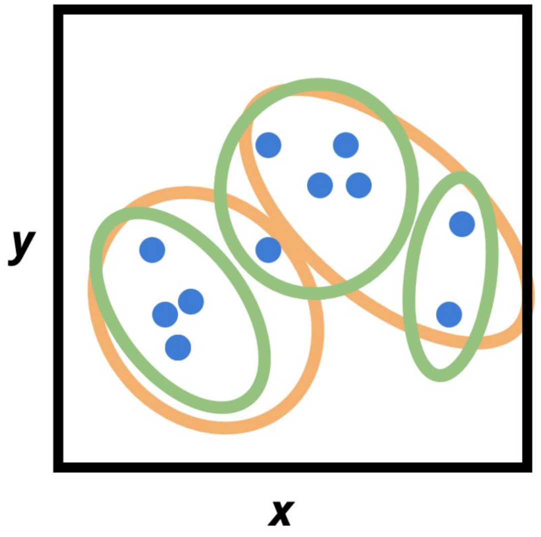
(x, y)



Clustering is an unsupervised learning method

all features are observed for all objects in the sample

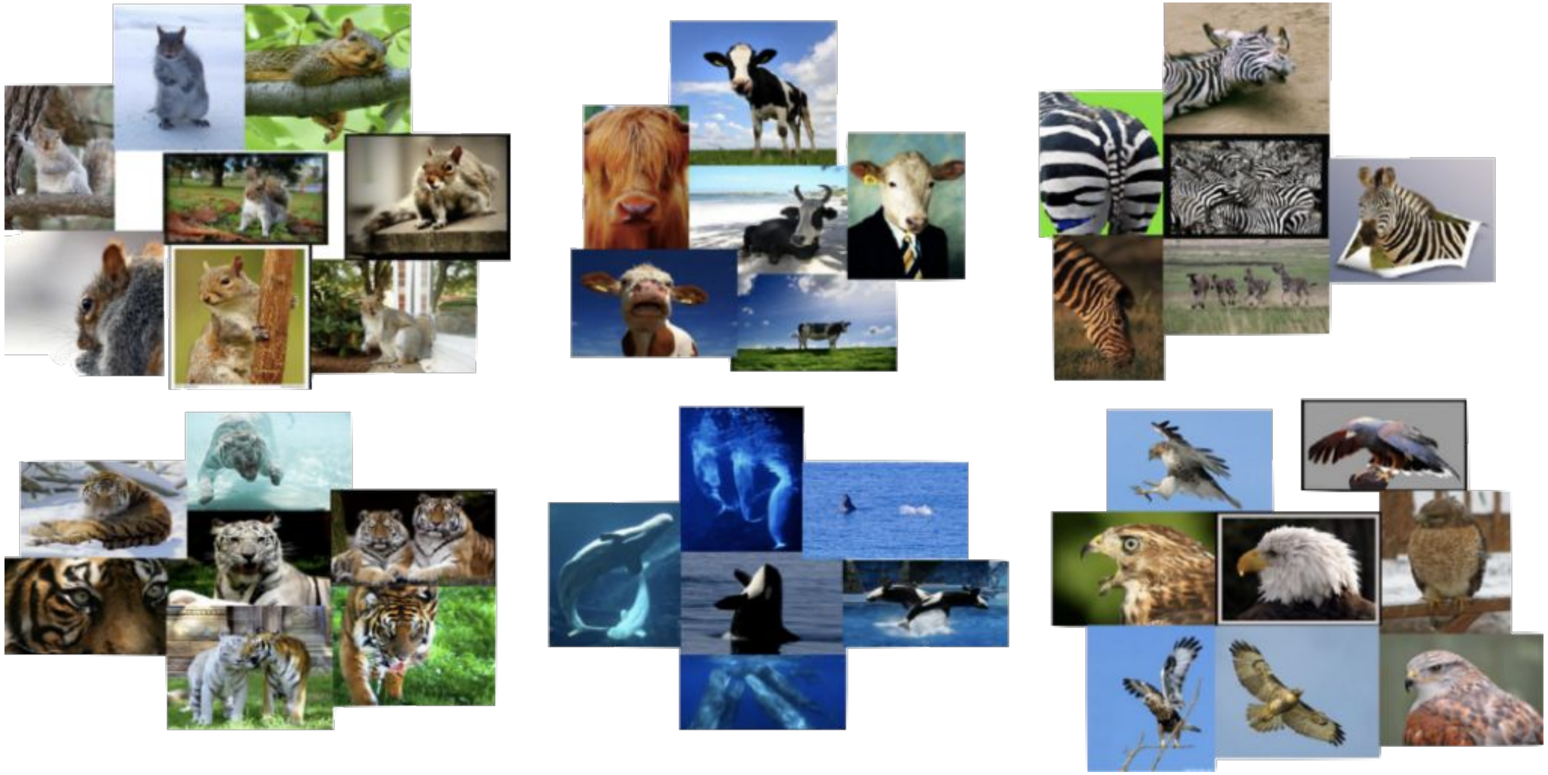
(x, y)



how should I group the observations in this feature space?

e.g.: how many groups should I make?

Optimal clustering



http://www-bcf.usc.edu/~soltanol/Applications.html

internal criterion:

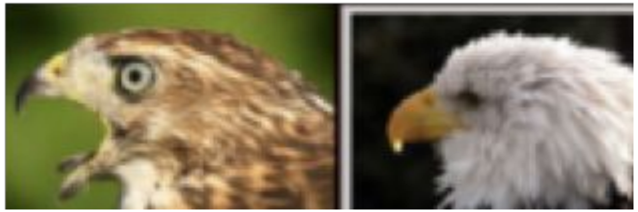
members of the cluster should be similar to each other (intra cluster compactness)

external criterion:

objects outside the cluster should be dissimilar from the objects inside the cluster

Tigers

Whales


raptors

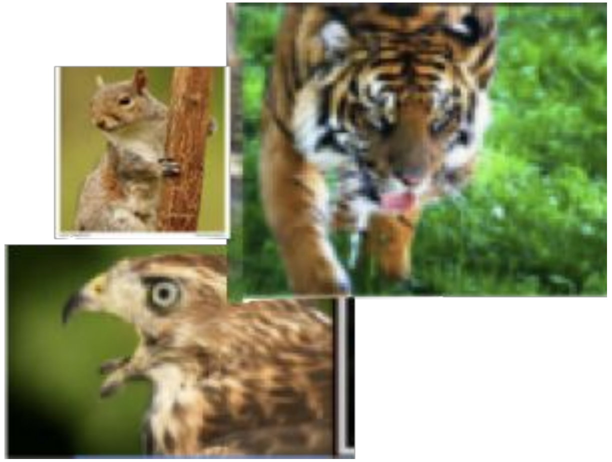
zoologist's clusters

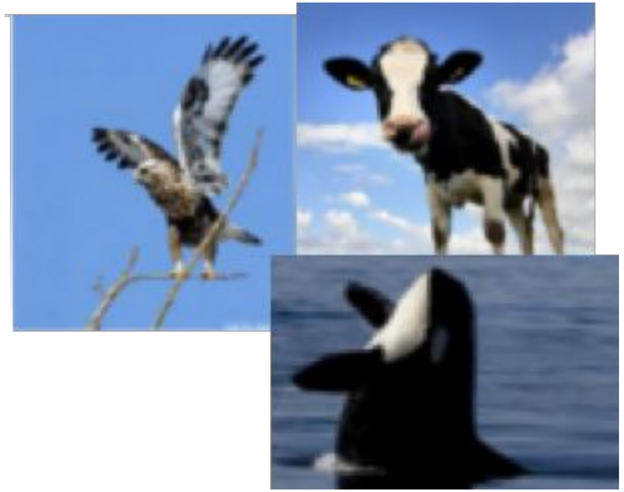
internal criterion:

members of the cluster should be similar to each other (intra cluster compactness)

external criterion:

objects outside the cluster should be dissimilar from the objects inside the cluster





yellow/green

photographer's clusters

black/white/blue

Optimal clustering

the optimal clustering depends on:

- how you define similarity/distance
- the purpose of the clustering

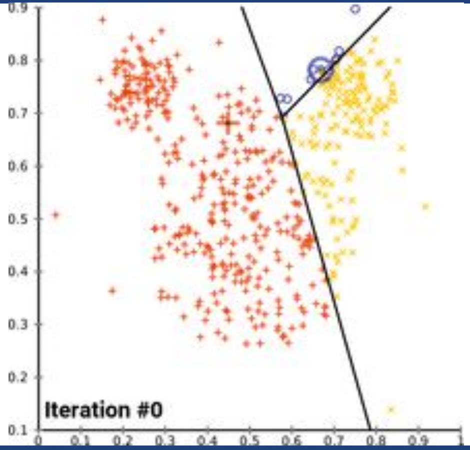
Optimal clustering

The ideal clustering algorithm should have:

- Scalability
- Ability to deal with different types of attributes
- Discovery of clusters with arbitrary shapes
- Minimal requirement for domain knowledge
- Deals with noise and outliers
- Insensitive to order
- Allows incorporation of constraints
- Interpretable

How does clustering work?

- Partitional
  - **Hard clustering:** K-means (McQueen ’67), K-medoids (Kaufman & Rausseeuw ’87)
  - **Soft Clustering:** Expectation Maximization (Dempster,Laird,Rubin ’77)
- Hierarchical
  - Agglomerative (bottom-up)
  - Divisive (top-down)
- also:
  - Density based
  - Grid based
  - Model based



Clustering by partitioning

K-means

Choose N “centers” guesses: random points in the feature space

repeat:

- Calculate distance between each point and each center
- Assign each point to the closest center
- Calculate the new cluster centers

until (convergence):

when clusters no longer change

ETH zürich

K-means

Objective: minimizing the aggregate distance within the cluster.

Order: #clusters #dimensions #iterations #datapoints **O(KdN)**

CONs:

- **Its non-deterministic:** the result depends on the (random) starting point
- **It only works where the mean is defined:** alternative is K-medoids which represents the cluster by its central member (median), rather than by the mean
- **Must declare the number of clusters upfront** (how would you know it?)

PROs:

Scales linearly with d and N

K-means

Objective: minimizing the aggregate distance within the cluster.

Order: #clusters #dimensions #iterations #datapoints **O(KdN)**

O(KdN):

complexity scales linearly with:

- *d* number of dimensions
- *N* number of datapoints
- *K* number of clusters

K-means: the objective function

Objective: minimizing the aggregate distance within the cluster.

total intra-cluster variance $\sum_k \sum_{i\in k}(\vec{x}_i - \vec{\mu}_k)^2$

Order: #clusters #dimensions #iterations #datapoints **O(KdN)**

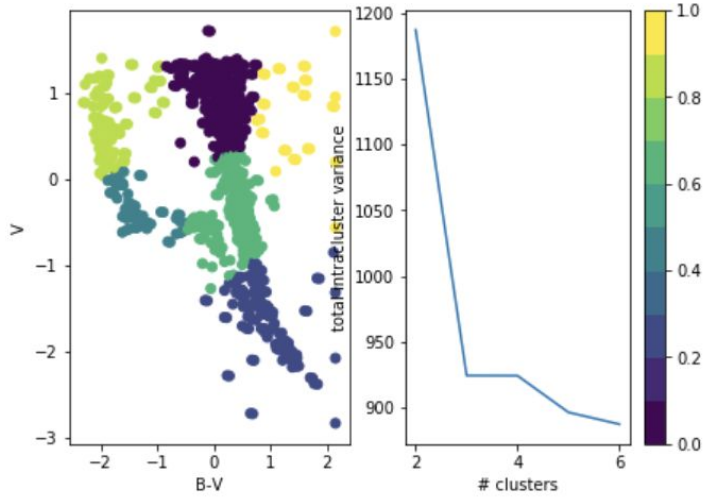
Must declare the number of clusters

either you know it because of domain knowledge

or

you choose it after the fact: "*elbow method*"

```
for i in range(1, nmaxc):
  c = KMeans(n_clusters=i, random_state=302).fit(X2)
  nc.append(c.inertia_)
  print("i.c. variance with {} clusters {:.2f}".format(i, c.inertia_))
pl.plot(range(1, nmaxc), nc)
```



ETH zürich

K-means: the objective function

Objective: minimizing the aggregate distance within the cluster.

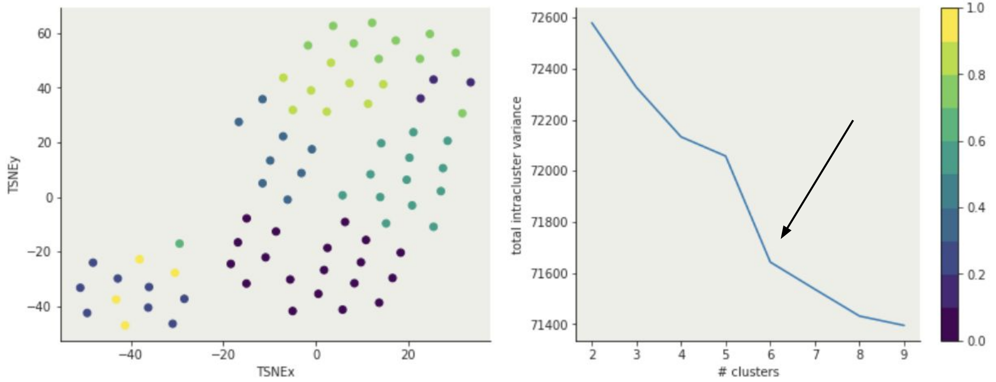
total intra-cluster variance $\sum_k \sum_{i \in k} (\vec{x}_i - \vec{\mu}_k)^2$

Order: #clusters #dimensions #iterations #datapoints **O(KdN)**

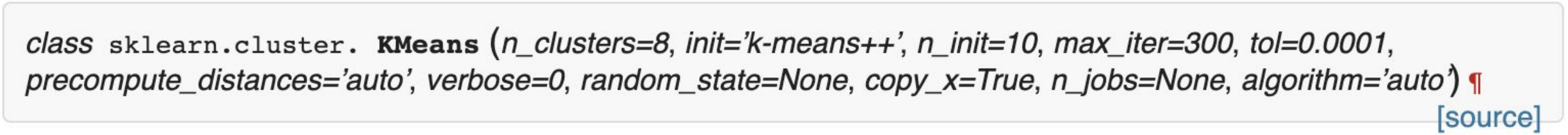
Must declare the number of clusters upfront (how would you know it?)

either domain knowledge or

after the fact: "elbow method"



K-means hyperparameters



```
class sklearn.cluster. KMeans (n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001,
precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=None, algorithm='auto') ¶
[source]
```

- *n_clusters* : number of clusters
- *init* : the initial centers or a scheme to choose the center

  'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.

  'random': choose k observations (rows) at random from data for the initial centroids.

  If an ndarray is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.
- *n_init* : if >1 it is effectively an ensemble method: runs n times with different initializations
- *random_state* : for reproducibility

ETH zürich

Convergence criteria

General

Any time you have an objective function (or loss function) you need to set up a tolerance : if your objective function did not change by more than ε since the last step you have reached convergence (i.e. you are satisfied)

ε is your tolerance

For clustering:

convergence can be reached if

no more than n data point changed cluster

n is your tolerance