Analyse discriminante

Application sous Python avec discrimintools 0.0.1

Duvérier DJIFACK ZEBAZE

Table des matières

1 Analyse Factorielle Discriminante

Sommaire

Ce chapitre a pour objectif de présenter rapidement les principales fonctionnalités offertes par le package « discrimintools » pour réaliser une Analyse Factorielle Discriminante ou Analyse Discriminante Descriptive.

1.1 Présentation des données

Nous allons illustrer ce chapitre à travers l'exemple des Vins de Bordeaux (Michel Tenenhaus, 2007). On cherche à relier la qualité des vins de Bordeaux à des caractéristiques météorologiques. La variable à expliquer y est la qualité du vin et prend 3 modalités : 1 = bon, 2 = moyen et 3 = médiocre. Les variables explicatives de la qualité du vin sont les suivantes : X_1 (Somme des températures moyennes journalières (°C)), X_2 (Durée d'insolation (h)), X_3 (Nombre de jours de grande chaleur) et X_4 (Hauteur des pluies (mm)).

```
# Chargement des données
import pandas as pd
donnee = pd.read_excel("./data/vin_bordelais.xls",index_col=1)
print(donnee)
```

```
##        Obs.  Temperature  Soleil  Chaleur  Pluie   Qualite
## Annee
## 1924      1         3064    1201       10    361     Moyen
## 1925      2         3000    1053       11    338  Mediocre
## 1926      3         3155    1133       19    393     Moyen
## 1927      4         3085     970        4    467  Mediocre
## 1928      5         3245    1258       36    294       Bon
```

```
## 1929      6         3267    1386       35    225       Bon
## 1930      7         3080     966       13    417  Mediocre
## 1931      8         2974    1189       12    488  Mediocre
## 1932      9         3038    1103       14    677  Mediocre
## 1933     10         3318    1310       29    427     Moyen
## 1934     11         3317    1362       25    326       Bon
## 1935     12         3182    1171       28    326  Mediocre
## 1936     13         2998    1102        9    349  Mediocre
## 1937     14         3221    1424       21    382       Bon
## 1938     15         3019    1230       16    275     Moyen
## 1939     16         3022    1285        9    303     Moyen
## 1940     17         3094    1329       11    339     Moyen
## 1941     18         3009    1210       15    536  Mediocre
## 1942     19         3227    1331       21    414     Moyen
## 1943     20         3308    1366       24    282       Bon
## 1944     21         3212    1289       17    302     Moyen
## 1945     22         3361    1444       25    253       Bon
## 1946     23         3061    1175       12    261     Moyen
## 1947     24         3478    1317       42    259       Bon
## 1948     25         3126    1248       11    315     Moyen
## 1949     26         3458    1508       43    286       Bon
## 1950     27         3252    1361       26    346     Moyen
## 1951     28         3052    1186       14    443  Mediocre
## 1952     29         3270    1399       24    306       Bon
## 1953     30         3198    1259       20    367       Bon
## 1954     31         2904    1164        6    311  Mediocre
## 1955     32         3247    1277       19    375       Bon
## 1956     33         3083    1195        5    441  Mediocre
## 1957     34         3043    1208       14    371  Mediocre
```

1.1.1 Objectifs

L'analyse factorielle discriminante est une méthode descriptive. Elle vise à produire un système de représentation de dimension réduite qui permet de discerner les classes lorsqu'on y projette les individus. Il s'agit d'une méthode d'analyse factorielle. On peut la voir comme une variante de l'analyse en composantes principales où les centres de classes sont les individus, pondérés par leurs effectifs, et avec une métrique particulière (SAPORTA, 2006). Les variables latentes (ou discriminantes) sont exprimées par des combinaisons linéaires des variables originelles. Elles sont deux à deux orthogonales. Elles cherchent à assurer un écartement maximal entre les centres de classes. In fine, l'objectif est de mettre en évidence les caractéristiques qui permettent de distinguer au mieux les groupes.

1.1.2 Problématique

L'analyse factorielle discriminante ou analyse discriminante descriptive permet de caractériser de manière multidimensionnelle l'appartenance des individus à des groupes prédéfinis, ceci à l'aide de plusieurs variables explicatives prises de façon simultanée. En effet, il s'agit de construire un nouveau système de représentation qui permet de mettre en évidence ces groupes. Les objectifs de l'analyse factorielle discriminante sont

double :

1. **Descriptif** : Mettre en évidence les caractéristiques qui permettent de distinguer au mieux les groupes ;
2. **Prédictif** : Classer automatiquement un nouvel individu (l'affecter à un groupe) à partir de ses caractéristiques

1.1.3 Rapport de corrélation

Nous mesurons le pouvons discriminant de chaque variables X_j en utilisant l'analyse de la variance à un facteur. Pour cela, nous utilisons le rapport de corrélation définit par :

$$\eta^2(X_j, y) = \frac{\text{Somme des carrés inter - classes}}{\text{Somme des carrés totale}} \tag{1.1}$$

Cet indicateur, compris entre 0 et 1, est basé sur la dispersion des moyennes conditionnelles. Il s'agit d'un indicateur de séparabilité des groupes :

— $\eta^2(X_j, y) = 0$, la discrimination est impossible, les moyennes conditionnelles sont confondues. La somme des carrés inter - classes est nulle.
— $\eta^2(X_j, y) = 1$, la discriminantion est parfaite, les points associés aux groupes sont agglutinés autour de leur moyenne respectives : la somme des carrés intra - classes est nulle, ce qui est équivalent à la somme des carrés inter - classes est egale à la somme des carrés totale.

```
# Pouvoir discriminant
from discrimintools.eta2 import eta2

R2 = {}
for name in donnee.columns[1:-1]:
    R2[name] = eta2(donnee["Qualite"],donnee[name])
R2 = pd.DataFrame(R2).T.sort_values(by=["pvalue"])
print(R2)
```

```
##                Sum. Intra   Sum. Inter    Eta2  F-stats  pvalue
## Temperature  237722.1212  420067.4082  0.6386  27.3893  0.0000
## Soleil       202192.3712  326909.0700  0.6179  25.0607  0.0000
## Chaleur        1664.3712    1646.5700  0.4973  15.3342  0.0000
## Pluie        178499.2121   97191.1702  0.3525   8.4396  0.0012
```

Toutes les p-values sont inférieures au seuil de 5%, par conséquent, il existe une différence significative dans la qualité du vin.

1.2 AFD

1.2.1 Chargement de discrimintools

Sage précaution avec les packages pour Python, nous affichons le numéro de la version de « discrimintools » utilisée dans ce tutoriel.

```
# version
import discrimintools
print(discrimintools.__version__)
```

```
## 0.0.1
```

Nous fonctionnons avec la version « 0.0.1 ».

```
from discrimintools import CANDISC
```

On crée une instance de la classe CANDISC, en lui passant ici des étiquettes pour les lignes et les variables.

Le constructeur de la classe CANDISC possède un paramètre `n_components` qui indique le nombre d'axes discriminants à garder. Par défaut, la valeur du paramètre `n_components` est fixée à `None`.

Réalisez l'AFD sur toutes observations en tapant la ligne de code suivante :

```
# Instanciation
my_cda = CANDISC(n_components=2,target=["Qualite"],priors="prop",
                 features=["Temperature","Soleil","Chaleur","Pluie"],
                 parallelize=False)
```

- `n_components` : le nombre d'axes discriminants à garder dans les résultats
- `target` : le label de la variable cible.
- `features` : les noms des variables explicatives. Si c'est None alors toutes les variables quantitatives seront utilisées.
- `priors` : les probabilités *a priori* d'appartenance aux classes
- `parallelize` : paralleliser l'algorithme.

On estime le modèle en appliquant la méthode `.fit` de la classe CANDISC sur le jeu de données à traiter.

```
# Apprentissage
my_cda.fit(donnee)
```

```
## CANDISC(features=['Temperature', 'Soleil', 'Chaleur', 'Pluie'], n_components=2,
##         priors='prop', target=['Qualite'])
```

1.2.2 Les valeurs propres

L'exécution de la méthode `my_cda.fit(donnee)` provoque le calcul de plusieurs attributs parmi lesquels `my_cda.eig_`.

```
print(my_cda.eig_)
```

```
##      Eigenvalue  Difference  Proportion  Cumulative
## LD1    3.278860    3.140286   95.945086   95.945086
## LD2    0.138574         NaN    4.054914  100.000000
```

L'attribut `my_cda.eig_` contient :

- en 1ère colonne : les valeurs propres en valeur absolue
- en 2ème colonne : les différences des valeurs propres
- en 3ème colonne : les valeurs propres en pourcentage de la variance totale (proportions)
- en 4ème colonne : les valeurs propres en pourcentage cumulé de la variance totale.

Le premier axe discriminant contient 96% de l'information totale disponible.

On peut obtenir un résumé des principaux résultats en utilisant la fonction `summaryCANDISC`.

```
from discrimintools import summaryCANDISC
summaryCANDISC(my_cda)
```

```
##                      Canonical Discriminant Analysis - Results
##
##
## Summary Information
##
##                infos  Value                  DF  DF value
## 0  Total Sample Size     34            DF Total        33
## 1          Variables      4   DF Within Classes        31
## 2            Classes      3  DF Between Classes         2
##
## Class Level information
##
##           Frequency  Proportion  Prior Probability
## Qualite
## Mediocre         12    0.352941           0.352941
## Bon              11    0.323529           0.323529
## Moyen            11    0.323529           0.323529
##
## Importance of components
##                            LD1      LD2
## Variance                 3.279    0.139
## Difference               3.140      NaN
## % of var.               95.945    4.055
## Cumulative % of var.    95.945  100.000
##
## Test of H0: The canonical correlations in the current row and all that follow are zero
##
##    statistic  DDL num.  DDL den.   Pr>F
## 0      8.451       8.0      56.0  0.000
## 1      1.340       3.0      29.0  0.281
##
## Group means:
##
##                   Bon  Mediocre     Moyen
## Temperature  3306.364  3037.333  3140.909
## Soleil       1363.636  1126.417  1262.909
```

```
## Chaleur        28.545    12.083    16.455
## Pluie         305.000   430.333   339.636
##
## Coefficients of canonical discriminants:
##
##                 LD1    LD2
## Temperature  -0.009  0.000
## Soleil       -0.007  0.005
## Chaleur       0.027 -0.128
## Pluie         0.006 -0.006
## intercept    32.876 -2.165
##
## Classification functions coefficients:
##
##                 Bon  Mediocre  Moyen
## Temperature   0.018    -0.018  0.001
## Soleil        0.013    -0.015  0.004
## Chaleur      -0.023     0.084 -0.069
## Pluie        -0.011     0.014 -0.004
## intercept   -72.590    65.609 -7.192
##
## Individuals (the 10 first)
##
##          LD1    LD2
## Annee
## 1924   0.883  0.872
## 1925   2.325  0.094
## 1926   0.995 -0.833
## 1927   2.727 -0.247
## 1928  -0.744 -1.721
## 1929  -2.231 -0.484
## 1930   2.747 -1.109
## 1931   2.534 -0.236
## 1932   3.731 -2.114
## 1933  -1.130 -1.368
##
## Correlations between Canonical and Original Variables
##
##              total.1  between.1  within.1  total.2  between.2  within.2
## Temperature   -0.901     -0.987    -0.724   -0.375     -0.211    -0.584
## Soleil        -0.897     -0.999    -0.701    0.116      0.003     0.176
## Chaleur       -0.771     -0.957    -0.525   -0.590     -0.336    -0.780
## Pluie          0.663      0.977     0.398   -0.361     -0.167    -0.421
##
## Class Means on Canonical Variables
##
##             LD1    LD2
## Qualite
## Bon      -2.122 -0.272
## Mediocre  2.079 -0.221
## Moyen    -0.146  0.513
```

Le champ `.coef_` nous intéresse particulièrement. Il correspond aux coefficients des fonctions discriminantes :

```
# Affichage brut des ceofficients
print(my_cda.coef_)
```

```
##                    LD1       LD2
## Temperature -0.008566  0.000046
## Soleil      -0.006774  0.005329
## Chaleur      0.027054 -0.127636
## Pluie        0.005866 -0.006175
```

La matrice est de dimension (4,2) puisque nous avons un problème à $(K = 3)$ classes (d'où $K - 1$ axes discriminants) et 4 descripteurs.

```
#dimensions
print(my_cda.coef_.shape)
```

```
## (4, 2)
```

Il ne faut pas oublier les constantes (*intercept*) des fonctions discriminantes.

```
# et les constantes pour chaque classe
print(my_cda.intercept_)
```

```
## LD1    32.876282
## LD2    -2.165279
## Name: intercept, dtype: float64
```

Nous pouvons dès lors adopter une présentation plus sympathique des fonctions discriminantes. Pour ce faire, nous utilisons la fonction `get_candisc_coef` en fixant le paramètre « `choice = "absolute"` ».

```
# Affichage des coefficients
from discrimintools import get_candisc_coef
coef = get_candisc_coef(my_cda,choice="absolute")
coef
```

```
##                     LD1       LD2
## Temperature  -0.008566  0.000046
## Soleil       -0.006774  0.005329
## Chaleur       0.027054 -0.127636
## Pluie         0.005866 -0.006175
## intercept    32.876282 -2.165279
```

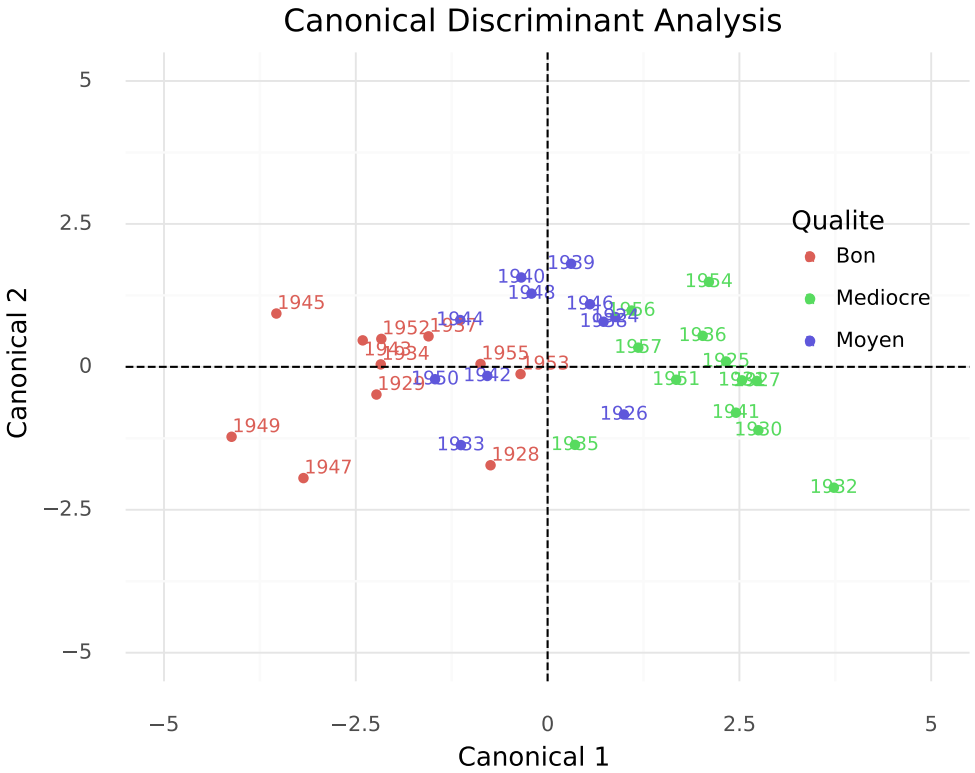
1.3 Représentations factorielles

1.3.1 Coordonnées des individus

```
# Coordonnées des individus
from discrimintools import get_candisc_ind
ind_coord = get_candisc_ind(my_cda)["coord"]
print(ind_coord.head(6))
```

```
##              LD1       LD2
## Annee
## 1924    0.882552  0.871537
## 1925    2.325456  0.094220
## 1926    0.994856 -0.832957
## 1927    2.726862 -0.247244
## 1928   -0.743596 -1.721167
## 1929   -2.230889 -0.484319
```

```
# Carte des individus
import plotnine as pn
from discrimintools import fviz_candisc
p = (fviz_candisc(my_cda,x_lim=(-5,5),y_lim=(-5,5),repel=True)+
      pn.theme(legend_direction="vertical",legend_position=(0.8,0.6)))
print(p)
```



1.3.2 Coordonnées des centres de classes

L'introduction des barycentre spermet de mieux situer la qualité relative des facteurs dans la discrimination des classes.

```
# Coordonnées des centres de classes
zk = my_cda.classes_["coord"]
print(zk)
```

```
##                LD1       LD2
## Qualite
## Bon      -2.121963 -0.271812
## Mediocre  2.079247 -0.221184
## Moyen    -0.146307  0.513104
```

1.4 Evaluation globale du modèle

1.4.1 Evaluation statistique des facteurs

1.4.1.1 Distance entre centres de classes

Dans le plan factoriel, les distances sont camptabilisées à l'aide d'une simple distance euclidienne.

```
# Distances entre centres de classes
print(my_cda.classes_["dist"])
```

```
##                 Bon   Mediocre     Moyen
## Bon        0.000000  17.652729  4.519311
## Mediocre  17.652729   0.000000  5.492267
## Moyen      4.519311   5.492267  0.000000
```

1.4.1.2 Pouvoir discriminant des facteurs

Le pouvoir discriminant des facteurs est traduit par les valeurs propres qui leurs sont associées.

```
print(my_cda.eig_)
```

```
##      Eigenvalue  Difference  Proportion  Cumulative
## LD1    3.278860    3.140286   95.945086   95.945086
## LD2    0.138574         NaN    4.054914  100.000000
```

1.4.1.3 Test MANOVA

discrimintools fournit un test de sgnificativité globale du modèle.

```
# Significativité globale du modèle
print(my_cda.statistics_["manova"])
```

```
##                   Multivariate linear model
## ============================================================
##
## ------------------------------------------------------------
##          Qualite          Value  Num DF  Den DF F Value Pr > F
## ------------------------------------------------------------
##            Wilks' lambda 0.2053 8.0000 56.0000  8.4505 0.0000
##           Pillai's trace 0.8880 8.0000 58.0000  5.7896 0.0000
##   Hotelling-Lawley trace 3.4174 8.0000 37.7500 11.7280 0.0000
##      Roy's greatest root 3.2789 4.0000 29.0000 23.7717 0.0000
## ============================================================
```

Nous nous intéressons en particulier à la ligne relative à « Wilks' Lambda ».

1.4.1.4 Performance globale

Nosu affichons les valeurs des statistiques suivantes : Lambda de Wilks, Transformation de Bartlett et de RAO.

```
# Performance globale
print(my_cda.statistics_["performance"])
```

```
##                 Stat      Value       p-value
## 0     Wilks' Lambda   0.205263           NaN
## 1  Bartlett -- C(8)  46.712169  1.739815e-07
## 2    Rao -- F(8,56)   8.450507  1.890358e-07
```

L'écartement entre les barycentres conditionnels est significatif à 5%. L'analyse discriminante est viable dans ce contexte.

1.4.2 Test sur un ensemble de facteurs

Combien de facteurs faut - il retenir ?.

```
# Test sur un ensemble de facteur
print(my_cda.statistics_["likelihood_test"])
```

```
##    statistic  DDL num.  DDL den.          Pr>F
## 0   8.450507       8.0      56.0  1.890358e-07
## 1   1.339549       3.0      29.0  2.807850e-01
```

1.4.3 Matrices de covariance

Elles sont directement fournies par l'objet « discrimintools »

1.4.4 Matrice de covariance intra - classe

```
# Covariance intra - classe
print(my_cda.cov_["within"])
```

```
##              Temperature       Soleil     Chaleur        Pluie
## Temperature  6991.827094  1714.255793  420.615865   392.273619
## Soleil       1714.255793  5946.834447  154.271168  -144.059715
## Chaleur       420.615865   154.271168   48.952094   -31.750446
## Pluie         392.273619  -144.059715  -31.750446  5249.976827
```

1.4.5 Matrice de covariance totale

```
# Covariance totale
print(my_cda.cov_["total"])
```

```
##               Temperature        Soleil      Chaleur        Pluie
## Temperature  19346.750865  12360.302768  1187.420415 -5130.448097
## Soleil       12360.302768  15561.807093   795.792388 -5317.760381
## Chaleur       1187.420415    795.792388    97.380623  -356.451557
## Pluie        -5130.448097  -5317.760381  -356.451557  8108.540657
```

1.4.6 Matrice de covariance inter - classe

```
# Matrice de covaiance inter - classe
print(my_cda.cov_["between"])
```

```
##               Temperature        Soleil     Chaleur        Pluie
## Temperature  12354.923771  10646.046975  766.804551 -5522.721715
## Soleil       10646.046975   9614.972646  641.521220 -5173.700666
## Chaleur        766.804551    641.521220   48.428528  -324.701111
## Pluie        -5522.721715  -5173.700666 -324.701111  2858.563830
```

1.4.7 Interprétation des facteurs

Elle permet la compréhension de la nature des facteurs.

1.4.7.1 Corrélation totale

```
# Correlation totale
print(my_cda.corr_["total"])
```

```
##                   LD1       LD2
## Temperature -0.900589 -0.374779
## Soleil      -0.896744  0.116190
## Chaleur     -0.770513 -0.590030
## Pluie        0.662815 -0.361294
```

1.4.7.2 Correlation intra - classe

```
# Correlation intra - classe
print(my_cda.corr_["within"])
```

```
##                    LD1       LD2
## Temperature -0.724221 -0.584256
## Soleil      -0.701280  0.176148
## Chaleur     -0.525372 -0.779910
## Pluie        0.398218 -0.420797
```

1.4.7.3 Correlation inter - classe

```
# Corrélation inter - classe
print(my_cda.corr_["between"])
```

```
##                    LD1       LD2
## Temperature -0.986651 -0.211244
## Soleil      -0.998654  0.002625
## Chaleur     -0.957391 -0.335599
## Pluie        0.976576 -0.166812
```

1.5 Prediction des classes

Considérons l'année 1958. Les données (hypothétiques) de cette année sont :

```
## Inidvidu supplémentaire
XTest = pd.DataFrame({"Temperature" : 3000,
                      "Soleil" : 1100,
                      "Chaleur" : 20,
                      "Pluie" : 300},index=[1958])
XTest
```

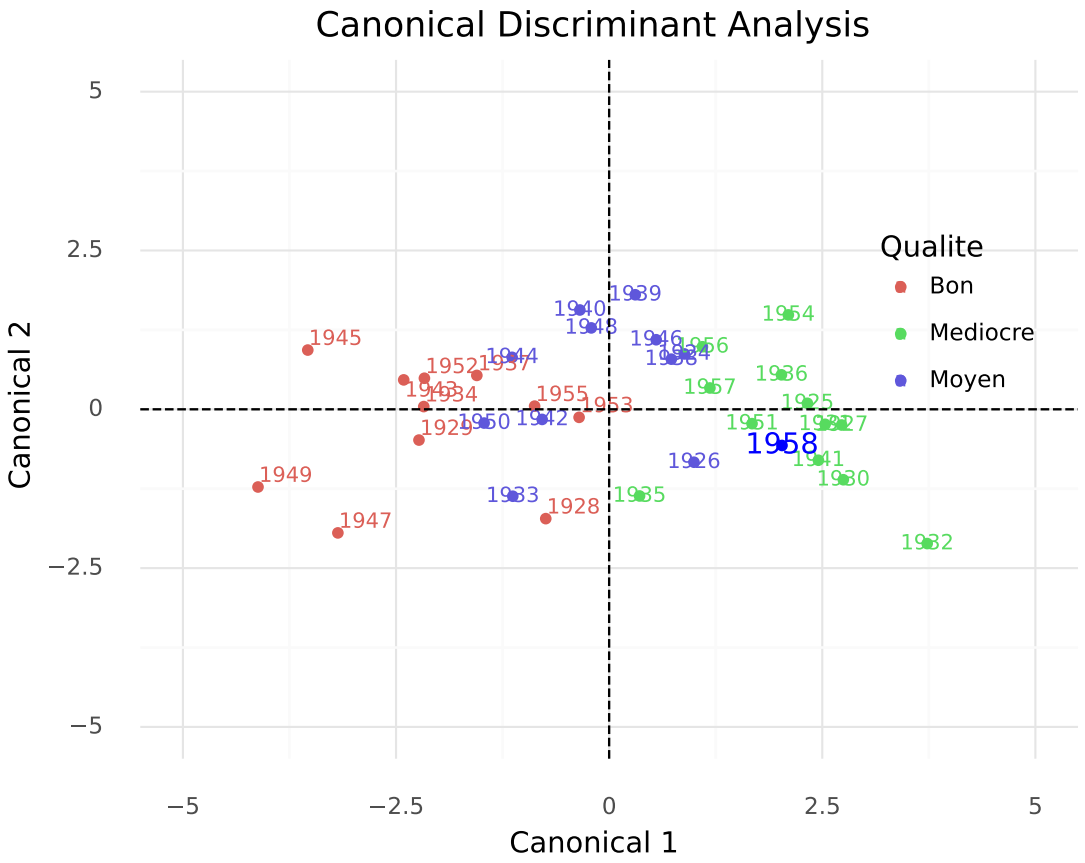
```
##       Temperature  Soleil  Chaleur  Pluie
## 1958         3000    1100       20    300
```

1.5.1 Coordonnées factorielles

```
# Coordonées factorielles
ind_sup_coord = my_cda.transform(XTest)
print(ind_sup_coord)
```

```
##            LD1       LD2
## 1958  2.027679 -0.569395
```

```
p = (p + pn.annotate("point", x = ind_sup_coord.iloc[:,0],
                     y = ind_sup_coord.iloc[:,1], color="blue")+
         pn.annotate("text", x = ind_sup_coord.iloc[:,0],
                     y = ind_sup_coord.iloc[:,1], label = "1958",color="blue"))
print(p)
```



La fonction `predict()` permet de produire les prédictions à partir de la matrice des explicatives en test.

```
# Prédiction simple
pred = my_cda.predict(XTest)
print(pred)
```

```
## 1958    Mediocre
## Name: prediction, dtype: object
```

1.5.2 Fonctions de classement explicites

La classe CANDISC de discrimintools retourne les fonctions de décision issues de l'analyse factorielle discriminante. Pour celà, il faut spécifier l'argument « `choice == "score"`.

```r
# Fonctions de décision - AFD
score_coef = get_candisc_coef(my_cda,choice = "score")
print(score_coef)
```

```
##                    Bon   Mediocre     Moyen
## Temperature   0.018164  -0.017821  0.001277
## Soleil        0.012925  -0.015263  0.003726
## Chaleur      -0.022716   0.084484 -0.069449
## Pluie        -0.010768   0.013562 -0.004026
## intercept   -72.590473  65.609287 -7.191833
```

1.5.3 Prédiction des classes sur l'échantillon d'apprentissage

```python
import numpy as np
# Prédiction sur XTrain
X = donnee[donnee.columns[:-1]]
y_pred = my_cda.predict(X)

# Distribution des classes prédictes
print(y_pred.value_counts())
```

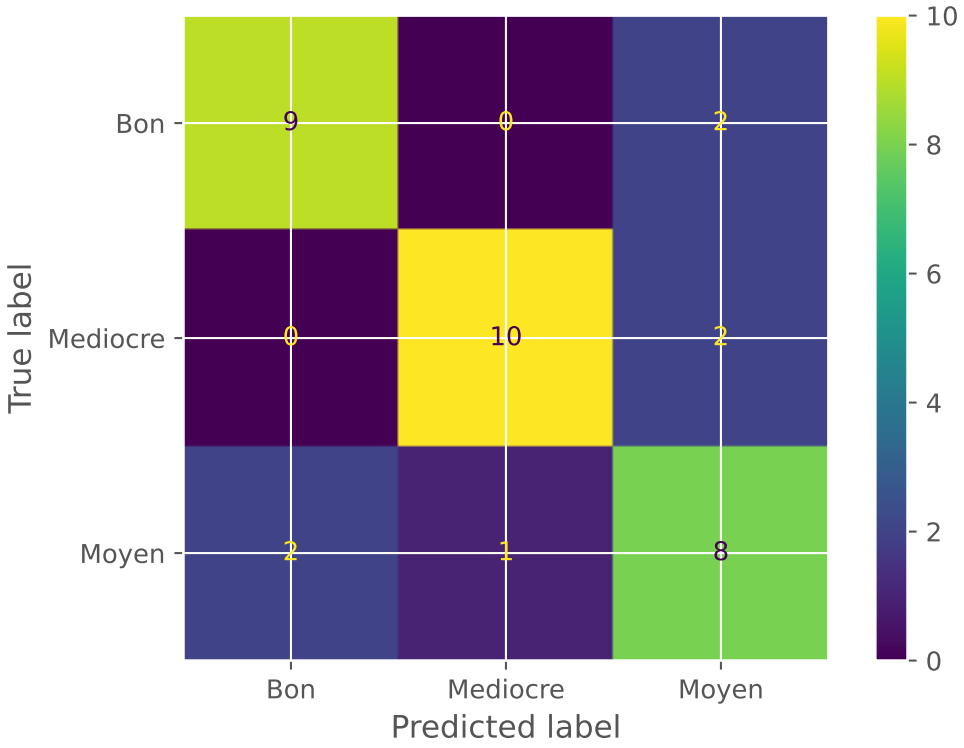
```
## prediction
## Moyen       12
## Mediocre    11
## Bon         11
## Name: count, dtype: int64
```

11 observations ont été prédite « Bon », 11 « Medicocre » et 12 « Moyen ».

1.5.4 Matrice de confusion et taux de bon classement

La matrice de confusion est issue de la confrontation entre ces prédictions et les classes observées. Nous faisons appel au module « metrics » de la librairie « scikit-learn ».

```python
# Matrice de confusion
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(donnee.Qualite,y_pred,labels=my_cda.classes_["classes"])
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=my_cda.classes_["classes"])
disp.plot();
plt.show()
```



La fonction `score()` nous donne le taux de reconnaissance (ou taux de succès).

```
# Taux de succès
print(my_cda.score(X,donnee.Qualite))
```

```
## 0.7941176470588235
```

Notre taux de succès est de 79%.

La fonction `classification_report()` génère un rapport sur les performances globales, mais aussi sur les reconnaissances par classe (rappel, précision et F-Measure[F1-Score])

```
# rapport
from sklearn.metrics import classification_report
print(classification_report(donnee.Qualite,y_pred))
```

```
##               precision    recall  f1-score   support
##
##          Bon       0.82      0.82      0.82        11
##     Mediocre       0.91      0.83      0.87        12
##        Moyen       0.67      0.73      0.70        11
##
##     accuracy                           0.79        34
##    macro avg       0.80      0.79      0.79        34
## weighted avg       0.80      0.79      0.80        34
```

Nous retrouvons, entre autres le taux de succès de 79%.

1.5.5 Probabilité d'appartenance

« discrimintools » peut aussi calculer les probabilités d'affectation aux classes avec `predict_proba()`. Elle permettent une analyse plus fine de la qualité du modèle, via la construction de la courbe ROC par exemple, dont le principe reste valable pour les problèmes multi - classes.

```
# Probabilité d'appartenance
print(my_cda.predict_proba(X).head(6))
```

```
##             Bon  Mediocre     Moyen
## Annee
## 1924   0.006695  0.344613  0.648692
## 1925   0.000045  0.958846  0.041109
## 1926   0.009222  0.698039  0.292739
## 1927   0.000009  0.986519  0.013472
## 1928   0.641715  0.031256  0.327029
## 1929   0.933408  0.000094  0.066499
```

1.6 Sélection de variables

Limiter le modèle aux variables explicatives pertinentes est primordial pour l'interprétation et le deploiement des modèles.

1.6.1 Backward selection

```
# Selection backward
from discrimintools import STEPDISC
backward = STEPDISC(my_cda,method="backward",alpha=0.01,
                    model_train=False,verbose=True)
```

```
##              Wilks L.  Partial L.         F   p-value
## Temperature  0.257007    0.201333  3.529197  0.042966
## Soleil       0.266324    0.229274  4.164681  0.026098
## Chaleur      0.219098    0.063145  0.943617  0.360034
## Pluie        0.248287    0.173285  2.934485  0.069660
##
##              Wilks L.  Partial L.         F   p-value
## Temperature  0.318676    0.312475  6.590138  0.004372
## Soleil       0.279988    0.217473  4.029702  0.028559
## Pluie        0.260568    0.159154  2.744539  0.080982
##
##              Wilks L.  Partial L.         F   p-value
## Temperature  0.382143    0.318139  6.998608  0.003202
## Soleil       0.361395    0.278993  5.804248  0.007398
```

Les variables sélectionnées sont les suivantes :

```
# Variables sélectionnées
selectedVar = backward.results_["selected"]
selectedVar
```

```
## ['Temperature', 'Soleil']
```

Nous entraînons le modèle avec les variables sélectionnées :

```
# Modèle réduit
my_cda2 = CANDISC(n_components=2,target=["Qualite"],priors="prop",
                  features=selectedVar,parallelize=False).fit(donnee)
```

```
# Summary
summaryCANDISC(my_cda2,to_markdown=False)
```

```
##                       Canonical Discriminant Analysis - Results
##
##
## Summary Information
##
##                 infos  Value                   DF  DF value
## 0  Total Sample Size     34            DF Total        33
## 1           Variables      2   DF Within Classes        31
## 2             Classes      3  DF Between Classes         2
##
## Class Level information
##
##           Frequency  Proportion  Prior Probability
## Qualite
## Mediocre         12    0.352941           0.352941
## Bon              11    0.323529           0.323529
## Moyen            11    0.323529           0.323529
##
## Importance of components
##                              LD1
## Variance                   2.643
## Difference                   NaN
## % of var.                100.000
## Cumulative % of var.     100.000
##
## Test of H0: The canonical correlations in the current row and all that follow are zero
##
##    statistic  DDL num.  DDL den.  Pr>F
## 0      81.94       1.0      31.0   0.0
##
## Group means:
##
##                    Bon  Mediocre     Moyen
## Temperature  3306.364  3037.333  3140.909
## Soleil       1363.636  1126.417  1262.909
```

```
##
## Coefficients of canonical discriminants:
##
##                 LD1
## Temperature  -0.007
## Soleil       -0.007
## intercept    32.868
##
## Classification functions coefficients:
##
##                 Bon  Mediocre  Moyen
## Temperature   0.015    -0.013 -0.000
## Soleil        0.015    -0.013 -0.000
## intercept   -68.037    56.571 -0.795
##
## Individuals (the 10 first)
##
##          LD1
## Annee
## 1924   1.046
## 1925   2.629
## 1926   0.876
## 1927   2.615
## 1928  -0.729
## 1929  -1.850
## 1930   2.683
## 1931   1.807
## 1932   1.972
## 1933  -1.662
##
## Correlations between Canonical and Original Variables
##
##              total.1  between.1  within.1
## Temperature   -0.933     -0.995    -0.813
## Soleil        -0.917     -0.993    -0.777
##
## Class Means on Canonical Variables
##
##             LD1
## Qualite
## Bon      -1.976
## Mediocre  1.802
## Moyen     0.010
```

1.6.2 Forward selection

```
# Selection forward
forward = STEPDISC(my_cda,method="forward",alpha=0.01,
                   model_train=False,verbose=True)
```

```
##              Wilks L.  Partial L.          F       p-value
## Temperature  0.361395    0.638605  27.389310  1.408416e-07
## Soleil       0.382143    0.617857  25.060741  3.345802e-07
## Chaleur      0.502688    0.497312  15.334220  2.344932e-05
## Pluie        0.647463    0.352537   8.439607  1.185298e-03
##
##          Wilks L.  Partial L.         F   p-value
## Soleil   0.260568    0.278993  5.804248  0.007398
## Chaleur  0.348810    0.034825  0.541217  0.175091
## Pluie    0.279988    0.225260  4.361330  0.021744
##
##          Wilks L.  Partial L.         F   p-value
## Chaleur  0.248287    0.047132  0.717220  0.264148
## Pluie    0.219098    0.159154  2.744539  0.080982
```

Les variables sélectionnées sont les suivantes :

```
# Variables sélectionnées
selectedVar2 = forward.results_["selected"]
selectedVar2
```

```
## ['Temperature', 'Soleil']
```

Nous entraînons le modèle avec les variables sélectionnées :

```
# Modèle réduit
my_cda3 = CANDISC(n_components=2,target=["Qualite"],priors="prop",
                  features=selectedVar2,parallelize=False).fit(donnee)

# Summary
summaryCANDISC(my_cda3,to_markdown=False)
```

```
##                      Canonical Discriminant Analysis - Results
##
##
## Summary Information
##
##                 infos  Value                   DF  DF value
## 0  Total Sample Size     34            DF Total        33
## 1           Variables      2   DF Within Classes        31
## 2             Classes      3  DF Between Classes         2
##
## Class Level information
##
##           Frequency  Proportion  Prior Probability
## Qualite
## Mediocre         12    0.352941           0.352941
## Bon              11    0.323529           0.323529
## Moyen            11    0.323529           0.323529
##
```

```
## Importance of components
##                             LD1
## Variance                  2.643
## Difference                  NaN
## % of var.               100.000
## Cumulative % of var.    100.000
##
## Test of H0: The canonical correlations in the current row and all that follow are zero
##
##    statistic  DDL num.  DDL den.  Pr>F
## 0      81.94       1.0      31.0   0.0
##
## Group means:
##
##                    Bon  Mediocre     Moyen
## Temperature  3306.364  3037.333  3140.909
## Soleil       1363.636  1126.417  1262.909
##
## Coefficients of canonical discriminants:
##
##                  LD1
## Temperature  -0.007
## Soleil       -0.007
## intercept    32.868
##
## Classification functions coefficients:
##
##                  Bon  Mediocre  Moyen
## Temperature   0.015    -0.013 -0.000
## Soleil        0.015    -0.013 -0.000
## intercept   -68.037    56.571 -0.795
##
## Individuals (the 10 first)
##
##          LD1
## Annee
## 1924   1.046
## 1925   2.629
## 1926   0.876
## 1927   2.615
## 1928  -0.729
## 1929  -1.850
## 1930   2.683
## 1931   1.807
## 1932   1.972
## 1933  -1.662
##
## Correlations between Canonical and Original Variables
##
##              total.1  between.1  within.1
## Temperature   -0.933     -0.995    -0.813
```

```
## Soleil        -0.917     -0.993    -0.777
##
## Class Means on Canonical Variables
##
##             LD1
## Qualite
## Bon      -1.976
## Mediocre  1.802
## Moyen     0.010
```

Bien qu'il soit possible de déduire un mécanisme de classement en analyse factorielle discriminante, sa finalité est bien différente de l'analyse discriminante linéaire, prédictive. Mais les deux approches se rejoignent.

2 Analyse Discriminante Linéaire

Sommaire

Ce chapitre a pour objectif de présenter rapidement les principales fonctionnalités offertes par le package « discrimintools » pour réaliser une Analyse Discriminante Linéaire.

2.1 Présentation des données

L'analyse discriminante linéaire fait partie des technique d'analyse discriminante prédictive. C'est une méthode prédictive où le modèle s'exprime sous la forme d'un système d'équations linéaires des variables explicatives. Il s'agit d'expliquer et de prédire l'appartenance d'un individu à une classe (groupe) prédéfinie à partir de ses caractéristiques mesurées à l'aide de variables prédictives.

2.1.1 Importation des données

Nous utilisons les données « alcool »(*cf.* fr_Tanagra_LDA_Python.pdf). Il s'agit de prédire le TYPE d'alcool (KIRSCH, MIRAB, POIRE) à partir de ses composants (butanol, méthanol, etc ; 8 variables).

```
# Chargement des données
import pandas as pd
DTrain = pd.read_excel("./data/Eau_de_vie_LDA.xlsx",sheet_name="TRAIN")
print(DTrain.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
```

```
## RangeIndex: 52 entries, 0 to 51
## Data columns (total 9 columns):
##  #   Column  Non-Null Count  Dtype
## ---  ------  --------------  -----
##  0   TYPE    52 non-null     object
##  1   MEOH    52 non-null     float64
##  2   ACET    52 non-null     float64
##  3   BU1     52 non-null     float64
##  4   BU2     52 non-null     float64
##  5   ISOP    52 non-null     int64
##  6   MEPR    52 non-null     float64
##  7   PRO1    52 non-null     float64
##  8   ACAL    52 non-null     float64
## dtypes: float64(7), int64(1), object(1)
## memory usage: 3.8+ KB
## None
```

2.1.2 Distribution relative

Nous calculons la distribution relative des classes :

```
# Distribution relative des classes
d = (DTrain.TYPE.value_counts(normalize=True).to_frame()
                .rename(columns={"index":"TYPE"}))
print(d)
```

```
##         proportion
## TYPE
## POIRE     0.384615
## KIRSCH    0.326923
## MIRAB     0.288462
```

Les classes semblent assez équilibrées.

2.2 LDA

2.2.1 Modélisation avec discrimintools

Sage précaution avec les packages pour Python, nous affichons le numéro de la version de « discrimintools » utilisée dans ce tutoriel.

```
# version
import discrimintools
print(discrimintools.__version__)
```

```
## 0.0.1
```

Nous fonctionnons avec la version « 0.0.1 ».

```
# Importation
from discrimintools import LDA
```

On crée une instance de la classe LDA, en lui passant ici des étiquettes pour les lignes et les variables.

```
# Instanciation
lda = LDA(target=["TYPE"],priors = "prop")
```

On estime le modèle en appliquant la méthode `.fit` de la classe LDA sur le jeu de données.

```
# Entraînement du modèle
lda.fit(DTrain)
```

```
## LDA(priors='prop', target=['TYPE'])
```

L'exécution de la méthode `lda.fit(D)` provoque le calcul de plusieurs attributs parmi lesquels `lda.coef_`. Ce champ nous intéresse particulièrement car il correspond aux coefficients des fonctions de classement.

```
# Coefficients des fonctions de score
print(lda.coef_)
```

```
##         KIRSCH     MIRAB     POIRE
## MEOH  0.003428  0.029028  0.033390
## ACET  0.006390  0.016413  0.007513
## BU1  -0.063681  0.405390  0.318047
## BU2  -0.000883  0.071352  0.114993
## ISOP  0.023082  0.029763 -0.008486
## MEPR  0.037494 -0.128942  0.061780
## PRO1  0.001971 -0.005413 -0.008318
## ACAL  0.066184 -0.226424 -0.130332
```

Le tableau est de dimension $(8, 3)$ puisque nous avons un problème à $(K = 3)$ classes (le nombre de modalités de la variable cible origine) et 8 descripteurs.

Il ne faut pas oublier les constantes (intercept) des fonctions linéaires :

```
# et les constantes pour chaque classe
print(lda.intercept_)
```

```
##              KIRSCH      MIRAB      POIRE
## Intercept -5.016453 -18.840685 -24.764879
```

```
# Summary
from discrimintools import summaryLDA
summaryLDA(lda)
```

```
##                      Linear Discriminant Analysis - Results
##
##
## Summary Information
##
##                infos  Value                   DF  DF value
## 0  Total Sample Size     52            DF Total        51
## 1          Variables      8   DF Within Classes        49
## 2            Classes      3  DF Between Classes         2
##
## Class Level information
##
##         Frequency  Proportion  Prior Probability
## TYPE
## POIRE          20    0.384615           0.384615
## KIRSCH         17    0.326923           0.326923
## MIRAB          15    0.288462           0.288462
##
## Group means:
##
##        KIRSCH    MIRAB     POIRE
## MEOH  371.676  934.200  1084.350
## ACET  203.018  235.067   185.250
## BU1     1.200   20.200    21.330
## BU2    21.018   13.567    49.380
## ISOP   81.588   90.933   118.050
## MEPR   28.894   29.400    50.000
## PRO1  790.771  195.267   317.400
## ACAL   12.012   12.353    14.495
##
## Coefficients of linear discriminants:
##
##            KIRSCH   MIRAB   POIRE
## MEOH        0.003   0.029   0.033
## ACET        0.006   0.016   0.008
## BU1        -0.064   0.405   0.318
## BU2        -0.001   0.071   0.115
## ISOP        0.023   0.030  -0.008
## MEPR        0.037  -0.129   0.062
## PRO1        0.002  -0.005  -0.008
## ACAL        0.066  -0.226  -0.130
## Intercept  -5.016 -18.841 -24.765
##
## Individuals (the 10 first) scores
##
##    KIRSCH   MIRAB   POIRE
## 0   1.368  -7.908 -11.389
## 1   4.894 -10.146 -13.263
## 2   3.772  -8.009 -13.542
## 3   0.853  -9.247 -13.117
## 4  -3.431 -17.240 -23.800
```

```
## 5   8.385  -5.734 -12.019
## 6   6.220  -9.387  -8.849
## 7  -3.069 -17.983 -23.098
## 8   0.338 -11.215 -11.580
## 9  -2.724 -22.471 -28.617
```

2.3 Inspection de l'objet LDA

— `call_["priors"]` correspond à la distribution relative des classes.

```
# distribution des classes
priors = lda.call_["priors"]
print(priors)
```

```
## TYPE
## POIRE     0.384615
## KIRSCH    0.326923
## MIRAB     0.288462
## Name: proportion, dtype: float64
```

— `summary_information_` correspond à la distribution absolue et relative des classes

```
# distribution absolue et relative des classes
print(lda.summary_information_)
```

```
##                infos  Value                  DF  DF value
## 0  Total Sample Size     52           DF Total        51
## 1          Variables      8   DF Within Classes        49
## 2            Classes      3  DF Between Classes         2
```

— `statistics_["Eta2"]` correspond au rapport de corrélation $\eta^2(X, y)$ entre les variables explicatives et la variable expliquée.

```
# Rapport de corrélation
print(lda.statistics_["Eta2"])
```

```
##         Sum. Intra    Sum. Inter    Eta2  F-stats  pvalue
## MEOH  1.970961e+06  5.002713e+06  0.7174  62.1861  0.0000
## ACET  7.395802e+05  2.141856e+04  0.0281   0.7095  0.4969
## BU1   1.774842e+03  4.427150e+03  0.7138  61.1126  0.0000
## BU2   1.384704e+05  1.293152e+04  0.0854   2.2880  0.1122
## ISOP  1.056640e+05  1.336308e+04  0.1123   3.0985  0.0541
## MEPR  1.203905e+04  5.362097e+03  0.3081  10.9121  0.0001
## PRO1  1.670689e+07  3.290220e+06  0.1645   4.8250  0.0122
## ACAL  3.241144e+03  6.735320e+01  0.0204   0.5091  0.6042
```

— `classes_["mean"]` indique les moyennes des variables conditionnellement aux classes

```
# moyennes conditionnelles des variables
print(lda.classes_["mean"])
```

```
##                MEOH        ACET    BU1  ...        MEPR        PRO1        ACAL
## KIRSCH   371.676471  203.017647   1.20  ...   28.894118  790.770588  12.011765
## MIRAB    934.200000  235.066667  20.20  ...   29.400000  195.266667  12.353333
## POIRE   1084.350000  185.250000  21.33  ...   50.000000  317.400000  14.495000
##
## [3 rows x 8 columns]
```

— `classes_["mahalanobis"]` indique la matrice des distances (au carré) de Mahalanobis

```
# Matrice des distances (au carré) de Mahalanobis
print(lda.classes_["mahalanobis"])
```

```
##            KIRSCH      MIRAB      POIRE
## KIRSCH   0.000000  27.371480  36.048105
## MIRAB   27.371480   0.000000   5.305086
## POIRE   36.048105   5.305086   0.000000
```

2.4 Evaluation globale du modèle

2.4.1 Statistiques multivariées

Le test de significativité globale du modèle est basé sur l'écartement entre les barycentres conditionnels pour l'analyse discriminante.

```
# MANOVA Test
print(lda.statistics_["manova"])
```

```
##                   Multivariate linear model
## ============================================================
##
## ------------------------------------------------------------
##            TYPE           Value   Num DF  Den DF F Value Pr > F
## ------------------------------------------------------------
##           Wilks' lambda 0.0667 16.0000 84.0000 15.0761 0.0000
##          Pillai's trace 1.3213 16.0000 86.0000 10.4630 0.0000
##  Hotelling-Lawley trace 8.1741 16.0000 65.2314 21.0816 0.0000
##     Roy's greatest root 7.3868  8.0000 43.0000 39.7042 0.0000
## ============================================================
```

Nous nous intéressons en particulier à la ligne relative à « Wilks' Lambda ».

2.4.2 Matrice de covariance

2.4.2.1 Matrice de covariance intra - classe

Elle est directement fournie par l'objet « discrimintools ».

```
# Matrice de covariance intra - classe
print(lda.cov_["within"])
```

```
##               MEOH          ACET  ...          PRO1        ACAL
## MEOH  40223.702461   7653.791777  ...   7923.707923  833.681423
## ACET   7653.791777  15093.472817  ...  14043.416983  462.680778
## BU1     299.526327   -110.066327  ...    272.533878    3.353327
## BU2   -2522.675774    148.729756  ...  24116.532085  -22.957528
## ISOP   3494.675210    293.413065  ...  -1034.013045   15.066136
## MEPR   1340.445258    223.269220  ...    305.218511    2.405330
## PRO1   7923.707923  14043.416983  ... 340956.952013  798.808215
## ACAL    833.681423    462.680778  ...    798.808215   66.145806
##
## [8 rows x 8 columns]
```

2.4.2.2 Matrice de covariance totale

La matrice de covariance totale est proposée par l'objet « discrimintools ».

```
# Matrice de covariance totale
print(lda.cov_["total"])
```

```
##               MEOH          ACET  ...          PRO1         ACAL
## MEOH  136738.708428   6617.583880  ...  -65773.762010  1082.729148
## ACET    6617.583880  14921.543661  ...   12047.594167   427.863371
## BU1     3173.906712    -99.421071  ...   -2032.141765    10.528265
## BU2      372.960935   -146.492066  ...   22370.955686    -4.595913
## ISOP    7655.131976     74.836048  ...   -3366.731373    32.959879
## MEPR    3593.549133     51.906735  ...    -731.993431    14.037722
## PRO1  -65773.762010  12047.594167  ...  392100.202304   626.809510
## ACAL    1082.729148    427.863371  ...     626.809510    64.872504
##
## [8 rows x 8 columns]
```

2.4.2.3 Matrice de covariance inter - classe

La matrice de covariance inter - classe est proposée par l'objet « discrimintools ».

```
# Matrice de covariance inter - classe
print(lda.cov_["between"])
```

```
##               MEOH         ACET  ...          PRO1        ACAL
## MEOH  96515.005967 -1036.207897  ... -73697.469933  249.047725
## ACET  -1036.207897  -171.929156  ...  -1995.822816  -34.817407
## BU1    2874.380385    10.645256  ...  -2304.675642    7.174939
## BU2    2895.636709  -295.221822  ...  -1745.576399   18.361615
## ISOP   4160.456766  -218.577017  ...  -2332.718327   17.893743
## MEPR   2253.103875  -171.362485  ...  -1037.211943   11.632392
```

```
## PRO1 -73697.469933 -1995.822816  ...  51143.250291 -171.998705
## ACAL    249.047725   -34.817407  ...   -171.998705   -1.273302
##
## [8 rows x 8 columns]
```

2.4.3 Autres indicateurs : Lambda de Wilks, Transformation de RAO et de Bartlett.

Ces trois indicateurs sont retournés par l'objet « discrimintools ».

```
# MANOVA test
global_perf = lda.statistics_["performance"]
print(global_perf)
```

```
##                 Stat       Value       p-value
## 0      Wilks' Lambda    0.066713           NaN
## 1  Bartlett -- C(16)  123.184510  0.000000e+00
## 2    Rao -- F(16,84)   15.076064  1.110223e-16
```

2.5 Evaluation des contributions des variables

Mesurer l'impact des variables est crucial pour l'interprétation du mécanisement d'affectation. Pour l'analyse discriminante, il est possible de produire une mesure d'importance des variables basée sur leurs contributions à la discrimination. Concrètement, il s'agit simplement d'opposer les lambdas de Wilks avec ou sans la variable à évaluer.

2.5.1 Affichage des contributions sous Python

Ces résultats sont fournis directement par l'objet « discrimintools »

```
# Evaluation statistique
stats_eval = lda.statistics_["statistical_evaluation"]
print(stats_eval)
```

```
##       Wilks L.  Partial L.   F(2, 42)   p-value
## MEOH  0.117975    0.565488  16.136067  0.000006
## ACET  0.074153    0.899667   2.341965  0.108572
## BU1   0.084183    0.792475   5.499262  0.007563
## BU2   0.095695    0.697142   9.122996  0.000513
## ISOP  0.072310    0.922600   1.761764  0.184196
## MEPR  0.087798    0.759852   6.636945  0.003128
## PRO1  0.092396    0.722038   8.084336  0.001071
## ACAL  0.075884    0.879150   2.886700  0.066885
```

2.6 Evaluation en Test

L'évaluation sur l'échantillon test est une approche priviligiée pour mesurer et comparer les performances des modèles de nature et de complexité différente. Dans cette

section, nous traitons la seconde feuille « TEST » comportant 50 observations de notre classeur Excel.

2.6.1 Importation des données

Nous chargeons la feuille « TEST ».

```
# chargement échantillon test
DTest = pd.read_excel("./data/Eau_de_vie_LDA.xlsx",sheet_name="TEST")
print(DTest.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 50 entries, 0 to 49
## Data columns (total 9 columns):
##  #   Column  Non-Null Count  Dtype
## ---  ------  --------------  -----
##  0   TYPE    50 non-null     object
##  1   MEOH    50 non-null     int64
##  2   ACET    50 non-null     int64
##  3   BU1     50 non-null     float64
##  4   BU2     50 non-null     float64
##  5   ISOP    50 non-null     int64
##  6   MEPR    50 non-null     int64
##  7   PRO1    50 non-null     int64
##  8   ACAL    50 non-null     float64
## dtypes: float64(3), int64(5), object(1)
## memory usage: 3.6+ KB
## None
```

Nous affichons pour vérification la distribution des classes.

```
# Distribution relative des classes
dtest = (DTest.TYPE.value_counts(normalize=True).reset_index()
                   .rename(columns={"index":"TYPE"}))
print(dtest)
```

```
##      TYPE  proportion
## 0   POIRE        0.38
## 1   MIRAB        0.34
## 2  KIRSCH        0.28
```

Elle est similaire à celle de l'échantillon « TRAIN ».

2.6.2 Prédiction des classes sur l'échantillon d'apprentissage

Il y a deux étapes dans l'évaluation :

1. Effectuer la prédiction à partir de la matrice des explicatives de l'échantillon test ;
2. Confronter les prédictions de l'étape 1 avec les classes observées.

2.6.3 Probabilité d'appartenance

L'objet « discrimintools » calcule les probabilités d'affectation aux classes avec `predict_proba()`. Elle permettent une analyse plus fine de la qualité du modèle, via la construction de la courbe ROC par exemple, dont le principe reste valable pour les problèmes multi - classes.

```
# Matrice X en Test
XTest = DTest[DTest.columns[1:]]
# Probabilité d'appartenance
print(lda.predict_proba(XTest).head(6))
```

```
##      KIRSCH         MIRAB         POIRE
## 0  1.000000  1.535523e-08  4.487814e-10
## 1  0.999989  1.109717e-05  3.307808e-07
## 2  0.999997  2.730580e-06  2.193638e-08
## 3  0.999992  7.523981e-06  5.271811e-08
## 4  0.999862  1.116948e-04  2.673034e-05
## 5  1.000000  1.745132e-08  1.549001e-10
```

2.6.4 Classe d'appartenance

L'objet « discrimintools » calcule les classes d'appartenance avec la fonction `predict()`. Elle permet de produire les prédictions à partir de la matrice des explicatives en test.

```
# Prédiction sur XTest
y_pred = lda.predict(XTest)
```

On calcule la distribution d'appartenance

```
# Distribution des classes prédictes
y_pred.value_counts(normalize=False).to_frame()
```

```
##             count
## prediction
## MIRAB          19
## POIRE          16
## KIRSCH         15
```

19 observations ont été prédite « MIRAB », 16 « POIRE » et 15 « KIRSCH ».

2.6.5 Matrice de confusion et taux de bon classement

La matrice de confusion est issue de la confrontation entre ces prédictions et les classes observées. Nous faisons appel au module « metrics » de la librairie « scikit-learn ».

```
# Matrice de confusion
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(DTest.TYPE,y_pred,labels=lda.classes_["classes"])
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=lda.classes_["classes"])
disp.plot(cmap=plt.cm.Blues,values_format='g');
plt.show()
```

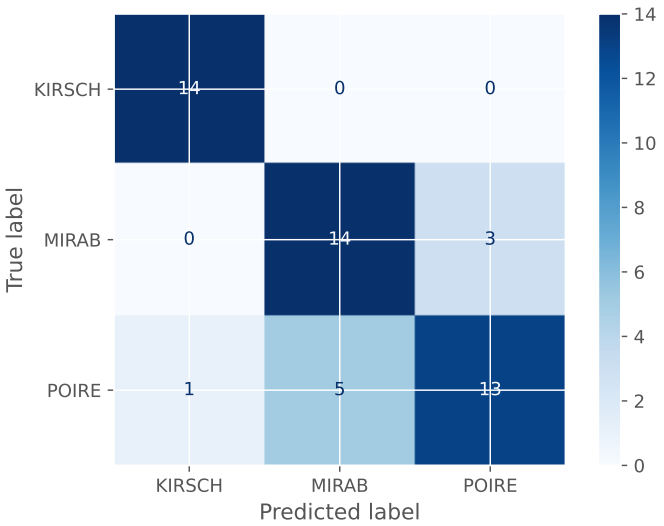


Figure 2.1 – Matrice de confusion

La fonction `score()` nous donne le taux de reconnaissance (ou taux de succès).

```
# Taux de succès
print(lda.score(XTest,DTest.TYPE))
```

```
## 0.82
```

Notre taux de succès est de 82%.

La fonction `classification_report()` génère un rapport sur les performances globales, mais aussi sur les reconnaissances par classe (rappel, précision et F-Measure[F1-Score])

```
# rapport
from sklearn.metrics import classification_report
print(classification_report(DTest.TYPE,y_pred))
```

```
##               precision    recall  f1-score   support
##
##       KIRSCH       0.93      1.00      0.97        14
##        MIRAB       0.74      0.82      0.78        17
##        POIRE       0.81      0.68      0.74        19
```

```
##
##     accuracy                           0.82        50
##    macro avg       0.83      0.84      0.83        50
## weighted avg       0.82      0.82      0.82        50
```

Nous retrouvons, entre autres le taux de succès de 82%.

2.7 Sélection de variables

Limiter le modèle aux variables explicatives pertinentes est primordial pour l'interprétation et le deploiement des modèles.

2.7.1 Backward selection

```
# Selection backward
from discrimintools import STEPDISC
backward=STEPDISC(lda,method="backward",alpha=0.01,model_train=True,verbose=True)
```

```
##       Wilks L.  Partial L.          F   p-value
## MEOH  0.117975    0.434512  16.136067  0.000006
## ACET  0.074153    0.100333   2.341965  0.108572
## BU1   0.084183    0.207525   5.499262  0.007563
## BU2   0.095695    0.302858   9.122996  0.000513
## ISOP  0.072310    0.077400   1.761764  0.184196
## MEPR  0.087798    0.240148   6.636945  0.003128
## PRO1  0.092396    0.277962   8.084336  0.001071
## ACAL  0.075884    0.120850   2.886700  0.066885
##
##       Wilks L.  Partial L.          F   p-value
## MEOH  0.129692    0.442449  17.061483  0.000004
## ACET  0.079826    0.094151   2.234643  0.119316
## BU1   0.092945    0.222011   6.135344  0.004528
## BU2   0.109809    0.341492  11.149568  0.000126
## MEPR  0.098171    0.263427   7.689228  0.001397
## PRO1  0.101216    0.285585   8.594537  0.000724
## ACAL  0.081813    0.116149   2.825379  0.070331
##
##       Wilks L.  Partial L.          F   p-value
## MEOH  0.147097    0.457327  18.540063  0.000001
## BU1   0.098601    0.190415   5.174419  0.009589
## BU2   0.122087    0.346156  11.647178  0.000087
## MEPR  0.107220    0.255494   7.549796  0.001517
## PRO1  0.112777    0.292178   9.081281  0.000499
## ACAL  0.086488    0.077032   1.836148  0.171437
##
##       Wilks L.  Partial L.          F   p-value
## MEOH  0.155113    0.442418  17.852822  0.000002
## BU1   0.110174    0.214984   6.161841  0.004313
```

```
## BU2   0.136572    0.366723  13.029454  0.000034
## MEPR  0.111753    0.226079   6.572738  0.003131
## PRO1  0.131479    0.342192  11.704517  0.000081
```

Les variables sélectionnées sont les suivantes :

```
# Variables sélectionnées
selectedVar = backward.results_["selected"]
selectedVar
```

```
## ['MEOH', 'BU1', 'BU2', 'MEPR', 'PRO1']
```

De plus, le paramètre « model_train » permet d'entraîner le modèle LDA avec les variables sélectionnées.

```
# Modèle réduit
lda2 = backward.results_["train"]
lda2
```

```
## LDA(features=['MEOH', 'BU1', 'BU2', 'MEPR', 'PRO1'],
##     priors=TYPE
## POIRE     0.384615
## KIRSCH    0.326923
## MIRAB     0.288462
## Name: proportion, dtype: float64,
##     target=['TYPE'])
```

```
# Summary
summaryLDA(lda2,to_markdown=False)
```

```
##                      Linear Discriminant Analysis - Results
##
##
## Summary Information
##
##                infos  Value                  DF  DF value
## 0  Total Sample Size     52           DF Total        51
## 1          Variables      5   DF Within Classes        49
## 2            Classes      3  DF Between Classes         2
##
## Class Level information
##
##         Frequency  Proportion  Prior Probability
## TYPE
## POIRE          20    0.384615           0.384615
## KIRSCH         17    0.326923           0.326923
## MIRAB          15    0.288462           0.288462
##
## Group means:
```

```
##
##        KIRSCH    MIRAB    POIRE
## MEOH  371.676  934.200  1084.35
## BU1     1.200   20.200    21.33
## BU2    21.018   13.567    49.38
## MEPR   28.894   29.400    50.00
## PRO1  790.771  195.267   317.40
##
## Coefficients of linear discriminants:
##
##             KIRSCH   MIRAB   POIRE
## MEOH         0.006   0.027   0.032
## BU1         -0.086   0.346   0.280
## BU2         -0.005   0.070   0.117
## MEPR         0.087  -0.026   0.058
## PRO1         0.003  -0.005  -0.008
## Intercept  -4.411 -16.919 -24.264
##
## Individuals (the 10 first) scores
##
##    KIRSCH   MIRAB   POIRE
## 0   1.222  -9.327 -12.467
## 1   2.097  -7.328 -11.323
## 2   2.140  -9.817 -14.200
## 3   1.044  -7.796 -12.160
## 4  -3.345 -15.344 -23.197
## 5   7.586  -7.363 -12.449
## 6   6.090  -1.736  -5.022
## 7  -2.769 -16.115 -22.891
## 8   0.752  -8.704 -10.746
## 9  -1.614 -19.595 -27.999
```

2.7.2 Forward selection

```
# Selection forward
forward = STEPDISC(lda,method="forward",alpha=0.01,model_train=True,verbose=True)
```

```
##       Wilks L.  Partial L.          F       p-value
## MEOH  0.282629    0.717371  62.186129  3.586020e-14
## ACET  0.971855    0.028145   0.709531  2.540218e-01
## BU1   0.286173    0.713827  61.112585  4.873879e-14
## BU2   0.914588    0.085412   2.288014  1.122087e-01
## ISOP  0.887731    0.112269   3.098457  5.406192e-02
## MEPR  0.691854    0.308146  10.912106  1.203236e-04
## PRO1  0.835465    0.164535   4.824978  1.222491e-02
## ACAL  0.979642    0.020358   0.509127  1.511647e-01
##
##       Wilks L.  Partial L.          F   p-value
## ACET  0.253614    0.102660   2.745708  0.074297
```

```
## BU1   0.192547    0.318729  11.228252  0.000100
## BU2   0.244101    0.136320   3.788072  0.029680
## ISOP  0.264061    0.065697   1.687604  0.195751
## MEPR  0.221217    0.217287   6.662572  0.002796
## PRO1  0.255676    0.095365   2.530037  0.090232
## ACAL  0.235697    0.166054   4.778852  0.012803
##
##       Wilks L.  Partial L.         F   p-value
## ACET  0.178725    0.071786  1.817445  0.173671
## BU2   0.170291    0.115585  3.071236  0.055772
## ISOP  0.174351    0.094502  2.452583  0.097018
## MEPR  0.147786    0.232468  7.117602  0.001994
## PRO1  0.176100    0.085419  2.194821  0.122666
## ACAL  0.173496    0.098943  2.580493  0.086432
##
##       Wilks L.  Partial L.         F   p-value
## ACET  0.138022    0.066069  1.627088  0.207606
## BU2   0.131479    0.110340  2.852582  0.067944
## ISOP  0.129820    0.121570  3.183082  0.050730
## PRO1  0.136572    0.075879  1.888507  0.162842
## ACAL  0.127365    0.138180  3.687719  0.032702
```

Les variables sélectionnées sont les suivantes :

```
# Variables sélectionnées
selectedVar2 = forward.results_["selected"]
selectedVar2
```

```
## ['MEOH', 'BU1', 'MEPR']
```

Nous entraînons le modèle avec les variables sélectionnées :

```
# Modèle réduit
lda3 = backward.results_["train"]
```

```
# Summary
summaryLDA(lda3,to_markdown=False)
```

```
##                     Linear Discriminant Analysis - Results
##
##
## Summary Information
##
##                infos  Value                  DF  DF value
## 0  Total Sample Size     52           DF Total        51
## 1          Variables      5   DF Within Classes        49
## 2            Classes      3  DF Between Classes         2
##
## Class Level information
##
```

```
##         Frequency  Proportion  Prior Probability
## TYPE
## POIRE          20    0.384615           0.384615
## KIRSCH         17    0.326923           0.326923
## MIRAB          15    0.288462           0.288462
##
## Group means:
##
##        KIRSCH    MIRAB    POIRE
## MEOH  371.676  934.200  1084.35
## BU1     1.200   20.200    21.33
## BU2    21.018   13.567    49.38
## MEPR   28.894   29.400    50.00
## PRO1  790.771  195.267   317.40
##
## Coefficients of linear discriminants:
##
##            KIRSCH   MIRAB   POIRE
## MEOH        0.006   0.027   0.032
## BU1        -0.086   0.346   0.280
## BU2        -0.005   0.070   0.117
## MEPR        0.087  -0.026   0.058
## PRO1        0.003  -0.005  -0.008
## Intercept  -4.411 -16.919 -24.264
##
## Individuals (the 10 first) scores
##
##    KIRSCH   MIRAB   POIRE
## 0   1.222  -9.327 -12.467
## 1   2.097  -7.328 -11.323
## 2   2.140  -9.817 -14.200
## 3   1.044  -7.796 -12.160
## 4  -3.345 -15.344 -23.197
## 5   7.586  -7.363 -12.449
## 6   6.090  -1.736  -5.022
## 7  -2.769 -16.115 -22.891
## 8   0.752  -8.704 -10.746
## 9  -1.614 -19.595 -27.999
```

3 La méthode DISQUAL

Sommaire

Ce chapitre a pour objectif de présenter rapidement les principales fonctionnalités offertes par le package « discrimintools » pour réaliser une analyse discriminante linéaire sur variables qualitatives.

3.1 Présentation de la méthode

La méthode DISQUAL (discrimination sur variables qualitatives) repose sur l'enchaînement de plusieurs étapes :

1. Réaliser une ACM sur les variables explicatives catégorielles.
2. Réaliser une ADL où l'on prédit la variable cible à l'aide des variables synthétiques (coordonnées factorielles des individus)
3. Exprimer les fonctions de classement définitives à partir des indicatrices des variables initiales.

3.2 Présentation des données

Nous allons illustrer ce chapitre en utilisant les données « Vote au congrès » (*cf* Ricco Rakotomalala, Pratique de l'Analyse Discriminante Linéaire, version 1.0, 2020). Ces données recensent les votes de (n = 435) parlementaires américains identifiés selon leur appartenance politique (Y = « group » avec $K = 2$ valeurs possibles $\{\text{republicain}, \text{democrat}\}$) sur différents thèmes en 1984.

Ces données ont été subdivisées en apprentissage ($n_{\text{train}} = 235$) et test ($n_{\text{test}} = 200$).

```
# Chargement des données - Base d'apprentissage
import pandas as pd
DTrain = pd.read_excel("./data/CongressVotePipeline.xlsx",sheet_name="train",
                       header=0)
DTrain.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 235 entries, 0 to 234
## Data columns (total 17 columns):
##  #   Column                      Non-Null Count  Dtype
## ---  ------                      --------------  -----
##  0   handicapped_infants         235 non-null    object
##  1   water_project_cost_sharin   235 non-null    object
##  2   adoption_of_the_budget_re   235 non-null    object
##  3   physician_fee_freeze        235 non-null    object
##  4   el_salvador_aid             235 non-null    object
##  5   religious_groups_in_schoo   235 non-null    object
##  6   anti_satellite_test_ban     235 non-null    object
##  7   aid_to_nicaraguan_contras   235 non-null    object
##  8   mx_missile                  235 non-null    object
##  9   immigration                 235 non-null    object
##  10  synfuels_corporation_cutb   235 non-null    object
##  11  education_spending          235 non-null    object
##  12  superfund_right_to_sue      235 non-null    object
##  13  crime                       235 non-null    object
##  14  duty_free_exports           235 non-null    object
##  15  export_administration_act   235 non-null    object
##  16  group                       235 non-null    object
## dtypes: object(17)
## memory usage: 31.3+ KB
```

3.2.0.1 Distribution relative

Nous calculons la distribution relative des classes :

```
# Distribution relative des classes
d = (DTrain.group.value_counts(normalize=True).to_frame()
                 .rename(columns={"index":"group"}))
print(d)
```

```
##             proportion
## group
## democrat      0.655319
## republican    0.344681
```

3.2.1 Relation entre descriptifs et cible - V de Cramer

Une première piste consiste à procéder à une simple analyse bivariée. Nous croisons chaque descripteur avec la variable cible. Nous disposons ainsi d'une première indication sur les liaisons individuelles de chaque descripteur avec « Fonction ».

```
# V de Cramer
import scientistmetrics as st
#
K = DTrain.shape[1]-1
cramerV = st.scientistmetrics(DTrain)
cramerV = (cramerV.iloc[:K,K].to_frame()
                  .sort_values(by="group",ascending=False))
print(cramerV)
```

```
##                                group
## physician_fee_freeze        0.920483
## adoption_of_the_budget_re   0.739235
## el_salvador_aid             0.727504
## education_spending          0.724472
## aid_to_nicaraguan_contras   0.670772
## crime                       0.644621
## mx_missile                  0.635674
## superfund_right_to_sue      0.595462
## anti_satellite_test_ban     0.588194
## duty_free_exports           0.544828
## religious_groups_in_schoo   0.499852
## handicapped_infants         0.465031
## export_administration_act   0.404828
## synfuels_corporation_cutb   0.362856
## immigration                 0.122598
## water_project_cost_sharin   0.035297
```

3.3 Analyse avec discrimintools

3.4 Modélisation avec discrimintools

Sage précaution avec les packages pour Python, nous affichons le numéro de la version de « discrimintools » utilisée dans ce tutoriel.

```
# version
import discrimintools
print(discrimintools.__version__)
```

```
## 0.0.1
```

Nous fonctionnons avec la version « 0.0.1 ».

```
# Importation
from discrimintools import DISQUAL
```

On crée une instance de la classe DISQUAL, en lui passant ici des étiquettes pour les variables explicatives et la variable cible.

```
# Instanciation
disqual = DISQUAL(n_components=None,target=["group"],priors="prop")
```

On estime le modèle en appliquant la méthode `.fit` de la classe DISQUAL sur le jeu de données.

```
# Entraînement du modèle
disqual.fit(DTrain)
```

```
## DISQUAL(priors='prop', target=['group'])
```

3.4.1 Inspection de l'objet DISQUAL

— `statistics_["chi2"]` correspond au test de chi2 entre variables qualitatives et la cible

```
# test statistique de chi2
print(disqual.statistics_["chi2"])
```

```
##                              statistic  ddl        pvalue
## physician_fee_freeze       199.112779    2  5.797082e-44
## adoption_of_the_budget_re  128.420017    2  1.300013e-28
## el_salvador_aid            124.376638    2  9.816490e-28
## education_spending         123.342020    2  1.646723e-27
## aid_to_nicaraguan_contras  105.734648    2  1.096511e-23
## crime                       97.651124    2  6.242059e-22
## mx_missile                  94.959085    2  2.398263e-21
## superfund_right_to_sue      83.325057    2  8.057379e-19
## anti_satellite_test_ban     81.303433    2  2.214032e-18
## duty_free_exports           69.756825    2  7.120295e-16
## religious_groups_in_schoo   58.715214    2  1.778907e-13
## handicapped_infants         50.819569    2  9.218725e-12
## export_administration_act   38.513132    2  4.334905e-09
## synfuels_corporation_cutb   30.941180    2  1.910769e-07
## immigration                  3.532094    2  1.710076e-01
## water_project_cost_sharin    0.292784    2  8.638189e-01
```

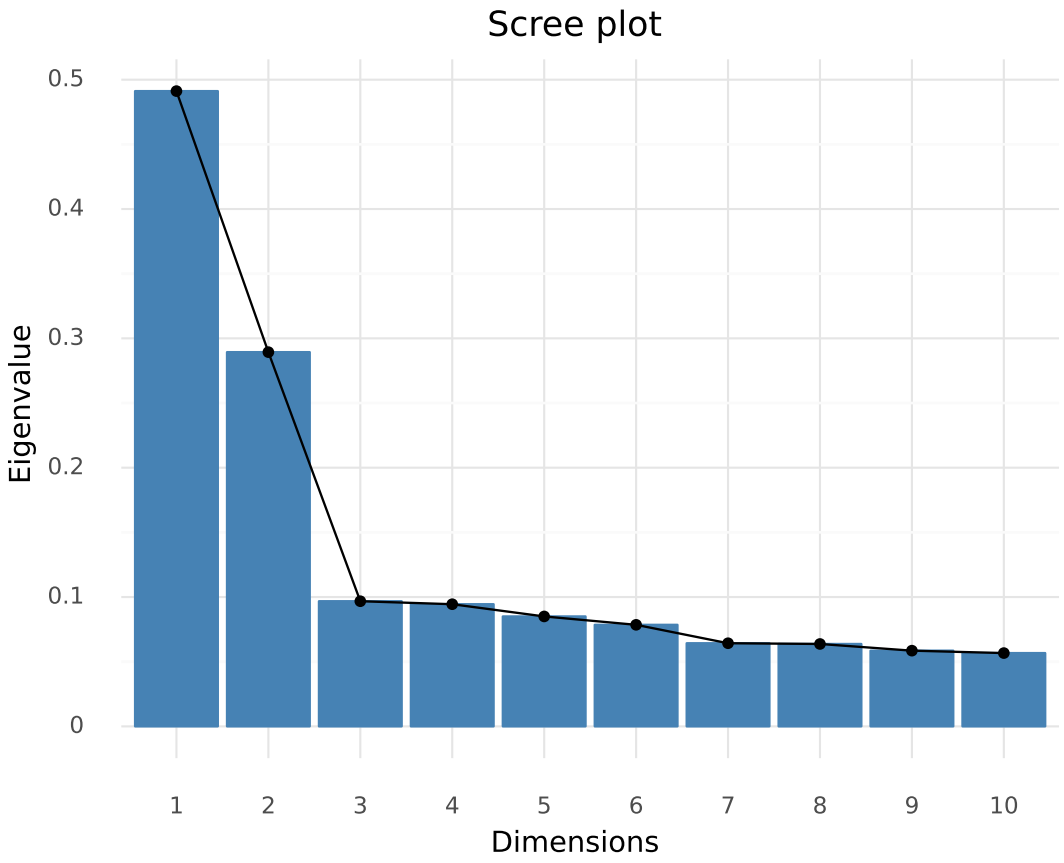
3.5 Analyse des correspondances multiples

La méthode disqual retourne l'objet « factor_model_ » pour l'ACM.

```
# MCA
mca = disqual.factor_model_
```

Visualisons les vaaleurs propres de l'ACM

```
# Valeurs propres
from scientisttools import fviz_screeplot
p = fviz_screeplot(mca,choice="eigenvalue")
print(p)
```



3.5.1 Coordonnées des modalités

Nous affichons les coordonnées des modalités.

```
# Coordonées des modalités
mca.var_["coord"].iloc[:,:2]
```

```
##                                    Dim.1     Dim.2
## handicapped_infants_n           0.513139 -0.012403
## handicapped_infants_other      -0.183899  2.056545
## handicapped_infants_y          -0.582207 -0.142277
## water_project_cost_sharin_n    -0.118659 -0.202458
## water_project_cost_sharin_other -0.007021  1.040837
## water_project_cost_sharin_y     0.117023 -0.058571
## adoption_of_the_budget_re_n     1.008799 -0.113044
## adoption_of_the_budget_re_other 0.220151  2.728064
## adoption_of_the_budget_re_y    -0.682156 -0.105725
```

```
## physician_fee_freeze_n          -0.694535 -0.095260
## physician_fee_freeze_other       0.107987  4.043269
## physician_fee_freeze_y           1.105472 -0.123046
## el_salvador_aid_n               -0.853549 -0.107062
## el_salvador_aid_other            0.034390  2.659709
## el_salvador_aid_y                0.989765 -0.131178
## religious_groups_in_schoo_n     -0.939717 -0.166661
## religious_groups_in_schoo_other -0.205421  3.399304
## religious_groups_in_schoo_y      0.623279 -0.063736
## anti_satellite_test_ban_n        1.023442 -0.164342
## anti_satellite_test_ban_other    0.421859  4.384608
## anti_satellite_test_ban_y       -0.705750 -0.081666
## aid_to_nicaraguan_contras_n      1.044946 -0.172575
## aid_to_nicaraguan_contras_other  0.710887  3.838595
## aid_to_nicaraguan_contras_y     -0.743465 -0.080833
## mx_missile_n                     0.887635 -0.097143
## mx_missile_other                -0.334515  0.919630
## mx_missile_y                    -0.738620 -0.000109
## immigration_n                   -0.092512 -0.085074
## immigration_other               -0.002443  5.180243
## immigration_y                    0.096692 -0.094533
## synfuels_corporation_cutb_n      0.151234 -0.201326
## synfuels_corporation_cutb_other  0.017498  2.839024
## synfuels_corporation_cutb_y     -0.262610 -0.069285
## education_spending_n            -0.693410 -0.119674
## education_spending_other         0.222486  2.135429
## education_spending_y             0.946292 -0.182346
## superfund_right_to_sue_n        -0.747397 -0.131916
## superfund_right_to_sue_other    -0.311813  2.160416
## superfund_right_to_sue_y         0.834518 -0.078354
## crime_n                         -0.910337 -0.125340
## crime_other                     -0.284037  2.171739
## crime_y                          0.742778 -0.055648
## duty_free_exports_n              0.649969 -0.121767
## duty_free_exports_other          0.229130  2.093520
## duty_free_exports_y             -0.797998 -0.101848
## export_administration_act_n      1.270045 -0.170180
## export_administration_act_other -0.455927  0.435683
## export_administration_act_y     -0.068768 -0.153196
```

3.5.2 Coefficients de projection

Les coefficients de projection appliqués sur les indicatrices, permettent d'obtenir les coordonnées factorielles des individus. Ils définissent les variables latentes.

```
# Fonction de projection
disqual.projection_function_.iloc[:,:2]
```

```
##                                 Dim.1     Dim.2
## handicapped_infants_n        0.045761 -0.001441
```

```
## handicapped_infants_other       -0.016400  0.238952
## handicapped_infants_y           -0.051921 -0.016531
## water_project_cost_sharin_n     -0.010582 -0.023524
## water_project_cost_sharin_other -0.000626  0.120936
## water_project_cost_sharin_y      0.010436 -0.006805
## adoption_of_the_budget_re_n      0.089964 -0.013135
## adoption_of_the_budget_re_other  0.019633  0.316977
## adoption_of_the_budget_re_y     -0.060834 -0.012284
## physician_fee_freeze_n          -0.061938 -0.011068
## physician_fee_freeze_other       0.009630  0.469793
## physician_fee_freeze_y           0.098585 -0.014297
## el_salvador_aid_n               -0.076119 -0.012440
## el_salvador_aid_other            0.003067  0.309035
## el_salvador_aid_y                0.088267 -0.015242
## religious_groups_in_schoo_n     -0.083804 -0.019365
## religious_groups_in_schoo_other -0.018319  0.394969
## religious_groups_in_schoo_y      0.055584 -0.007406
## anti_satellite_test_ban_n        0.091270 -0.019095
## anti_satellite_test_ban_other    0.037621  0.509453
## anti_satellite_test_ban_y       -0.062938 -0.009489
## aid_to_nicaraguan_contras_n      0.093188 -0.020052
## aid_to_nicaraguan_contras_other  0.063397  0.446011
## aid_to_nicaraguan_contras_y     -0.066302 -0.009392
## mx_missile_n                     0.079159 -0.011287
## mx_missile_other                -0.029832  0.106853
## mx_missile_y                    -0.065870 -0.000013
## immigration_n                   -0.008250 -0.009885
## immigration_other               -0.000218  0.601899
## immigration_y                    0.008623 -0.010984
## synfuels_corporation_cutb_n      0.013487 -0.023392
## synfuels_corporation_cutb_other  0.001560  0.329870
## synfuels_corporation_cutb_y     -0.023419 -0.008050
## education_spending_n            -0.061838 -0.013905
## education_spending_other         0.019841  0.248118
## education_spending_y             0.084390 -0.021187
## superfund_right_to_sue_n        -0.066653 -0.015328
## superfund_right_to_sue_other    -0.027807  0.251021
## superfund_right_to_sue_y         0.074422 -0.009104
## crime_n                         -0.081183 -0.014563
## crime_other                     -0.025330  0.252337
## crime_y                          0.066241 -0.006466
## duty_free_exports_n              0.057964 -0.014148
## duty_free_exports_other          0.020434  0.243249
## duty_free_exports_y             -0.071165 -0.011834
## export_administration_act_n      0.113262 -0.019773
## export_administration_act_other -0.040659  0.050623
## export_administration_act_y     -0.006133 -0.017800
```

3.6 Analyse discriminante sur facteurs

Avec l'analyse discriminante linéaire, nous cherchons à prédire le « group » d'appartenance politique à partir des « n_components » composantes de l'ACM. n_components est un hyperparamètre de l'algorithme DISQUAL. Le réduire améliore les propriétés de régularisation, mais nous prenons le risque de ne pas capter suffisamment les informations véhiculées par les données. L'augmenter nous fait prendre le risque du surapprentissage.

3.6.1 Coefficients de l'ADL

Les coefficients des fonctions de classement sont :

```
# Coefficients
disqual.lda_model_.coef_
```

```
##       democrat  republican
## Z1   -7.989576   15.190057
## Z2    0.125551   -0.238702
## Z3    5.361641  -10.193737
## Z4    2.894756   -5.503610
## Z5    0.871930   -1.657744
## Z6   -2.720325    5.171976
## Z7   -1.990073    3.783596
## Z8    0.438413   -0.833526
## Z9   -0.556042    1.057166
## Z10   1.062908   -2.020837
## Z11   0.125935   -0.239432
## Z12   1.622582   -3.084910
## Z13  -2.508054    4.768398
## Z14   0.813802   -1.547229
## Z15  -1.216593    2.313028
## Z16  -1.428645    2.716190
## Z17  -1.589206    3.021454
## Z18  -4.321235    8.215680
## Z19  -0.083495    0.158744
## Z20  -4.308874    8.192180
## Z21  -3.224343    6.130233
## Z22   3.098009   -5.890042
## Z23  -0.802653    1.526032
## Z24  -0.935585    1.778766
## Z25   1.521233   -2.892221
## Z26   7.675348  -14.592637
## Z27   1.308562   -2.487884
## Z28  -6.135812   11.665619
## Z29   0.606964   -1.153982
## Z30  -4.706786    8.948704
## Z31 -10.410249   19.792325
## Z32  -4.730903    8.994556
```

3.6.2 Coefficients de DISQUAL

Nous exprimons les fonctions de classement dans l'espace originel des indicatrices.

```
# Coefficients
disqual.coef_
```

```
##                                  democrat  republican
## handicapped_infants_n           -0.155292    0.295247
## handicapped_infants_other        0.835278   -1.588059
## handicapped_infants_y            0.116795   -0.222054
## water_project_cost_sharin_n     -0.254918    0.484658
## water_project_cost_sharin_other -0.397060    0.754904
## water_project_cost_sharin_y      0.345095   -0.656107
## adoption_of_the_budget_re_n     -1.148006    2.182629
## adoption_of_the_budget_re_other  1.739083   -3.306404
## adoption_of_the_budget_re_y      0.644624   -1.225581
## physician_fee_freeze_n           3.915654   -7.444577
## physician_fee_freeze_other       1.263054   -2.401362
## physician_fee_freeze_y          -6.360063   12.091972
## el_salvador_aid_n                0.192299   -0.365606
## el_salvador_aid_other           -0.232696    0.442409
## el_salvador_aid_y               -0.201358    0.382829
## religious_groups_in_schoo_n     -0.221397    0.420928
## religious_groups_in_schoo_other -0.180218    0.342637
## religious_groups_in_schoo_y      0.153531   -0.291898
## anti_satellite_test_ban_n        0.726035   -1.380363
## anti_satellite_test_ban_other    0.281802   -0.535771
## anti_satellite_test_ban_y       -0.499898    0.950424
## aid_to_nicaraguan_contras_n      0.123551   -0.234899
## aid_to_nicaraguan_contras_other -1.314518    2.499207
## aid_to_nicaraguan_contras_y     -0.015919    0.030266
## mx_missile_n                    -0.934568    1.776833
## mx_missile_other                -0.306693    0.583094
## mx_missile_y                     0.838073   -1.593372
## immigration_n                    0.294473   -0.559862
## immigration_other               -1.428927    2.716725
## immigration_y                   -0.256921    0.488467
## synfuels_corporation_cutb_n     -0.532975    1.013311
## synfuels_corporation_cutb_other -0.585379    1.112943
## synfuels_corporation_cutb_y      1.002123   -1.905270
## education_spending_n             0.398455   -0.757556
## education_spending_other         0.120096   -0.228331
## education_spending_y            -0.584639    1.111535
## superfund_right_to_sue_n         0.041243   -0.078412
## superfund_right_to_sue_other    -0.651848    1.239317
## superfund_right_to_sue_y         0.022094   -0.042006
## crime_n                         -0.297718    0.566032
## crime_other                      1.095669   -2.083123
## crime_y                          0.158022   -0.300437
## duty_free_exports_n             -0.017415    0.033111
```

```
## duty_free_exports_other         -0.974848    1.853415
## duty_free_exports_y              0.135347   -0.257327
## export_administration_act_n      0.122343   -0.232603
## export_administration_act_other  0.194168   -0.369158
## export_administration_act_y     -0.109851    0.208852
```

Toutes les indicatrices dont représentées.

3.7 Evaluation en Test

L'évaluation sur l'échantillon test est une approche priviligiée pour mesurer et comparer les performances des modèles de nature et de complexité différente. Dans cette section, nous traitons la seconde feuille « test » comportant 200 observations de notre classeur Excel.

3.7.1 Importation des données

Nous chargeons la feuille « test ».

```
# chargement échantillon test
DTest = pd.read_excel("./data/CongressVotePipeline.xlsx",sheet_name="test",header=0)
print(DTest.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 200 entries, 0 to 199
## Data columns (total 17 columns):
##  #   Column                      Non-Null Count  Dtype
## ---  ------                      --------------  -----
##  0   handicapped_infants         200 non-null    object
##  1   water_project_cost_sharin   200 non-null    object
##  2   adoption_of_the_budget_re   200 non-null    object
##  3   physician_fee_freeze        200 non-null    object
##  4   el_salvador_aid             200 non-null    object
##  5   religious_groups_in_schoo   200 non-null    object
##  6   anti_satellite_test_ban     200 non-null    object
##  7   aid_to_nicaraguan_contras   200 non-null    object
##  8   mx_missile                  200 non-null    object
##  9   immigration                 200 non-null    object
##  10  synfuels_corporation_cutb   200 non-null    object
##  11  education_spending          200 non-null    object
##  12  superfund_right_to_sue      200 non-null    object
##  13  crime                       200 non-null    object
##  14  duty_free_exports           200 non-null    object
##  15  export_administration_act   200 non-null    object
##  16  group                       200 non-null    object
## dtypes: object(17)
## memory usage: 26.7+ KB
## None
```

Nous affichons pour vérification la distribution des classes.

```
# Distribution relative des classes
dtest = (DTest.group.value_counts(normalize=True).reset_index()
                    .rename(columns={"index":"group"}))
print(dtest)
```

```
##         group  proportion
## 0    democrat       0.565
## 1  republican       0.435
```

3.7.2 Prédiction des classes sur l'échantillon d'apprentissage

Il y a deux étapes dans l'évaluation :

1. Effectuer la prédiction à partir de la matrice des explicatives de l'échantillon test ;
2. Confronter les prédictions de l'étape 1 avec les classes observées.

3.7.3 Probabilité d'appartenance

L'objet « discrimintools » calcule les probabilités d'affectation aux classes avec `predict_proba()`. Elle permettent une analyse plus fine de la qualité du modèle, via la construction de la courbe ROC par exemple, dont le principe reste valable pour les problèmes multi - classes.

```
# Matrice X en Test
XTest = DTest.drop(columns=["group"])
# Probabilité d'appartenance
print(disqual.predict_proba(XTest).head(6))
```

```
##         democrat    republican
## 0   1.816241e-11  1.000000e+00
## 1   4.244324e-07  9.999996e-01
## 2   9.995927e-01  4.073076e-04
## 3   1.000000e+00  4.289600e-11
## 4   9.999999e-01  8.454981e-08
## 5   1.000000e+00  1.392334e-10
```

3.7.4 Classe d'appartenance

L'objet « discrimintools » calcule les classes d'appartenance avec la fonction `predict()`. Elle permet de produire les prédictions à partir de la matrice des explicatives en test.

```
# Prédiction sur XTest
y_pred = disqual.predict(XTest)
```

On calcule la distribution d'appartenance

```
# Distribution des classes prédictes
y_pred.value_counts(normalize=False).to_frame()
```

```
##             count
## prediction
## democrat      111
## republican     89
```

111 observations ont été prédite « democrat » et 89 « republican ».

3.7.5 Matrice de confusion et taux de bon classement

La matrice de confusion est issue de la confrontation entre ces prédictions et les classes observées. Nous faisons appel au module « metrics » de la librairie « scikit-learn ».

```
# Matrice de confusion
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(DTest.group,y_pred,
                      labels=disqual.lda_model_.classes_["classes"])
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=disqual.lda_model_.classes_["classes"])
disp.plot(cmap=plt.cm.Blues,values_format='g');
plt.show()
```

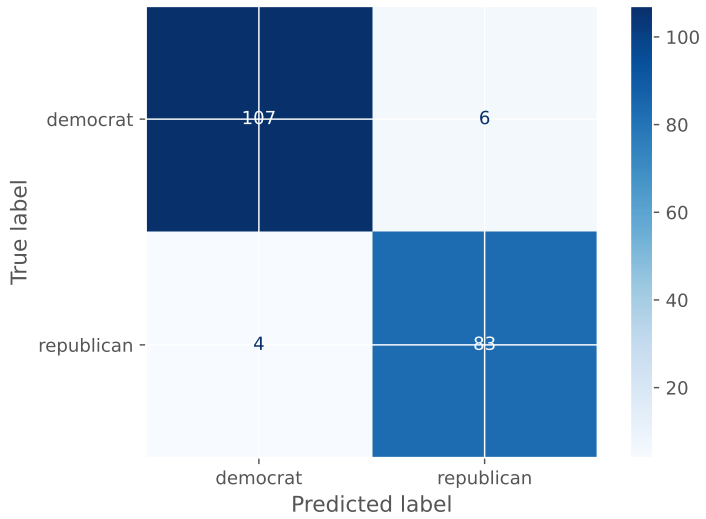


Figure 3.1 – Matrice de confusion

La fonction `score()` nous donne le taux de reconnaissance (ou taux de succès).

```
# Taux de succès
score = disqual.score(XTest,DTest.group)
print(score)
```

```
## 0.95
```

Notre taux de succès est de 95%.

La fonction `classification_report()` génère un rapport sur les performances globales, mais aussi sur les reconnaissances par classe (rappel, précision et F-Measure[F1-Score])

```
# rapport
from sklearn.metrics import classification_report
print(classification_report(DTest.group,y_pred))
```

```
##               precision    recall  f1-score   support
##
##     democrat       0.96      0.95      0.96       113
##   republican       0.93      0.95      0.94        87
##
##     accuracy                           0.95       200
##    macro avg       0.95      0.95      0.95       200
## weighted avg       0.95      0.95      0.95       200
```

Nous retrouvons, entre autres le taux de succès de 95%s.

4 Analyse des Correspondances Discriminante

Sommaire

Ce chapitre a pour objectif de présenter rapidement les principales fonctionnalités offertes par le package « discrimintools » pour réaliser une Analyse des Correspondances Discriminante .

4.1 Présentation des données

L'analyse des correspondances discriminante (ACD) est le pendant de l'analyse factorielle discriminante pour les descripteurs catégoriels. On la reconnaît sous les traits de l'analyse discriminante barycentrique. Lorsque le nombre de classes est supérieur à 2, l'approche passe par un tableau de contingence particulier soumis à une analyse factorielle des correspondances (AFC).

4.1.1 Importation des données

Nous illustrons l'analyse des correspondances discriminante à l'aide d'un exemple sur les données « Races Canines » extraites de l'ouvrage de Tenenhaus. Il s'agit de prédire la variable « Fonction » (utilite, chasse, compagnie) de $(n = 27)$ chiens à partir de leurs caractéristiques (Taille, Poids, etc. 6 variables).

```
# Chargement des données
import pandas as pd
# Données actives
```

```
DTrain = pd.read_csv("./data/races_canines.txt",sep="\t",encoding='latin-1',
                     index_col=0)
print(DTrain.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 27 entries, Beauceron to Terre-Neuve
## Data columns (total 7 columns):
##  #   Column        Non-Null Count  Dtype
## ---  ------        --------------  -----
##  0   Taille        27 non-null     object
##  1   Poids         27 non-null     object
##  2   Velocite      27 non-null     object
##  3   Intelligence  27 non-null     object
##  4   Affection     27 non-null     object
##  5   Agressivite   27 non-null     object
##  6   Fonction      27 non-null     object
## dtypes: object(7)
## memory usage: 1.7+ KB
## None
```

4.1.2 Distribution relative

Nous calculons la distribution relative des classes :

```
# Distribution relative des classes
d = (DTrain.Fonction.value_counts(normalize=True).reset_index()
                .rename(columns={"index":"Fonction"}))
print(d)
```

```
##     Fonction  proportion
## 0  compagnie    0.370370
## 1     chasse    0.333333
## 2    utilite    0.296296
```

4.2 Analyse bivariée

Une première piste consiste à procéder à une simple analyse bivariée. Nous croisons chaque descripteur avec la variable cible. Nous disposons ainsi d'une première indication sur les liaisons individuelles de chaque descripteur avec « Fonction ».

```
# V de Cramer
import scientistmetrics as st
cramerV = st.scientistmetrics(DTrain)
cramerV = (cramerV.iloc[:6,6].to_frame()
                  .sort_values(by="Fonction",ascending=False).T)
print(cramerV)
```

```
##           Affection     Poids    Taille  Agressivite  Velocite  Intelligence
## Fonction   0.739394  0.672298  0.550325     0.511811  0.396357        0.2769
```

Nous avons quelques relations qui sont assez fortes : Affection avec un V de Cramer de 0.74 ; Poids avec un V de Cramer de 0.67 ; Taille avec un V de Cramer de 0.55 et Agressivite avec un V de Cramer de 0.51. Il semble donc possible d'expliquer la fonction des chiens à partir de leurs caractéristiques. Mais il faut le faire de manière multivariée c'est - à - dire en tenant compte du rôle simultané de l'ensemble des descripteurs.

4.3 Analyse avec discrimintools

4.4 Modélisation avec discrimintools

Sage précaution avec les packages pour Python, nous affichons le numéro de la version de « discrimintools » utilisée dans ce tutoriel.

```
# version
import discrimintools
print(discrimintools.__version__)
```

```
## 0.0.1
```

Nous fonctionnons avec la version « 0.0.1 ».

```
# Importation
from discrimintools import DISCA
```

On crée une instance de la classe DISCA, en lui passant ici des étiquettes pour les variables explicatives et la variable cible.

```
# Instanciation
disca = DISCA(n_components=None,target=["Fonction"],priors="prop")
```

On estime le modèle en appliquant la méthode `.fit` de la classe DISCA sur le jeu de données.

```
# Entraînement du modèle
disca.fit(DTrain)
```

```
## DISCA(priors='prop', target=['Fonction'])
```

4.4.1 Inspection de l'objet DISCA

— `call_["priors"]` correspond à la distribution relative des classes.

```
# distribution des classes
print(disca.call_["priors"].to_frame())
```

```
##            proportion
## Fonction
## compagnie    0.370370
## chasse       0.333333
## utilite      0.296296
```

— `statistics_["chi2"]` correspond au test de chi2 entre variables qualitatives et la cible

```
# test statistique de chi2
print(disca.statistics_["chi2"])
```

```
##                statistic  ddl    pvalue
## Poids          24.407143    4  0.000066
## Affection      14.760989    2  0.000623
## Taille         16.354286    4  0.002579
## Agressivite     7.072665    2  0.029120
## Velocite        8.483333    4  0.075394
## Intelligence    4.140385    4  0.387340
```

— `statistics_["categories"]` correspond à la distribution absolue et relative des colonnes

```
# distribution absolue et relative des classes
print(disca.statistics_["categories"])
```

```
##                         Frequence  Proportion
## Taille_Taille+                5.0    0.030864
## Taille_Taille++              15.0    0.092593
## Taille_Taille-                7.0    0.043210
## Poids_Poids+                 14.0    0.086420
## Poids_Poids++                 5.0    0.030864
## Poids_Poids-                  8.0    0.049383
## Velocite_Veloc+               8.0    0.049383
## Velocite_Veloc++              9.0    0.055556
## Velocite_Veloc-              10.0    0.061728
## Intelligence_Intell+         13.0    0.080247
## Intelligence_Intell++         6.0    0.037037
## Intelligence_Intell-          8.0    0.049383
## Affection_Affec+             14.0    0.086420
## Affection_Affec-             13.0    0.080247
## Agressivite_Agress+          13.0    0.080247
## Agressivite_Agress-          14.0    0.086420
```

4.5 Analyse des classes

4.5.1 Coordonnées des classes

L'objet « disca » fournit les coordonnées des points - classes.

```
# Coordonnées des points - classes
print(disca.classes_["coord"])
```

```
##                Dim.1     Dim.2
## chasse     -0.167113 -0.476797
## compagnie   0.714651  0.162645
## utilite    -0.705312  0.333090
```

On projette ces points - classes dans le plan :

```
# Projection des points classes
from plotnine import *
gcoord = disca.classes_["coord"]
p = (ggplot(gcoord,aes(x="Dim.1",y="Dim.2",label=gcoord.index))+
        geom_point(aes(color=gcoord.index))+
        geom_text(aes(color=gcoord.index),
                  adjust_text={'arrowprops': {'arrowstyle': '-','lw':1.0}})+
        geom_hline(yintercept=0,colour="black",linetype="--")+
        geom_vline(xintercept=0,colour="black",linetype="--")+
        theme(legend_direction="vertical",legend_position=(0.2,0.5))+
        labs(color="Fonction"))
print(p)
```

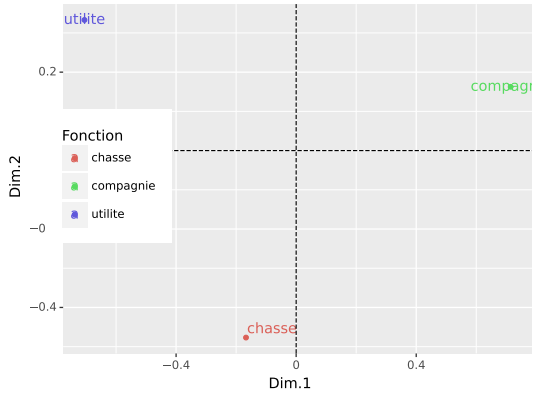


Figure 4.1 – Carte des points - classes

Visiblement, « compagnie » et « utilite » s'opposent sur le premier facteur. « chasse » se démarque des deux autres sur le second facteur.

4.5.2 Distances entre centres de classes

Les distances entre centres de classes permettent de situer les proximités entre les groupes sur l'ensemble des facteurs. La distance euclidienne entre les classes dans le répère factoriel est la suivante :

```
# Distance euclidienne
DE = disca.classes_["dist"]
print(DE)
```

```
##              chasse  compagnie   utilite
## chasse     0.000000   1.186395  0.945574
## compagnie  1.186395   0.000000  2.045346
## utilite    0.945574   2.045346  0.000000
```

Les trois types de fonctions forment un triangle approximativement isocèle dans le plan factoriel.

Ajoutons ces distances sur le plan factoriel :

```
# Projection des points classes avec distances entre classes
p = (ggplot(gcoord,aes(x="Dim.1",y="Dim.2",label=gcoord.index))+
        geom_point(aes(color=gcoord.index))+
        geom_text(aes(color=gcoord.index),
                  adjust_text={'arrowprops': {'arrowstyle': '-','lw':1.0}})+
        geom_hline(yintercept=0,colour="black",linetype="--")+
        geom_vline(xintercept=0,colour="black",linetype="--")+
        theme(legend_direction="vertical",legend_position=(0.2,0.3))+
        annotate("segment",x=gcoord.iloc[0,0],y=gcoord.iloc[0,1],
                           xend=gcoord.iloc[1,0],yend=gcoord.iloc[1,1],
                           color="blue")+
        annotate("segment",x=gcoord.iloc[0,0],y=gcoord.iloc[0,1],
                           xend=gcoord.iloc[2,0],yend=gcoord.iloc[2,1],
                           color="blue")+
        annotate("segment",x=gcoord.iloc[1,0],y=gcoord.iloc[1,1],
                           xend=gcoord.iloc[2,0],yend=gcoord.iloc[2,1],
                           color="blue")+
        # Add test
        annotate('text', x = -0.3, y = 0.2,label = DE.iloc[0,1].round(2),
                 size = 10, angle='35')+
        annotate('text', x = 0.4, y = 0.2,label = DE.iloc[0,2].round(2),
                 size = 10, angle='-60')+
        annotate('text', x = 0, y = -0.25,label = DE.iloc[2,1].round(2),
                 size = 10, angle='-10')+
        labs(color="Fonction"))
print(p)
```

4.5.3 Qualité de la représentation des classes

Il suffit de passer les coordonnées au carré et de diviser par la somme en ligne. Sous discrimintools, elles correspondent à la qualité de représentation des points - lignes de l'analyse factorielle des correspondances.

```
# Qualité de représentation
gcos2 = disca.classes_["cos2"]
print(gcos2)
```

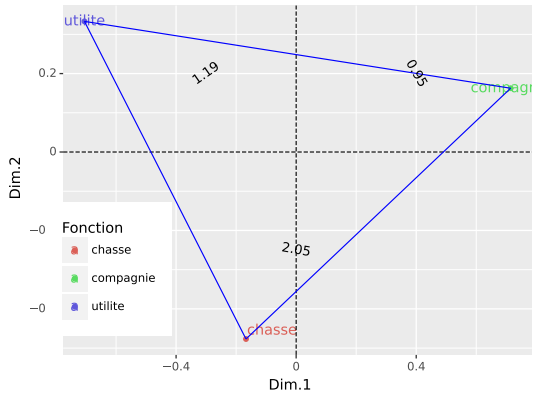


Figure 4.2 – Carte des points - classes

```
##                Dim.1     Dim.2
## chasse      0.109405  0.890595
## compagnie   0.950755  0.049245
## utilite     0.817642  0.182358
```

Le graphique (Figure 4.1) ne laissait aucun doute, mais c'est toujours mieux quand les chiffres confirment : les informations portées par « compagnie » et « utilite » sont bien captées par le premier facteur. « chasse » est mieux situé sur le second facteur. Et la somme en ligne dans le tableau des COS2 fait bien 100%.

4.5.4 Contributions des classes

Sous discrimintools, elles correspondent aux contributions des points - lignes de l'analyse factorielle des correspondances.

```
# Contribution des groupes
gcontrib = disca.classes_["contrib"]
print(gcontrib)
```

```
##                Dim.1      Dim.2
## chasse      2.691509  63.975158
## compagnie  54.691459   8.271504
## utilite    42.617032  27.753338
```

Le premier axe oppose les fonctions « compagnie » et « utilite ». Elles déterminent (**contributions** = $54.69\% + 42.62\%$) 97.31% de l'information portée par le facteur. Elles sont aussi très bien représentées puisque 95.08% (resp. 81.76%) de l'information véhiculée par « compagnie » (resp. « utilite ») est restrancrite sur cet axe.

Le second axe permet surtout de distinguer la fonction « chasse » des deux premiers.

4.6 Structures canoniques

Les structures canoniques correspondent aux représentations des modalités colonnes du tableau de contingence - et donc des modalités des variables prédictives - dans le

répère factoriel.

4.6.1 Poids, distance à l'origine et inertie

```
# Informations sur les modalités
mod_infos = disca.var_["infos"]
print(mod_infos)
```

```
##                            dist     marge   inertia
## Taille_Taille+         0.672309  0.030864  0.013951
## Taille_Taille++        0.672309  0.092593  0.041852
## Taille_Taille-         1.022203  0.043210  0.045150
## Poids_Poids+           0.508474  0.086420  0.022343
## Poids_Poids++          1.541104  0.030864  0.073302
## Poids_Poids-           1.055492  0.049383  0.055015
## Velocite_Veloc+        0.651920  0.049383  0.020988
## Velocite_Veloc++       0.540918  0.055556  0.016255
## Velocite_Veloc-        0.494975  0.061728  0.015123
## Intelligence_Intell+   0.349767  0.080247  0.009817
## Intelligence_Intell++  0.502079  0.037037  0.009336
## Intelligence_Intell-   0.360122  0.049383  0.006404
## Affection_Affec+       0.712498  0.086420  0.043871
## Affection_Affec-       0.767305  0.080247  0.047246
## Agressivite_Agress+    0.531131  0.080247  0.022638
## Agressivite_Agress-    0.493193  0.086420  0.021021
```

4.6.2 Coordonnées des points modalités

```
# Coordonnées des points modalités
mod_coord = disca.var_["coord"]
print(mod_coord)
```

```
##                             Dim.1     Dim.2
## Taille_Taille+          0.615447 -0.270601
## Taille_Taille++        -0.672278 -0.006473
## Taille_Taille-          1.000991  0.207157
## Poids_Poids+           -0.158972 -0.482984
## Poids_Poids++          -1.199302  0.967820
## Poids_Poids-            1.027765  0.240335
## Velocite_Veloc+         0.465513 -0.456397
## Velocite_Veloc++       -0.524295 -0.133070
## Velocite_Veloc-         0.099455  0.484880
## Intelligence_Intell+    0.311993  0.158107
## Intelligence_Intell++ -0.491839  0.100882
## Intelligence_Intell-  -0.138108 -0.332586
## Affection_Affec+        0.656065  0.277906
## Affection_Affec-       -0.706532 -0.299283
```

```
## Agressivite_Agress+   -0.430926  0.310489
## Agressivite_Agress-    0.400145 -0.288311

# Ajout de la variable
modcoord = mod_coord.copy()
modcoord.loc[:,"variable"] = [x.split("_")[0] for x in mod_coord.index]

# Projection des points modalités
p = (ggplot(modcoord,aes(x="Dim.1",y="Dim.2",label=mod_coord.index))+
        geom_point(aes(color=modcoord.variable))+
        geom_text(aes(color=modcoord.variable),
                  adjust_text={'arrowprops': {'arrowstyle': '-','lw':1.0}})+
        geom_hline(yintercept=0,colour="black",linetype="--")+
        geom_vline(xintercept=0,colour="black",linetype="--")+
        theme(legend_direction="vertical",legend_position=(0.2,0.5))+
        labs(color="Variable"))
print(p)
```

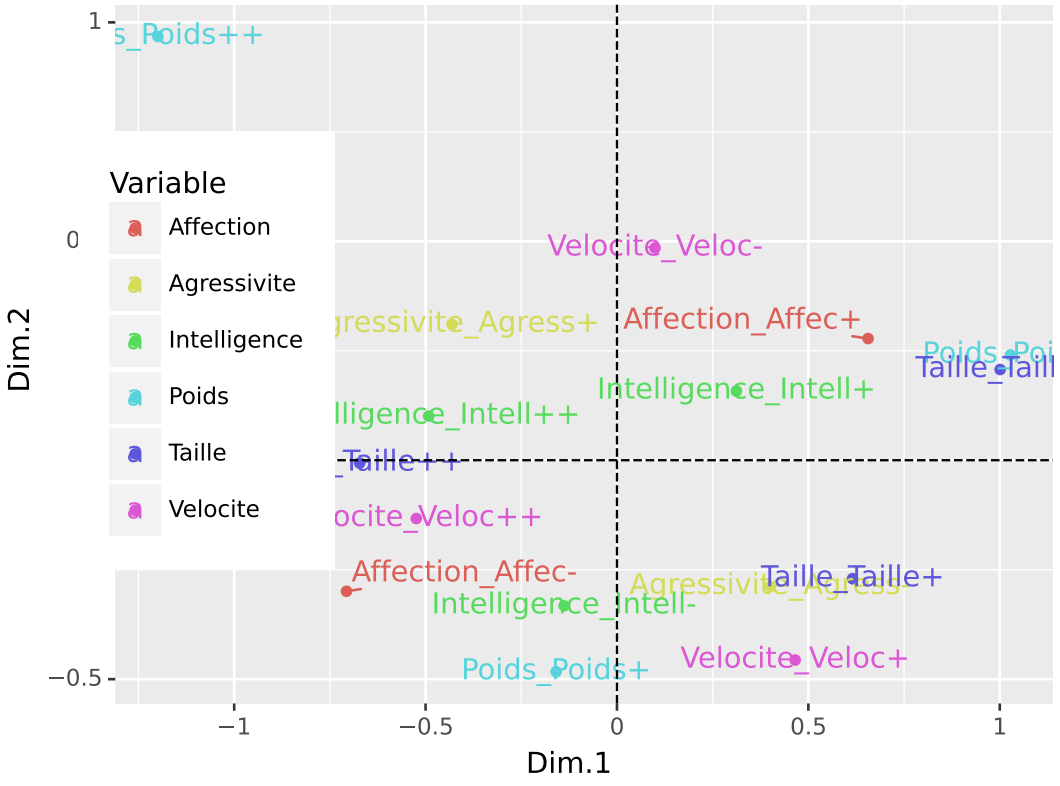


Figure 4.3 – Carte des points - modalités

4.6.3 Contributions des points modalités aux facteurs

Les contributions des points modalités sont :

```
# Contributions des points modalités
mod_contrib = disca.var_["contrib"]
print(mod_contrib)
```

```
##                            Dim.1      Dim.2
## Taille_Taille+          3.380111   1.908012
## Taille_Taille++        12.099553   0.003275
## Taille_Taille-         12.518107   1.565485
## Poids_Poids+            0.631468  17.019410
## Poids_Poids++          12.835325  24.406737
## Poids_Poids-           15.081961   2.408099
## Velocite_Veloc+         3.094092   8.684110
## Velocite_Veloc++        4.415434   0.830525
## Velocite_Veloc-         0.176536  12.252351
## Intelligence_Intell+    2.258456   1.693550
## Intelligence_Intell++   2.590466   0.318224
## Intelligence_Intell-    0.272339   4.611574
## Affection_Affec+       10.754785   5.634746
## Affection_Affec-       11.582076   6.068189
## Agressivite_Agress+     4.308521   6.531111
## Agressivite_Agress-     4.000770   6.064603
```

4.7 Affectation des classes

4.8 Fonction discriminante canonique

L'exécution de la méthode `disca.fit(DTrain)` provoque le calcul de plusieurs attributs parmi lesquels `disca.coef_`. Ce champ nous intéresse particulièrement car il correspond aux coefficients des fonctions de classement. Ces fonctions canoniques permettent de projeter des individus non étiquetés dans l'espace factoriel.

```
# Coefficients des fonctions discriminantes canoniques
print(disca.coef_)
```

```
##                              Dim.1     Dim.2
## Taille_Taille+            0.174416 -0.131042
## Taille_Taille++          -0.190522 -0.003135
## Taille_Taille-            0.283679  0.100319
## Poids_Poids+             -0.045052 -0.233892
## Poids_Poids++            -0.339879  0.468680
## Poids_Poids-              0.291266  0.116385
## Velocite_Veloc+           0.131925 -0.221016
## Velocite_Veloc++         -0.148584 -0.064441
## Velocite_Veloc-           0.028185  0.234810
## Intelligence_Intell+      0.088418  0.076566
## Intelligence_Intell++    -0.139386  0.048854
## Intelligence_Intell-     -0.039140 -0.161059
## Affection_Affec+          0.185927  0.134580
```

```
## Affection_Affec-      -0.200229 -0.144932
## Agressivite_Agress+   -0.122123  0.150359
## Agressivite_Agress-    0.113400 -0.139619
```

4.8.1 Coordonnées des individus

A partir des fonctions discriminantes canoniques, on détermine les coordonnées des individus.

```
# Coordonnées factorielles des individus
ind_coord = disca.ind_["coord"]
print(ind_coord)
```

```
##                     Dim.1     Dim.2
## Chien
## Beauceron       -0.231936  0.060037
## Basset           0.241638  0.295881
## Berger All      -0.459741  0.032325
## Boxer            0.413511 -0.224446
## Bull-Dog         0.990876  0.523040
## Bull-Mastif     -0.963955  0.754635
## Caniche          0.866811  0.039503
## Chihuahua        0.863318  0.285415
## Cocker           0.646090  0.581656
## Colley           0.003587 -0.229940
## Dalmatien        0.649034 -0.514423
## Doberman        -0.845897 -0.247187
## Dogue All       -1.040478  0.245472
## Epag. Breton     0.421230 -0.542135
## Epag. Français  -0.102060 -0.666027
## Fox-Hound       -0.745651 -0.457100
## Fox-Terrier      0.859092  0.357192
## Gd Bleu Gasc    -0.465142 -0.613675
## Labrador         0.649034 -0.514423
## Levrier         -0.510127 -0.747077
## Mastiff         -0.863708  0.544722
## Pekinois         0.863318  0.285415
## Pointer         -0.610374 -0.537164
## St-Bernard      -0.736151  0.782347
## Setter          -0.382570 -0.509452
## Teckel           0.990876  0.523040
## Terre-Neuve     -0.500627  0.492370
```

```
# Ajout de la colonne Fonction
rowcoord = pd.concat([ind_coord,DTrain["Fonction"]],axis=1)
# Projection des points modalités
p = (ggplot(rowcoord,aes(x="Dim.1",y="Dim.2",label=rowcoord.index))+
        geom_point(aes(color=rowcoord.Fonction))+
        geom_text(aes(color=rowcoord.Fonction),
                  adjust_text={'arrowprops': {'arrowstyle': '-','lw':1.0}})+
```

```
        geom_hline(yintercept=0,colour="black",linetype="--")+
        geom_vline(xintercept=0,colour="black",linetype="--")+
        theme(legend_direction="vertical",legend_position=(0.5,0.2))+
        labs(color="Fonction")+
        annotate("text",x=gcoord["Dim.1"].values,y=gcoord["Dim.2"].values,
                 label=gcoord.index,color=["red","green","violet"]))
print(p)
```

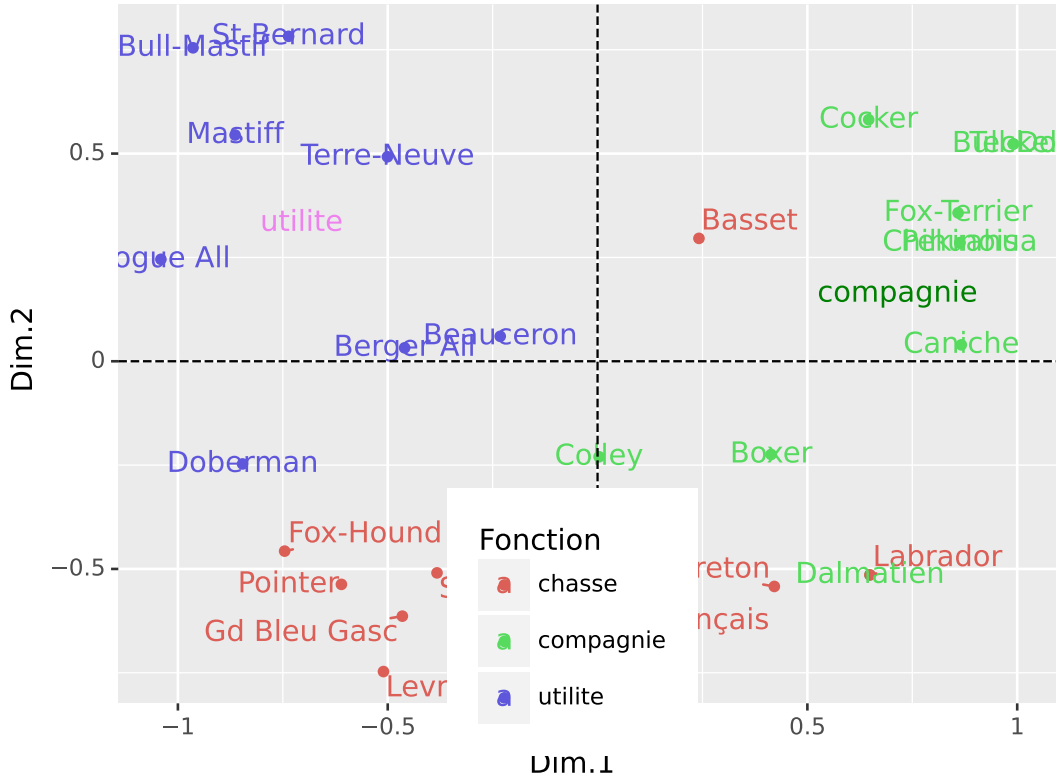


Figure 4.4 – Carte des individus

4.8.2 Valeurs propres associées aux facteurs

Les valeurs propres associées aux facteurs sont celles issues de l'analyse factorielle des correspondances.

```
# Valeurs propres
from scientisttools import get_eig
eig = get_eig(disca.factor_model_)
print(eig)
```

```
##         eigenvalue  difference  proportion  cumulative
## Dim.1     0.345864    0.227414    74.48927    74.48927
## Dim.2     0.118450         NaN    25.51073   100.00000
```

La valeur propre (λ) indique l'inertie (la variance) expliquée par l'appartenance aux groupes sur chaque axe. En les additionnant, nous avons l'inertie expliquée par l'appartenance aux groupes dans l'espace complet soit 0.4643136. Cette inertie indique la quantité d'information que l'on peut modéliser dans la relation entre la cible Fonction et les descripteurs. Le premier facteur explique 74.49% de l'inertie totale.

On peut représenter graphiquement ces valeurs propres

```
# Scree plot
from scientisttools import fviz_screeplot
p = fviz_screeplot(disca.factor_model_,choice="proportion",add_labels=True)
print(p)
```

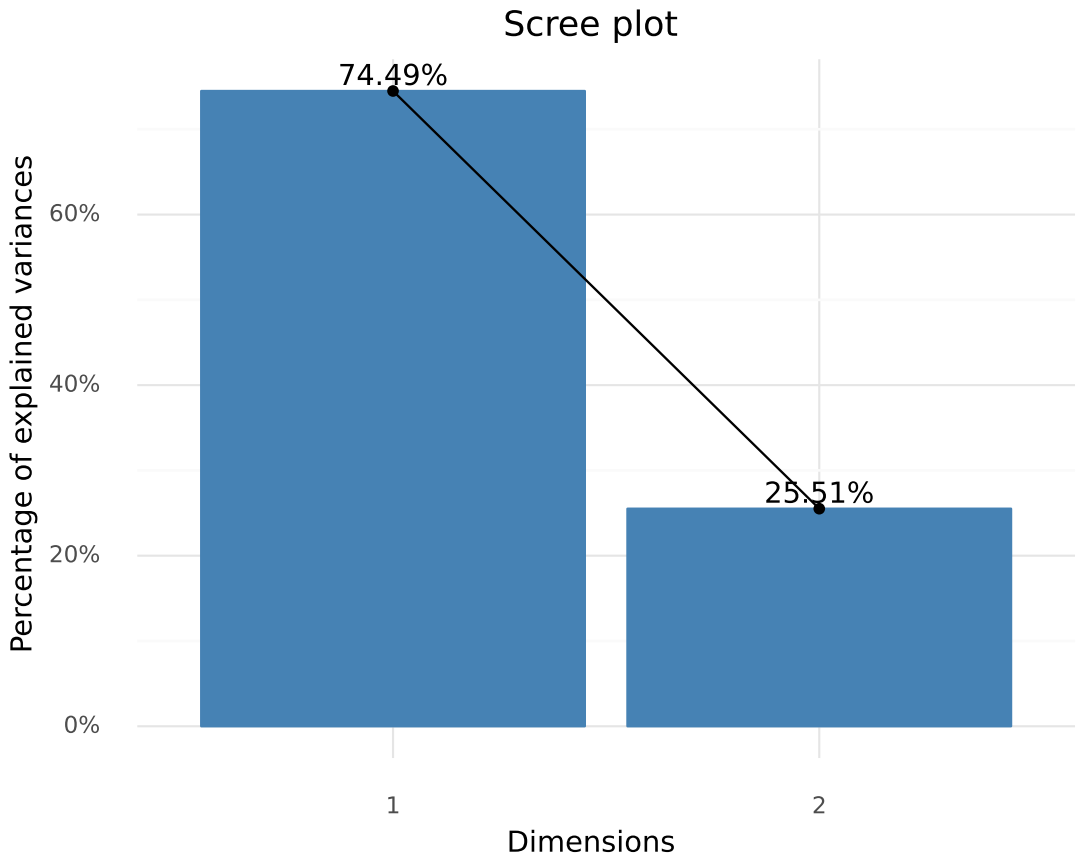


Figure 4.5 – Scree plot

4.8.3 Rapport de corrélation

Le champ `anova_["Eta2"]` correspond aux carrés des rapports de corrélation.

```
# Rapport de corrélation
print(disca.anova_["Eta2"])
```

```
## Dim.1    0.735104
## Dim.2    0.518040
## Name: Eta2, dtype: float64
```

4.8.4 Corrélation canonique

La corrélation canonique est la racine carré du rapport de corrélation.

```
# Corrélation canonique
print(disca.anova_["canonical_Eta2"])
```

```
## Dim.1    0.857382
## Dim.2    0.719750
## Name: Eta2, dtype: float64
```

4.9 Traitement d'individus supplémentaires

Les fonctions discriminantes canoniques nous permettent de positionner les individus suppémentaires dans le répère factoriel.

4.9.1 Importation des données

Nous chargeons les individus supplémentaires.

```
# Individus supplémentaires
Dsup = pd.read_excel("./data/races_canines_acm.xls",
                     header=0,sheet_name=1,index_col=0)
print(Dsup)
```

```
##           Taille    Poids Velocite Intelligence Affection Agressivite
## Chien
## Medor    Taille+   Poids-   Veloc-     Intell++    Affec-     Agress+
## Djeck   Taille++  Poids++   Veloc+      Intell+    Affec+     Agress-
## Taico    Taille-   Poids+  Veloc++     Intell++    Affec+     Agress+
## Rocky    Taille+   Poids+   Veloc+      Intell-    Affec+     Agress-
## Boudog   Taille-   Poids-  Veloc++      Intell+    Affec-     Agress+
## Wisky    Taille+  Poids++   Veloc-      Intell-    Affec+     Agress+
```

4.9.2 Coordonnées des individus supplémentaires

L'objet « DISCA » contient la fonction `transform()` bien connue des utilisateurs de scikit-learn. Elle permet d'obtenir les coordonnées des individus dans l'espace factoriel.

```
# Coordonnées des individus supplémentaires
ind_sup_coord = disca.transform(Dsup)
```

On rajoute ces individus au plan factoriel

```
# Projection des points modalités
p = (ggplot(rowcoord,aes(x="Dim.1",y="Dim.2",label=rowcoord.index))+
        geom_point(aes(color=rowcoord.Fonction))+
```

Table 4.1 – Coordonnées des individus supplémentaires

	Dim.1	Dim.2
Medor	0.0321289	0.2744329
Djeck	-0.0107306	0.3160556
Taico	0.0144601	0.1357781
Rocky	0.5214770	-0.7520483
Boudog	0.1924262	0.2342553
Wisky	-0.1126134	0.6963258

```
        geom_text(aes(color=rowcoord.Fonction),
                  adjust_text={'arrowprops': {'arrowstyle': '-','lw':1.0}})+
        geom_hline(yintercept=0,colour="black",linetype="--")+
        geom_vline(xintercept=0,colour="black",linetype="--")+
        theme(legend_direction="vertical",legend_position=(0.5,0.2))+
        labs(color="Variable")+
        annotate("text",x=ind_sup_coord["Dim.1"].values,
                        y=ind_sup_coord["Dim.2"].values,
                     label=ind_sup_coord.index))
print(p)
```

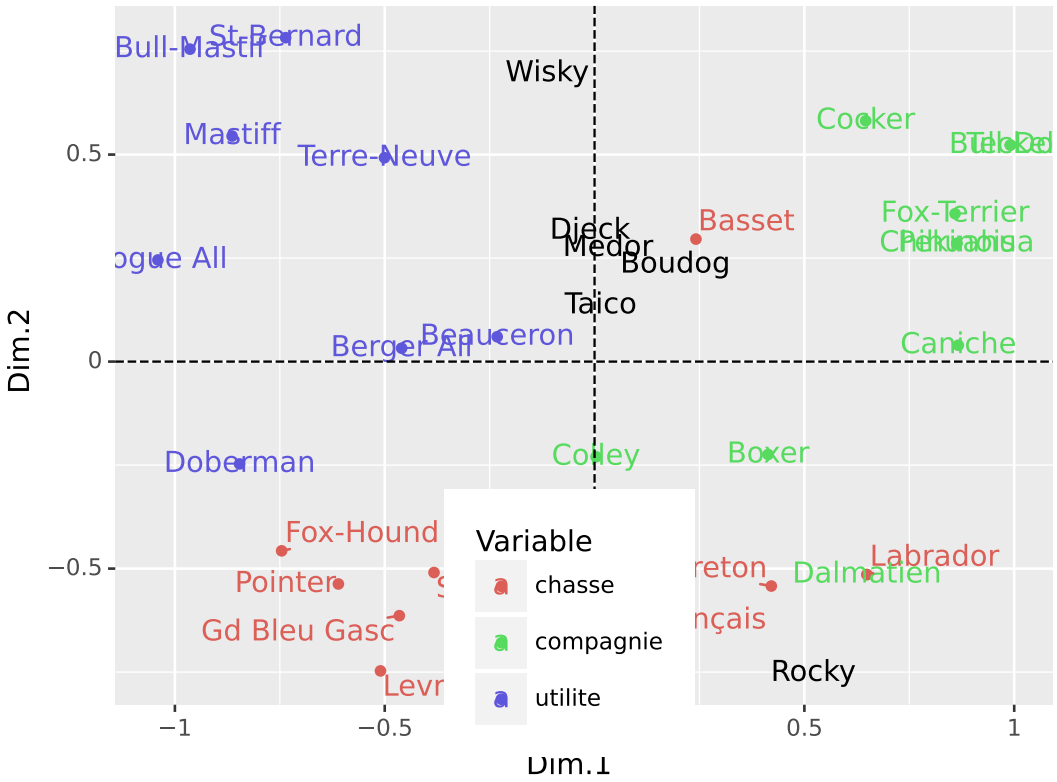


Figure 4.6 – Carte des individus

4.9.3 Distances euclidiennes aux classes

La fonction `decision_function()` permet de calculer les distances euclidiennes aux centres de classes.

```
# Distances euclidiennes aux classes
disca.decision_function(Dsup)
```

```
##           chasse  compagnie   utilite
## Medor   0.604044   0.478333  0.547259
## Djeck   0.653071   0.549714  0.482733
## Taico   0.408217   0.490990  0.557003
## Rocky   0.549920   0.873981  2.682536
## Boudog  0.634864   0.277847  0.815701
## Wisky   1.379187   0.969182  0.483231
```

4.9.4 Probabilités d'affectation

L'objet « discrimintools » calcule les probabilités d'affectation aux classes avec `predict_proba()`.

```
# probabilité d'affectation
print(disca.predict_proba(Dsup))
```

```
##           chasse  compagnie   utilite
## Medor   0.322822   0.381961  0.295217
## Djeck   0.318679   0.372868  0.308453
## Taico   0.345874   0.368724  0.285402
## Rocky   0.444261   0.419785  0.135955
## Boudog  0.318441   0.422972  0.258588
## Wisky   0.266300   0.363212  0.370487
```

4.9.5 Prédiction

On effectue la prédiction à partir de la matrice des explicatives des individus supplémentaires.

```
# Prediction des individus supplémentaires
ypred = disca.predict(Dsup)
ypred
```

```
## Chien
## Medor     compagnie
## Djeck     compagnie
## Taico     compagnie
## Rocky        chasse
## Boudog    compagnie
## Wisky       utilite
## Name: prediction, dtype: object
```