MUIInf
Máster Oficial Universitario en
Ingeniería Informática
muiinf.webs.upv.es

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
etsinf

Data Science (CDA)

Practical 9:

Clustering (with python)

José Hernández Orallo (jorallo@dsic.upv.es)
(adapting material from Cèsar Ferri).

In this practical, we will use the library "clusters" and several cluster methods from it.

Required files (on the poliformat):

- clusters.py
- blogdata.txt
- zebo.txt

Other requirements

- pillow library (PIL), as it is used by clusters.py. If not installed, this can be installed with "pip install pillow" (e.g., Windows).
  - In windows, for python 2.7, this would be:
    …\Pyhon27\Scripts\pip install pillow
  - In windows, for python 3.x, this would be (some code not tested on 3.x):
    …\Pyhon38\Scripts\pip install pillow

Additional files (not really needed)

- generatefeedvector.py
- feedlist.txt

Dataset: "blogdata.txt". Data from blogs:

RSS (Really Simple Syndication): employs a family of standard web feed formats to publish frequently updated information: blog entries, news headlines, audio, video...

There is a Python package for dealing with RSS (universal feed parser), found here: https://pypi.python.org/pypi/feedparser

With the file, generatefeedvector.py, we can collect a list of blogs (feedlist.txt) and download the last posts into a structure (a bag of words representation).

As we don't have time to run the program to download the information, you have some old results of the program in a file known as "blogdata.txt".

A bag of words is just a dataset where the columns represent how many times the word appears in a document

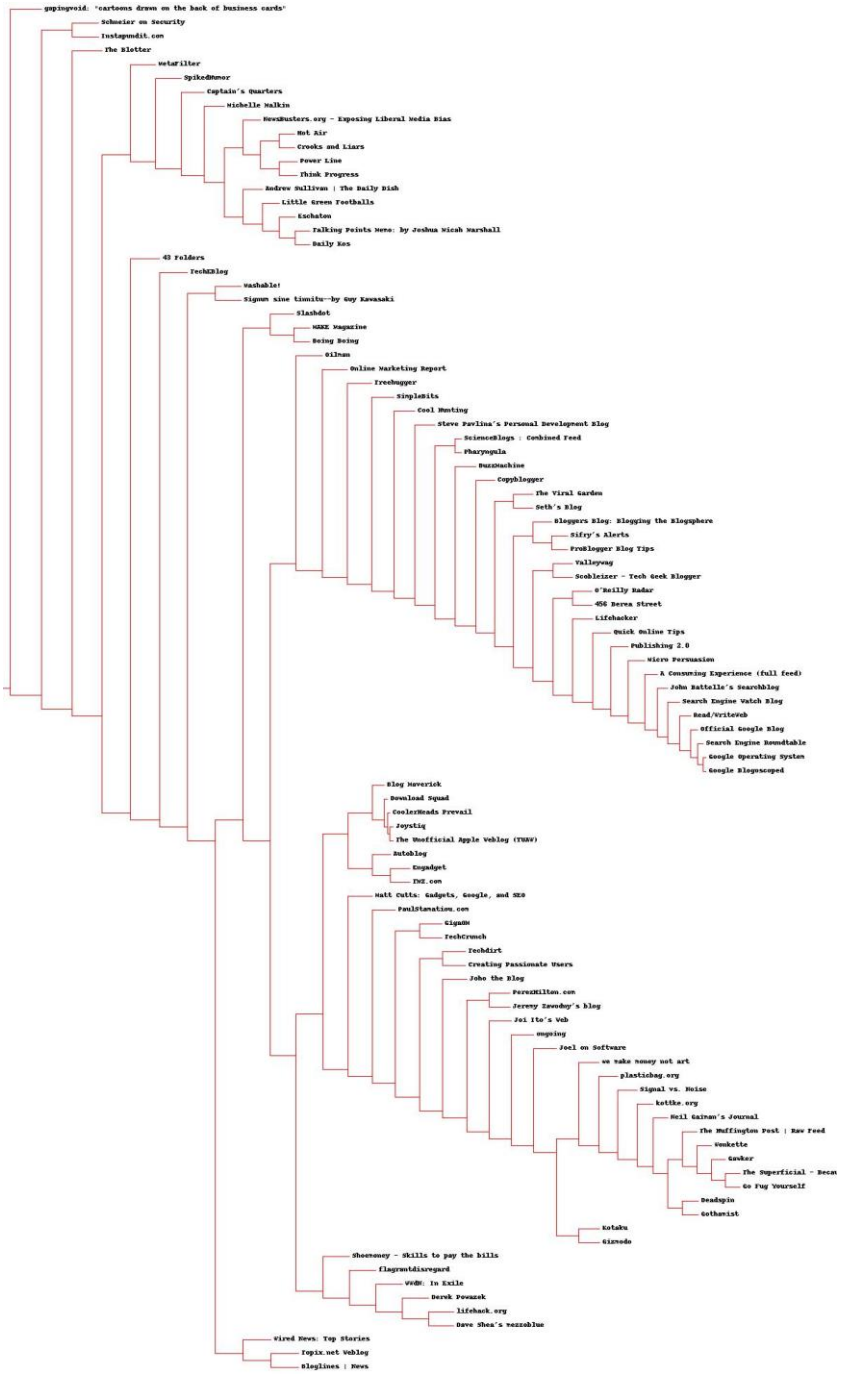
We're going to analyse that data. For instance, here we see three blogs and four words.

	"china"	"kids"	"music"	"yahoo"
Gothamist	0	3	3	0
GigaOM	6	0	0	2
Quick Online Tips	0	2	2	22

Bag of words usually have many columns, including thousands of words of a language (stop words such as "the", "a", "is", etc., are usually discarded).

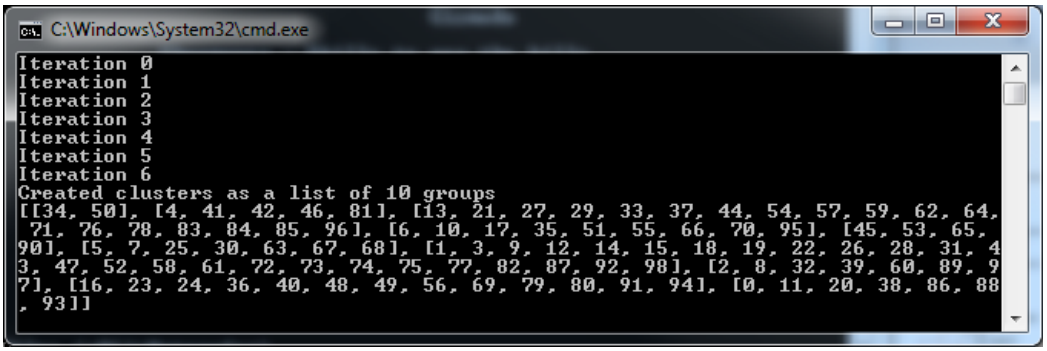
Exercise 1

Construct a dendrogram (hcluster) using the "blogdata.txt" data. Show the dendrogram on the screen and also export it to a file.
You should get something like this:



Exercise 2

Generate a non-hierarchical clustering using kmeans (function "kcluster"), with k=10. You should get something like this:



Multidimensional scaling:

With the same "blogdata.txt" data, we are going to show the data using multidimensional scaling, a technique for visualising the level of similarity of individual cases of a dataset in a 2-dimensional plot. In particular, it can show the information contained in a distance matrix.

In other words, we calculate the distance on the n-dimensional space of words and then we try to map this onto a 2D space using multidimensional scaling, so that similar objects appear together.

The python package "clusters" also has multidimensional scaling, in particular the following algorithm:

1. Plot randomly the elements in a 2D space
2. Compute distances between elements in the 2D space
3. Compute real distances between elements
4. Compare real distance and distance in the plot (error)
5. Move elements in order to reduce error
6. Repeat 4-5 until convergence

Exercise 3

Show the multidimensional scaling, using the function “scaledown” and export it to a file.
We should get something as follows:

Transposing the data:

Sometimes we are interested in clustering words instead of blogs. We can invert the bag of words (swap rows and columns).

There is a function in clusters.py that transposes the data.

```
def rotatematrix(data):
  newdata=[]
  for i in range(len(data[0])):
    newrow=[data[j][i] for j in range(len(data))]
    newdata.append(newrow)
  return newdata
```

Exercise 4

Use the above function to create another dendrogram but now clustering words instead of blogs. The dendrogram will be massive.

Also apply kmeans to this data.

Dataset: "zebo.txt". Data from social networks:

zebo.com was a social network where members showed the items/goals they desire. With "Beautiful Soup", a python library for scraping web pages, we obtained the following zebo data: "zebo.txt", containing 500 users and 35 objects.

As we want to clusters the users depending on the items they desire we need a good distance between the vector of 35 objects.

For boolean attributes a common metric is Tanimoto distance (Jaccard Index). It is is defined as the size of the intersection divided by the size of the union of the sample sets.

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

This is the definition of the Tanimoto distance found in cluster.py (under the misspelt name tanamoto):

```
def tanamoto(v1,v2):
  c1,c2,shr=0,0,0

  for i in range(len(v1)):
    if v1[i]!=0: c1+=1 # in v1
    if v2[i]!=0: c2+=1 # in v2
    if v1[i]!=0 and v2[i]!=0: shr+=1 # in both

  return 1.0-(float(shr)/(c1+c2-shr))
```

Exercise 5

Perform a dendrogram using the Tanimoto distance.

Something like this is expected: