Python in a Nutshell

Part IV: Scikits

Manel Velasco,[1] PhD and Alexandre Perera,[1,2] PhD

[1]Departament d'Enginyeria de Sistemes, Automatica i Informatica Industrial (ESAII)
Universitat Politecnica de Catalunya

[2]Centro de Investigacion Biomedica en Red en Bioingenieria, Biomateriales y Nanomedicina (CIBER-BBN)
Alexandre.Perera@upc.edu Manel.Velasco@upc.edu

Introduction to Python for Engineering and Statistics
Febraury, 2013

Contents I

Contents II

Machine Learning

Scikits

- SciKits (short for SciPy Toolkits), are add-on packages for SciPy, hosted and developed separately from the main SciPy distribution.
- The SciKits cover a broad spectrum of application domains, including financial computation, audio processing, geosciences, computer vision, engineering, machine learning, medical computing and bioinformatics.
- All SciKits are available under the 'scikits' namespace and are licensed under OSI-approved licenses.
- Packages are packaged as toolkits (instead of in the main, monolithic SciPy distribution) when:
  1. The package is deemed too specialized to live in SciPy itself or
  2. The package has a GPL (or similar) license which is incompatible with SciPy's BSD license or
  3. The package is meant to be included in SciPy, but development is still in progress.

Scikits

Avaliable kits

The list of available kits can be found at
http://scikits.appspot.com/scikits.

Scikits

Main scikits

scikit-aero	Aeronautical engineering calculations in Python.
scikit-image	Image processing routines for SciPy.
scikit-rf	Object Oriented Microwave Engineering.
audiolab	A python module to make noise from numpy arrays (sic).
timeseries	Time series manipulation.
learn	Machine learning Sci-kit.

pandas

timeseries

The scikits.timeseries module is no longer undergoing active development. There is an outstanding list of bugs that are unlikely to be fixed. The plan is for the core functionality of this module to be implemented in pandas.

pandas

Python Data Analysis Library (pandas)

Python has long been great for data munging and preparation, but less so for data analysis and modeling. *pandas* helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R.

pandas features

pandas features

- A fast and efficient DataFrame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format.
- Intelligent data alignment and integrated handling of missing data: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form.
- High performance merging and joining of data sets.

pandas

```
In [1]: index = date_range('1/1/2000', periods=8)

In [2]: s = Series(randn(5), index=['a', 'b', 'c', 'd', 'e'])

In [3]: df = DataFrame(randn(8, 3), index=index,
   ...:                columns=['A', 'B', 'C'])
   ...:

In [4]: wp = Panel(randn(2, 5, 4), items=['Item1', 'Item2'],
   ...:            major_axis=date_range('1/1/2000', periods=5),
   ...:            minor_axis=['A', 'B', 'C', 'D'])
   ...:
```

pandas

```
In [39]: df
Out[39]:
          one      three         two
a    2.213700        NaN -1.436616
b    0.115002   0.640476 -1.107900
c    0.008462   0.697707  0.844137
d         NaN   0.331810 -0.429735

In [40]: df.mean(0)
Out[40]:
one      0.779055
three    0.556664
two     -0.532528
```

audiolab

audiolab

Is essentially a wrapper around Erik de Castro Lopo's excellent libsndfile:

http://www.mega-nerd.com/libsndfile/

Audiolab is a python package to read/write audio files from numpy arrays. Matlab have functions such as wavread, wavwrite, soundsc, etc. Main features are:

- Reading all formats supported by sndfile directly into numpy arrays.
- Writing all formats supported by sndfile directly from numpy arrays.
- A matlab-like API (e.g. wavread) for some file formats. Wav, aiff, flac, au, and ogg formats are supported.
- A play function to output data from numpy array into the sound device of your computer (Only ALSA for linux and CoreAudio for Mac OS X is implemented ATM).

audiolab overview

```
from audiolab import wavread
data, fs, enc = wavread('test.wav')
import numpy as np
from scikits.audiolab import Sndfile

f = Sndfile('test.wav', 'r')

# Sndfile instances can be queried for the audio file meta-data
fs = f.samplerate
nc = f.channels
enc = f.encoding

# Reading is straightforward
data = f.read_frames(1000)

# This reads the next 1000 frames, e.g. from 1000 to 2000,
# but as single precision
data_float = f.read_frames(1000, dtype=np.float32)
```

audiolab

```
import numpy as np
from scikits.audiolab import Format, Sndfile

filename = 'foo.wav'

# Create some data to save as audio data: one
# second of stereo white noise
data = np.random.randn(48000, 2)

# Create a Sndfile instance for writing wav files @ 48000 Hz
format = Format('wav')
f = Sndfile(filename, ' w', format, 2, 48000)

# Write the first 500 frames of the signal.
# Note that the write_frames method
# uses tmp's numpy dtype to determine how to
# write to the file; sndfile also
# converts the data on the fly if necessary
f.write_frames(data[:500]); f.close()
```

audiolab

```
import numpy as np
from scikits.audiolab import play

# output one second of stereo
#    gaussian white noise at 48000 hz

play(0.05 * np.random.randn(2, 48000))
```

Introduction

Datasets

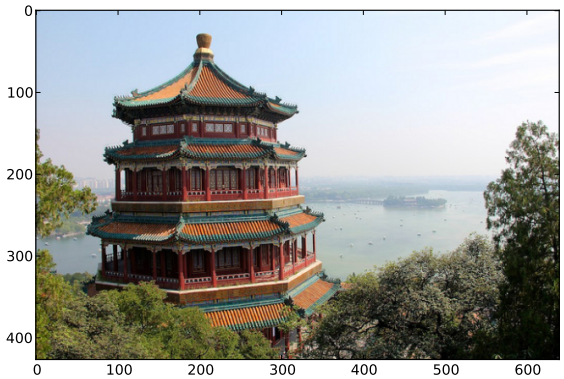
There are three different datasets in sklearn:

1. Sample images.
2. Toy Datasets.
3. Sample Generators.

Image sets

The scikit also embed a couple of sample JPEG images (china and flower) published under Creative Commons license by their authors.

```
import numpy as np
import pylab as pl
from sklearn.datasets import load_sample_ir
china = load_sample_image("china.jpg")
```



Toy datasets

load_boston() Regression Load and return the boston house-prices dataset.

load_iris() Classification Load and return the iris dataset.

load_diabetes() Regression Load and return the diabetes dataset.

load_iris() Classification Load and return the digits dataset.

load_linnerud() Multivariate Regression Load and return the linnerud dataset.

These functions return a *bunch* (which is a dictionary that is accessible with the *dict.key* syntax). All datasets have at least two keys,

- data, containing an array of shape n samples $\times$ n features and
- target, a numpy array of length n features, containing the targets.

Data Generators

A number of functions exists to create the most esoteric data distributions:

make_classification() n-class classification problem (+ multilabel).

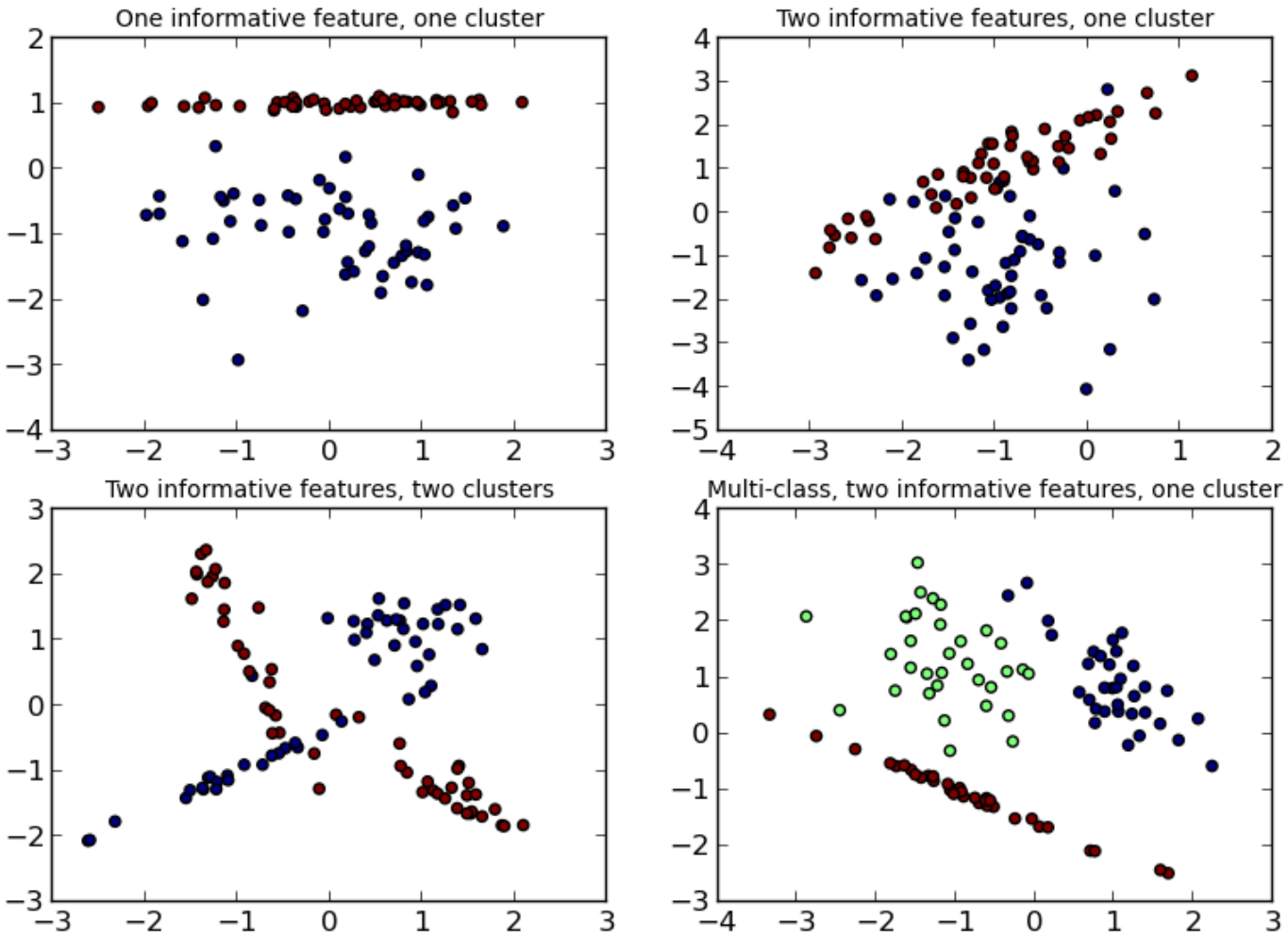
make_regression() Generate a regression problem.

make_swiss_roll() Generate swiss roll datasets.

make_s_curve Generates S curve datasets.

All of them returning a tuple (X, y) consisting of a n samples $\times$ n features numpy array X and an array of length n samples containing the targets y.

Data Generators



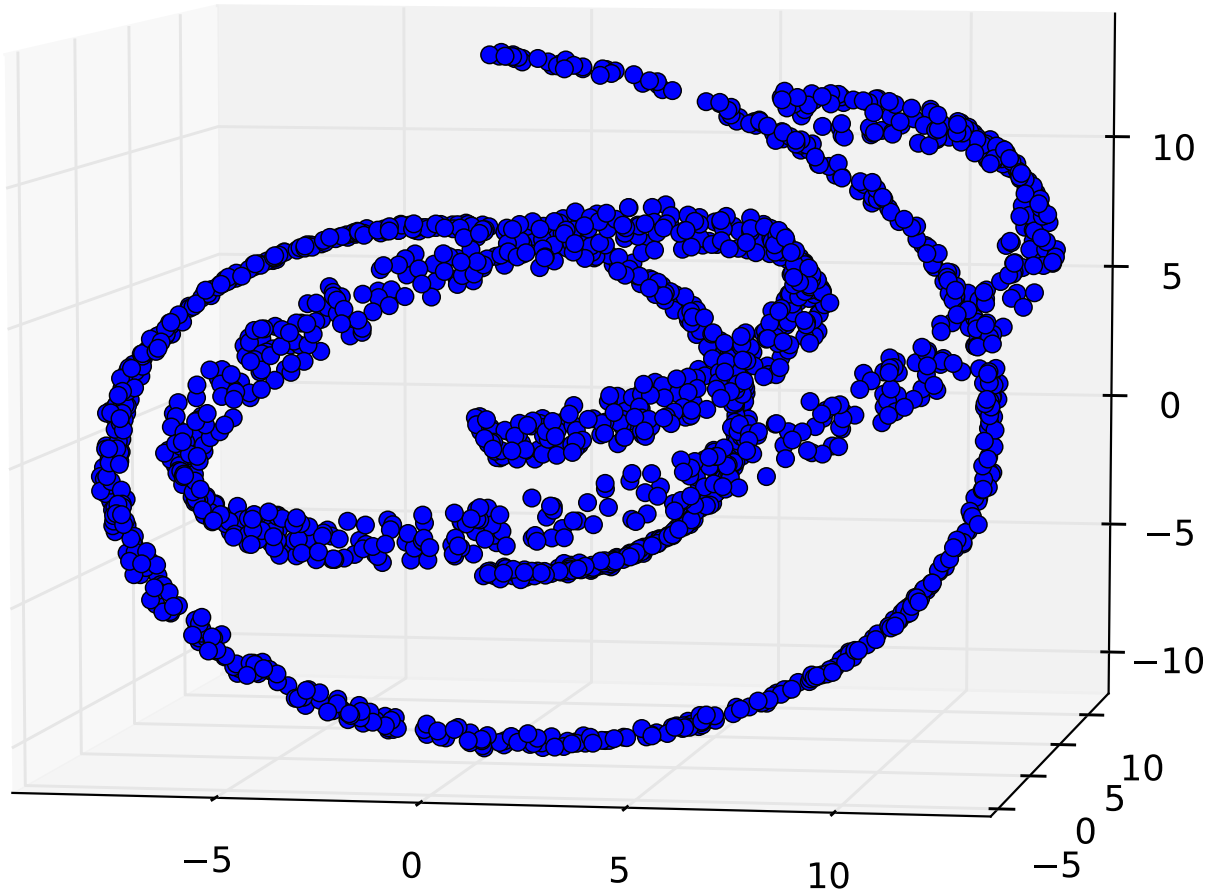
Challenge

5 mins challenge

Generate and plot a dataset with two non-overlapping swiss rolls on $\mathbb{R}^3$ with 1000 samples each.

Tip, you can generate a rotation matrix with help of this function:

```
import numpy as np
def rot(angle, R = np.zeros((3,3))):
    cx, cy, cz = np.cos(angle)
    sx, sy, sz = np.sin(angle)
    R.flat = (cx*cz -sx*cy*sz, cx*sz + sx*cy*cz, sx*sy,\
     -sx*cz - cx*cy*sz, -sx*sz +cx*cy*cz, cx*sy,\
     sy*sz, -sy*cz, cy)
    return R
```


10
5
0
−5
−10
−5
0
5
10
10
5
0
−5

Introduction to clustering in python

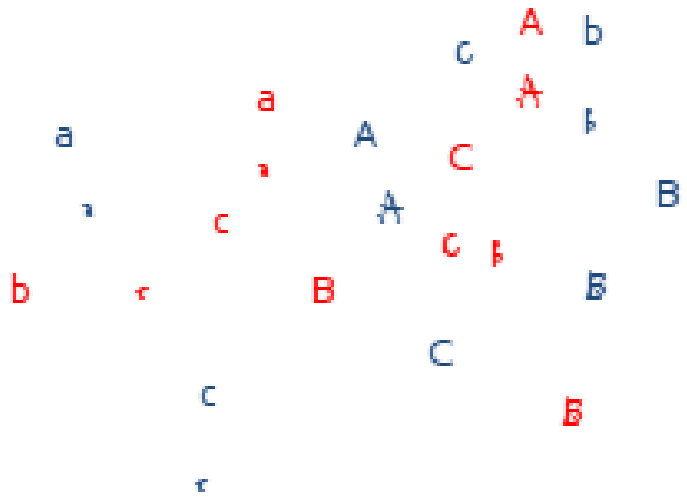
Clustering

Or Cluster Analysis, is the task of grouping a set of objects in such a way that objects in the same groups (clusters) are more similar to each other than to those in other groups.

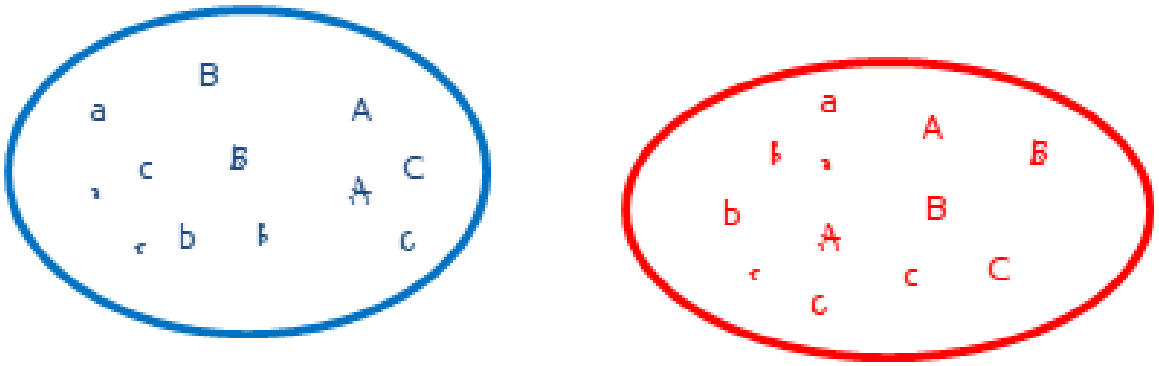
It is useful in a large amount of applications:

- Exploratory analysis.
- Machine Learning & Pattern Recognition.
- Image analysis.
- Bioinformatics.
- Market Analysis.

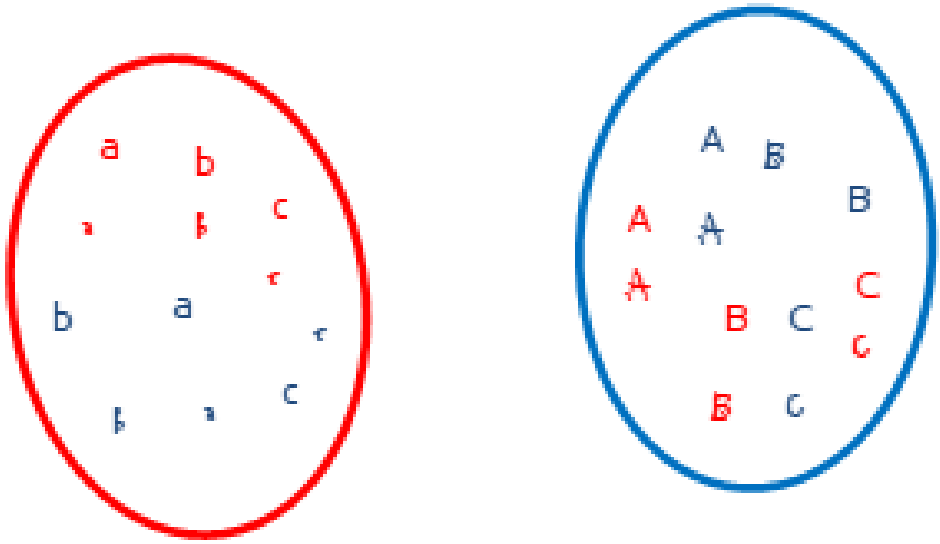
Clustering



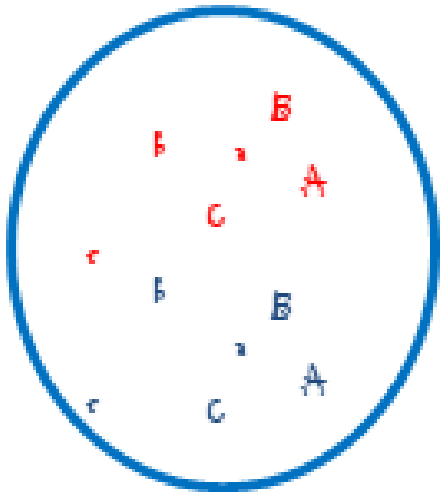
Clustering

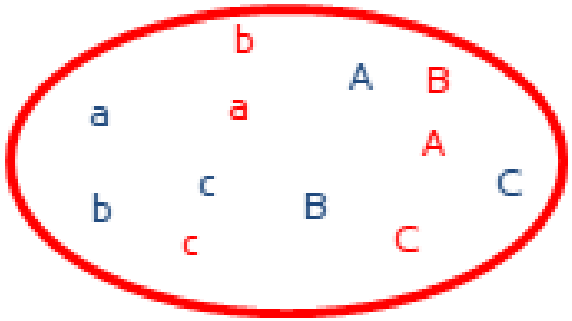


Clustering

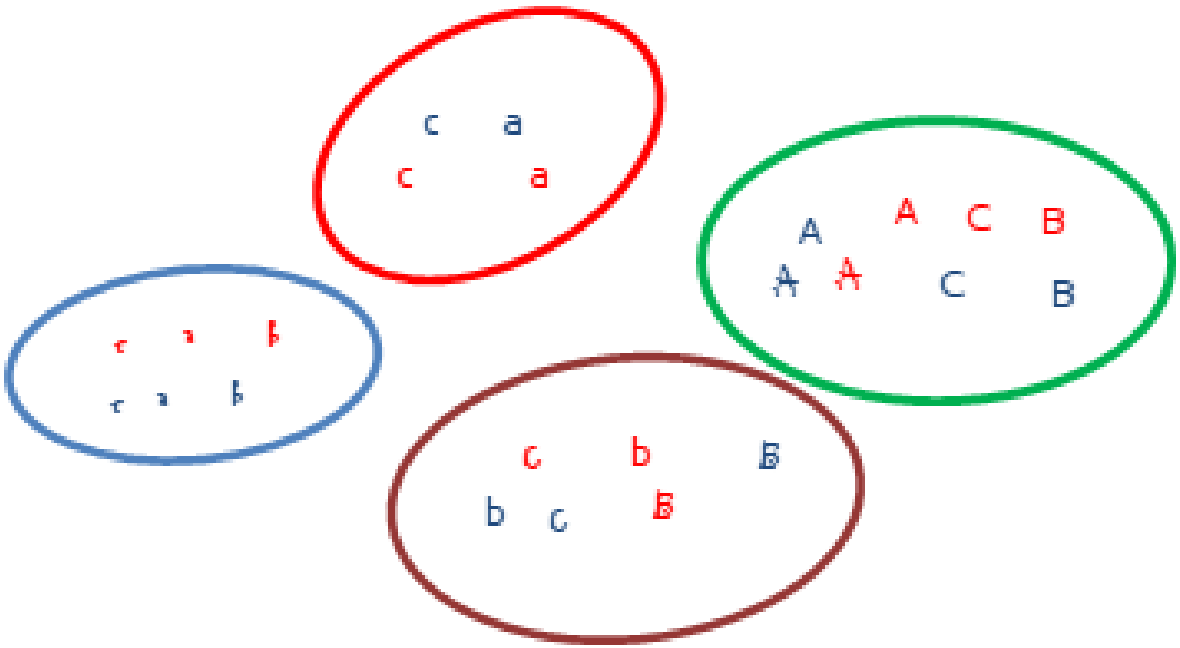


Clustering

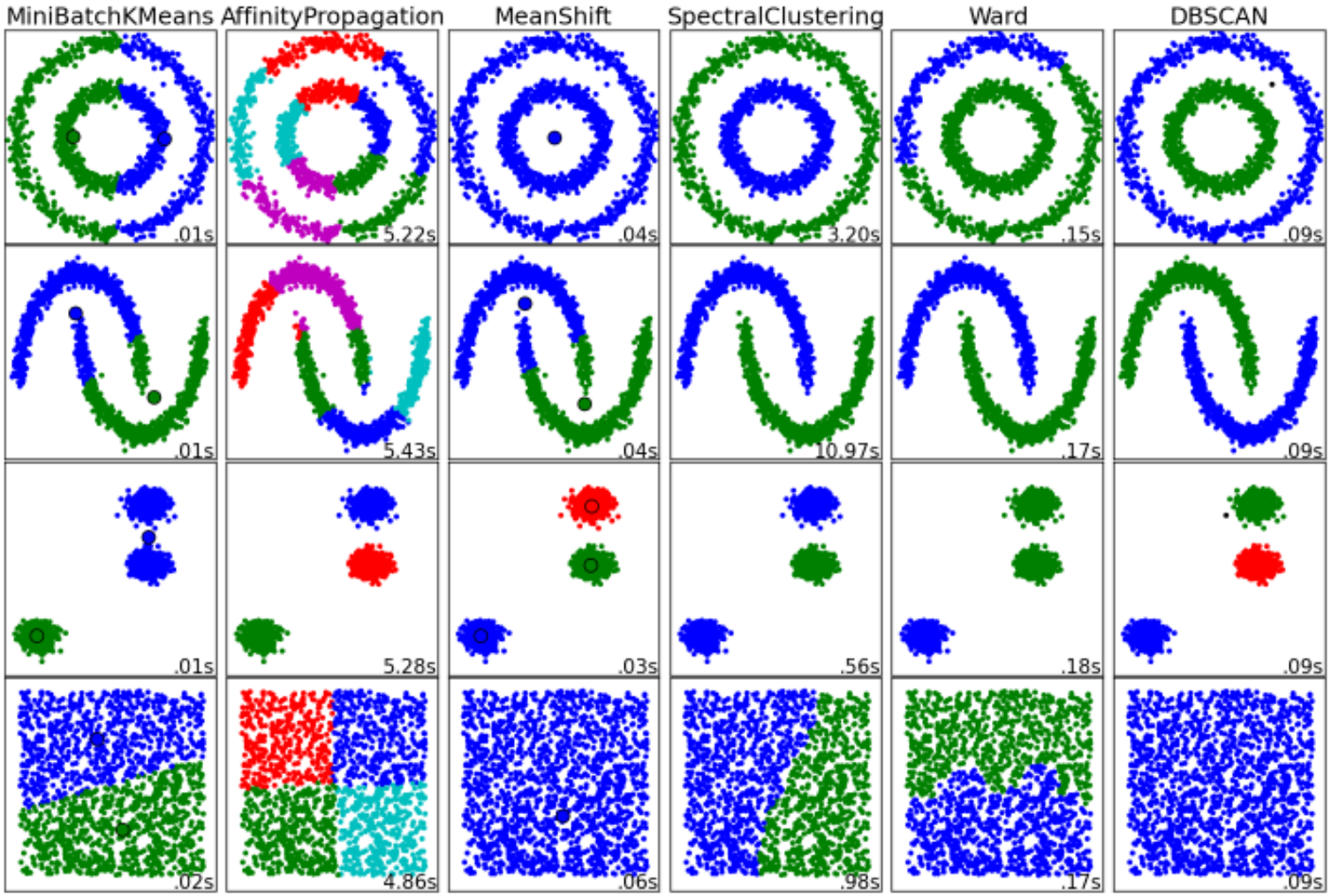




Clustering



Clustering algorithms in sklearn



Notes on Clustering input

Input data formats

One important thing to note is that the algorithms implemented in this module take different kinds of matrix as input.

- MeanShift and KMeans take data matrices of shape [n_samples, n_features]. The are obtained from `sklearn.feature\_extraction` module.
- AffinityPropagation and SpectralClustering take similarity matrices of shape [n_samples, n_samples]. These are obtained from `sklearn.metrics.pairwise module`.

sklearn.metrics.pairwise

sklearn.metrics.pairwise

Contains both distances and kernels. There are a number of ways to convert between a distance metric and a similarity measure, such as a kernel. Let D be the distance, and S be the kernel:

$$s(x, y) = e^{\frac{-d(x,y)}{\gamma}} \tag{1}$$

$$s(x, y) = \frac{1}{\frac{d(x,y)}{max(D)}} \tag{2}$$

sklearn.metrics.pairwise

sklearn.metrics.pairwise

Contains both distances and kernels. There are a number of ways to convert between a distance metric and a similarity measure, such as a kernel. Let D be the distance, and S be the kernel:

$$s(x, y) = e^{\frac{-d(x,y)}{\gamma}} \tag{1}$$

$$s(x, y) = \frac{1}{\frac{d(x,y)}{max(D)}} \tag{2}$$

sklearn.metrics.pairwise.rbf_kernel(X, Y=None, gamma=0)

Compute the rbf (gaussian) kernel between X and Y:

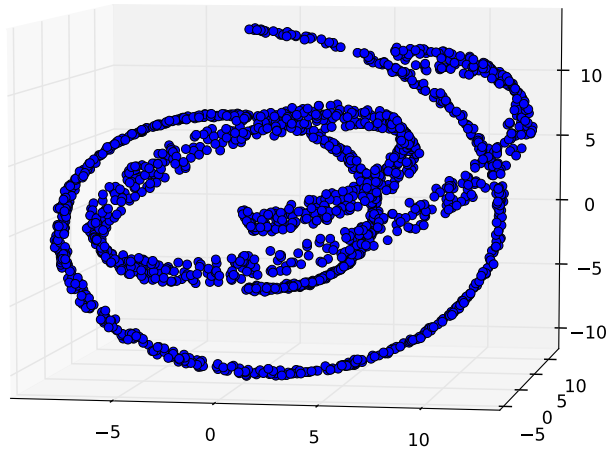
$$K(x, y) = e^{-\gamma||x-y||^2} \tag{3}$$

Learning Clustering in python

Let's recover our previous challenge (the two swiss rolls).

Could we automagically assign each sample to an egg roll? Open your ipython and:

- Use first k-means.
- Try another clustering algorithm.



Principal Components Analysis

PCA

PCA is an orthogonal linear transformation that creates a data model with a new coordinate system with a criteria for maximum variance captured.

$$w_1 = \arg \max_{||w||=1} \left\{ E(w^T X)^2 \right\} \tag{4}$$

Principal Component Analysis

Most popular and commonly used methods. Sometimes included in the "Factor Analysis" methods.

- It's the second most main tool for visualization
  - (First is always to plot the signals!)

Principal Component Analysis

Most popular and commonly used methods. Sometimes included in the "Factor Analysis" methods.

- It's the second most main tool for visualization
  - (First is always to plot the signals!)
- Main goal of PCA is to capture main directions of variance in input space.

Principal Component Analysis

Most popular and commonly used methods. Sometimes included in the "Factor Analysis" methods.

- It's the second most main tool for visualization
  - (First is always to plot the signals!)
- Main goal of PCA is to capture main directions of variance in input space.
- Dimensionality reduction:

Principal Component Analysis

Most popular and commonly used methods. Sometimes included in the "Factor Analysis" methods.

- It's the second most main tool for visualization
  - (First is always to plot the signals!)
- Main goal of PCA is to capture main directions of variance in input space.
- Dimensionality reduction:
  - Allows for projecting the dataset onto a low dimensional subspace.

Principal Component Analysis

Most popular and commonly used methods. Sometimes included in the "Factor Analysis" methods.

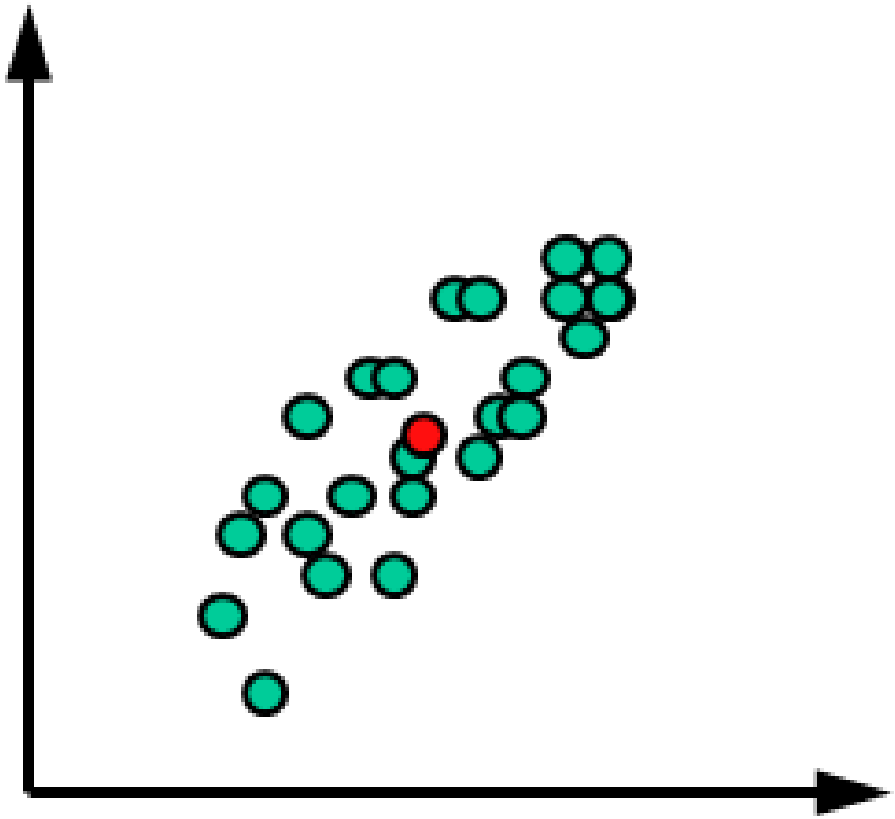
- It's the second most main tool for visualization
  - (First is always to plot the signals!)
- Main goal of PCA is to capture main directions of variance in input space.
- Dimensionality reduction:
  - Allows for projecting the dataset onto a low dimensional subspace.
  - Form of compression, or data modeling.

Principal Component Analysis

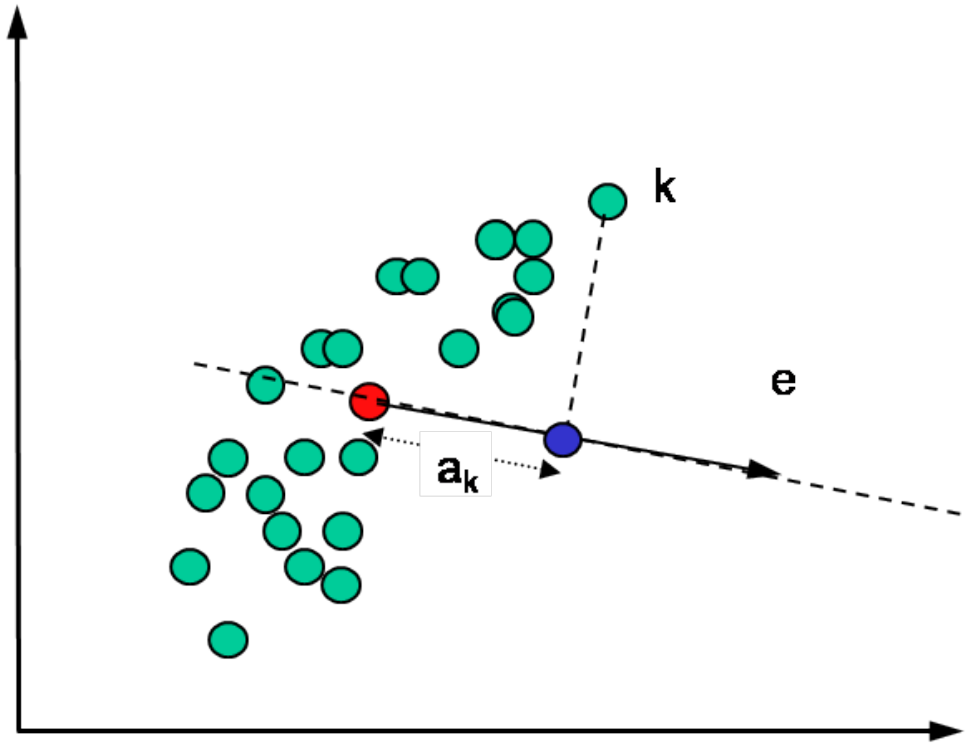
Most popular and commonly used methods. Sometimes included in the "Factor Analysis" methods.

- It's the second most main tool for visualization
  - (First is always to plot the signals!)
- Main goal of PCA is to capture main directions of variance in input space.
- Dimensionality reduction:
  - Allows for projecting the dataset onto a low dimensional subspace.
  - Form of compression, or data modeling.
  - Filtering.

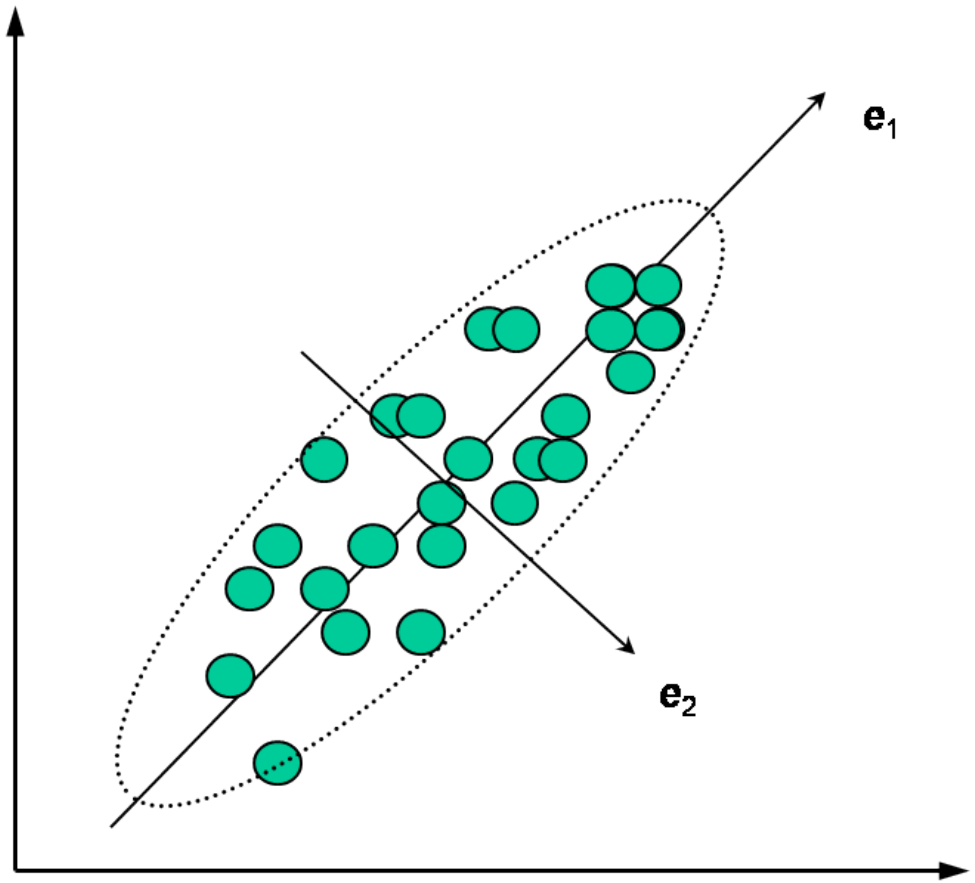
Principal Component Analysis



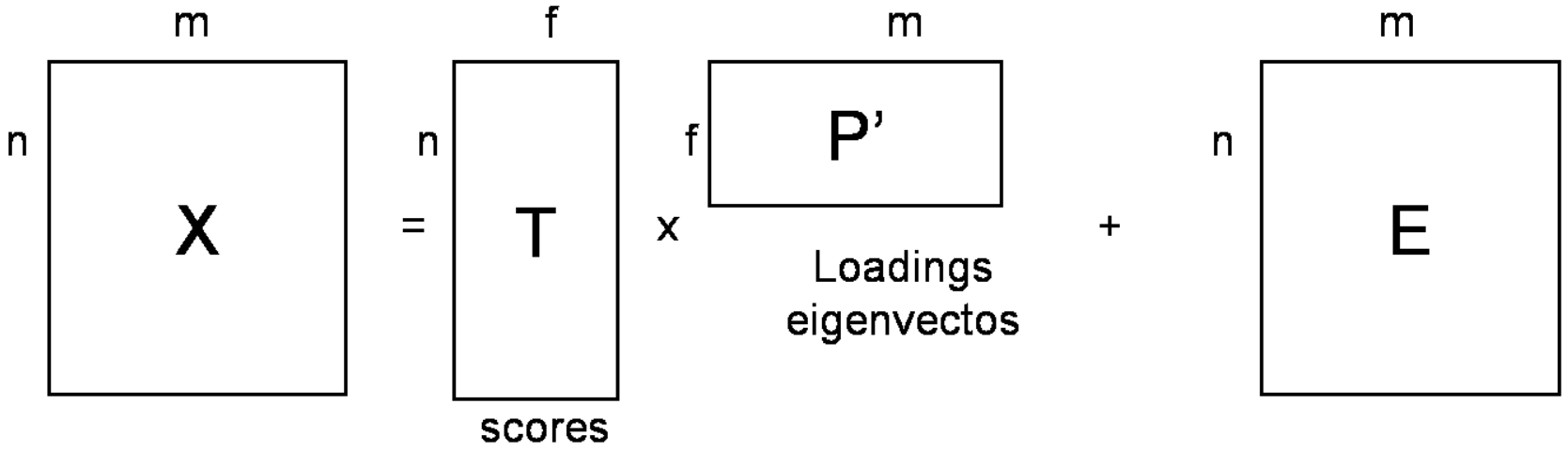
Principal Component Analysis



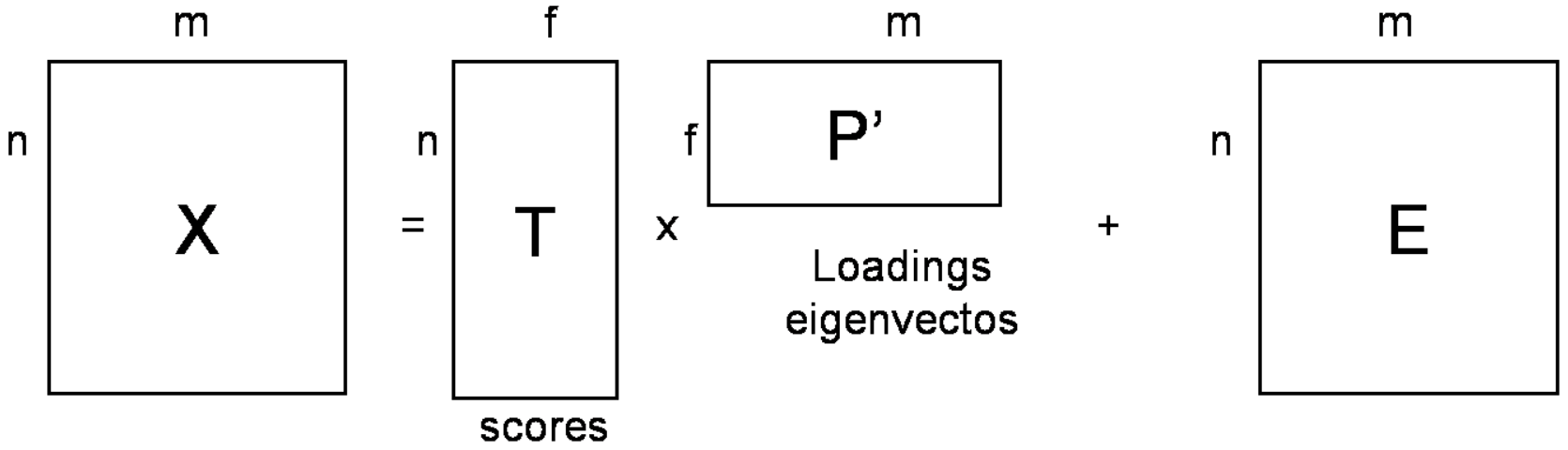
PCA

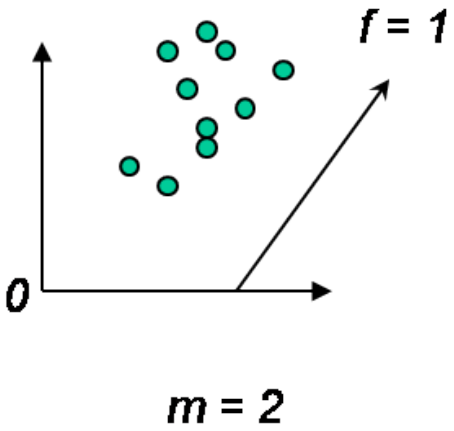


PCA

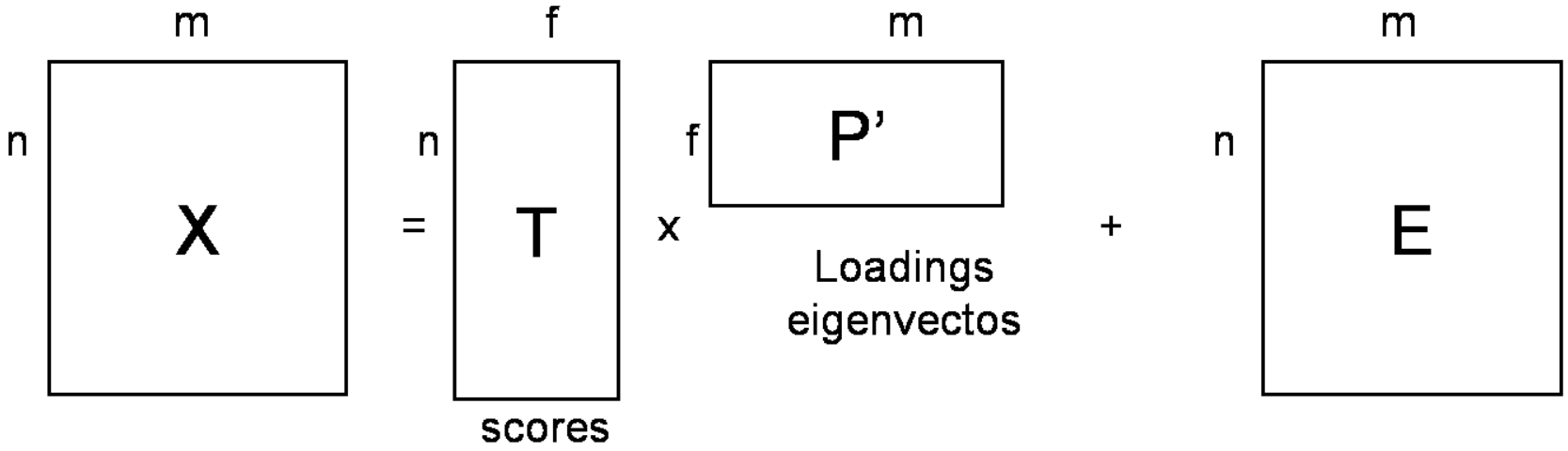


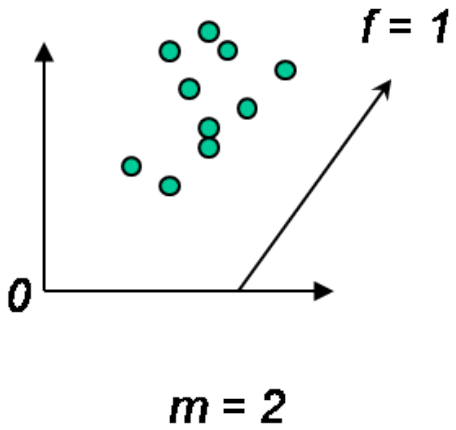
PCA

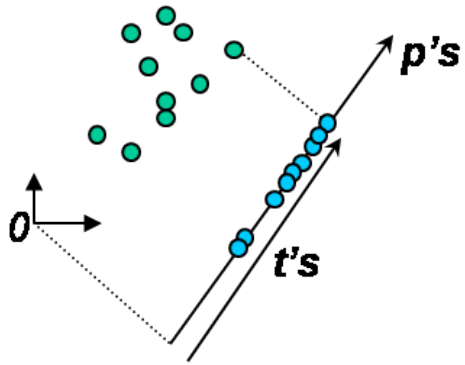




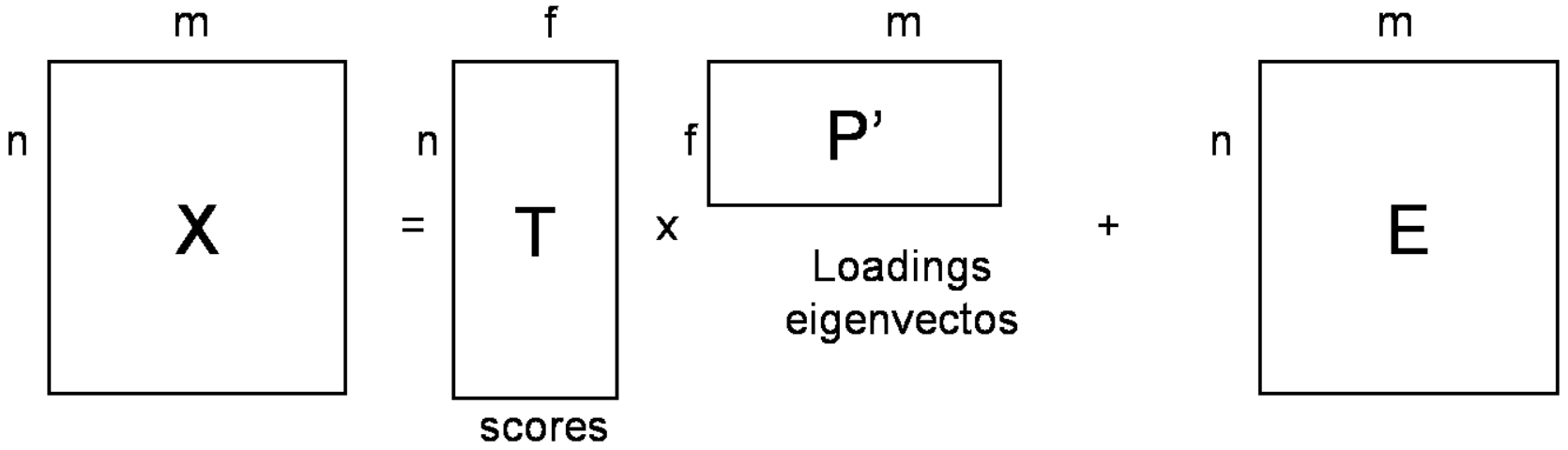
PCA

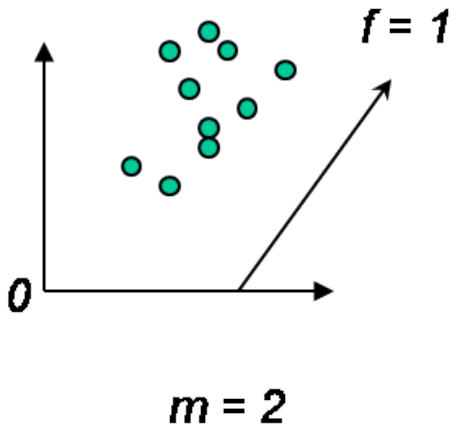


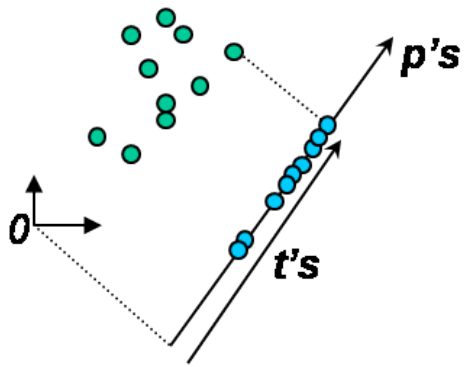


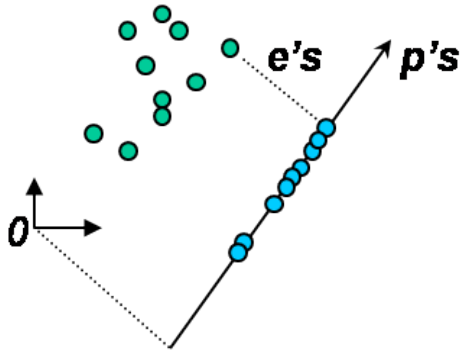


PCA


m
f
m
m
n
X
=
n
T
x
f
P'
+
n
E
scores
Loadings
eigenvectos


f = 1
0
m = 2


p's
0
t's


e's
p's
0

The Iris dataset

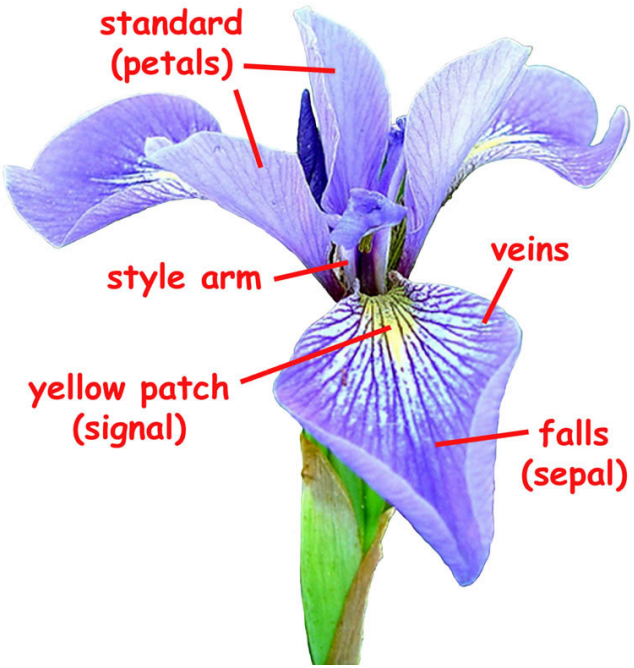
Iris dataset

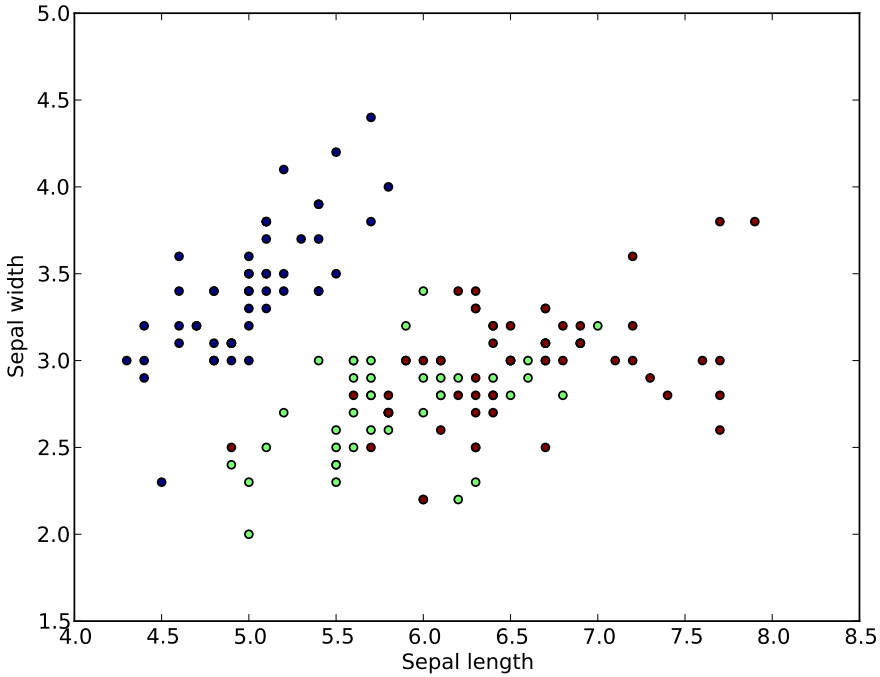
The iris dataset is a classical classification task consisting in identifying 3 different types of irises (Setosa, Versicolour, and Virginica) from their petal and sepal length and width.

```
>>> import numpy as np
>>> from sklearn import datasets
>>> import matplotlib.pylab as pl
>>> iris = datasets.load_iris()
>>> X = iris.data
>>> y = iris.target
>>> X[:2,:]
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2]])
>>> np.unique(y)
array([0, 1, 2])
```

```
pl.scatter(X[:, 0], X[:, 1],
      c=y)
pl.xlabel('Sepal length')
pl.ylabel('Sepal width')
```

Iris Dataset





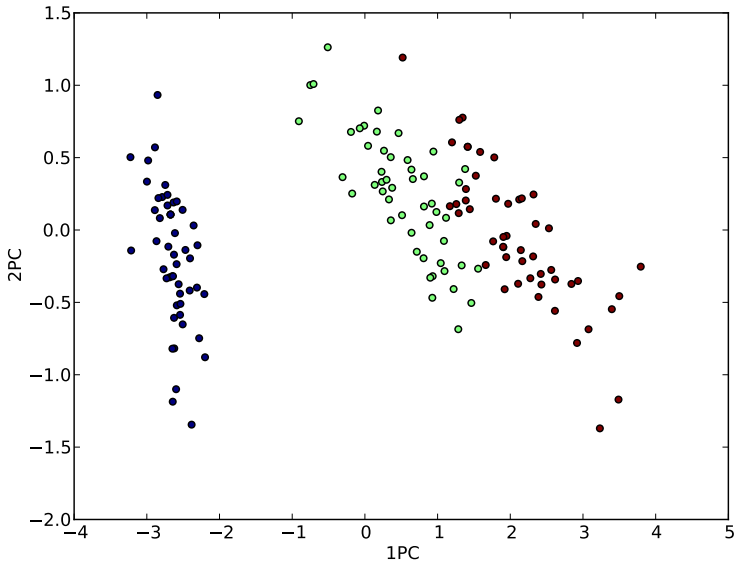
PCA in python

It's very easy now to construct the PCA projection:

```
>>> from sklearn import decomposition
>>> pca = decomposition.PCA(n_components=2)
>>> pca.fit(X)
PCA(copy=True, n_components=2, whiten=False)
>>> T = pca.transform(X)
```

PCA in python

```
import pylab as pl
pl.scatter(T[:, 0], T[:, 1], \
    c=y)
pl.xlabel('1PC')
pl.ylabel('2PC')
```



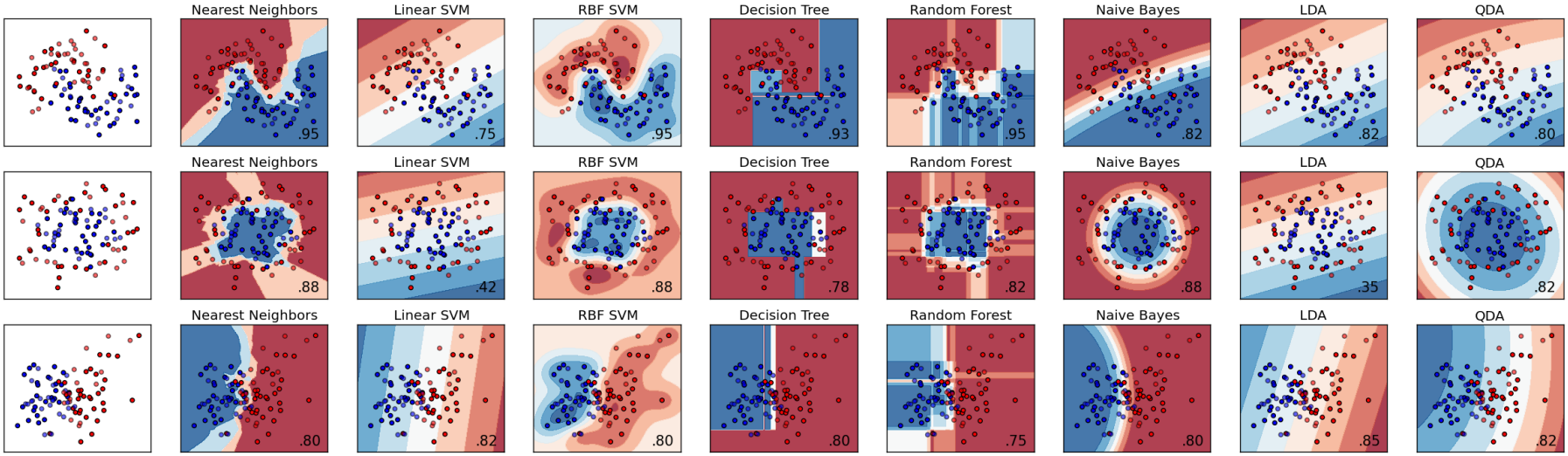
Supervised Learning

Supervised Learning

Consists in learning the link between two datasets:

- An observed data X and
- A variable y usually called target or labels.

Classification algorithms in sklearn



k-nearest neighbours

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(7)
>>> knn.fit(T, y)
KNeighborsClassifier(algorithm='auto', leaf_size=30, n_neighbors
           warn_on_equidistant=True, weights='uniform')
>>> knn.predict([[0.1, 0.2]])
array([1])
>>> yPred = knn.predict(T)
>>> (yPred==y).mean()
0.9799999999999998
```

Support Vector Classification

```
>>> from sklearn import svm
>>> C,gamma = 1,0.7
>>> svcg = svm.SVC(kernel='rbf', gamma=gamma, C=C).fit(T, y)
>>> svcg.predict([-.7,7])
array([ 2.])
>>> yPred = svcg.predict(T)
>>> (yPred==y).mean()
0.95333333333333337
```

sklearn.metrics

sklearn.metrics

roc_curve	Compute Receiver operating characteristic (ROC).
precision_recall_curve	Compute precision-recall pairs for different probability thresholds.
accuracy_score	Accuracy classification score.
confusion_matrix	Confusion matrix.
matthews_corrcoef	Matthews Correlation coefficient
classification_report	Build a text report showing the main classification metrics.

sklearn.metrics

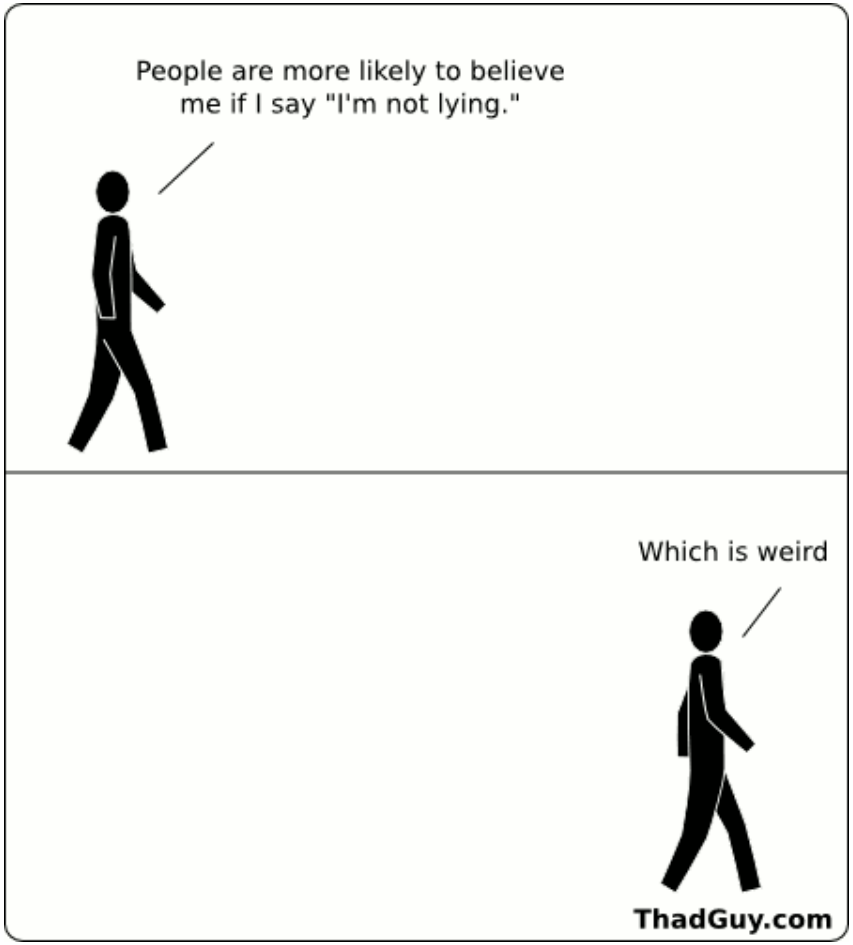
```
>>> from sklearn import metrics
>>> metrics.recall_score(yPred,y)
0.9533333333333337
>>> metrics.confusion_matrix(yPred,y)
array([[50,  0,  0],
       [ 0, 47,  4],
       [ 0,  3, 46]])
```

```
metrics.classification_report(yPred,y)
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        50
          1       0.94      0.92      0.93        51
          2       0.92      0.94      0.93        49

navg / total       0.95      0.95      0.95       150
```

Cross-validation



Validity and Over-Fitting

In multivariate predictive models, over-fitting occurs when a large number of predictor variables is fit to a small N of subjects. A model may "fit" well or perfectly, even if no real relationship. Simon, JNCI 2003

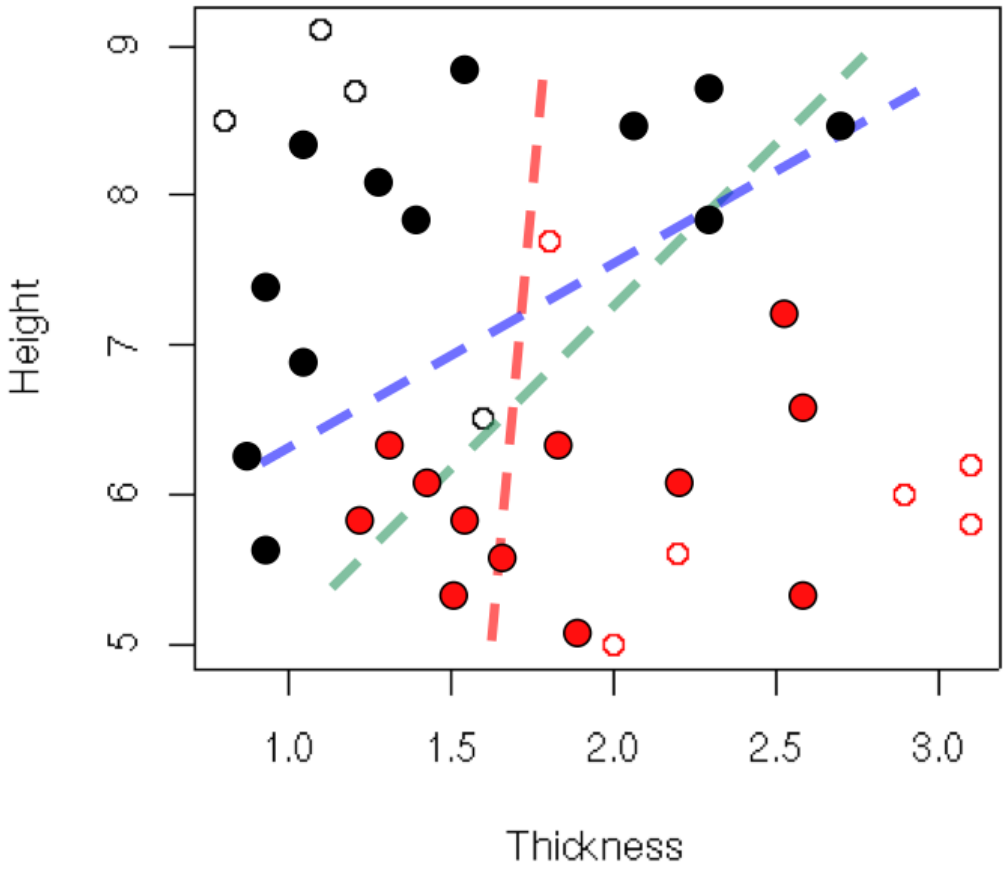
Validity and Over-Fitting

In multivariate predictive models, over-fitting occurs when a large number of predictor variables is fit to a small N of subjects. A model may "fit" well or perfectly, even if no real relationship. Simon, JNCI 2003

Direct Consequence of Over-fitting

Model performance results are not reproducible in a new set of data.

Over-Fitting



Validation Schemes

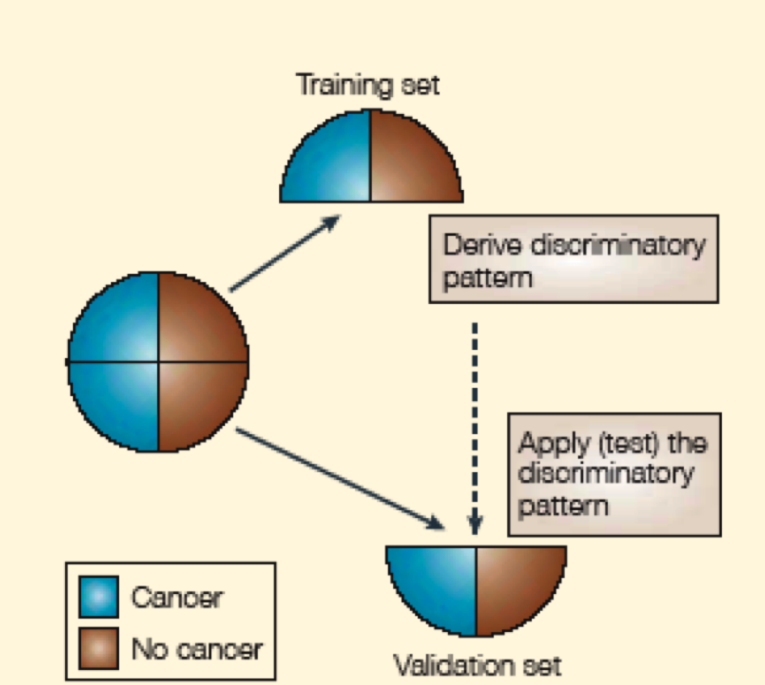


Figure 1 | **Method of dividing original sample to assess reproducibility and overfitting.**

Ransohoff. Nat Rev Cancer 2004

Cross Validation

sklearn.cross_validation.train_test_split

Split arrays or matrices into random train and test subsets.

```
>>> from sklearn import cross_validation
>>> xTrain, xVal, yTrain, yVal = cross_validation.train_test_split(
...     X, y, test_size=0.4)
...
>>> xTrain.shape,xVal.shape
((90, 4), (60, 4))
```

Cross Validation

sklearn.cross_validation.train_test_split

Split arrays or matrices into random train and test subsets.

```
>>> from sklearn import cross_validation
>>> xTrain, xVal, yTrain, yVal = cross_validation.train_test_split(
...     X, y, test_size=0.4)
...
>>> xTrain.shape,xVal.shape
((90, 4), (60, 4))
```

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(7).fit(xTrain, yTrain)
>>> yPredVal = knn.predict(xVal)
>>> (yPredVal==yVal).mean()
0.93333333333333335
```

Cross-validation metrics

Can we compute some statistics on the performance of the classifiers?
`cross_val_score` exists!

```
>>> knn = neighbors.KNeighborsClassifier(1)
>>> recalls = cross_validation.cross_val_score(
...     knn, X, y, cv = 6)
...
>>> recalls
array([ 0.92,  0.96,  0.96,  1.  ,  0.96,  0.96])
>>> recalls.mean()
0.9599999999999996
>>> recalls.std()
0.023094010767585018
```

Leave one out

LOO

Each training set is constructed by taking all samples except sample k and the model validates over k. Then just iterate through k.

```
>>> from sklearn.cross_validation import LeaveOneOut
>>> from sklearn import svm
>>> loo = LeaveOneOut(y.size)
>>> cl = svm.SVC(kernel='linear', C=1)
>>> recalls = [(y[indVal] == cl.fit(X[indTrain,:],y[indTrain]).
...          predict(X[indVal,:])).mean() \
...              for indTrain, indVal in loo]
...
>>> len(recalls)
150
>>> recalls[:5]
[1.0, 1.0, 1.0, 1.0, 1.0]
```

k-nearest neighbours vs. SVC Challenge

Challenge

1. Get all variables from Iris dataset.
2. Compute a 2D-PCA model of the Iris dataset.
3. Compute and plot the decision boundaries for a k-NN and SVC classifier of your choice.

Hint: use np.mesgrid()

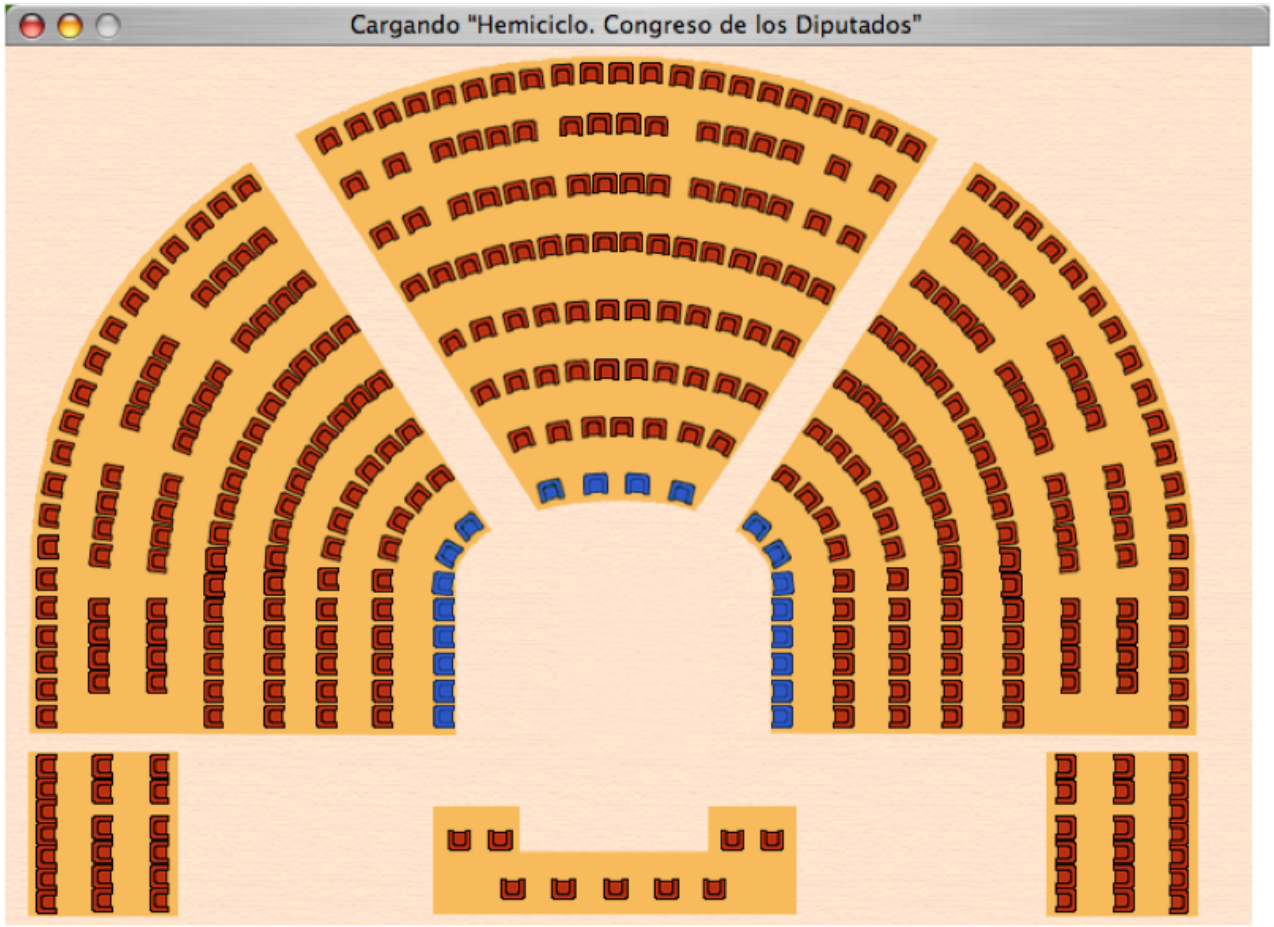
An Eigenfaces Session with python

This session aims to demonstrate the use of scikit in python using an eigenfaces exercise.

1. First will construct a class for storing the information of the chamber of representatives.
2. Later we will import the image dataset,
3. compute some descriptive statistics,
4. perform an analysis of principal components,
5. clustering analysis and finally,
6. will use a non-linear classifier to assess the gender from just the image of the representative.

www.congreso.es

- Public database (members of the Chamber of Representatives)
- Small *python* script extracted picture and info for each representative



www.congreso.es

- Public database (members of the Chamber of Representatives)
- Small *python* script extracted picture and info for each representative

www.congreso.es

- Public database (members of the Chamber of Representatives)
- Small *python* script extracted picture and info for each representative

www.congreso.es

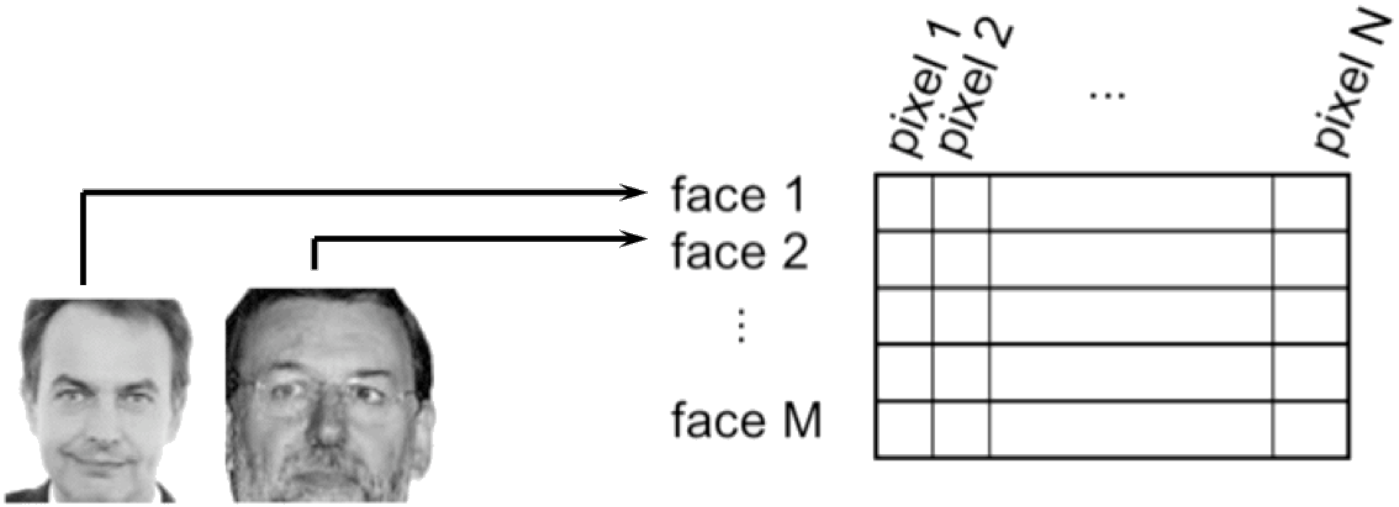
- Public database (members of the Chamber of Representatives)
- Small *python* script extracted picture and info for each representative

www.congreso.es

- Public database (members of the Chamber of Representatives)
- Small *python* script extracted picture and info for each representative

www.congreso.es

- Images imported and converted to B/W
- 350 members, $86 \times 85 = 7310$ pixels



Preparing to import the data

We will need to define some classes to prepare the image readout. A nice way of doing this is defining a *Diputado* class, followed by a *Parlamento* A *Diputado* has a Name, Surnames, ID, Picture, Gender and Political Affiliation.

A Diputado class definition I

```
class Diputado(object):
    def __init__(self,ind,fileRoot="./"):
        self.name=""
        self.surname=""
        self.ind=ind
        self.picfile=""
        self.party=""
        self.gender=""
        self.fileRoot=fileRoot
        self.ext='c.jpg'
    def setName(self,name):
        self.name=name
    def setSurname(self,surname):
        self.surname=surname
    def setParty(self,party):
        self.party=party
    def setGender(self,gender):
        self.gender=gender
```

```
    def getName(self):
        return self.name
    def getSurname(self):
        return self.surname
    def getInd(self):
        return self.ind
    def getPicfile(self):
        return self.fileRoot + str(s
    def getParty(self):
        return self.party
    def getGender(self):
        return self.gender
```

Parlament class

```
class Parlament(object):
    def __init__(self):
        self.elements=[]
        self.inds=[]
        self.ndips=0
    def add(self, diputado):
        self.elements.append(diputado)
        self.inds.append(diputado.getInd())
        self.ndips +=1
    def getInds(self):
        return [self.elements[i].getInd() for i in range(self.ndips)]
    def len(self):
        return len(self.elements)
    def __getitem__(self, key):
        if isinstance(key, slice):
            indices = key.indices(self.ndips)
            return [self[ii] for ii in xrange(*key.indices(self.len()))]
        else:
            return self.elements[key]
    def getName(self,key):
        return self.elements[key].getName()
```

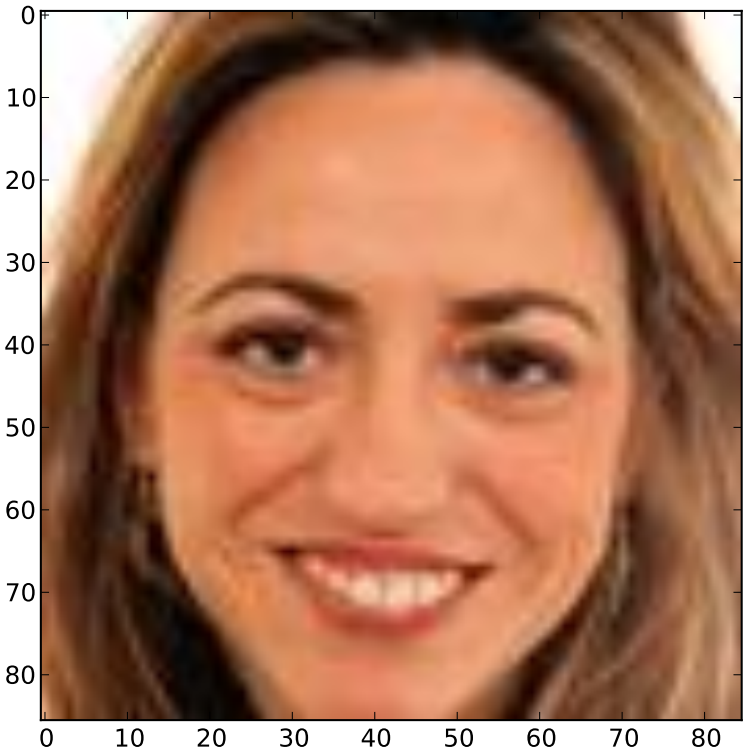
Importing data images

```
import csv
with open('db/index.csv','rb') as csvfile:
    r = csv.reader(csvfile, delimiter=';')
    r.next()
    p = Parlament()
    for row in r:
        n = Diputado(int(row[0]),"db/db/")
        n.setName(row[1].strip())
        n.setSurname(row[2].strip())
        n.setGender(row[3].strip())
        n.setParty(row[4].strip())
        p.add(n)
```

Let's see what we have

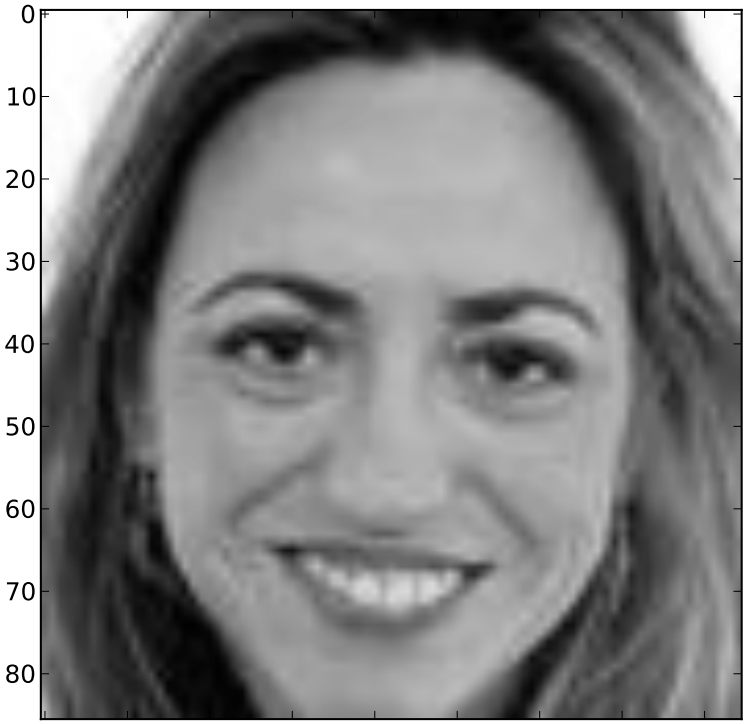
```
from scipy import ndimage,misc
from numpy import shape,prod,unique,sqrt
import matplotlib.pyplot as pl
I = ndimage.imread(p[1].getPicfile())
pl.imshow(I)
shape(I)
'|'.join([p[1].getName(), p[1].getSurname(), p[1].getGender()])
```

Let's see what we have



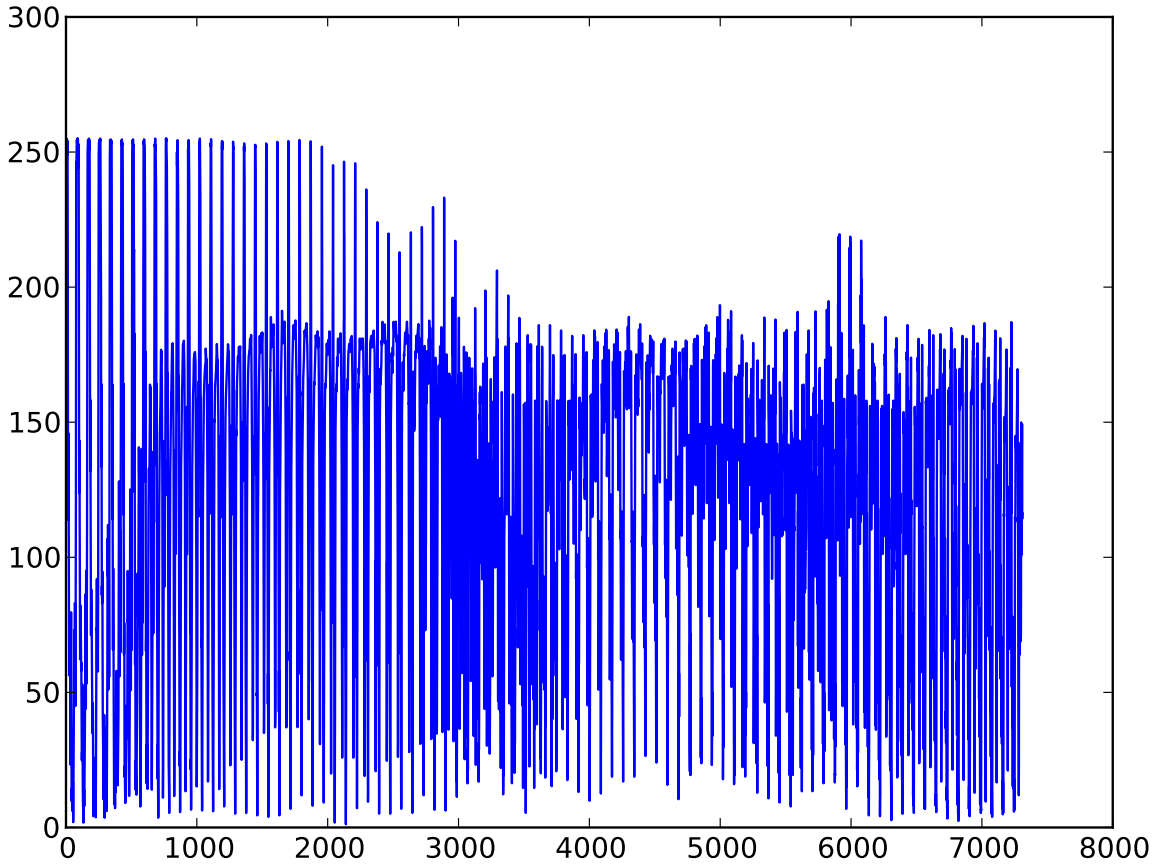
Converting to gray

```
I = ndimage.imread(p[1].getPicfile(), flatten=1)
img=pl.imshow(I)
img.set_cmap('gray')
Is=shape(I)
Is
```



And everything is a Signal...

```
Iv = np.reshape(I,prod(Is))
pl.plot(Iv)
```



Construct the final dataset

```
>>> X=np.array([ np.reshape(ndimage.imread(p[i].getPicfile(), \
...                    flatten=1), prod(Is)) \
...                for i in range(0,p.len())])
...
>>> shape(X)
(348, 7310)
>>> Yg=np.array([ p[i].getGender() \
...     for i in range(0,p.len())])
...
>>> Yp=np.array([ p[i].getParty() \
...     for i in range(0,p.len())])
...
>>> Yp[0:5],Yg[0:5],shape(X)
(array(['GS', 'GS', 'GC-CiU', 'GP', 'GP'],
      dtype='|S10'), array(['H', 'M', 'H', 'H', 'H'],
      dtype='|S1'), (348, 7310))
```

Principal Component Analysis

Now begins the interesting part. Let's try a PCA projection.

```
>>> from sklearn.decomposition import PCA
>>> from sklearn import preprocessing
>>> sX = preprocessing.scale(X)
>>> ncomp=5
>>> mod=PCA(n_components=ncomp)
>>> mod.fit(sX)
PCA(copy=True, n_components=5, whiten=False)
>>> ev = mod.explained_variance_ratio_
>>> print(ev)
[ 0.21537468  0.08870527  0.07254977  0.05253376  0.04564393]
```

Principal Component Analysis

The projection over the embedding.

```
>>> Xpca = mod.transform(sX)
>>> Xpca[0:3,:]
array([[-35.67224884,  -2.5693078 ,  13.0386591 ,   7.5424757 ,
        -10.67783451],
       [ 55.11857986, -19.39882469, -26.91612053, -21.87864685,
         -6.68953943],
       [-36.31109619,  -0.79103142,  -9.56684589, -20.49104309,
         -3.42452836]], dtype=float32)
```

Eigendiputados

The principal components can be interpreted.

```
eigendips = mod.components_.reshape((ncomp, Is[0], Is[1]))

pl.figure(figsize=(9 ,3))

pl.subplot(131)
pl.imshow(eigendips[0], cmap=pl.cm.gray)
pl.axis('off')

pl.subplot(132)
pl.imshow(eigendips[1], cmap=pl.cm.gray)
pl.axis('off')

pl.subplot(133)
pl.imshow(-eigendips[2], cmap=pl.cm.gray)
pl.axis('off')

pl.subplots_adjust(wspace=0.01, hspace=0.01, top=1, bottom=0, left=0, r
```

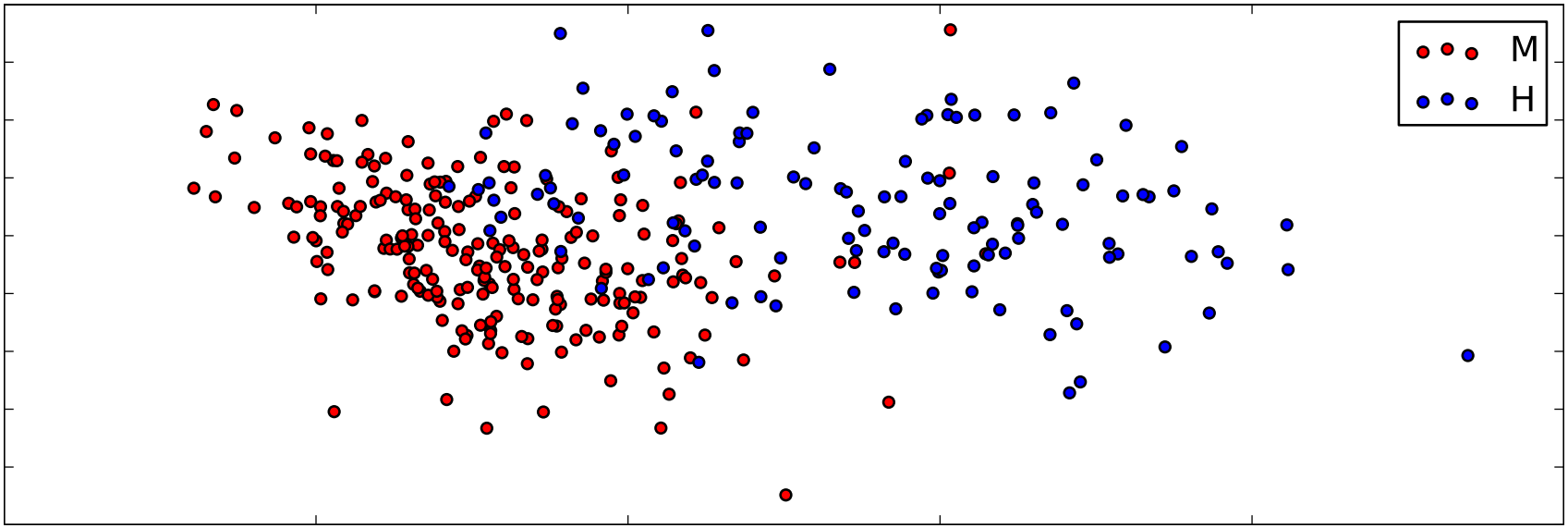
Eigendiputados

Score plots

```
colors=['blue','red']
clut = dict(zip(['M','H'],['red','blue']))
""" for one shot plot cols=[ clut[i] for i in Yg]"""

pl.scatter(Xpca[np.where(Yg=='H'),0],\
    Xpca[np.where(Yg=='H'),1],c='red')
pl.scatter(Xpca[np.where(Yg=='M'),0],\
    Xpca[np.where(Yg=='M'),1],c='blue')
pl.legend(('M','H'))
pl.xlabel('1PC ('+ "%s" % float( "%2.1g" % (100*ev[0])) +"%)")
pl.ylabel('2PC ('+ "%s" % float( "%2.1g" % (100*ev[1])) +"%)")
```

Score plots

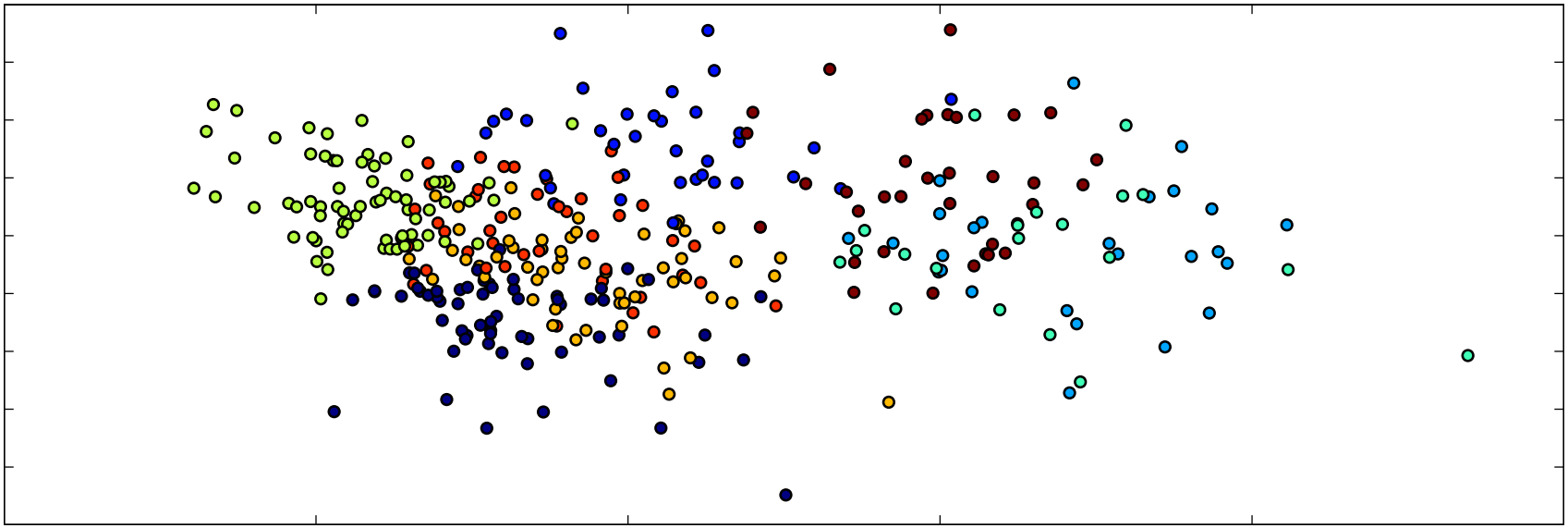


k-means

With the data already imported, it's easy to group similar pictures with a clustering algorithm.

```
>>> from sklearn import cluster
>>>
>>> k_means =  cluster.KMeans(k=8,n_init=10)
>>> k_means.fit(Xpca)
KMeans(copy_x=True, init='k-means++', k=8, max_iter=300, n_init=
    precompute_distances=True,
    random_state=<mtrand.RandomState object at 0x7f6d01af51b0>,
    verbose=0)
>>>
>>> pl.scatter(Xpca[:,0],Xpca[:,1],c=k_means.labels_.astype(np.i
<matplotlib.collections.PathCollection object at 0x3afcd90>
```

k-means



k-means

We can plot the samples that have been grouped within the same cluster. For doing so we have to reshape the vectors corresponding to each sample matching the original images. This is extremely easy to do in python!

```
def plotg(g):
    indg = np.where(k_means.labels_ == g)
    nx = int(sqrt(shape(indg)[1]))+1
    ny = nx+1
    f = pl.figure()

    for e,i in enumerate(indg[0]):
        f.add_subplot(ny,nx,e)
        pl.imshow(X[i,:].reshape(Is), cmap=pl.cm.gray)
        pl.axis('off')
```

k-means groups

```
>>> print(unique(k_means.labels_))
[0 1 2 3 4 5 6 7]
>>> plotg(5), plotg(0)
(None, None)
```

Supervised Analysis

Next, we will build two classifiers and will try to asses gender just from the feature vector (pixel intensities). We will test k-nn classifier and compare it with a Support Vector Classifier, trained with a grid search. We will also show how to use a cross-validation scheme provided by sklearn scipy toolkit.

Supervised analysis

```
>>> nYg = np.array([ int(y=='H') for y in Yg])
>>> print nYg[0:5]
[1 0 1 1 1]
```

k-Neighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import cross_validation
k_fold = cross_validation.KFold(n=p.len(), k=3, indices=True)
scores = list()
for train_indices, test_indices in k_fold:
    Xtrain = X[train_indices,:]
    Ytrain = nYg[train_indices]
    Xval = X[test_indices,:]
    Yval = nYg[test_indices]
    knn = KNeighborsClassifier()
    knn.fit(Xtrain,Ytrain)
    scores.append(knn.score(Xval,Yval))
```

```
>>> [knn.fit(X[train,:], nYg[train]).score(X[test,:], nYg[test])
...      for train, test in k_fold]
...
[0.8189655172413793, 0.8965517241379310343, 0.9051724137931034]
```

Support Vector Classifier

```
>>> from sklearn import svm
>>> svc = svm.SVC(C=1, kernel='linear')
>>> [svc.fit(X[train,:], nYg[train]).score(X[test,:], nYg[test])
...     for train, test in k_fold]
...
[0.84482758620689657, 0.8965517241379105, 0.8965517241379105]
```

Support Vector Classifier

In fact, we should cross-validate with some balance on the class folds.

```
k_fold_class_balanced = \
    cross_validation.StratifiedKFold(nYg,k=5)

svc = svm.SVC(C=1, kernel='linear')
[svc.fit(X[train,:], nYg[train]).score(X[test,:], nYg[test])
    for train, test in k_fold_class_balanced]
```

SCV tunning

```
>>> from sklearn.grid_search import GridSearchCV
>>> gammas = np.logspace(-6, -1, 5)
>>> clf = GridSearchCV(estimator=svc, param_grid=dict(gamma=gammas), n_
>>> clf.fit(X[train,:], nYg[train])
GridSearchCV(cv=None,
       estimator=SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
  kernel='linear', probability=False, shrinking=True, tol=0.001,
  verbose=False),
       fit_params={}, iid=True, loss_func=None, n_jobs=-1,
       param_grid={'gamma': array([  1.00000e-06,   1.77828e-05,   3.16
         1.00000e-01])},
       pre_dispatch='2*n_jobs', refit=True, score_func=None, verbose=0)
>>> print clf.best_score_
0.888888888889
>>> print clf.best_estimator_.gamma
1e-06
>>> print clf.score(X[test,:], nYg[test])
0.898550724638
```

Performance

```
>>> from sklearn.metrics import classification_report
>>> ypred = clf.predict(X[test,:])
>>> ytest = nYg[test]
>>> ygtest = Yg[test]
>>> print classification_report(ytest, ypred, target_names=['H'
             precision    recall  f1-score   support

          H       0.91      0.80      0.85        25
          M       0.89      0.95      0.92        44

avg / total       0.90      0.90      0.90        69
```

Performance

```
>>> from sklearn.metrics import confusion_matrix
>>> print confusion_matrix(ytest, ypred)
[[20  5]
 [ 2 42]]
```