# ADL sous Python avec scientisttools

Duvérier DJIFACK ZEBAZE

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

## 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.

### 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("./donnee/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
```

### Distribution relative

Nous calculons la distribution relative des classes :

```
# Distribution relative des classes
d = (DTrain.TYPE.value_counts(normalize=True).reset_index()
                .rename(columns={"index":"TYPE","TYPE":"p(k)"}))
```

**Table 1** – Distribution relative des classes

| TYPE | p(k) |
|---|---|
| POIRE | 0.3846154 |
| KIRSCH | 0.3269231 |
| MIRAB | 0.2884615 |

Les classes semblent assez équilibrées.

## LDA

### Modélisation avec scientisttools

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

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

```
## 0.0.9
```

Nous fonctionnons avec la version « 0.0.9 ».

```
# Importation
from scientisttools.discriminant_analysis 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(features_labels=DTrain.columns[1:].values,
          target=["TYPE"],
          distribution = "homoscedastik",
          row_labels=DTrain.index,
          priors = None,
          parallelize=False)
```

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(features_labels=array(['MEOH', 'ACET', 'BU1', 'BU2', 'ISOP', 'MEPR', 'PRO1', 'ACAL'],
##       dtype=object),
##     row_labels=RangeIndex(start=0, stop=52, step=1), 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_)
```

**Table 2** – Coefficients des fonctions de score

| | KIRSCH | MIRAB | POIRE |
|---|---|---|---|
| MEOH | 0.0034282 | 0.0290285 | 0.0333902 |
| ACET | 0.0063904 | 0.0164128 | 0.0075135 |
| BU1 | -0.0636813 | 0.4053900 | 0.3180471 |
| BU2 | -0.0008832 | 0.0713520 | 0.1149928 |
| ISOP | 0.0230822 | 0.0297634 | -0.0084863 |
| MEPR | 0.0374935 | -0.1289417 | 0.0617800 |
| PRO1 | 0.0019711 | -0.0054127 | -0.0083181 |
| ACAL | 0.0661839 | -0.2264238 | -0.1303319 |

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_)
```

**Table 3** – Constantes pour chaque classe

| | KIRSCH | MIRAB | POIRE |
|---|---|---|---|
| Intercept | -5.016453 | -18.84069 | -24.76488 |

## Inspection de l'objet LDA

— `priors_` correspond à la distribution relative des classes.

```
# distribution des classes
print(lda.priors_)
```

**Table 4** – Distribution relative pour chaque classe

| POIRE | KIRSCH | MIRAB |
|---|---|---|
| 0.3846154 | 0.3269231 | 0.2884615 |

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

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

**Table 5** – Distribution absolue et relative pour chaque classe

| n(k) | p(k) |
|---|---|
| 20 | 0.3846154 |
| 17 | 0.3269231 |
| 15 | 0.2884615 |

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

```
# Rapport de corrélation
print(lda.correlation_ratio_)
```

**Table 6** – Rapport de correlation

| | Sum. Intra | Sum. Inter | correlation ratio | F-stats | pvalue |
|---|---|---|---|---|---|
| MEOH | 1970961.421 | 5002712.7092 | 0.7174 | 62.1861 | 0.0000 |
| ACET | 739580.168 | 21418.5587 | 0.0281 | 0.7095 | 0.4969 |
| BU1 | 1774.842 | 4427.1503 | 0.7138 | 61.1126 | 0.0000 |
| BU2 | 138470.430 | 12931.5223 | 0.0854 | 2.2880 | 0.1122 |
| ISOP | 105664.001 | 13363.0759 | 0.1123 | 3.0985 | 0.0541 |
| MEPR | 12039.049 | 5362.0975 | 0.3081 | 10.9121 | 0.0001 |
| PRO1 | 16706890.649 | 3290219.6689 | 0.1645 | 4.8250 | 0.0122 |
| ACAL | 3241.144 | 67.3532 | 0.0204 | 0.5091 | 0.6042 |

— `gmean_` indique les moyennes des variables conditionnellement aux classes

```
# moyennes conditionnelles des variables
print(lda.gmean_)
```

**Table 7** – Moyennes conditionnelles

| | MEOH | ACET | BU1 | BU2 | ISOP | MEPR | PRO1 | ACAL |
|---|---|---|---|---|---|---|---|---|
| KIRSCH | 371.6765 | 203.0176 | 1.20 | 21.01765 | 81.58824 | 28.89412 | 790.7706 | 12.01176 |
| MIRAB | 934.2000 | 235.0667 | 20.20 | 13.56667 | 90.93333 | 29.40000 | 195.2667 | 12.35333 |
| POIRE | 1084.3500 | 185.2500 | 21.33 | 49.38000 | 118.05000 | 50.00000 | 317.4000 | 14.49500 |

— `mean_` indique les moyennes des variables explicatives

```
# moyennes des variables
print(lda.mean_)
```

**Table 8** – Moyennes des variables

| MEOH | ACET | BU1 | BU2 | ISOP | MEPR | PRO1 | ACAL |
|---|---|---|---|---|---|---|---|
| 808.0481 | 205.4288 | 14.42308 | 29.77692 | 98.30769 | 37.15769 | 436.925 | 13.06538 |

— `std_` indique les écart types des variables explicatives

```
# écarts types des variables
print(lda.std_)
```

**Table 9** – Ecart - types des variables

| MEOH | ACET | BU1 | BU2 | ISOP | MEPR | PRO1 | ACAL |
|---|---|---|---|---|---|---|---|
| 369.782 | 122.1538 | 11.02759 | 54.48546 | 48.31008 | 18.47157 | 626.179 | 8.054347 |

— `squared_mdist_` indique la matrice des distances (au carré) de Mahalanobis

```
# Matrice des distances (au carré) de Mahalanobis
print(lda.squared_mdist_)
```

**Table 10** – Ecart - types des variables

| | KIRSCH | MIRAB | POIRE |
|---|---|---|---|
| KIRSCH | 0.00000 | 27.371480 | 36.048105 |
| MIRAB | 27.37148 | 0.000000 | 5.305086 |
| POIRE | 36.04810 | 5.305086 | 0.000000 |

# Evaluation globale du modèle

## 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.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 ».

## Matrice de covariance

### Matrice de covariance intra - classe

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

```
# Matrice de covariance intra - classe
print(lda.wcov_)
```

**Table 11** – Matrice de covariance intra - classe

| | MEOH | ACET | BU1 | BU2 | ISOP | MEPR | PRO1 | ACAL |
|---|---|---|---|---|---|---|---|---|
| MEOH | 40223.70 | 7653.79 | 299.53 | -2522.68 | 3494.68 | 1340.45 | 7923.71 | 833.68 |
| ACET | 7653.79 | 15093.47 | -110.07 | 148.73 | 293.41 | 223.27 | 14043.42 | 462.68 |
| BU1 | 299.53 | -110.07 | 36.22 | 22.44 | 70.24 | 21.94 | 272.53 | 3.35 |
| BU2 | -2522.68 | 148.73 | 22.44 | 2825.93 | -431.05 | -91.43 | 24116.53 | -22.96 |
| ISOP | 3494.68 | 293.41 | 70.24 | -431.05 | 2156.41 | 615.87 | -1034.01 | 15.07 |
| MEPR | 1340.45 | 223.27 | 21.94 | -91.43 | 615.87 | 245.69 | 305.22 | 2.41 |
| PRO1 | 7923.71 | 14043.42 | 272.53 | 24116.53 | -1034.01 | 305.22 | 340956.95 | 798.81 |
| ACAL | 833.68 | 462.68 | 3.35 | -22.96 | 15.07 | 2.41 | 798.81 | 66.15 |

### Matrice de covariance totale

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

```
# Matrice de covariance totale
print(lda.tcov_)
```

**Table 12** – Matrice de covariance totale

| | MEOH | ACET | BU1 | BU2 | ISOP | MEPR | PRO1 | ACAL |
|---|---|---|---|---|---|---|---|---|
| MEOH | 136738.71 | 6617.58 | 3173.91 | 372.96 | 7655.13 | 3593.55 | -65773.76 | 1082.73 |
| ACET | 6617.58 | 14921.54 | -99.42 | -146.49 | 74.84 | 51.91 | 12047.59 | 427.86 |
| BU1 | 3173.91 | -99.42 | 121.61 | 85.72 | 182.12 | 79.11 | -2032.14 | 10.53 |
| BU2 | 372.96 | -146.49 | 85.72 | 2968.67 | -178.40 | 72.00 | 22370.96 | -4.60 |
| ISOP | 7655.13 | 74.84 | 182.12 | -178.40 | 2333.86 | 754.03 | -3366.73 | 32.96 |
| MEPR | 3593.55 | 51.91 | 79.11 | 72.00 | 754.03 | 341.20 | -731.99 | 14.04 |
| PRO1 | -65773.76 | 12047.59 | -2032.14 | 22370.96 | -3366.73 | -731.99 | 392100.20 | 626.81 |
| ACAL | 1082.73 | 427.86 | 10.53 | -4.60 | 32.96 | 14.04 | 626.81 | 64.87 |

### Matrice de covariance inter - classe

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

```
# Matrice de covariance inter - classe
print(lda.bcov_)
```

**Table 13** – Matrice de covariance inter - classe

| | MEOH | ACET | BU1 | BU2 | ISOP | MEPR | PRO1 | ACAL |
|---|---|---|---|---|---|---|---|---|
| MEOH | 96515.01 | -1036.21 | 2874.38 | 2895.64 | 4160.46 | 2253.10 | -73697.47 | 249.05 |
| ACET | -1036.21 | -171.93 | 10.65 | -295.22 | -218.58 | -171.36 | -1995.82 | -34.82 |
| BU1 | 2874.38 | 10.65 | 85.39 | 63.28 | 111.88 | 57.17 | -2304.68 | 7.17 |
| BU2 | 2895.64 | -295.22 | 63.28 | 142.74 | 252.65 | 163.42 | -1745.58 | 18.36 |
| ISOP | 4160.46 | -218.58 | 111.88 | 252.65 | 177.46 | 138.15 | -2332.72 | 17.89 |
| MEPR | 2253.10 | -171.36 | 57.17 | 163.42 | 138.15 | 95.50 | -1037.21 | 11.63 |
| PRO1 | -73697.47 | -1995.82 | -2304.68 | -1745.58 | -2332.72 | -1037.21 | 51143.25 | -172.00 |
| ACAL | 249.05 | -34.82 | 7.17 | 18.36 | 17.89 | 11.63 | -172.00 | -1.27 |

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

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

```
# MANOVA test
print(lda.global_performance_)
```

**Table 14** – Performance globale

| Stat | Value | p-value |
|---|---|---|
| Wilks' Lambda | 0.0667132 | NaN |
| Bartlett – C(16) | 123.1845099 | 0 |
| Rao – F(16,84) | 15.0760641 | 0 |

## 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.

### Affichage des contributions sous Python

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

```
# Evaluation statistique
print(lda.statistical_evaluation_)
```

**Table 15** – Contribution des variables

| | Wilks L. | Partial L. | F(2, 42) | p-value |
|---|---|---|---|---|
| MEOH | 0.1180 | 0.5655 | 16.1361 | 0.0000 |
| ACET | 0.0742 | 0.8997 | 2.3420 | 0.1086 |
| BU1 | 0.0842 | 0.7925 | 5.4993 | 0.0076 |
| BU2 | 0.0957 | 0.6971 | 9.1230 | 0.0005 |
| ISOP | 0.0723 | 0.9226 | 1.7618 | 0.1842 |
| MEPR | 0.0878 | 0.7599 | 6.6369 | 0.0031 |
| PRO1 | 0.0924 | 0.7220 | 8.0843 | 0.0011 |
| ACAL | 0.0759 | 0.8792 | 2.8867 | 0.0669 |

Dans le tableau 15, nous distinguons le lambda de Wilks lorsque la variable est rétirée (Wilks L ; 0.1179746 si on retire la variable MEOH par exemple), le ratio entre le lambda global (0.0667132) et ce dernier (Partial L ; pour MEOH : 0.0667132 / 0.1179746 = 0.565488), et la statistique de test de significativité de l'écart (F = 16.1360671) qui suit une distribution de Fisher à $(K-1=2)$ et $(N-K-p+1=42)$.

## Evaluation en Test

L'avaluation 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, nnous traitons la seconde feuille « TEST » comportant 50 observations de notre classeur Excel.

### Importation des données

Nous chargeons la feuille « TEST ».

```
# chargement échantillon TEST
DTest = pd.read_excel("./donnee/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","TYPE":"p(k)"}))
```

**Table 16** – Distribution relative des classes - TEST

| TYPE | p(k) |
|---|---|
| POIRE | 0.38 |
| MIRAB | 0.34 |
| KIRSCH | 0.28 |

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

### 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.

### Probabilité d'appartenance

L'objet « scientisttools » 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
```

### Classe d'appartenance

L'objet « scientisttools » 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
import numpy as np
# print(np.unique(y_pred,return_counts=True))
y_pred.value_counts(normalize=False)
```

```
## predict
## MIRAB      19
## POIRE      16
## KIRSCH     15
## dtype: int64
```

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

## 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_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=lda.classes_)
disp.plot(cmap=plt.cm.Blues,values_format='g');
plt.show()
```

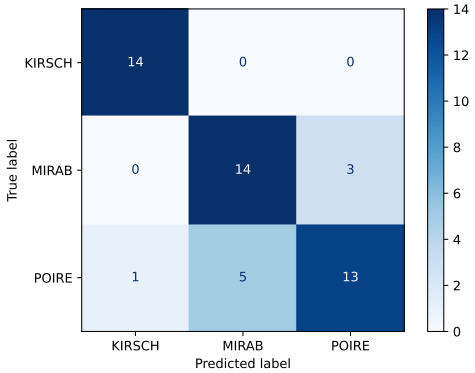


**Figure 1** – Matrice de confusion

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%.

Pour plus d'informations sur l'ADL sous scientisttools, consulter le notebook

https://github.com/enfantbenidedieu/scientisttools/blob/master/notebooks/lda_example.ipynb.