ACM

February 5, 2021

In [2]:
```
#chargement des donnees - index_col = 0 pour indiquer que la colonne nř0 est un label
import pandas
D = pandas.read_excel("ACM.xlsx",sheet_name="ACM",index_col=0)
```

In [3]:
```
#affichage des caracteristiques
print(D.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8 entries, Beauceron to Labrador
Data columns (total 3 columns):
Taille       8 non-null object
Velocite     8 non-null object
Affection    8 non-null object
dtypes: object(3)
memory usage: 256.0+ bytes
None
```

In [5]:
```
#recuperation des variables actives
DActives = D[['Taille','Velocite','Affection']]
print(DActives)
```

```
              Taille Velocite Affection
Chien
Beauceron    Taille++  Veloc++    Affec+
Basset         Taill-    Velo-     Affe-
Berger-All   Taille++  Veloc++    Affec+
Boxer         Taille+   Veloc+    Affec+
Bull-Dog       Taill-    Velo-    Affec+
Bull-Mastif  Taille++    Velo-     Affe-
Caniche        Taill-   Veloc+    Affec+
Labrador      Taille+   Veloc+    Affec+
```

In [6]:
```
#recuperation des infos -
#nombre de variables
p = DActives.shape[1]
#nombre d'observations
n = DActives.shape[0]
```

In [7]:
```
#codage en 0/1  les noms de modalites sont explicites
#pas necessaire de prefixer les indicatrices par les noms de variables
X = pandas.get_dummies(DActives,prefix='',prefix_sep='')
print(X)
```

```
             Taill-  Taille+  Taille++  Velo-  Veloc+  Veloc++  Affe-  Affec+
Chien
Beauceron         0        0         1      0       0        1      0       1
Basset            1        0         0      1       0        0      1       0
Berger-All        0        0         1      0       0        1      0       1
Boxer             0        1         0      0       1        0      0       1
Bull-Dog          1        0         0      1       0        0      0       1
Bull-Mastif       0        0         1      1       0        0      1       0
Caniche           1        0         0      0       1        0      0       1
Labrador          0        1         0      0       1        0      0       1
```

In [9]:
```
#librairie numpy
import numpy
#profil individu moyen
ind_moy = numpy.sum(X.values,axis=0)/(n*p)
print(ind_moy)
```

```
[ 0.125       0.08333333  0.125       0.125       0.125       0.08333333
  0.08333333  0.25      ]
```

In [11]:
```
#distance du KHI-2 entre beauceron (nř0) et basset (nř1)
print(numpy.sum(1/ind_moy*(X.values[0,:]/p-X.values[1,:]/p)**2))

#idem entre basset(nř1) et caniche(nř6)
print(numpy.sum(1/ind_moy*(X.values[1,:]/p-X.values[6,:]/p)**2))
```

```
5.77777777778
3.55555555556
```

In [13]:
```
#distance a l'origine du basset (nř1)
print(numpy.sum(1/ind_moy*(X.values[1,:]/p-ind_moy)**2))
#distance a l'origine du caniche (nř6)
print(numpy.sum(1/ind_moy*(X.values[6,:]/p-ind_moy)**2))
```

```
2.11111111111
1.22222222222
```

In [14]:
```
#distance a l'origine des individus (obs.)
disto_ind = numpy.apply_along_axis(arr=X.values,axis=1,func1d=lambda x:numpy.sum(1 /i
#poids des obs.
```

```
poids_ind = numpy.ones(X.shape[0])/n
#inertie
inertie_ind = poids_ind*disto_ind
#afffichage
print(pandas.DataFrame(numpy.transpose([poids_ind,disto_ind,inertie_ind]),index=D.inde
#poids (1/n), leurs distances a lorigine, leurs inerties [1/2()]
```

```
                0         1         2
Chien
Beauceron    0.125  1.666667  0.208333
Basset       0.125  2.111111  0.263889
Berger-All   0.125  1.666667  0.208333
Boxer        0.125  1.666667  0.208333
Bull-Dog     0.125  1.222222  0.152778
Bull-Mastif  0.125  2.111111  0.263889
Caniche      0.125  1.222222  0.152778
Labrador     0.125  1.666667  0.208333
```

In [15]:
```
#inerrie totale
inertie_tot_ind = numpy.sum(inertie_ind)
print(inertie_tot_ind)
```

```
1.66666666667
```

In [ ]:
```
#Linformation totale portee par les donnees, comptabilisee via les distances entre ind
```

In [ ]:
```
# 2 Analyse des associations entre les modalites
```

In [ ]:
```
#Distance entre modalites
```

In [17]:
```
#somme en colonne
somme_col = numpy.sum(X.values,axis=0)
print(somme_col)
```

```
[3 2 3 3 3 2 2 6]
```

In [78]:
```
# avec les listes , slicing ne fonctionne pas
M=[[1,2,3],[4,5,6],[7,8,9]]
M[: , 0]
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-78-57336eea6fa7> in <module>()
```

```
      1 # avec les listes , slicing ne fonctionne pas
      2 M=[[1,2,3],[4,5,6],[7,8,9]]
----> 3 M[: , 0]


TypeError: list indices must be integers or slices, not tuple
```

In [76]:
```
# on doit travailler avec array (Tableau) ( ca marche tres bien)
M=numpy.array([[1,2,3],[4,5,6],[7,8,9]])
M[: , 0]
```

Out[76]:
```
array([1, 4, 7])
```

In [18]:
```
#distance entre taille- (2) et velocite- (5)
print(numpy.sum(n*((X.values[:,2]/somme_col[2]-X.values[:,5]/somme_col[5])**2)))
```

```
1.33333333333
```

In [19]:
```
#distance entre taille- (2) et velocite+ (3)
print(numpy.sum(n*((X.values[:,2]/somme_col[2]-X.values[:,3]/somme_col[3])**2)))
```

```
3.55555555556
```

In [ ]:
```
#Distance a lorigine
```

In [21]:
```
#profil moyen des variables-modalites
moda_moy = numpy.ones(X.shape[0])/n
#distance a l'origine taille-
print(numpy.sum(n*((X.values[:,0]/somme_col[0]-moda_moy)**2)))
```

```
1.66666666667
```

In [22]:
```
#distance a l'origine taille+
print(numpy.sum(n*((X.values[:,1]/somme_col[1]-moda_moy)**2)))
```

```
3.0
```

In [ ]:
```
#Inertie totale
```

In [25]:
```
#poids des variables_modalites (points modalites)
poids_moda = somme_col/(n*p)
#distance a l'origine des points modalites
disto_moda = numpy.apply_along_axis(arr=X.values/somme_col,axis=0,func1d=lambda x: nu
#inertie
inertie_moda = poids_moda * disto_moda
#affichage
print(pandas.DataFrame(numpy.transpose([poids_moda,disto_moda,inertie_moda]),index =X
```

```
             Poids     Disto   Inertie
Taill-    0.125000  1.666667  0.208333
Taille+   0.083333  3.000000  0.250000
Taille++  0.125000  1.666667  0.208333
Velo-     0.125000  1.666667  0.208333
Veloc+    0.125000  1.666667  0.208333
Veloc++   0.083333  3.000000  0.250000
Affe-     0.083333  3.000000  0.250000
Affec+    0.250000  0.333333  0.083333
```

In [ ]:
```
#Remarque
#La valeur de linertie totale obtenue via le point de vue des modalites est strictemen
#exprimee a partir de lanalyse des proximites entre les individus .
#Nous avons deux manieres dapprehender les donnees, mais il sagit bien des memes donne
#Nous constatons de surcroit que les modalites (Taille+, Veloc++, Affec-) sont celles
#contribuent le plus a linformation globale.
```

In [27]:
```
#inertie totale des points-modalites
inertie_tot_moda = numpy.sum(inertie_moda)
print(inertie_tot_moda)
```

```
1.66666666667
```

In [24]:
```
#ACM via une ACP sur le tableau des profils
```

In [ ]:
```
#Nous reduisons les variables par la racine carree de leur variance c.-a-d. lecart-typ
```

In [29]:
```
profil = numpy.apply_along_axis(arr=X.values,axis=1,func1d=lambda x:x/numpy.sum(x))
print(pandas.DataFrame(profil,index=X.index,columns=X.columns))
```

```
              Taill-   Taille+  Taille++     Velo-    Veloc+   Veloc++  \
Chien
Beauceron    0.000000  0.000000  0.333333  0.000000  0.000000  0.333333
Basset       0.333333  0.000000  0.000000  0.333333  0.000000  0.000000
Berger-All   0.000000  0.000000  0.333333  0.000000  0.000000  0.333333
Boxer        0.000000  0.333333  0.000000  0.000000  0.333333  0.000000
Bull-Dog     0.333333  0.000000  0.000000  0.333333  0.000000  0.000000
Bull-Mastif  0.000000  0.000000  0.333333  0.333333  0.000000  0.000000
Caniche      0.333333  0.000000  0.000000  0.000000  0.333333  0.000000
Labrador     0.000000  0.333333  0.000000  0.000000  0.333333  0.000000

                Affe-    Affec+
Chien
Beauceron    0.000000  0.333333
Basset       0.333333  0.000000
Berger-All   0.000000  0.333333
Boxer        0.000000  0.333333
```

```
Bull-Dog     0.000000  0.333333
Bull-Mastif  0.333333  0.000000
Caniche      0.000000  0.333333
Labrador     0.000000  0.333333
```

```
In [30]: #Nous reduisons les variables par la racine carree de leur variance c.-a-d. lecart-ty
         #reduire les profils
         profil = profil/numpy.std(profil,axis=0,ddof=0)
         print(profil)
```

```
[[ 0.          0.          2.06559112  0.          0.          2.30940108
   0.          2.30940108]
 [ 2.06559112  0.          0.          2.06559112  0.          0.
   2.30940108  0.        ]
 [ 0.          0.          2.06559112  0.          0.          2.30940108
   0.          2.30940108]
 [ 0.          2.30940108  0.          0.          2.06559112  0.          0.
   2.30940108]
 [ 2.06559112  0.          0.          2.06559112  0.          0.          0.
   2.30940108]
 [ 0.          0.          2.06559112  2.06559112  0.          0.
   2.30940108  0.        ]
 [ 2.06559112  0.          0.          0.          2.06559112  0.          0.
   2.30940108]
 [ 0.          2.30940108  0.          0.          2.06559112  0.          0.
   2.30940108]]
```

```
In [31]: #ponderation des modalites
         pond_moda = (n-somme_col)/(n*p)
         print(pond_moda)
```

```
[ 0.20833333  0.25        0.20833333  0.20833333  0.20833333  0.25        0.25
  0.08333333]
```

```
In [32]: #appliquer la ponderation aux profils
         profil = profil*numpy.sqrt(pond_moda)
         print(profil)
```

```
[[ 0.          0.          0.94280904  0.          0.          1.15470054
   0.          0.66666667]
 [ 0.94280904  0.          0.          0.94280904  0.          0.
   1.15470054  0.        ]
 [ 0.          0.          0.94280904  0.          0.          1.15470054
   0.          0.66666667]
 [ 0.          1.15470054  0.          0.          0.94280904  0.          0.
   0.66666667]
```

```
 [ 0.94280904  0.          0.          0.94280904  0.          0.          0.
   0.66666667]
 [ 0.          0.          0.94280904  0.94280904  0.          0.
   1.15470054  0.        ]
 [ 0.94280904  0.          0.          0.          0.94280904  0.          0.
   0.66666667]
 [ 0.          1.15470054  0.          0.          0.94280904  0.          0.
   0.66666667]]
```

In [ ]:
```
#numpy.linalg.eig
#
```

In [35]:
```
R=numpy.corrcoef(X.values,rowvar=False)
```

In [36]:
```
pr=numpy.linalg.eig(R)
print(pr)
```

```
(array([  3.56910019e+00,   2.65189894e+00,   1.27835795e+00,
        -1.50908880e-16,   3.40048343e-01,   1.60594584e-01,
         7.53627951e-17,   0.00000000e+00]), array([[ -1.97572068e-01,   3.51721627e-01,   6.3
         -5.97614305e-01,   2.23488183e-01,   1.75104646e-01,
          5.30093358e-02,   3.01605041e-17],
       [  3.62565037e-01,   2.39130868e-01,  -4.89295285e-01,
         -5.34522484e-01,  -3.89068180e-01,   3.67239448e-01,
          4.74129913e-02,   6.69823572e-17],
       [ -1.26715960e-01,  -5.65606777e-01,  -1.94425636e-01,
         -5.97614305e-01,   1.24504976e-01,  -5.03573594e-01,
          5.30093358e-02,  -4.71717356e-17],
       [ -4.71079787e-01,   1.79135691e-01,   2.21190999e-02,
          4.00891461e-16,  -5.77583745e-01,  -2.34006718e-01,
         -5.95258656e-01,  -1.51851997e-17],
       [  3.96335443e-01,   3.37794725e-01,  -1.64530188e-01,
          1.14945139e-16,   5.10772644e-01,  -2.89328891e-01,
         -5.95258656e-01,  -2.94097915e-17],
       [  8.35667175e-02,  -5.77945774e-01,   1.59220436e-01,
         -1.24706289e-15,   7.46970818e-02,   5.85106998e-01,
         -5.32415528e-01,   5.76014892e-17],
       [ -4.62316770e-01,   9.74194769e-02,  -3.67676828e-01,
         -4.65480057e-16,   3.02577398e-01,   2.23681287e-01,
          1.84450459e-16,   7.07106781e-01],
       [  4.62316770e-01,  -9.74194769e-02,   3.67676828e-01,
          4.65480057e-16,  -3.02577398e-01,  -2.23681287e-01,
          9.59407461e-17,   7.07106781e-01]]))
```

In [38]:
```
t=sorted(pr[0],reverse=True)
v=t/sum(t)
w=numpy.cumsum(v)
```

```
        t=[t,v,w]
        print(pandas.DataFrame(numpy.transpose(t),columns=['Val.P', '%','cumsum %'],index=rang
```

```
          Val.P             %  cumsum %
0  3.569100e+00  4.461375e-01  0.446138
1  2.651899e+00  3.314874e-01  0.777625
2  1.278358e+00  1.597947e-01  0.937420
3  3.400483e-01  4.250604e-02  0.979926
4  1.605946e-01  2.007432e-02  1.000000
5  7.536280e-17  9.420349e-18  1.000000
6  0.000000e+00  0.000000e+00  1.000000
7 -1.509089e-16 -1.886361e-17  1.000000
```

```
In [ ]: #LACP ne sait pas quil y a des redondances artificielles dans les
        #donnees, dou les facteurs aux variances nulles en excedent.
```

```
In [40]: #coordonnees fact. des observations (1er plan)
         print(pandas.DataFrame(pr[1][:,:2],index=X.index,columns=['Fact.1','Fact.2']))
```

```
              Fact.1    Fact.2
Chien
Beauceron   -0.197572  0.351722
Basset       0.362565  0.239131
Berger-All  -0.126716 -0.565607
Boxer       -0.471080  0.179136
Bull-Dog     0.396335  0.337795
Bull-Mastif  0.083567 -0.577946
Caniche     -0.462317  0.097419
Labrador     0.462317 -0.097419
```