1 Introduction

Utilisation de l'outil Pipeline du package « scikit-learn » sous Python. Illustration avec la méthode DISQUAL de discrimination sur variables qualitatives (Saporta, 2006).

En inventoriant le package « fanalysis » d'Olivier Garcia dédié à l'analyse factorielle (ACP, AFC et ACM) sous Python, mon attention a été attirée par l'outil Pipeline du package « scikit-learn » mis en avant lors de la présentation de l'ACM (analyse des correspondances multiples). Un Pipeline est un méta-opérateur qui permet d'enchaîner plusieurs calculs, pourvu que les classes mises à contribution implémentent les fonctions **fit()** (apprentissage) et **transform()** (projection). Les mécanismes de classes de Python et la forte cohérence des objets de « scikit-learn » font merveille ici. Cette notion d'opérateur encapsulant plusieurs autres qui s'exécutent séquentiellement n'est pas sans rappeler les metanodes dans des logiciels de data mining tels que Knime. J'avais pu en explorer le fonctionnement lors de la programmation de la validation croisée par exemple.

Nous nous appuierons sur l'étude de la méthode DISQUAL de Gilbert Saporta pour montrer l'intérêt de la classe Pipeline de « scikit-learn ». DISQUAL (discrimination sur variables qualitatives) permet de réaliser une analyse discriminante prédictive sur des variables explicatives qualitatives en faisant succéder deux techniques statistiques : dans un premier temps, une ACM est opérée sur les descripteurs, nous obtenons une description des données dans un espace factoriel ; dans un second temps, on lance une analyse discriminante linéaire (ADL), expliquant la variable cible à partir des facteurs de l'ACM. DISQUAL cumule un double avantage : elle rend réalisable l'analyse discriminante linéaire dans une configuration qu'elle ne sait pas appréhender nativement (explicatives qualitatives) ; on peut en moduler les propriétés de régularisation, et donc la robustesse au surapprentissage, en jouant sur le nombre de facteurs de l'ACM à retenir pour l'analyse discriminante.

On note surtout dans le contexte de ce tutoriel que DISQUAL est constituée deux techniques statistiques qui se succèdent (ACM + ADL). L'utilisation de Pipeline est complètement indiquée.

2 La méthode DISQUAL

Avant d'étudier l'outil Pipeline, nous présentons l'approche DISQUAL en détaillant les étapes. Nous travaillerons sous Tanagra / Excel tout d'abord, puis nous reproduisons les calculs sous Python.

2.1 Données

Nous utilisons les données des « votes au congrès » (Congressional Voting Records Data Set). L'objectif est d'identifier le groupe ("group") d'appartenance politique des parlementaires ("democrat" ou "republican") à partir de leurs votes sur différents thèmes qui leur sont soumis. Les valeurs possibles sont "y" (qui synthétise les situations « voted for, paired for, and announced for »), "no" (« voted against, paired against, and announced against »), "other" (« voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known »). Dans un premier temps, nous travaillons sur une version réduite de la base avec $p = 4$ variables explicatives. Nous disposons de $n = 435$ observations.

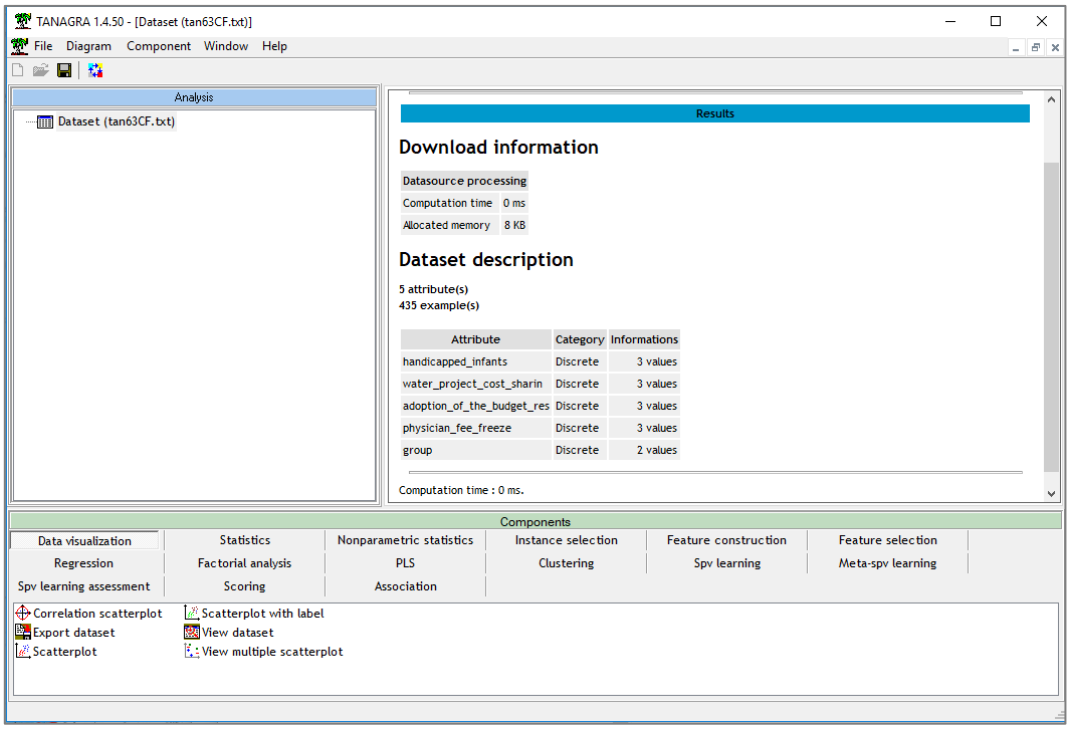
2.2 Détail des calculs avec Tanagra

2.2.1 Importation des données

Nous chargeons le fichier « CongressVotePipeline.xlsx » dans le tableur Excel. Nous nous plaçons sur la 3ème feuille « subset » comportant 5 colonnes (p = 4 explicatives + 1 cible) et 435 observations. A l'aide de la macro complémentaire "tanagra.xla", nous envoyons les données vers Tanagra[1].

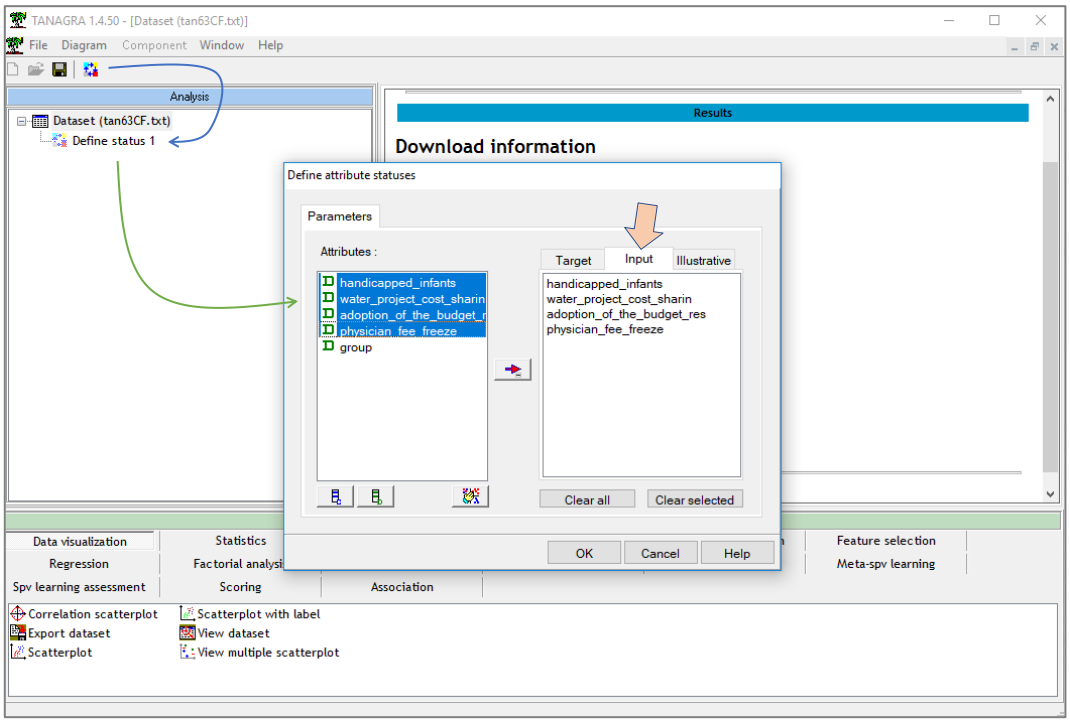
Tanagra est automatiquement démarré et les données chargées. Pour chaque variable, nous disposons de son type, elles sont toutes qualitatives, et du nombre de modalités.

[1] Voir le tutoriel "L'add-in Tanagra pour Excel 2007 et 2010" (Août 2010), http://tutoriels-data-mining.blogspot.com/2010/08/ladd-in-tanagra-pour-excel-2007-et-2010.html. Un outil est également disponible pour les tableurs des suites bureautiques Libre Office et Open Office, voir http://tutoriels-data-mining.blogspot.com/2011/07/tanagra-addon-pour-openoffice-33.html

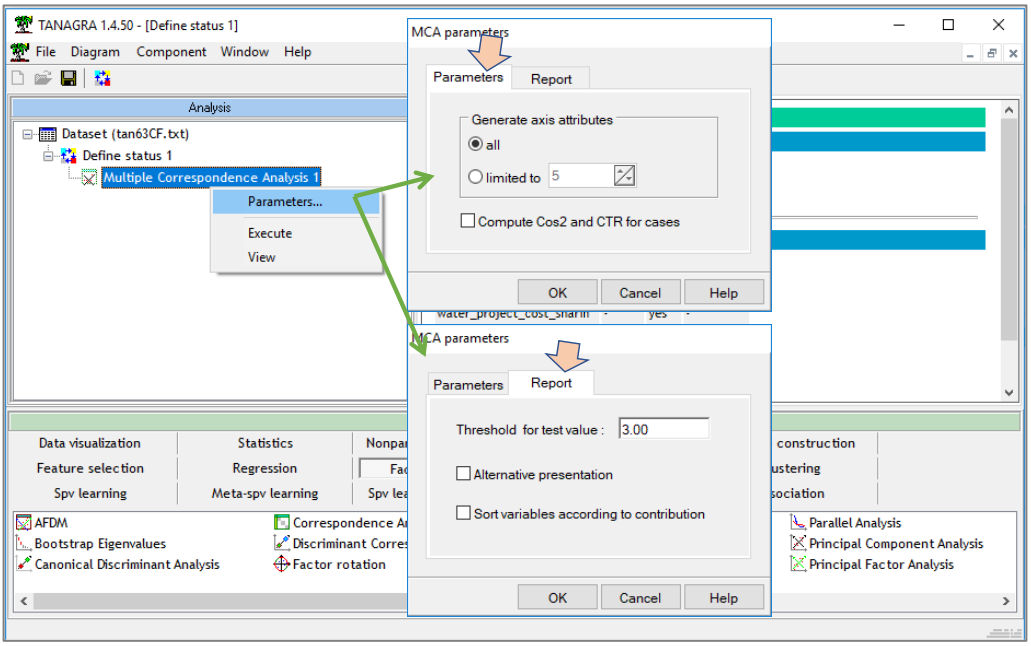


2.2.2 Analyse des correspondances multiples

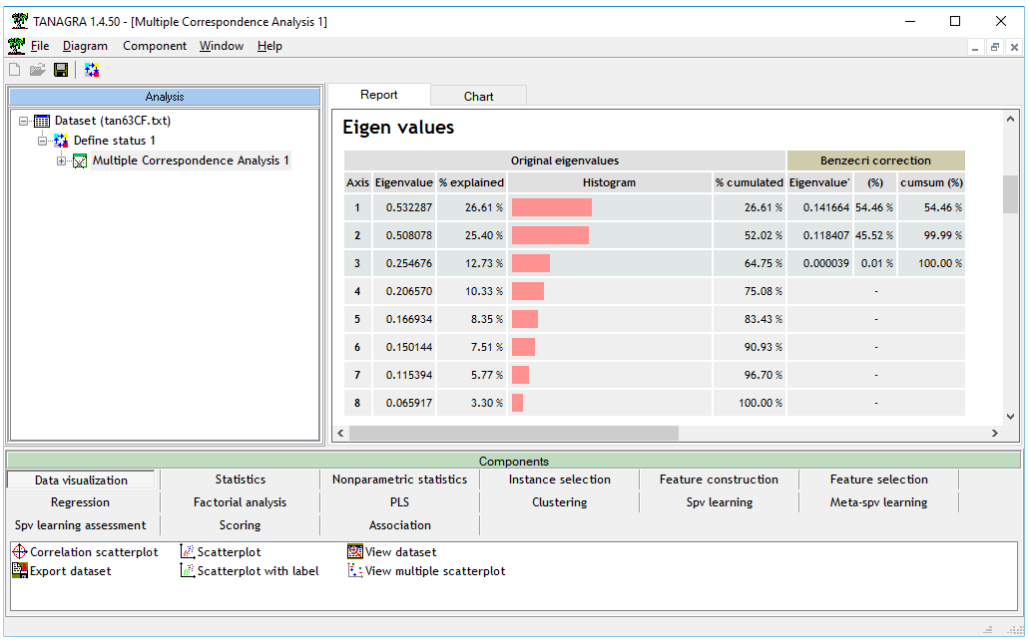
Nous spécifions les p = 4 variables (handicapped_infants, water_project_cost_sharin, adoption_of_the_budget_res, physician_fee_freeze) à utiliser pour l'ACM à l'aide du composant DEFINE STATUS. Nous les plaçons en INPUT.



Puis nous insérons le composant MULITPLE CORRESPONDANCE ANALYSIS (onglet **FACTORIAL ANALYSIS**). Nous le paramétrons de manière à générer tous les facteurs (Parameters : Generate axis attributes = all) et à présenter les tableaux de résultats en respectant l'ordre initial des variables dans la base (Report : Sort variables according to contribution est désélectionné).



Nous exécutons les calculs et visualisons les résultats en cliquant sur le menu contextuel VIEW.



En inspectant le tableau des valeurs propres, on peut penser que 2 facteurs permettent de décrire correctement les données. Mais cela ne veut pas qu'elles suffisent pour prédire au mieux les valeurs de "group" dans l'analyse discriminante. La variable cible ne prend pas part à l'ACM. Les facteurs sont construits de manière à maximiser la dispersion des modalités (la somme des carrés des rapports de corrélation avec les axes, cf. « Cours ACM », Août 2013).

Nous disposons ensuite des coordonnées factorielles des modalités (couples « variable = valeur »).

Values	Overall			Coordinate							
Attribute = Value	Mass	Sq.Dist	Inertia	coord_1	coord_2	coord_3	coord_4	coord_5	coord_6	coord_7	coord_8
handicapped_infants = n	0.1356	0.8432	0.1144	-0.3848	0.5064	-0.1651	0.0608	-0.6177	0.1601	0.0016	0.0239
handicapped_infants = other	0.0069	35.2500	0.2431	4.0292	1.8537	-0.4061	-0.0079	-0.2848	-3.9076	0.0985	-0.2325
handicapped_infants = y	0.1075	1.3262	0.1425	0.2270	-0.7581	0.2344	-0.0763	0.7978	0.0487	-0.0083	-0.0153
water_project_cost_sharin = n	0.1103	1.2656	0.1397	-0.1756	-0.2102	-0.9899	0.4192	0.1743	-0.0335	-0.0583	-0.0024
water_project_cost_sharin = other	0.0276	8.0625	0.2224	1.3730	0.7657	-0.3697	-2.2685	-0.0263	0.5300	-0.1627	-0.0069
water_project_cost_sharin = y	0.1121	1.2308	0.1379	-0.1651	0.0185	1.0657	0.1457	-0.1652	-0.0974	0.0975	0.0040
adoption_of_the_budget_res = n	0.0983	1.5439	0.1517	-0.6402	0.8866	0.0155	-0.0862	0.3554	-0.1457	-0.0055	0.4390
adoption_of_the_budget_res = other	0.0063	38.5455	0.2437	4.3242	2.3774	-0.1856	1.5292	0.4292	1.6977	2.9512	-0.2141
adoption_of_the_budget_res = y	0.1454	0.7194	0.1046	0.2447	-0.7026	-0.0024	-0.0082	-0.2589	0.0247	-0.1246	-0.2874
physician_fee_freeze = n	0.1420	0.7611	0.1080	0.2517	-0.7257	-0.0585	-0.0664	-0.2231	-0.0457	0.1508	0.2978
physician_fee_freeze = other	0.0063	38.5455	0.2437	4.3681	2.1093	0.7884	2.0066	0.1605	1.3450	-2.8817	0.4790
physician_fee_freeze = y	0.1017	1.4576	0.1483	-0.6227	0.8817	0.0326	-0.0320	0.3013	-0.0199	-0.0314	-0.4453

Figure 1 - Coordonnées factorielles des modalités

Plus bas dans la feuille de résultats, les coefficients des fonctions de projection permettant de calculer les coordonnées factorielles des individus à partir des indicatrices issues du codage disjonctif complet du tableau de données initial. Il y a une relation directe entre ces coefficients et les coordonnées factorielles des modalités, nous l'exploiterons plus loin dans ce tutoriel (Section 2.3.4).

Factor Score Coefficients

Applied to indicator matrix i.e. columns are dummy variables

Attribute = Value	Axis_1	Axis_2	Axis_3	Axis_4	Axis_5	Axis_6	Axis_7	Axis_8
handicapped_infants = n	-0.1318	0.1776	-0.0818	0.0335	-0.3780	0.1033	0.0012	0.0233
handicapped_infants = other	1.3807	0.6502	-0.2012	-0.0043	-0.1743	-2.5212	0.0725	-0.2264
handicapped_infants = y	0.0778	-0.2659	0.1161	-0.0420	0.4882	0.0314	-0.0061	-0.0149
water_project_cost_sharin = n	-0.0602	-0.0737	-0.4904	0.2306	0.1067	-0.0216	-0.0429	-0.0023
water_project_cost_sharin = other	0.4705	0.2686	-0.1831	-1.2478	-0.0161	0.3419	-0.1197	-0.0068
water_project_cost_sharin = y	-0.0566	0.0065	0.5279	0.0801	-0.1011	-0.0629	0.0717	0.0039
adoption_of_the_budget_res = n	-0.2194	0.3110	0.0077	-0.0474	0.2175	-0.0940	-0.0040	0.4274
adoption_of_the_budget_res = other	1.4817	0.8338	-0.0919	0.8412	0.2626	1.0954	2.1719	-0.2084
adoption_of_the_budget_res = y	0.0838	-0.2464	-0.0012	-0.0045	-0.1584	0.0159	-0.0917	-0.2798
physician_fee_freeze = n	0.0862	-0.2545	-0.0290	-0.0365	-0.1365	-0.0295	0.1110	0.2899
physician_fee_freeze = other	1.4968	0.7398	0.3905	1.1037	0.0982	0.8678	-2.1208	0.4664
physician_fee_freeze = y	-0.2134	0.3092	0.0161	-0.0176	0.1844	-0.0128	-0.0231	-0.4336

Figure 2 - Fonction permettant de calculer les coordonnées factorielles des individus

Prenons l'exemple du premier individu de la base pour expliciter l'idée. Ses valeurs sont :

handicapped_infants	water_project_cost_sharin	adoption_of_the_budget_res	physician_fee_freeze
n	n	n	n

Sa position sur le premier axe sera obtenue avec :

$$-0.1318 + (-0.0602) + (-0.2194) + 0.0862 = -0.3251$$

Pour le second facteur, nous aurons :

$$0.1776 + (-0.0737) + 0.3110 + (-0.2545) = 0.1603$$

Ainsi, les individus sont plongés dans un espace factoriel, totalement fidèle à l'original au sens de l'ACM, si nous conservons l'ensemble des (q = M – p = 12 – 4 = 8) facteurs, où M est le nombre total de modalités (4 variables x 3 modalités chacune = 12). Voici les coordonnées factorielles des 10 premières observations.

examples	Axe 1	Axe 2	Axe 3	Axe 4	Axe 5	Axe 6	Axe 7	Axe 8
1	-0.3251	0.1603	-0.5935	0.1801	-0.1903	-0.0418	0.0652	0.7384
2	-0.3251	0.1603	-0.5935	0.1801	-0.1903	-0.0418	0.0652	0.7384
3	-0.3251	0.1603	-0.5935	0.1801	-0.1903	-0.0418	0.0652	0.7384
4	-0.3251	0.1603	-0.5935	0.1801	-0.1903	-0.0418	0.0652	0.7384
5	-0.3251	0.1603	-0.5935	0.1801	-0.1903	-0.0418	0.0652	0.7384
6	1.3760	0.6832	-0.6931	1.0687	-0.1451	1.1476	2.2412	0.1025
7	1.9193	0.9395	0.1245	-0.1151	-0.4543	1.3289	-2.3311	0.2031
8	-0.6248	0.7241	-0.5484	0.1990	0.1306	-0.0252	-0.0688	0.0149
9	1.3760	0.6832	-0.6931	1.0687	-0.1451	1.1476	2.2412	0.1025
10	-0.0219	-0.3971	-0.6023	0.2230	-0.5662	0.0681	-0.0225	0.0311

Figure 3 - Coordonnées factorielles des 10 premières observations

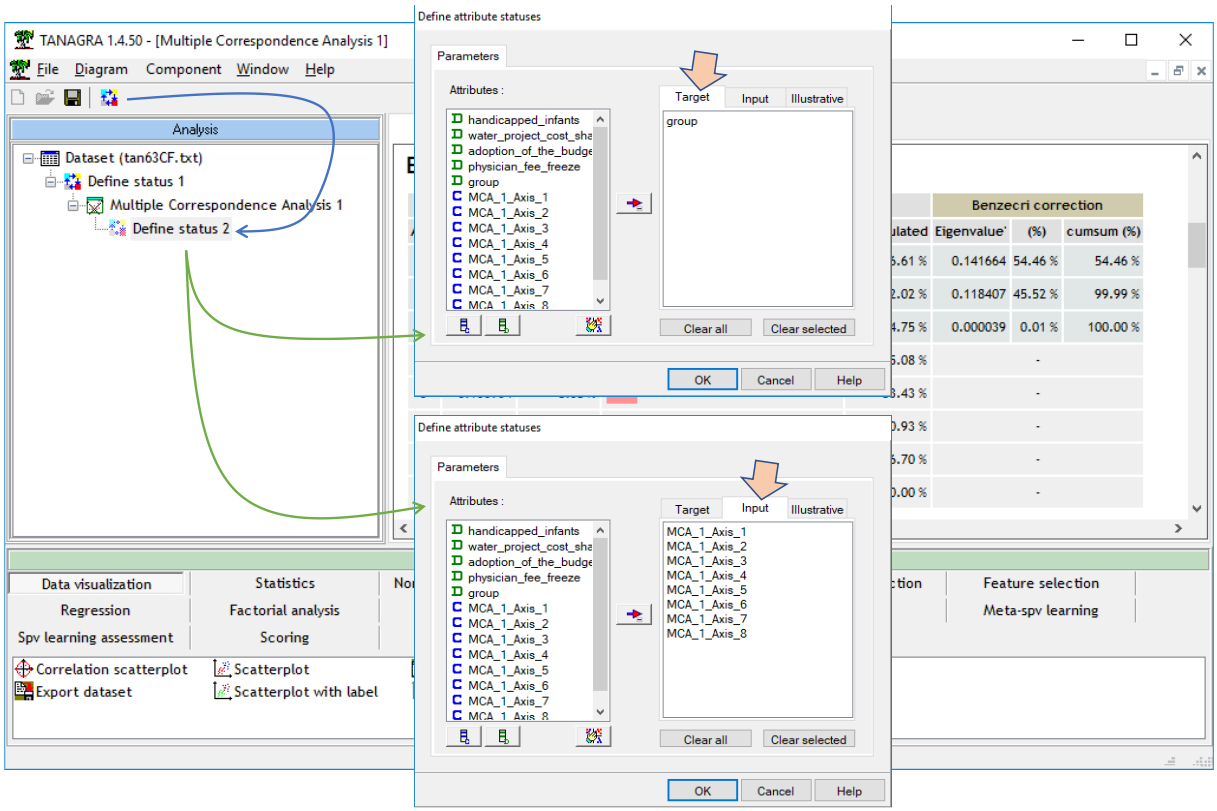
2.2.3 Analyse discriminante sur facteurs

De fait, nous disposons d'un nouveau tableau de données pour l'analyse discriminante avec la cible "group" et q = 8 variables explicatives.

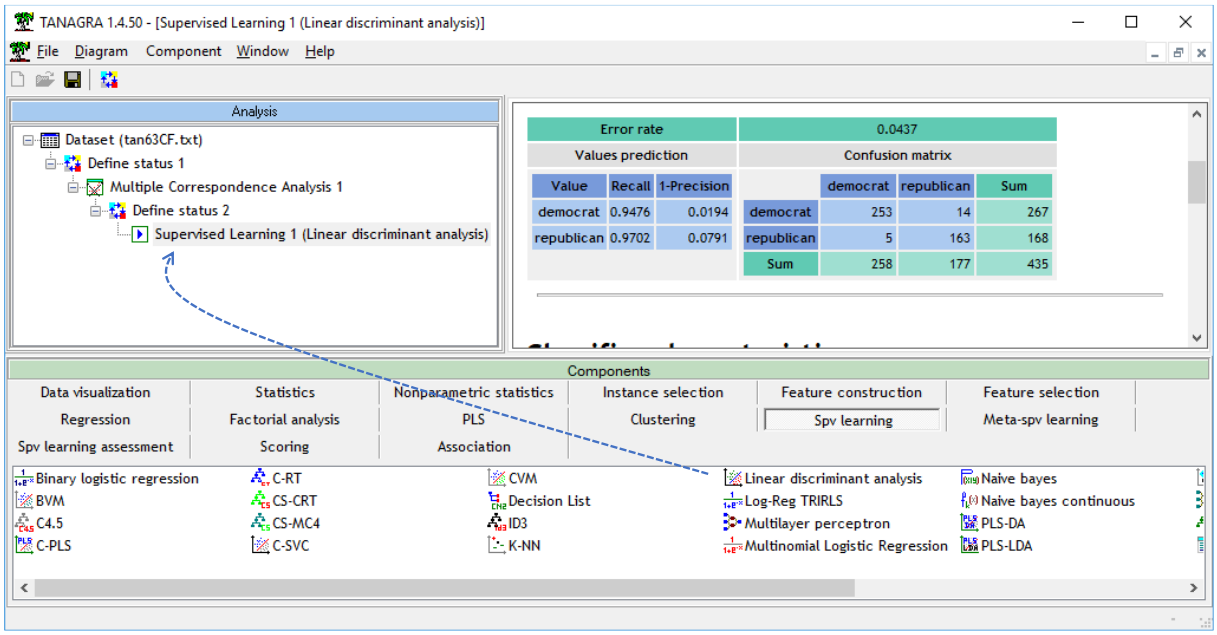
examples	Axe 1	Axe 2	Axe 3	Axe 4	Axe 5	Axe 6	Axe 7	Axe 8	group
1	-0.3251	0.1603	-0.5935	0.1801	-0.1903	-0.0418	0.0652	0.7384	democrat
2	-0.3251	0.1603	-0.5935	0.1801	-0.1903	-0.0418	0.0652	0.7384	democrat
3	-0.3251	0.1603	-0.5935	0.1801	-0.1903	-0.0418	0.0652	0.7384	democrat
4	-0.3251	0.1603	-0.5935	0.1801	-0.1903	-0.0418	0.0652	0.7384	democrat
5	-0.3251	0.1603	-0.5935	0.1801	-0.1903	-0.0418	0.0652	0.7384	democrat
6	1.3760	0.6832	-0.6931	1.0687	-0.1451	1.1476	2.2412	0.1025	democrat
7	1.9193	0.9395	0.1245	-0.1151	-0.4543	1.3289	-2.3311	0.2031	democrat
8	-0.6248	0.7241	-0.5484	0.1990	0.1306	-0.0252	-0.0688	0.0149	democrat
9	1.3760	0.6832	-0.6931	1.0687	-0.1451	1.1476	2.2412	0.1025	democrat
10	-0.0219	-0.3971	-0.6023	0.2230	-0.5662	0.0681	-0.0225	0.0311	democrat

Figure 4 - Tableau de données pour l'analyse discriminante

Dans Tanagra, nous insérons de nouveau un composant DEFINE STATUS, nous plaçons "group" en TARGET, les facteurs "MCA_1_Axis_1" ... "MCA_1_Axis_8" en INPUT.



Il reste à introduire le composant LINEAR DISCRIMINANT ANALYSIS (onglet **Spv Learning**).



Parmi les résultats, nous disposons des fonctions de classement qui permettent d'associer les individus aux groupes à partir de leurs coordonnées factorielles.

LDA Summary

	Classification functions		Statistical Evaluation			
Attribute	democrat	republican	Wilks L.	Partial L.	F(1,426)	p-value
MCA_1_Axis_1	3.0027	-4.7721	0.3196	0.4659	488.2775	0.0000
MCA_1_Axis_2	-5.5278	8.7853	0.7010	0.2124	1579.5657	0.0000
MCA_1_Axis_3	0.3332	-0.5295	0.1499	0.9933	2.8766	0.0906
MCA_1_Axis_4	0.2474	-0.3932	0.1493	0.9970	1.2861	0.2574
MCA_1_Axis_5	-3.3555	5.3328	0.2157	0.6902	191.2287	0.0000
MCA_1_Axis_6	0.0613	-0.0974	0.1489	0.9999	0.0573	0.8109
MCA_1_Axis_7	0.8174	-1.2991	0.1516	0.9819	7.8445	0.0053
MCA_1_Axis_8	4.9439	-7.8572	0.2062	0.7221	163.9187	0.0000
constant	-2.2782	-5.4728		-		

Figure 5 - Fonctions de classements sur facteurs issus de l'analyse discriminante

Puisque nous sommes dans un cadre binaire (variable cible à K = 2 modalités, "republican" vs. "democrat"), nous pouvons construire une fonction score unique opposant la modalité positive (republican) à la négative (democrat) (n'y voyez surtout pas une quelconque inclination politique de ma part, c'est pour être raccord avec les résultats de Python plus loin, section 0).

Attribute	democrat	republican	fonction score
MCA_1_Axis_1	3.0027	-4.7721	-7.7748
MCA_1_Axis_2	-5.5278	8.7853	14.3131
MCA_1_Axis_3	0.3332	-0.5295	-0.8627
MCA_1_Axis_4	0.2474	-0.3932	-0.6405
MCA_1_Axis_5	-3.3555	5.3328	8.6883
MCA_1_Axis_6	0.0613	-0.0974	-0.1586
MCA_1_Axis_7	0.8174	-1.2991	-2.1165
MCA_1_Axis_8	4.9439	-7.8572	-12.8011
constant	-2.2782	-5.4728	-3.1946

Figure 6 - Coefficients de la fonction score

La fonction score permet d'affecter les classes aux individus à partir de leur description. Prenons l'exemple de notre premier individu (Figure 4), nous calculons son score comme suit :

$$-7.7748 \times (-0.3215) + 14.3131 \times 0.1603 + \cdots + (-12.8011) \times 0.7384 + (-3.1946) = \mathbf{-9.2123}$$

Le score étant négatif, le modèle lui affecte la modalité "democrat". A juste titre puisqu'il s'agit bien de son groupe d'appartenance (1er individu, Figure 4).

2.2.4 Reconstitution des coefficients sur indicatrices

Le dispositif fonctionne, mais reconnaissons qu'il n'est pas très pratique. Pour un individu supplémentaire, il faudrait calculer tout d'abord ses coordonnées factorielles à partir des fonctions de

projection (Figure 2), puis calculer son score à l'aide de la fonction dédiée (Figure 6). De plus, on ne discerne pas le mécanisme d'affectation aux classes dans la fonction score parce que la lecture des facteurs n'est pas aisée. Par exemple, on a plus tendance à être un "positif" (republican) lorsque "Axe 2" et "Axe 5" prennent des valeurs positives. Il faudrait se plonger dans l'interprétation de ces facteurs pour comprendre le profil de votes sous-jacent. C'est juste très difficile, voire ingérable.

Il nous faut plutôt exprimer la fonction score à partir des modalités. Nous le faisons en effectuant un simple produit matriciel entre les coefficients des fonctions de projection et ceux de la fonction score. La constante étant restituée telle quelle. Soit,

Attribute = Value	Axis_1	Axis_2	Axis_3	Axis_4	Axis_5	Axis_6	Axis_7	Axis_8
handicapped_infants = n	-0.1318	0.1776	-0.0818	0.0335	-0.3780	0.1033	0.0012	0.0233
handicapped_infants = other	1.3807	0.6502	-0.2012	-0.0043	-0.1743	-2.5212	0.0725	-0.2264
handicapped_infants = y	0.0778	-0.2659	0.1161	-0.0420	0.4882	0.0314	-0.0061	-0.0149
water_project_cost_sharin = n	-0.0602	-0.0737	-0.4904	0.2306	0.1067	-0.0216	-0.0429	-0.0023
water_project_cost_sharin = other	0.4705	0.2686	-0.1831	-1.2478	-0.0161	0.3419	-0.1197	-0.0068
water_project_cost_sharin = y	-0.0566	0.0065	0.5279	0.0801	-0.1011	-0.0629	0.0717	0.0039
adoption_of_the_budget_res = n	-0.2194	0.3110	0.0077	-0.0474	0.2175	-0.0940	-0.0040	0.4274
adoption_of_the_budget_res = other	1.4817	0.8338	-0.0919	0.8412	0.2626	1.0954	2.1719	-0.2084
adoption_of_the_budget_res = y	0.0838	-0.2464	-0.0012	-0.0045	-0.1584	0.0159	-0.0917	-0.2798
physician_fee_freeze = n	0.0862	-0.2545	-0.0290	-0.0365	-0.1365	-0.0295	0.1110	0.2899
physician_fee_freeze = other	1.4968	0.7398	0.3905	1.1037	0.0982	0.8678	-2.1208	0.4664
physician_fee_freeze = y	-0.2134	0.3092	0.0161	-0.0176	0.1844	-0.0128	-0.0231	-0.4336

X

Attribute	fonction score
MCA_1_Axis_1	-7.7748
MCA_1_Axis_2	14.3131
MCA_1_Axis_3	-0.8627
MCA_1_Axis_4	-0.6405
MCA_1_Axis_5	8.6883
MCA_1_Axis_6	-0.1586
MCA_1_Axis_7	-2.1165
MCA_1_Axis_8	-12.8011
constant	-3.1946

Nous obtenons une nouvelle version de la fonction score, plus facile à déployer et à interpréter.

Attribute = Value	F.Score
handicapped_infants = n	0.0154
handicapped_infants = other	0.3786
handicapped_infants = y	-0.0438
water_project_cost_sharin = n	0.7384
water_project_cost_sharin = other	1.2891
water_project_cost_sharin = y	-1.0443
adoption_of_the_budget_res = n	2.6215
adoption_of_the_budget_res = other	0.1343
adoption_of_the_budget_res = y	-1.7777
physician_fee_freeze = n	-9.3931
physician_fee_freeze = other	-2.8583
physician_fee_freeze = y	13.2855
constant	-3.1946

Figure 7 - Fonction score exprimée à partir des indicatrices

Pour notre premier individu de coordonnées

handicapped_infants	water_project_cost_sharin	adoption_of_the_budget_res	physician_fee_freeze
n	n	n	n

Nous additionnons les points (sans oublier la constante) :

$$0.0154 + 0.7384 + 2.6215 + (-9.3931) + (-3.1946) = -9.2123$$

Le résultat et a fortiori la conclusion sont complétement cohérents avec le calcul du score à partir de la description factorielle. Heureusement.

L'interprétation surtout fait un très grand bond en avant. En effet, au regard des valeurs des coefficients (Figure 7), on constate que le vote pour le thème "physician fee freeze" est déterminant dans l'identification du groupe politique. Ses indicatrices présentent les coefficients les plus élevés en valeur absolue.

2.3 Calculs sous Python

Maintenant que nous avons compris le principe, essayons de reproduire les calculs sous python à l'aide de l'ACM du package "fanalysis" et l'analyse discriminante linéaire de "scikit-learn".

2.3.1 Importation des données

Nous chargeons la feuille « subset » du classeur « CongressVotePipeline.xlsx ».

```
#modif. du dossier de travail
import os
os.chdir("… votre dossier …")

#librairie pandas
import pandas

#chargement de la feuille de données
#version des données à 4 variables explicatives
vote_subset = pandas.read_excel("CongressVotePipeline.xlsx",sheet_name="subset",header=0)
print(vote_subset.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435 entries, 0 to 434
Data columns (total 5 columns):
handicapped_infants           435 non-null object
water_project_cost_sharin     435 non-null object
adoption_of_the_budget_res    435 non-null object
physician_fee_freeze          435 non-null object
group                         435 non-null object
dtypes: object(5)
memory usage: 17.1+ KB
```

2.3.2 Analyse des correspondances multiples

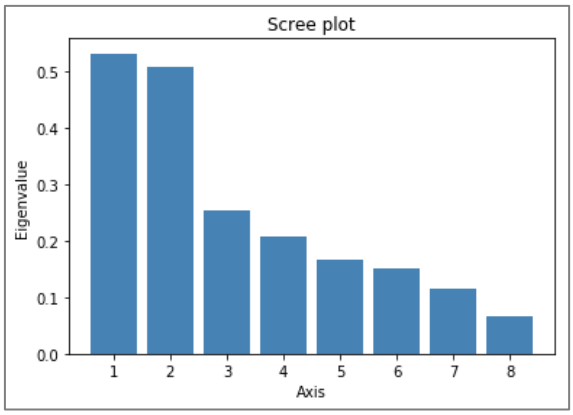
Nous importons la classe MCA de la librairie "fanalysis" après avoir installé cette dernière. Nous instancions l'objet et lançons l'analyse sur les descripteurs. Nous affichons les valeurs propres.

```
#importation de la librairie
from fanalysis.mca import MCA

#instanciation
acm = MCA(var_labels=vote_subset.columns[:4])

#apprentissage
coord = acm.fit_transform(vote_subset.iloc[:,:4].values)

#affichage des valeurs propres
print(acm.eig_)
```

```
[[5.32286802e-01 5.08077671e-01 2.54676393e-01 2.06570177e-01
  1.66933993e-01 1.50144110e-01 1.15394061e-01 6.59167923e-02]
 [2.66143401e+01 2.54038836e+01 1.27338196e+01 1.03285089e+01
  8.34669965e+00 7.50720551e+00 5.76970307e+00 3.29583961e+00]
 [2.66143401e+01 5.20182237e+01 6.47520433e+01 7.50805522e+01
  8.34272518e+01 9.09344573e+01 9.67041604e+01 1.00000000e+02]]
```

Nous disposons successivement : des valeurs propres absolues, relatives, et leur cumul. Une représentation graphique est possible.

```
#valeurs propres - graphique
print(acm.plot_eigenvalues())
```



Les coordonnées des modalités colonnes sont disponibles.

```
#coordonnées des colonnes
print(acm.col_topandas())
```

```
                                col_coord_dim1      ...        col_cos2_dim8
handicapped_infants_n                -0.384746      ...             0.000680
handicapped_infants_other             4.029186      ...             0.001534
handicapped_infants_y                 0.227004      ...             0.000176
water_project_cost_sharin_n          -0.175612      ...             0.000004
water_project_cost_sharin_other       1.373027      ...             0.000006
water_project_cost_sharin_y          -0.165065      ...             0.000013
adoption_of_the_budget_res_n         -0.640196      ...             0.124803
adoption_of_the_budget_res_other      4.324168      ...             0.001189
adoption_of_the_budget_res_y          0.244694      ...             0.114802
```

```
physician_fee_freeze_n                0.251688      ...            0.116479
physician_fee_freeze_other            4.368061      ...            0.005951
physician_fee_freeze_y               -0.622687      ...            0.136021
```

Nous retrouvons les valeurs de Tanagra (Figure 1). Pour chaque axe, nous pouvons calculer les coefficients des fonctions de projection en les multipliant par

$$\frac{1}{p\sqrt{\lambda_h}}$$

Où p est le nombre de variables actives de l'ACM, λ_h est la valeur propre associée au facteur n°h. Sous Python, ça donne :

```
#nombre var. actives
p = vote_subset.shape[1]-1
print(p)

#calcul des fonctions de projection
import numpy
fproj = numpy.zeros(acm.col_coord_.shape)

#pour chaque colonne
for j in range(fproj.shape[1]):
    fproj[:,j] = acm.col_coord_[:,j]/(p*numpy.sqrt(acm.eig_[0,j]))

#affichage fonction
print(fproj)

#affichage plus avenant des deux premiers facteurs
print(pandas.DataFrame(fproj,index=acm.col_labels_))
```

Soit, pour le premier et le dernier facteur (Figure 2) :

```
                                    0    ...            7
handicapped_infants_n          -0.131838    ...   -0.023315
handicapped_infants_other       1.380653    ...    0.226421
handicapped_infants_y           0.077786    ...    0.014895
water_project_cost_sharin_n    -0.060176    ...    0.002300
water_project_cost_sharin_other 0.470486    ...    0.006753
water_project_cost_sharin_y    -0.056562    ...   -0.003927
adoption_of_the_budget_res_n   -0.219371    ...   -0.427423
adoption_of_the_budget_res_other 1.481732   ...    0.208432
adoption_of_the_budget_res_y    0.083848    ...    0.279829
physician_fee_freeze_n          0.086244    ...   -0.289933
physician_fee_freeze_other      1.496773    ...   -0.466373
physician_fee_freeze_y         -0.213372    ...    0.433579
```

Nous exploiterons cette matrice "fproj" lorsqu'il s'agira d'exprimer la fonction score de l'analyse discriminante à partir des indicatrices des modalités.

La fonction **fit_transform()** de l'ACM a produit une matrice correspondant aux coordonnées factorielles des observations. Elle sera utilisée en entrée de l'analyse discriminante. Nous en vérifions les dimensions :

```
#taille du tableau de données présenté à l'ADL
print(coord.shape)
```

```
(435, 8)
```

Nous affichons les 10 premières lignes pour vérification :

```
#10 premières lignes
print(coord[:10,:])
```

```
[[-0.3251412  -0.16031225 -0.59346066 -0.18010007  0.19030782  0.04178947  -0.06523741 -0.73837111]
 [-0.3251412  -0.16031225 -0.59346066 -0.18010007  0.19030782  0.04178947  -0.06523741 -0.73837111]
 [-0.3251412  -0.16031225 -0.59346066 -0.18010007  0.19030782  0.04178947  -0.06523741 -0.73837111]
 [-0.3251412  -0.16031225 -0.59346066 -0.18010007  0.19030782  0.04178947  -0.06523741 -0.73837111]
 [-0.3251412  -0.16031225 -0.59346066 -0.18010007  0.19030782  0.04178947  -0.06523741 -0.73837111]
 [ 1.37596252 -0.68318124 -0.69308334 -1.06865807  0.14514713 -1.14756756  -2.24119738 -0.10251636]
 [ 1.91926791 -0.93954926  0.12445061  0.11514013  0.4542653  -1.32891944   2.33107531 -0.20310668]
 [-0.62475706 -0.72408537 -0.54836065 -0.19904672 -0.13057386  0.02516957   0.06884158 -0.01485958]
 [ 1.37596252 -0.68318124 -0.69308334 -1.06865807  0.14514713 -1.14756756  -2.24119738 -0.10251636]
 [-0.02192213  0.39708369 -0.60232152 -0.222973    0.56617773 -0.06813171   0.02246883 -0.03111944]]
```

Ils sont raccords avec ceux de Tanagra (Figure 3) aux signes près. La disparité des signes n'est pas un problème, nous savons que l'orientation des axes est arbitraire en analyse factorielle. Ce sont les concomitances et oppositions qui importent. Cette divergence n'aura aucune influence sur le résultat final c.-à-d. l'expression de la fonction score à partir des indicatrices des modalités.

2.3.3 Analyse discriminante sur facteurs

Nous utilisons la classe LinearDiscriminantAnalysis de "scikit-learn" pour l'ADL, la fonction **fit()** prend en entrée les descripteurs issus de l'ACM et la variable cible. La cible étant binaire, l'outil fournit directement les coefficients de la fonction score :

```
#classe pour l'analyse discriminante
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

#instanciation
adl = LinearDiscriminantAnalysis()

#apprentissage
adl.fit(coord,vote_subset.group)
```

```
#affichage des coefficients de la fonction score
print(adl.coef_)
```

```
[[ -7.7748435  -14.31314024  -0.86273619   0.64052874  -8.68830833
    0.15860569   2.11651849  12.80110157]]
```

Ils sont également raccords avec ceux de Tanagra (Figure 6), aux signes près toujours. La constante est également cohérente.

```
#la constante
print(adl.intercept_)
```

```
[-3.19459941]
```

2.3.4 Reconstitution des coefficients sur indicatrices

Pour exprimer la fonction score à partir des indicatrices, nous la multiplions avec la matrice des coefficients des fonctions de projection de l'ACM.

```
#reconstitution de la fonction sur indicatrices
fpValues = numpy.dot(fproj,numpy.transpose(adl.coef_))
print(fpValues)

#affichage plus joli
print(pandas.DataFrame(fpValues,index=acm.col_labels_))
```

```
handicapped_infants_n              0.015427
handicapped_infants_other          0.378558
handicapped_infants_y             -0.043762
water_project_cost_sharin_n        0.738387
water_project_cost_sharin_other    1.289068
water_project_cost_sharin_y       -1.044336
adoption_of_the_budget_res_n       2.621544
adoption_of_the_budget_res_other   0.134340
adoption_of_the_budget_res_y      -1.777715
physician_fee_freeze_n            -9.393089
physician_fee_freeze_other        -2.858302
physician_fee_freeze_y            13.285504
```

Nous retrouvons à l'identique les coefficients des indicatrices de la fonction score (Figure 7). Les signes sont totalement cohérents cette fois-ci. Les disparités consécutives au mode de calcul des facteurs de l'ACM n'ont eu aucun impact sur le résultat final.

3 Pipeline sous Python

Invoquer explicitement l'ACM puis l'ADL n'est pas très sorcier finalement. Il en est de même lors du déploiement, lorsqu'on souhaite appliquer DISQUAL sur des individus supplémentaires, il faudrait d'abord appeler la fonction transform() de l'ACM puis, sur le résultat, appeler predict() de l'ADL pour obtenir la prédiction. L'affaire devient un peu plus complexe lorsqu'il s'agit d'englober ces opérations dans une structure plus complexe telle que la validation croisée par exemple, ou mieux encore, une recherche du nombre de facteurs à retenir par validation croisée. Il faut imbriquer des boucles et les erreurs sont vites arrivées. Dans ce contexte, pouvoir encapsuler les calculs dans une structure unique, plus facile à manipuler, se révèle très pratique et nous facilite grandement la vie. C'est ce que nous verrons dans cette section.

3.1 Principe du pipeline avec "scikit-learn"

3.1.1 Importation des données

Nous importons une version étendue de la base « Vote au congrès » dans cette section, elle fera office d'échantillon d'apprentissage. Nous chargeons la feuille « train » du classeur Excel.

```
#importation des données d'apprentissage
vote_train = pandas.read_excel("CongressVotePipeline.xlsx",sheet_name="train",header=0)
print(vote_train.info())
```

Nous disposons maintenant de 17 colonnes (p = 16 variables explicatives + 1 cible) et n_{train} = 235 observations.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 235 entries, 0 to 234
Data columns (total 17 columns):
handicapped_infants          235 non-null object
water_project_cost_sharin    235 non-null object
adoption_of_the_budget_re    235 non-null object
physician_fee_freeze         235 non-null object
el_salvador_aid              235 non-null object
religious_groups_in_schoo    235 non-null object
anti_satellite_test_ban      235 non-null object
aid_to_nicaraguan_contras    235 non-null object
mx_missile                   235 non-null object
immigration                  235 non-null object
synfuels_corporation_cutb    235 non-null object
education_spending           235 non-null object
superfund_right_to_sue       235 non-null object
crime                        235 non-null object
```

```
duty_free_exports            235 non-null object
export_administration_act    235 non-null object
group                        235 non-null object
dtypes: object(17)
```

3.1.2 Construction du Pipeline

Nous utilisons la classe Pipeline pour construire la méthode DISQUAL.

```
#construction du pipeline
from sklearn.pipeline import Pipeline
disqual = Pipeline([ ("acm",MCA(var_labels=vote_train.columns[:-1])),
                     ("adl",LinearDiscriminantAnalysis())])
```

Nous avons nommé notre objet "disqual". Dans l'appel du constructeur, nous spécifions une liste [ ] de méthodes. Pour chacune d'entre elles, nous avons un tuple ( ) avec : un identifiant (en rouge dans le code ci-dessus), un appel du constructeur avec éventuellement les paramètres associés.

Attention ! Le dispositif fonctionne parce que ces deux classes (MCA et LinearDiscriminantAnalysis) implémentent les fonctions fit() et transform().

3.1.3 Apprentissage du modèle prédictif

Pour construire le modèle prédictif, nous faisons appel à la méthode fit() de Pipeline. Elle se charge de réaliser automatiquement ce que nous avions défini manuellement précédemment c.-à-d. appel de fit_transform() pour l'ACM, puis appel de fit() de l'ADL avec en entrée le fruit du calcul de l'ACM.

```
#apprentissage
disqual.fit(vote_train.iloc[:,:-1].values,vote_train.group)
```

3.1.4 Inspection du modèle

Les techniques incluses dans le Pipeline sont nommées. Nous pouvons y accéder sélectivement avec la propriété **named_steps**. Pour connaître le nombre de facteurs produits par l'ACM (acm) par exemple, nous ferons :

```
#accès à l'ACM, nombre de composants
print(disqual.named_steps["acm"].n_components_)
```

```
32
```

Ainsi, le nombre de variables prédictives (facteurs issus de l'ACM) présentées à l'ADL est égal à 32.

Pour se faire une idée des valeurs propres de l'ACM, nous faisons appel à **plot_eigenvalues()**.

```
#afficher les valeurs propres
disqual.named_steps["acm"].plot_eigenvalues()
```

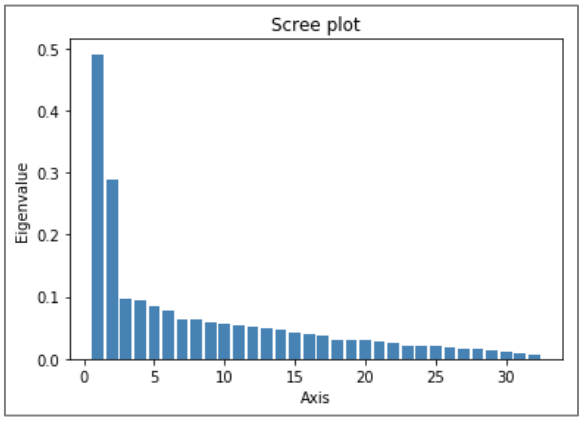


Figure 8 - Graphique des valeurs propres

Avec deux facteurs, nous disposons, semble-t-il, d'une description satisfaisante des données. Mais cela ne veut pas dire qu'elles suffisent pour prédire efficacement la variable cible "group". Nous traiterons cette question cruciale plus loin (Section 3.2).

Nous pouvons aussi afficher les coefficients de la fonction score de l'analyse discriminante (adl).

```
#coefs. de la fonction score
print(disqual.named_steps["adl"].coef_)
```

```
[[-23.1796328    0.36425284 -15.55537716   8.39836565  -2.52967472
   -7.89230055   5.77366907  -1.27193945  -1.61320859  -3.08374413
    0.3653672    4.7074921   -7.27645188   2.36103053   3.5296209
    4.14483567  -4.61066035  12.53691493   0.24223927 -12.50105317
    9.35457676  -8.98805132  -2.32868567  -2.71435029  -4.41345391
   22.26798533  -3.79644671  17.80143093  -1.76094629 -13.65548986
   30.20257336 -13.72545908]]
```

Ramener la fonction score sur les indicatrices des modalités est un jeu d'enfant (voir Section 2.3.4).

3.1.5 Déploiement sur des données test

Nous utilisons un échantillon test pour évaluer les performances du modèle. Nous chargeons les données de la feuille « test » (n_{test} = 200)

```
#chargement de l'échantillon test
vote_test = pandas.read_excel("CongressVotePipeline.xlsx",sheet_name="test",header=0)
print(vote_test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 17 columns):
handicapped_infants          200 non-null object
```

```
water_project_cost_sharin    200 non-null object
adoption_of_the_budget_re    200 non-null object
physician_fee_freeze         200 non-null object
el_salvador_aid              200 non-null object
religious_groups_in_schoo    200 non-null object
anti_satellite_test_ban      200 non-null object
aid_to_nicaraguan_contras    200 non-null object
mx_missile                   200 non-null object
immigration                  200 non-null object
synfuels_corporation_cutb    200 non-null object
education_spending           200 non-null object
superfund_right_to_sue       200 non-null object
crime                        200 non-null object
duty_free_exports            200 non-null object
export_administration_act    200 non-null object
group                        200 non-null object
dtypes: object(17)
```

Et nous appliquons l'objet « disqual » en faisant appel à la fonction predict().

```
#prediction
pVote = disqual.predict(vote_test.iloc[:,:-1].values)
print(numpy.unique(pVote,return_counts=True))
```

```
(array(['democrat', 'republican'], dtype='<U10'), array([111,  89], dtype=int64))
```

111 parlementaires sont désignées "democrat", 89 "republican".

Nous confrontons ces prédictions avec les classes observées dans une matrice de confusion.

```
#matrice de confusion
import sklearn.metrics as metrics
print(metrics.confusion_matrix(vote_test.group,pVote))
```

```
[[107   6]
 [  4  83]]
```

Le taux de reconnaissance est …

```
#taux de reconnaissance
print(metrics.accuracy_score(vote_test.group,pVote))
```

```
0.95
```

… de 95%.

3.2 Identification du nombre "optimal" de facteurs pour DISQUAL

La détermination du nombre de facteurs de l'ACM à introduire dans l'ADL est une question clé. Elle conditionne les performances prédictives du modèle. Le graphique des valeurs propres semble suggérer l'utilisation de 2 facteurs (Figure 8). En réalité, ce dispositif nous dit que 2 facteurs

permettent de décrire les données (dans l'espace des variables explicatives) avec une bonne approximation. Mais il ne nous dit pas qu'elles suffisent pour prédire au mieux la variable cible. Et pour cause, l'ACM ne tient pas compte de cette dernière dans la construction des facteurs.

En pratique, on s'appuie sur une démarche empirique où l'on évalue différentes configurations sur les données. On choisit la solution qui maximise une mesure de performance, souvent le taux de reconnaissance (ça peut être aussi l'AUC de le courbe ROC, etc.). Puisqu'il n'est pas souhaitable de mettre à contribution l'échantillon test dans ce processus, on passe par la validation croisée. Nous utilisons la classe GridSearchCV de "scikit-learn".

Nous spécifions tout d'abord la liste des configurations à évaluer.

```
#nombres de facteurs à tester
parametres = [{"acm__n_components":[1,2,5,10,15,20,25,30,32]}]
```

La clé du dictionnaire (au sens de Python) est définie par l'identifiant de l'objet (acm), suivi de deux underscores '__' et le nom du paramètre à manipuler. Nous avons ensuite une liste avec les différentes valeurs de "n_components" à tester.

Remarque sur la sélection des facteurs pour l'ADL. On pourrait être tenté de sélectionner les facteurs en nous basant sur les tests de significativité (Figure 5). Par exemple, à 5% dans notre modèle, nous ne conserverions les facteurs n° 1, 2, 5, 7, 8 (ceux qui présentent une p-value inférieure à 0.05). Ce n'est pas une bonne idée. En effet, un facteur est calculé à partir des résidus des précédents, son existence leur est tributaire. Prendre un facteur (ex. le n°5) sans ceux qui le précèdent (les n°1, 2, 3 et 4) n'a pas de sens. Dans la sélection, nous devons raisonner en évaluant tour à tour les solutions à q* **premiers** facteurs. C'est ce que nous réalisons dans cette section en mesurant les performances en validation croisée.

Nous instancions ensuite l'objet validation-croisée (classe KFold). Nous ne sommes pas obligés de le faire en réalité. L'intérêt ici est d'indiquer un paramétrage qui permet de rendre l'expérimentation reproductible à l'identique.

```
#initialiser un objet validation croisée
from sklearn import model_selection
cvEval = model_selection.KFold(n_splits=5,shuffle=True,random_state=1)
```

Nous demandons une 5-fold (n_splits = 5) validation croisée, les observations sont au préalable mélangées au hasard (shuffle = True), et le générateur de nombres aléatoires est initialisé à (random_state = 1).

Enfin, nous initialisons l'outil de recherche GridSearchCV, il prend en paramètre notamment l'objet "disqual" défini plus haut (Section 3.1.2) et nous lançons les calculs.

```
#outil grille de recherche
grid=model_selection.GridSearchCV(estimator=disqual,param_grid=parametres,scoring="accuracy",cv=cvEval)

#lancement de la recherche
grid.fit(vote_train.iloc[:,:-1].values,vote_train.group)
```

Le taux de reconnaissance (accuracy) en validation croisée est utilisé pour évaluer les modèles. Nous affichons le résultat (la moyenne du taux sur l'ensemble des blocs[2]) pour chaque valeur du paramètre "n_components".

```
#résultats
print(pandas.DataFrame(grid.cv_results_).loc[:,["params","mean_test_score"]])
```

```
                    params  mean_test_score
0   {'acm__n_components': 1}         0.893617
1   {'acm__n_components': 2}         0.893617
2   {'acm__n_components': 5}         0.923404
3  {'acm__n_components': 10}         0.931915
4  {'acm__n_components': 15}         0.927660
5  {'acm__n_components': 20}         0.936170
6  {'acm__n_components': 25}         0.944681
7  {'acm__n_components': 30}         0.948936
8  {'acm__n_components': 32}         0.948936
```

La meilleure solution pour la prédiction correspond à q* = 30 ou q* = 32 facteurs. Il ne fallait pas se limiter à 2 facteurs comme pouvait le suggérer l'éboulis des valeurs propres de l'ACM (Figure 8) en tous les cas.

Nous pouvons tracer l'évolution du taux de reconnaissance en fonction du nombre de composantes utilisées pour disposer d'une vision plus globale des résultats.

```
#erreur en CV
err_cv = numpy.array(pandas.DataFrame(grid.cv_results_).loc[:,["mean_test_score"]])
```

[2] Voir « Validation croisé, Bootstrap – Diapos » (Février 2015) pour le principe de la validation croisée ; http://tutoriels-data-mining.blogspot.com/2015/02/validation-croisee-bootstrap-diapos.html

```
print(err_cv)

#librairie graphique
import matplotlib.pyplot as plt

#paramètres - nombre de facteurs testés
prm = numpy.array(parametres[0]["acm__n_components"])

#graphique
plt.plot(prm,err_cv)
plt.title("CV accuracy rate vs. factor number")
plt.ylabel("Accuracy rate")
plt.xlabel("Factor number")
plt.show()
```

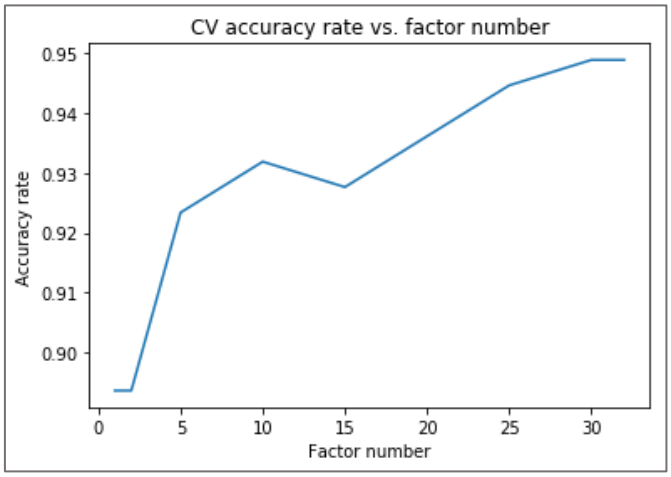


Figure 9 - Taux de reconnaissance en fonction du nombre de facteurs

3.3 Préférence à la régularisation

Le taux étant calculé en validation croisée sur les données, il est forcément entaché d'une certaine imprécision. Plutôt que la simple courbe moyenne, nous pouvons définir un bandeau de confiance à l'instar de ce que propose le package "glmnet" sous R[3]. Les bornes basses et hautes sont définies par 1 x écart-type (de la validation croisée) de part et d'autre de la courbe du taux.

```
#ecart-type de l'erreur en validation croisée
std_cv = numpy.array(pandas.DataFrame(grid.cv_results_).loc[:,["std_test_score"]])
print(std_cv)

#graphique error bar avec les intervalles de confiance
plt.errorbar(prm,err_cv,yerr=std_cv,ecolor="silver")
```

[3] Voir « Régressions ridge et elasticnet sous R », Mai 2018, page 28 ; http://tutoriels-data-mining.blogspot.com/2018/05/regressions-ridge-et-elasticnet-sous-r.html

```
plt.plot(prm,err_cv-std_cv,color="silver",linestyle="dashed",linewidth=1)
plt.plot(prm,err_cv+std_cv,color="silver",linestyle="dashed",linewidth=1)
plt.show()
```

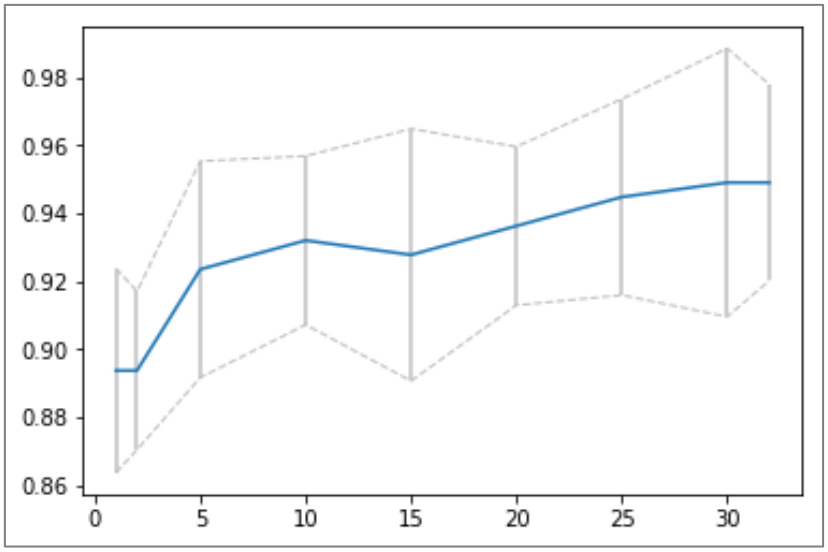


Figure 10 - Bandeau de confiance du taux de reconnaissance en validation croisée

On peut introduire la règle de l'écart-type en choisissant le modèle le plus fortement régularisé (q* le plus petit possible) dont le taux de reconnaissance est supérieur à la borne basse de l'intervalle de confiance du meilleur taux (cf. « Régressions ridge et elasticnet sous R », pages 17 et 18).

Déterminons d'abord les solutions (q*) qui sont concernées :

```
#bornes basses de l'intervalle de confiance
lbound = err_cv - std_cv
print(lbound)

#accuracy de référence – la borne basse de la dernière, celle de q = 32
ref = lbound[-1]

#les facteurs dont le taux sont supérieurs à cette référence
prm[numpy.where(err_cv > ref)[0]] #array([ 5, 10, 15, 20, 25, 30, 32])

#graphique error bar - seuil matérialisé
plt.errorbar(prm,err_cv,yerr=std_cv,ecolor="silver")
plt.plot(prm,numpy.repeat(ref,len(prm)),color="coral",linestyle="dashed",linewidth=1)
plt.show()
```

Le taux de référence est égal à 92.007%. Il est matérialisé par la ligne en pointillé rouge dans le graphique (Figure 11).

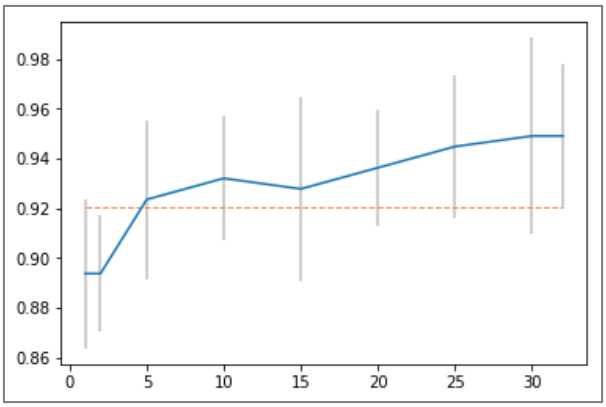


Figure 11 - Taux de référence pour la règle de l'écart-type

q* = 5 est la plus petite valeur du nombre de facteurs pour laquelle le taux en validation croisée est supérieur à ce seuil. Construisons le modèle correspondant et calculons les performances en test.

```
#disqual avec moins d'axes
disqualReg = Pipeline([("acm",MCA(n_components=5,var_labels=vote_train.columns[:-1])),
                       ("adl",LinearDiscriminantAnalysis())])

#apprentissage
disqualReg.fit(vote_train.iloc[:,:-1].values,vote_train.group)

#prédiction
pReg = disqualReg.predict(vote_test.iloc[:,:-1].values)

#taux de reconnaissance
print(metrics.accuracy_score(vote_test.group,pReg))
```

Le taux en test de 90.5%, moindre par rapport au modèle utilisant la totalité des facteurs (95%).

Je l'avais déjà remarqué dans de nombreux tutoriels. Sur des bases « classiques » où le ratio n (nombre d'observations) est largement supérieur à p (nombre d'explicatives), la préférence à la régularisation n'est pas payante. La situation ne s'inverse que lorsque le ratio lui-même s'inverse. Il en est de même du principe de parcimonie qui procède de la même philosophie.

4 Conclusion

L'outil Pipeline est particulièrement indiqué lorsque nous opérons une succession de traitements statistiques (ex. un enchaînement de transformations de variables). Certes un bon programmeur pourrait s'en passer. Mais en prenant en charge les étapes intermédiaires, il nous facilite grandement la vie et nous permet d'être plus efficace.

5 Références

LeMakiStatheux, « La méthode DISQUAL ».

Saporta G., ''Probabilités, analyse des données et statistique'', Dunod, 2006 ; Section 18.4, « Discrimination sur variables qualitatives ».

Tufféry S., ''Data Mining et Statistique Décisionnelle – L'intelligence des données'', Dunod, 2012 ; Section 12.7, « L'analyse discriminante sur variables qualitatives (méthode DISQUAL) ».

''SCIKIT-LEARN – Machine Learning in Python'', http://scikit-learn.org/stable/index.html