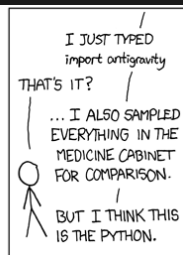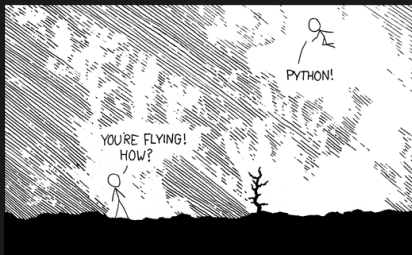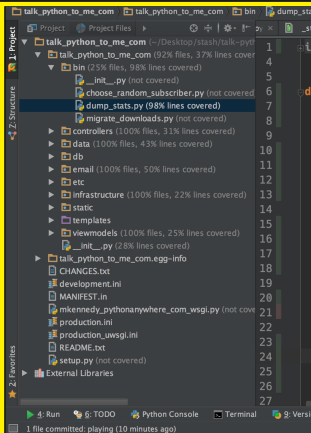# Machine Learning

Lecture Zero – September 27th, 2018
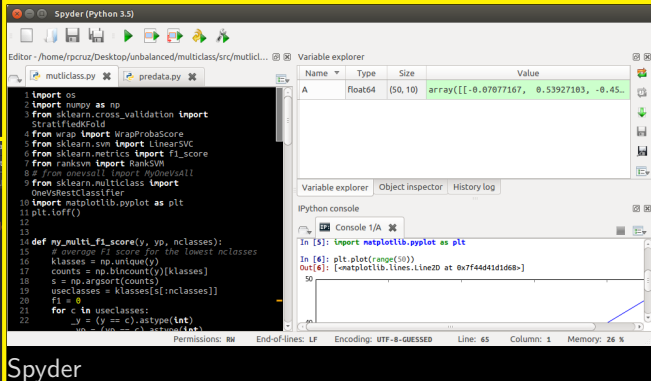
Ricardo Cruz   rpcruz@inesctec.pt
Eduardo Castro   emcastro@inesctec.pt
João Ribeiro Pinto   jtpinto@fe.up.pt
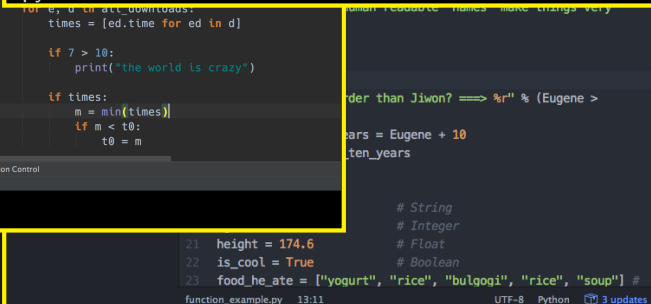
Editors

Spyder (Python 3.5)

Editor · /home/rpcruz/Desktop/unbalanced/multiclass/src/mutlicl...

multiclass.py   predata.py

```
1  import os
2  import numpy as np
3  from sklearn.cross_validation import
   StratifiedKFold
4  from wrap import WrapProbaScore
5  from sklearn.svm import LinearSVC
6  from sklearn.metrics import f1_score
7  from ranksvm import RankSVM
8  # from onevsall import MyOneVsAll
9  from sklearn.multiclass import
   OneVsRestClassifier
10 import matplotlib.pyplot as plt
11 plt.ioff()
12
13
14 def my_multi_f1_score(y, yp, nclasses):
15   # average F1 score for the lowest nclasses
16   klasses = np.unique(y)
17   counts = np.bincount(y)[klasses]
18   s = np.argsort(counts)
19   useclasses = klasses[s[:nclasses]]
20   f1 = 0
21   for c in useclasses:
22     _y = (y == c).astype(int)
```

Variable explorer

Name	Type	Size	Value
A	float64	(50, 10)	array([[-0.07077167, 0.53927103, -0.45...

Variable explorer   Object inspector   History log

IPython console

Console 1/A

In [5]: import matplotlib.pyplot as plt

In [6]: plt.plot(range(50))
Out[6]: [<matplotlib.lines.Line2D at 0x7f44d41d168>]

Permissions: RW   End-of-lines: LF   Encoding: UTF-8-GUESSED   Line: 65   Column: 1   Memory: 26 %

Spyder

talk_python_to_me_com   talk_python_to_me_com   bin   dump_sta

```
talk_python_to_me_com (~/Desktop/stash/talk-pyt
  talk_python_to_me_com (92% files, 37% lines cover
    bin (25% files, 98% lines covered)
      __init__.py (not covered)
      choose_random_subscriber.py (not covere
      dump_stats.py (98% lines covered)
      migrate_downloads.py (not covered)
    controllers (100% files, 31% lines covered)
    data (100% files, 43% lines covered)
    db
    email (100% files, 50% lines covered)
    etc
    infrastructure (100% files, 22% lines covered)
    static
    templates
    viewmodels (100% files, 25% lines covered)
    __init__.py (28% lines covered)
  talk_python_to_me_com.egg-info
  CHANGES.txt
  development.ini
  MANIFEST.in
  mkennedy_pythonanywhere_com_wsgi.py (not
  production.ini
  production_uwsgi.ini
  README.txt
  setup.py (not covered)
External Libraries
```

4: Run   6: TODO   Python Console   Terminal   9: Version Control

1 file committed: playing (10 minutes ago)

PyCharm

```
        ore, u  th all downloads:
        times = [ed.time for ed in d]

    if 7 > 10:
        print("the world is crazy")

    if times:
        m = min(times)
        if m < t0:
            t0 = m
```

rder than Jiwon? ===> %r" % (Eugene >

ears = Eugene + 10
ten_years

```
                    # String
                    # Integer
                    # Float
                    # Boolean
21  height = 174.6
22  is_cool = True
23  food_he_ate = ["yogurt", "rice", "bulgogi", "rice", "soup"]  #
```

function_example.py   13:11   UTF-8   Python   3 updates

Atom

1/58

Agenda

Theory	Practice
~Introduction~	
Numpy	Implement Euclidean distance
Cycles/functions	K-means algorithm
File handling	Use a real dataset
Matplotlib	Plots
Classes/modules	—
Scikit-learn	Use its k-means
Keras+TF	Demo
~Finishing touches~	
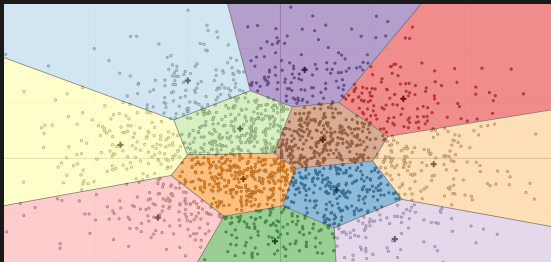
Introduction

Introduction
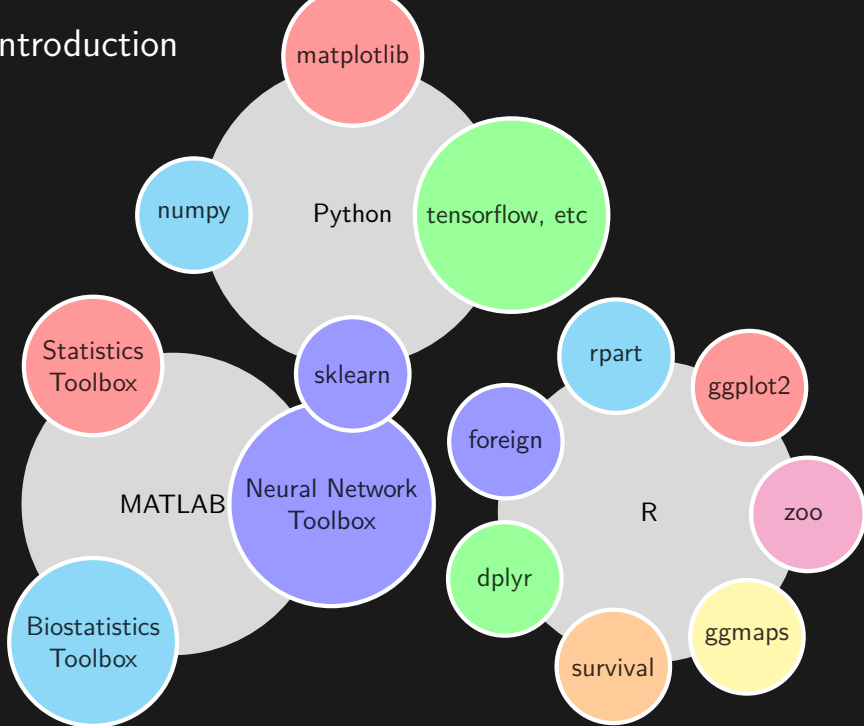
Introduction

- Furthermore, Python is a **general** purpose language
  - It is used in everything from user interfaces to web servers
  - Linear algebra is **not** a primary citizen

Introduction: Syntax

Python

```python
1 def is_prime(n):
2   for m in range(2, n):
3     if n % m == 0:
4       return False
5   return True
```

MATLAB

```matlab
1 function ret = is_prime(n)
2   ret = true;
3   for m=1:(n-1)
4     if mod(n, m) == 0
5       ret = false;
6       break
7     end
8   end
9 end
```

Warning: In Python you **must** properly indent your code.

NumPy: linear algebra package
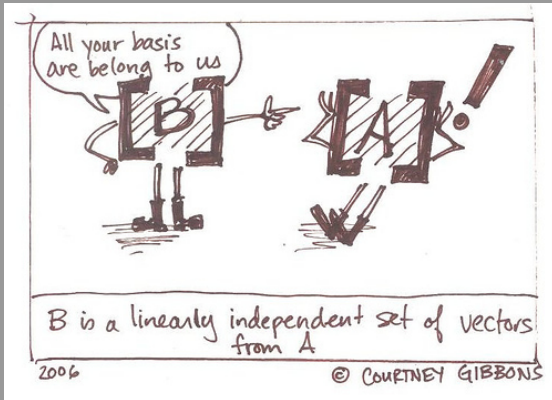
NumPy

The *de-facto* package for linear algebra is numpy.

Usually, it is shortended to np:

```python
import numpy as np
```

NumPy features two main structures:

- np.ndarray — array of multiple dimensions
- np.matrix — matrix (two dimensions)

Why ever use np.matrix when np.ndarray supports two dimensions?

NumPy vs MATLAB

Summary of **NumPy for MATLAB users** [1] [2]

MATLAB	NumPy
size(a)	a.shape
a(1:5,:)	a[0:5,:] or a[0:5] or a[:5]
a(end-4:end,:)	a[-5:]
a.'	a.transpose() or a.T
a * b	a.dot(b)
a .* b	a * b

The main conceptual difference is that Numpy supports **arithmetic broadcasting.** That is, you can do the following element-wise multiplication: (6,3) * (6,1). It automatically assumes you want to multiply by column. In MATLAB, you would have to use bsxfun(@times,r,A) or first use repmat().

---

[1] https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html
[2] http://mathesaurus.sourceforge.net/matlab-numpy.html

NumPy Example

Create $B_{ij} = \begin{cases} 5, & \text{if } A_{ij} > 1 \\ 0, & \text{otherwise.} \end{cases}$

Suggestions:
```
B = A.copy()
B[A > 1] = 5
B[A <= 1] = 0
```

──────────────────────

```
B = np.zeros(A.shape)
B[A > 1] = 5
```

──────────────────────

```
B = (A > 1) * 5
```

──────────────────────

```
B = np.asarray(
  [[5 if aij > 1 else 0 for aij in ai] for
ai in A])
```

It is probably a good
idea to finish this off
with:

`B = B.astype(np.int8)`

to save memory!

API

Python Lists

1. Create a list – `l = [1, 5, 2]`
2. Access item – `l[1]`
3. Modify item – `l[1] = 7`
4. Add item – `l.append(8)`, `l += [9, 10]`

NumPy

1. Create ndarray from list – `np.array()`
2. Create ndarray of zeros/ones – `np.zeros()`, `np.ones()`
3. Sample Uniform(0,1) – `np.rand`, `np.random.random()`
4. Sample Normal(0,1) – `np.randn`, `np.random.normal()`
5. Horizontal concatenation – `np.vstack()`, `np.c_[]`
6. Vertical concatenation – `np.hstack()`, `np.r_[]`
7. Any concatenation – `np.concatenate(list, axis)`

Project: Implement Euclidean distance

Get the project files on **https://github.com/jtrpinto/ml2018**.

1. Implement a function called **euclidean_distance(a, b)**
   - Given two vectors **a** and **b**, computes the square-root of the squared difference between the two vectors
   - $d = \|x_1 - x_2\|_2^2$
   - TIP: you might want to use vectorisation!
2. Use it for the two example vectors given in the script
   v1 = [1.1, 2.5, 4.4, 0.1, 2.3, 3.4]
   v2 = [2.0, 2.2, 1.0, 1.0, 2.5, 3.4]

Possibly useful functions:

`np.random.randn, np.mean, np.std, np.sqrt, np.square, np.linalg.norm`

Project: Implement Euclidean distance (solution)

```python
1 # Implement the euclidean_distance function here:
2 def euclidean_distance(a, b):
3     # Computes and returns the Euclidean distance between vectors
             'a' and 'b'
4     distance = np.sqrt(np.sum(np.square(a - b)))
5     return distance
6
7 # Check if it is working! Here are two vectors:
8 v1 = np.array([1.1, 2.5, 4.4, 0.1, 2.3, 3.4])
9 v2 = np.array([2.0, 2.2, 1.0, 1.0, 2.5, 3.4])
10
11 # Call the euclidean_distance function below to compute the
       distance between them:
12 dist = euclidean_distance(v1, v2)
13
14 print('Distance between v1 and v2: ', dist) # (For these vectors
       , it should be something like 3.6482...)
```

Cycles & Functions

Loops

Python has some interesting functions that can help you in loops.

```python
for n in range(10, 30, 2):
    print(n)
# 10 12 14 16 18 20 22 24 26 28

for i, n in enumerate(range(10, 20, 2)):
    print((i, n))
# (0, 10) (1, 12) (2, 14) (3, 16) (4, 18)

for a, b in zip(range(10, 20), range(40, 50)):
    print((a, b))
# (10, 40) (11, 41) (12, 42) (13, 43) (14, 44) (15, 45) (16, 46)
    (17, 47) (18, 48) (19, 49)
```

You will probably use these functions a lot in your loops: range, enumerate
and zip.

List Comprehension and Functools

List comprehension

```
1 [i**2 for i in range(5)]
```

```
1 [i for i in range(5) if i % 2 ==
      0]
```

Functools

```
1 map(lambda x: x**2, range(5))
```

```
1 filter(lambda x: x % 2 == 0,
          range(5))
```

```
1 functools.reduce(lambda a,b: a+b
    , range(5))  # sum numbers
    (10)
```

○ List comprehension is considered the most Pythonic approach.

○ But...

Project: Implement the k-means algorithm

Given data X_{ij}, the user first defines K clusters.

The algorithm:

1. Random cluster centroids are selected randomly using the MacQueen method;
2. Each observation is assigned the closest cluster:
   - $c_i(t+1) = \arg\min_k \|x_i - \tilde{c}_k\|_2^2$
3. Compute the new centroid of each cluster (using the Euclidean distance you implemented before):
   - $\forall k, j \colon \tilde{c}_{kj} = \frac{1}{\sum_{i=1}^N \mathbb{1}_{c_i=k}} \sum_{i=1}^N \mathbb{1}_{c_i=k} X_{ij}$
4. Repeat steps 2-3 for a set number of iterations.

Please place this inside a `kmeans`$\colon \mathbb{Z}, \mathbb{R}^{N \times M} \to \mathbb{Z}^N$ function, which maps K and X to its respective clusters.

Note: you can also implement a convergence condition that stops the loop as soon as the clusters stabilise.

Project: Implement the k-means algorithm (solution)

```python
def k_means(X, k=3, n_iterations=10):
    # (...)
    # The k_means algorithm is iterative. Start a loop here for '
        n_iterations':
    for ii in range(n_iterations):
        # In each loop, for each data point:
        for index in range(X.shape[0]):
            # Compute the Euclidean distance between the data
                point and each centroid:
            distance = [euclidean_distance(X[index], cc) for cc
                in centroids]

            # Then, assign each data point to the cluster with
                the nearest centroid:
            membership[index] = np.argmin(distance)

        # Now, recompute the centroids of each cluster, computing
            the mean of the cluster data points:
        for cc in range(k):
            centroids[cc] = np.mean(X[membership == cc], axis=0)
    # (...)
```

Kmeans: usage example

K-means can be used for e.g. image compression...
24 (8×3) bits $\rightarrow$ k=64 (6 bits)
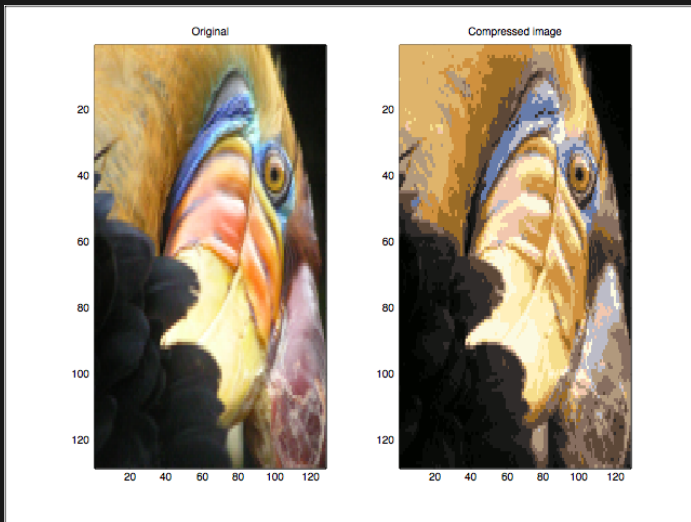
Matplotlib
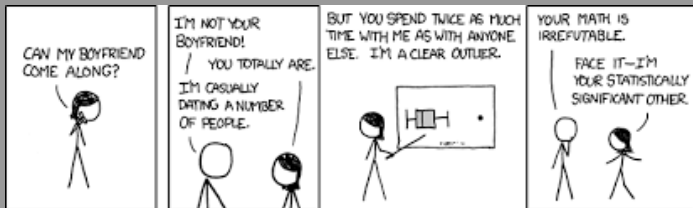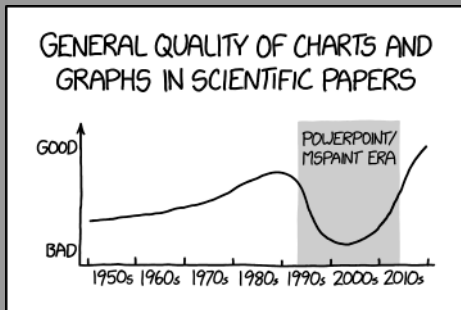
Matplotlib

The *de-facto* package for plotting graphics in python is matplotlib.

Matplotlib contains an API[3] called `pyplot` that is inspired in MATLAB.

See: http://matplotlib.org/1.4.3/api/pyplot_api.html

---

[3]API = Application Programmer's Interface

Matplotlib

Example using synthetically created data:


```
1 from matplotlib.pyplot as plt
2 from sklearn.datasets import make_blobs
3 X, y = make_blobs(centers=3)
```
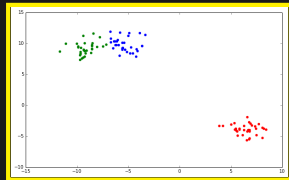
```
1 plt.scatter(X[y == 0, 0], X[y == 0, 1], color='red')
2 plt.scatter(X[y == 1, 0], X[y == 1, 1], color='green')
3 plt.scatter(X[y == 2, 0], X[y == 2, 1], color='blue')
4 plt.show()
```

OR

```
1 colors = ['red', 'green', 'blue']
2 plt.scatter(X[:, 0], X[:, 1], color=[colors[_y] for _y in y])
3 plt.show()
```

OR

```
1 colors = plt.cm.rainbow(np.linspace(0, 1, 3))
2 plt.scatter(X[:, 0], X[:, 1], color=colors[y])
3 plt.show()
```

Project: k-Means in a simulated dataset

1. Use matplotlib to **plot the simulated dataset**
2. Use the implemented k-Means function to **cluster the dataset** (note: use $k = 3$)
3. Use matplotlib again, to **plot the k-Means clustering result** (use a different color for the points of each cluster)
4. At last, try to **plot the movement of the centroids** over the iterations
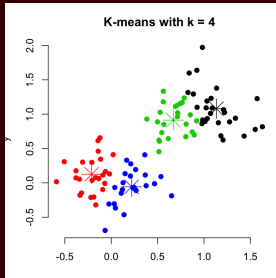
Project: k-Means in a simulated dataset (solution)

```python
1 # Now, to test it, we shall use a simulated dataset:
2 X, _ = additional_stuff.simulated_dataset()
3
4 # We can use matplotlib to visualise the simulated data points:
5 # First, we plot the data points:
6 pl.plot(X[:,0], X[:,1], 'x')
7 # Then, some additional stuff:
8 pl.title('Simulated Dataset')
9 pl.xlabel('x_1')
10 pl.ylabel('x_2')
11 pl.show()
12
13 # Now, we can apply our implementation of 'k-means' to this
       dataset:
14 membership, centroids, centroids_history = k_means(X, k=3)
```

Project: k-Means in a simulated dataset (solution)
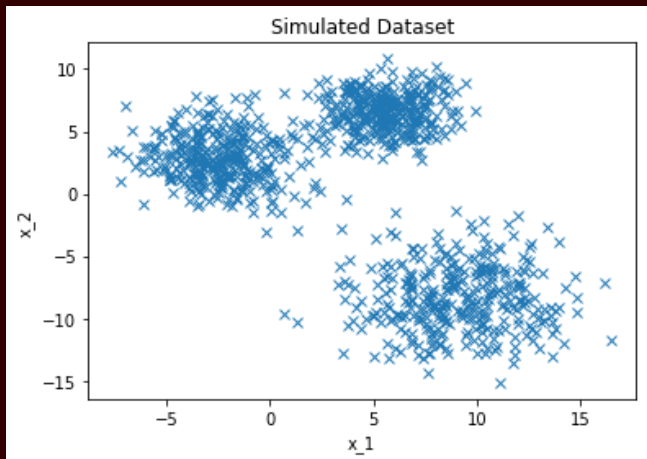
Project: k-Means in a simulated dataset (solution)

```
1 # Using matplotlib, we can plot the result:
2 # First, plot the data points:
3 colors = ['red', 'green', 'blue']
4 pl.scatter(X[:, 0], X[:, 1], marker='x', color=[colors[int(_y)]
     for _y in membership])
5 # Then, plot the final cluster centroids:
6 pl.plot(centroids[:,0], centroids[:,1], 'o', color='black')
7 # And some additional stuff:
8 pl.title('k-Means Clustering results')
9 pl.xlabel('x_1')
10 pl.ylabel('x_2')
11 pl.show()
```

Project: k-Means in a simulated dataset (solution)
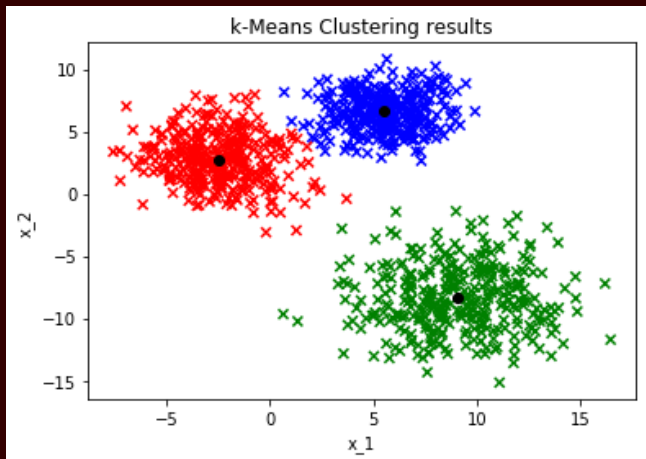
Project: k-Means in a simulated dataset (solution)

```
1 # We can also see the movement of each centroid during the
     iterations:
2 # Plot the data points again (use alpha to make them transparent)
     :
3 pl.scatter(X[:, 0], X[:, 1], marker='x', alpha=0.1, color=[colors
     [int(_y)] for _y in membership])
4 # And plot each centroid's movement:
5 pl.plot(centroids_history[:,0,0], centroids_history[:,0,1], '--',
     marker='o', color='red')
6 pl.plot(centroids_history[:,1,0], centroids_history[:,1,1], '--',
     marker='o', color='green')
7 pl.plot(centroids_history[:,2,0], centroids_history[:,2,1], '--',
     marker='o', color='blue')
8 # Then, plot the final cluster centroids:
9 pl.plot(centroids[:,0], centroids[:,1], 'o', color='black')
10 # And some additional stuff:
11 pl.title('k-Means Centroid Movement')
12 pl.xlabel('x_1')
13 pl.ylabel('x_2')
14 pl.show()
```

Project: k-Means in a simulated dataset (solution)
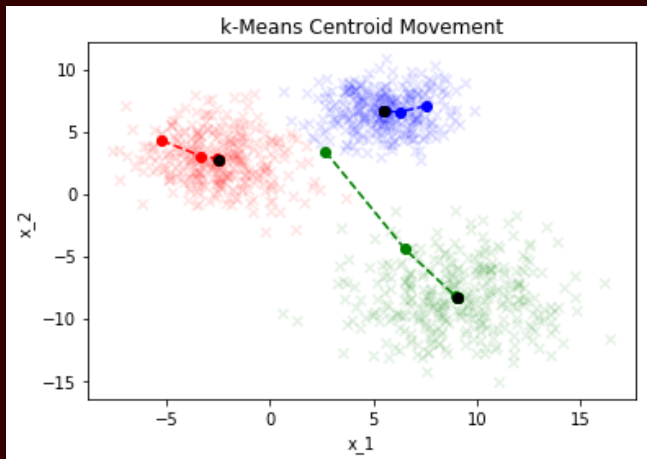
File Handling

Open files using Numpy

Open a CSV file and add some random noise.

```python
import numpy as np
d = np.loadtxt('file.csv', [np.float64, np.float64, np.int8],
               delimiter=',', skiprows=1)
X = d[:, :-1]
y = d[:, -1]
```

Open files using Numpy

Open a CSV file and add some random noise.

```
1 import numpy as np
2 d = np.loadtxt('file.csv', [np.float64, np.float64, np.int8],
3                delimiter=',', skiprows=1)
4 X = d[:, :-1]
5 y = d[:, -1]
```

By default numpy always works with `np.float64`. But you may want to change `dtype` when opening a CSV file or creating a vector in order to use less memory. The following types are available:

`np.int`	8, 16, 32 and 64 bits
`np.uint`	8, 16, 32 and 64 bits
`np.float`	16, 32 and 64 bits
`np.complex`	64 and 128 bits

Complex numbers are represented using two 32-bits or 64-bits floats.

By default numpy always works with `np.float64`. If you work with GPUs, it's a good idea to convert everything to 32-bits.

Open images using Scikit-image

```python
from skimage.io import imread
img = imread('filename.png')

print(img.shape)
# (512, 512, 3)  # if color (RGB)
```

Pandas

If you are used to R `data.frame` then you are going to miss things like accessing columns by column name.

There is a widely used data-frame package for Python called `pandas`.

(This package is particularly useful when dealing with time series because it supports a lot of timeseries functionality we are not going to cover.)

Opening is faster than numpy...

```python
import pandas as pd
df = pd.read_csv('titanic.csv')
df = pd.read_excel('a.xlsx')

df.groupby('gender').mean()
#                 age       height
# gender
# female   53.870850   158.201459
# male     53.491803   158.971695

df.to_csv('blabla.csv')
df.to_excel('blabla.xlsx')
```

Indexing

```python
df['nome-coluna']

df.loc['nome-linha']
df.ix[5]      # nro linha
df.ix[:, 5]   # linha,coluna
```

Save Session

There are two ways to save your session/variables to a file:
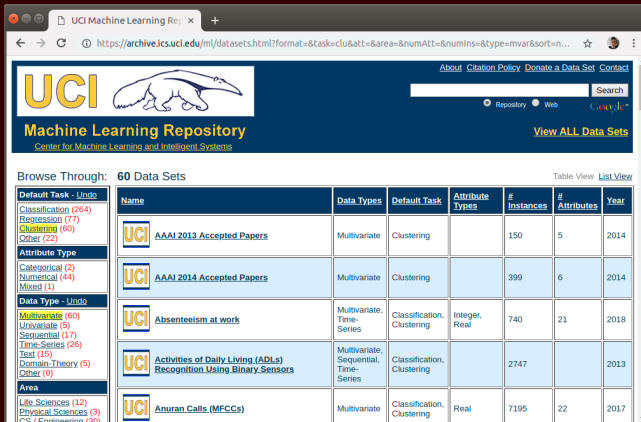
- `np.save`: numpy binary format
- `pickle`: binary file (python version-specific)
- `json`: text file (universal)

Pickle is the native Python approach:

```python
import pickle
pickle.dump(d, open('save.pickle', 'wb'))
# and then you can just load it again:
d = pickle.load(open('save.pickle', 'rb'))
```

Project: Use a real dataset

We are going to use the Iris dataset from "UCI Machine Learning"

It includes 150 samples of flowers of three classes (iris-setosa, iris-versicolor, iris-virginica) and includes measurements of sepal width and length and petal width and length.

Project: Use a real dataset

Tasks:

1. **Open 'iris.txt'** using pandas read_csv() function;
2. **Check the dataset** using Spyder's Variable Explorer;
3. Make a **numpy array with just two features** of the dataset;
4. Use k-means to **cluster the dataset** (use $k = 3$);
5. Encode the labels and **print the mutual information score**;
6. Use matplotlib to **plot the results**.

Project: Use a real dataset (solution)

```python
# We can also apply our k-means algorithm to a real dataset.
# To do that, we can use pandas to import the dataset:
data = pd.read_csv('iris.txt')

# To use k-means, let's select just two columns of the dataset:
X = np.array(data[['petal width', 'petal length']])

# And perform k-Means clustering with k = 3 (no. of classes)
membership, centroids, _ = k_means(X, k=3)

# Getting and encoding the labels:
# (...)

# And plot the results:
colors = ['red', 'green', 'blue']
pl.scatter(X[:, 0], X[:, 1], marker='x', color=[colors[int(_y)]
    for _y in membership])
pl.plot(centroids[:,0], centroids[:,1], 'o', color='black')
pl.title('Clustering Result')
pl.xlabel('Petal width (cm)')
pl.ylabel('Petal length (cm)')
pl.show()
```

# Project: Use a real dataset (solution)
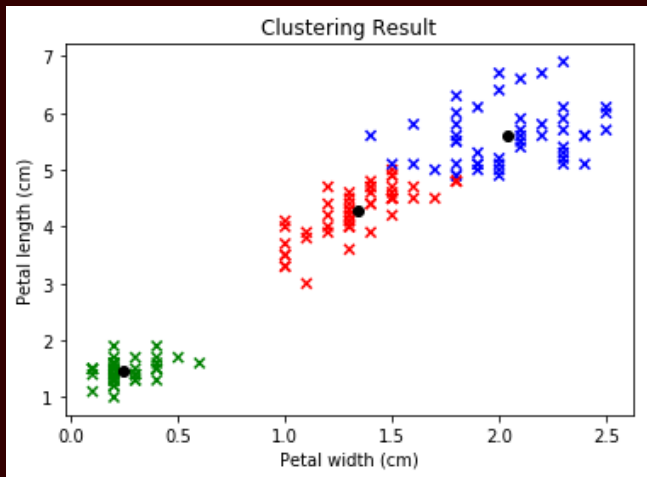
Project: Use an image

Tasks:

1. **Open the image** 'lemons.jpg' using scikit-image io.imread() function;
2. **Visualise the image** using matplotlib imshow() function;
3. Change the shape from $(w, h, c) \rightarrow (w \times h, c)$ with numpy's reshape() function;
4. Use **k-means on the reshaped image**;
5. Reshape the k-means result back: $(w \times h, c) \rightarrow (w, h, c)$;
6. **Visualise the result** using the imshow() function;
7. **Save the result** using scikit-image: io.imsave(filename, image).

Project: Use an image (solution)

```python
1 # First, we use scikit-image to import the image:
2 image = io.imread('lemons.jpg')
3
4 # Visualise the image using matplotlib:
5 pl.imshow(image)
6 pl.show()
7
8 # Each pixel should be considered a single sample (in this case,
       with three
9 # features - RGB). We need to restructure the image accordingly:
10 image_reshaped=image.reshape((image.shape[0]*image.shape[1], 3))
11
12 # Now, we finally apply k-means to it:
13 membership, _ , _ = k_means(image_reshaped, k=2)
14
15 # Reshaping and visualising the result:
16 result = membership.reshape((image.shape[0], image.shape[1]))
17 pl.imshow(result)
18 pl.show()
19
20 # Save the result as an image:
21 io.imsave('result.jpg', result)
```

Project: Use an image (solution)
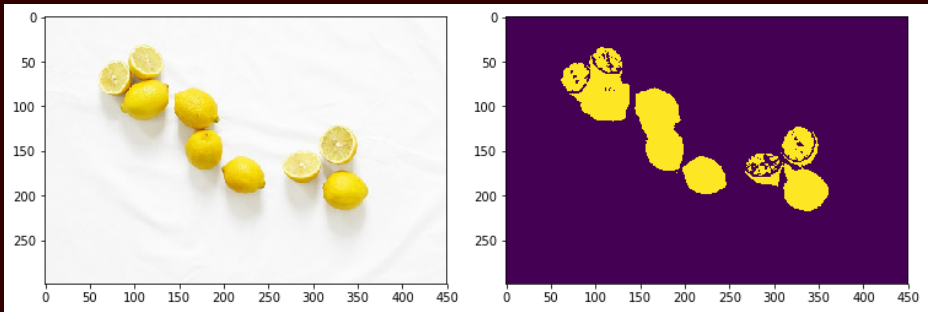
Classes & Modules

Classes

```
 1 class Animal:
 2   def talk(self):
 3     raise NotImplementedError('Implement me')
 4
 5 class Cat(Animal):
 6   def talk(self):
 7     print("Miau")
 8
 9 class Human(Animal):
10   def talk(self):
11     print("Bla bla")
12
13 Cat().talk()  # what is the output?
```

Two notes:

- Python is very dynamic: there are no formal contracts like abstract methods
- in Python, the reference to the object itself is passed explicitly

Modules

Different ways to access a module:

```
1 import fib
2 fib.whatever()
```

```
1 from fib import whatever
2 whatever()
```

```
1 from fib import *
2 whatever()
```

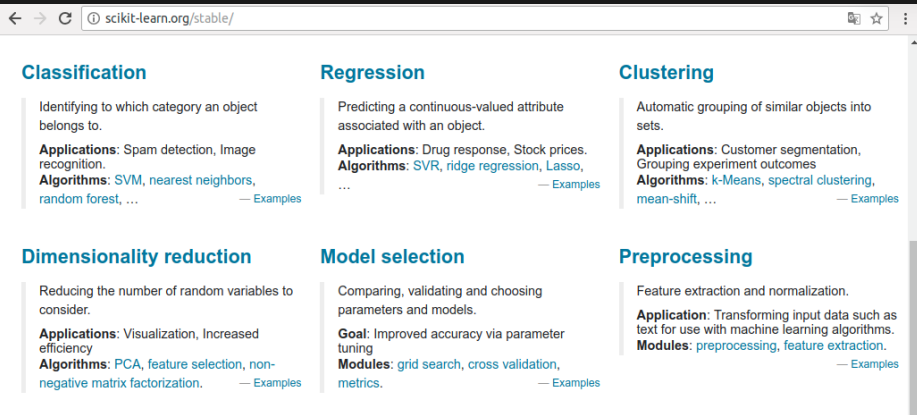(import everything into your namespace — usually not recommended)

Inside each directory, you ~~must~~ may have a `__init__.py` file where you can put initialization code.

Scikit-learn

Scikit-learn

Scikit-learn has literally **everything!**

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, …   — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, …
                                   — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, …   — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics.   — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.   — Examples

Scikit-learn interface

Estimator:	Implements the fit method to learn from data.
	For Supervised Learning:
| | ```
estimator = estimatorObj.fit(data, labels)
``` |
| | and Unsupervised Learning: |
| | ```
estimator = estimatorObj.fit(data)
``` |
| **Predictor:** | ```
labels = predictorObj.predict(data)
``` |
| | May implement predict_proba to return the degree of certainty. |
| **Transformer:** | Filters or modifies the data: |
| | ```
new_data = transformerObj.transform(data)
``` |
| **Model:** | Measures goodness of fit: |
| | ```
score = modelObj.score(data, labels)
``` |

Project: Apply K-means with sklearn

Tasks:

1. Find the sklearn documentation for the K-means algorithms;
2. Go to the code you have previously written for the Iris dataset and substitute your k-means function by the KMeans object of sklearn. You should:
   - **2.1** Create the object.
   - **2.2** Estimate the parameters with the data (fit(.)).
   - **2.3** For each sample predict the assigned cluster (predict(.)).
3. Plot and check you have obtained an equivalent result.

Project: Apply K-means with sklearn

Tasks:

1. Find the sklearn documentation for the K-means algorithms;
2. Go to the code you have previously written for the Iris dataset and substitute your k-means function by the KMeans object of sklearn. You should:
   - 2.1 Create the KMeans object.
   - 2.2 Estimate the parameters with the data (fit(.)).
   - 2.3 For each sample predict the assigned cluster (predict(.)).
3. Plot and check you have obtained an equivalent result.

Solution:

```
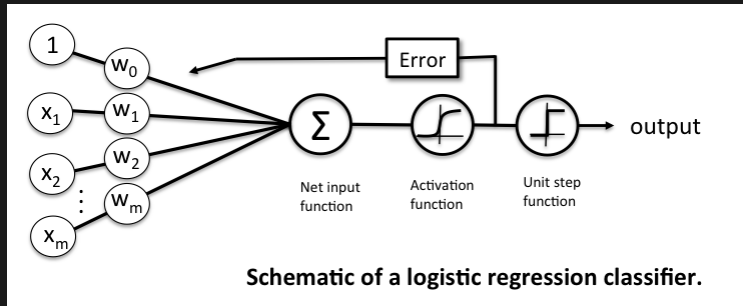1    from sklearn.cluster import KMeans as kmeans
2    model = kmeans(n_clusters=3)
3    model.fit(X)
4    membership = model.predict(X)
5    centroids = model.cluster_centers_
```

Classification with Logistic Regression

Schematic of a logistic regression classifier.

Project: Apply Logistic Regression for Classification

We now focus on the problem of categorizing data.

1. Find the sklearn documentation for the K-means algorithms;
2. Split the data into training and testing.
3. Create the LogisticRegression object and estimate the parameters with the training data.
4. Obtain the accuracy of the model in the training and testing data (score(.)).

Further exercises:

5. Run the same code again. Did you obtain the same result?
6. Try the same exercise with another classifier.

Useful functions: `train_test_split`, `LogisticRegression`, `fit`, `score`.

Project: Apply Logistic Regression for Classification (solution)

Solution:

```python
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(X, y)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

print("Train set: ", logreg.score(X_train, y_train))
print("Test set: ", logreg.score(X_test, y_test))
```

Note: If you want to obtain the same exact results you need to remove the randomness in the process. Some common sources of variation are the train and test split and the initial parameters of the logistic regression.

Note: Because different classifiers have the same interface, we can simply substitute the 3rd line by another model. In this way we can quickly estimate the performance of multiple models for any machine learning problem we want to solve.

Plotting the Feature Space

Using the provided function (plot_fs()) plot the feature space. This function might be useful for other pojects!
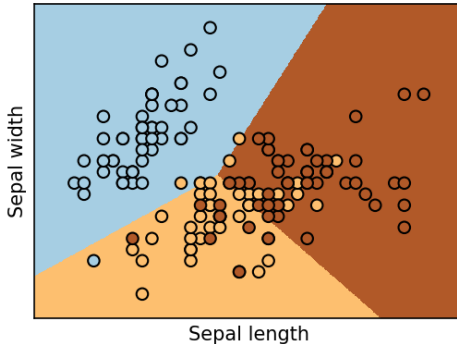
Keras & TensorFlow

Keras

Keras is a high-level neural networks API.

Models in Keras (and Tensorflow) are defined as a graph of operations which transform the data from the input to the output. The simplest possible model in Keras is a `Sequential` which represents a linear stack of operations. We construct the model by adding layers:

```python
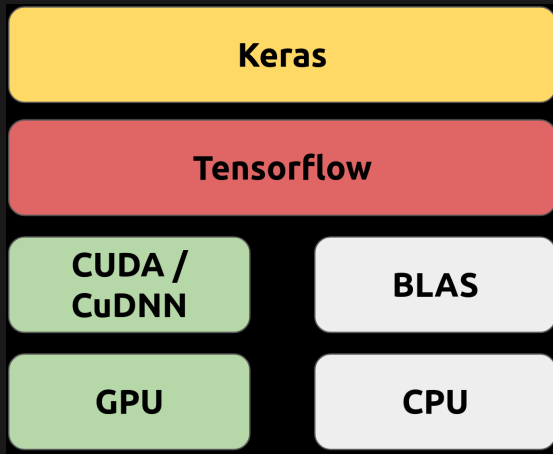from keras.layers import Dense
from keras.models import Sequential
model = keras.Sequential()
model.add(Dense(64, activation='relu', input_dim=100))
model.add(Dense(10, activation='softmax'))
```

After creating the model we need to compile it before we start the training process.

```python
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

Keras

After a Keras model is compiled its interface is similar to models in sklearn:

<table>
<tr><th>Sklearn</th><th>Keras</th></tr>
</table>

Fit the training data:

```
1    model.fit
```

```
1    model.fit
```

Predict the output for input data:

```
1    model.predict
```

```
1    model.predict
```

Evaluate the performance for input data:

```
1    model.score
```

```
1    model.evaluate
```

Logistic Regression in Keras

A Logistic Regression classifier can easily be implemented in Keras as a one layer neural network. For :

```
1    model = Sequential()
2    model.add(Dense(n_categories, input_dim=n_inputs, activation=
         'softmax'))
3    # The model is optimized by stochastic gradient descent with
         momentum
4    sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
5    model.compile(optimizer=sgd, loss='categorical_crossentropy',
         metrics=['accuracy'])
```

Note that although we are doing logistic regression the structure and optimization of this model is different from the sklearn's logistic regression. More on this will be explained on the lecture about logistic regression.

Project: Implement Logistic Regression in Keras

We now focus on the problem of categorizing data.

1. Find the keras documentation for the sequential model;
2. Go to the code you have previously written for classification of the Iris dataset and substitute the sklearn implementation of the logistic regression by your own keras implementation.
3. Fit the training data and obtain the accuracy for the train and test data.
4. Obtain a plot of the feature space (use plot_fs())

When to use sklearn vs Keras

Keras is used to work with neural networks. It allows faster training due to using GPU accelaration. Generally, if you are working with `Deep Learning` models you will be using Keras or a similar module. For traditional machine learning algorithms you use sklearn.

In this course sklearn will be more useful.

MNIST in Keras Demo

An example where modules like keras are used often: Image Recognition with Convolutional Neural Networks.

```
1    model = Sequential()
2    model.add(Conv2D(32, kernel_size=(3, 3),
             activation='relu',input_shape=
             input_shape))
3    model.add(Conv2D(64, (3, 3), activation='
             relu'))
4    model.add(MaxPooling2D(pool_size=(2, 2)))
5    model.add(Dropout(0.25))
6    model.add(Flatten())
7    model.add(Dense(128, activation='relu'))
8    model.add(Dropout(0.5))
9    model.add(Dense(num_classes, activation='
             softmax'))
10   model.compile(loss=keras.losses.
             categorical_crossentropy, optimizer=
             keras.optimizers.Adadelta(), metrics
             =['accuracy'])
11   model.fit(x_train, y_train,
12       batch_size=batch_size,
13       epochs=epochs,
14       verbose=1,
15       validation_data=(x_test, y_test))
16   score = model.evaluate(x_test, y_test,
             verbose=0)
```


Logistic Regression: 81%
CNN: 99%

Finishing Touches

Pointers

References

In programming languages when calling a function, there are two possible behaviors:

- pass by value (or copy)
- pass by reference

Python uses what some people like to call 'pass-by-object-reference'. Everything is an object and the reference to the object is copied (not the object itself).

```python
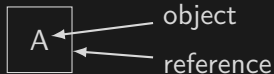1 def fn(b):
2   print(id(b))
3   print(a is b)
4 a = []
5 print(id(a))
6 fn(a)
```

Output:

```
1 139663236727240
2 139663236727240
3 True
```


object

reference

References

```
1 a = [1 ,2 ,3]
2 b = a
3 b [1] = 7
4 print(a)
```

What is the value of a?

```
1 a = [1 ,2 ,3]
2 b = a
3 b = [1 ,7 ,3]
4 print(a)
```

What is the value of a?

```
1 a = 2
2 b += 5
3 print(a)
```

What is the value of a?

Python2 vs Python3

Python2 will be discontinued in 2020. The major differences are that:

- In Python3, functions like `range()` are generators/iterators, while in Python2 they are lists. In most use cases, this will make no difference to you.

- In Python3, strings are codified in Unicode by default. This makes it nicer to work with when using, for instance, Portuguese characters like ç or á.

- Probably the when you'll notice the most is that in Python2 `print 'Hello World'` was valid. In Python3, you need to use parenthesis, `print('Hello World')`

- There are many other small ones. **Most importantly, not all packages are yet compatible with Python3.** (Though 99% are.)

Python Implementations

There are several implementations of the Python language:

- CPython
  - $\boxed{\text{source code}} \xrightarrow{\text{is compiled to}} \boxed{\text{bytecode}} \rightarrow \text{execute}$
  - CPython is the most popular implementation of Python, and is developed by the Python Foundation
  - This is the implementation people *mean* when they say Python

- PyPy
  - $\boxed{\text{source code}} \rightarrow \boxed{\text{bytecode}} \rightarrow \text{execute} \leftrightarrow \boxed{\text{machine code}}$
  - This is also a very popular and highly supported implementation
  - Faster; this is how Java and MATLAB work

- Jython
  - $\boxed{\text{source code}} \rightarrow \boxed{\text{Java bytecode}}$
  - The Java interpretor: $\boxed{\text{bytecode}} \rightarrow \text{execute} \leftrightarrow \boxed{\text{machine code}}$
  - It outsources work to Java

PEP8

Using a pretty style makes reading your code much easier.

In Python, there is a coding style that is recommended called PEP8.

Exercise: Enable code style checking in Spyder and fix the errors.

Under the Preferences window:

Ricardo Cruz   rpcruz@inesctec.pt
Eduardo Castro   emcastro@inesctec.pt
João Ribeiro Pinto   jtpinto@fe.up.pt