

**Rapport de projet de Python**

Analyse Discriminante Linéaire

*Sous la direction de M. Ricco Rakotomalala*

*Réalisé par : Aleksandra KRUCHININA, Aymeric DELEFOSSE et Mamadou DIALLO*

L3 MIASHS – IDS 2020-2021

Sommaire

[Introduction 3](#_Toc61286438)

[Module : calculations.py 3](#_Toc61286439)

[verification\_NA 3](#_Toc61286440)

[recodification\_var\_expl : 3](#_Toc61286441)

[freq\_relat 4](#_Toc61286442)

[means\_class 4](#_Toc61286443)

[cov\_matrix 4](#_Toc61286444)

[pooled\_cov\_matrix 4](#_Toc61286445)

[Wilks 5](#_Toc61286446)

[wilks\_log 5](#_Toc61286447)

[p\_value 6](#_Toc61286448)

[Module : reporting.py 7](#_Toc61286449)

[Classe HTML (Objet) 7](#_Toc61286450)

[Méthodes : 7](#_Toc61286451)

[create\_html\_head 7](#_Toc61286452)

[stepdisc\_html\_output 7](#_Toc61286453)

[discrim\_html\_output 8](#_Toc61286454)

[discrim\_html\_output\_datapane 8](#_Toc61286455)

[Classe PDF(FPDF) (Objet) 8](#_Toc61286456)

[footer 8](#_Toc61286457)

[discrim\_pdf\_output 8](#_Toc61286458)

[Module : discriminant\_analysis.py 10](#_Toc61286459)

[Classe LinearDiscriminantAnalysis (Objet) 10](#_Toc61286460)

[Attributs 10](#_Toc61286461)

[Méthodes 11](#_Toc61286462)

[Pseudo-code 16](#_Toc61286463)

[Table des matières 18](#_Toc61286464)

# Introduction

Pour réaliser notre application, nous avons choisi de créer une classe LDA, LinearDiscriminantAnalysis. Cette solution permet de créer des instances, de mieux structurer et d’organiser le code. De plus, cela permet d’utiliser les variables calculées dans certaines fonctions (qui n’étaient pas utilisées dans leurs retours) à partir d’autres fonctions. Par exemple, l’attribut **coef\_** qui est construit dans la fonction **fit()** et qui est utilisé ultérieurement dans la fonction **predict().**

Le code est divisé en trois modules : **calculations**, **reporting** et **discriminant\_analysis**.

# Module : calculations.py

C’est le module qui regroupe des fonctions de vérification et de modification de données ainsi que des fonctions permettant d'effectuer les calculs nécessaires pour la pratique de l'analyse discriminante linéaire (lambda de Wilks, matrice de covariance, etc.).

## verification\_NA

On a repris l’approche de la procédure DISCRIM de SAS : les valeurs nulles ne sont pas prises en compte pour l’analyse, donc les observations précédant ces valeurs sont annulées.

**Fonction verification\_NA(inputData : matrice de réels, targetValues  : vecteur de réels) :**

**Début**

Définir n et p

Concaténation des X (inputData) et y (targetValues)

Suppression des observations/lignes avec les valeurs nulles

Calcul du nombre de lignes supprimées

Redéfinition de X et y sans lignes supprimées

Affichage du nombre de lignes supprimées

Renvoyer nouveaux X et y

**Fin**

## recodification\_var\_expl :

La fonction *recodification\_var\_expl()* de notre module **calculations**  permet de convertir des variables qualitatives en variables quantitatives. L’Analyse de Discriminante Linéaire (ADL), ne cadrant tout simplement pas avec les variables qualitatives, nous oblige de recoder ces variables avant l’application de cette méthode statistique sur nos données. Plusieurs approches existent afin de recoder des variables qualitatives. Parmi elles, nous trouvons la méthode DISQUAL ainsi que la méthode des *dummy variables* (celle que nous utiliserons), pour ne citer que ces deux-là.  La méthode des *dummy variables* (Ricco Rakotomalala, *Pratique de l’Analyse Discriminante Linéaire*, section 3.3, p.76) octroie M-1 variables à chaque variables qualitatives possédant M modalités (notion modalité de référence). À ceux-ci, nous attribuons soit 0, soit 1, représentant la présence ou non de catégories.

À savoir :

* Cette fonction n’est utile qu’en cas de présence de **variables qualitatives** dans les **variables explicatives** des données ;
* Si nous sommes en présence que de variables quantitatives, il faut directement implémenter la fonction **fit()** de la classe LDA.

**Fonction recodification\_var\_expl(inputData : matrice de réels) :**

Création d’un dictionnaire vide et lui

Attribuer des valeurs soit vraie ou fausse selon que la variable est positive ou pas

Créer une liste de valeurs booléen

Convertir le type de la liste en type object de python si sa valeur égale à faux

Recoder les variables de type object en tenant compte de la modalité de référence

Jointure des données initiales avec les données recodées

Suppression de variables non numériques de nos données finales

Retourner les données recodées

## freq\_relat

La fonction calcule les fréquences relatives.

**Fonction freq\_relat(targetValues : matrice de réels, n : integer):**

**Début**

Calcul du nombre d'effectifs par classes (effClassValues)

Calcul des fréquences relatives par classes (freqClassValues)

Renvoyer ces deux vecteurs

**Fin**

## means\_class

La fonction calcule des moyennes conditionnelles selon le groupe d'appartenance.

**Fonction means\_class(inputData : matrice de réels, targetValues : vecteur de réels):**

**Début**

Calcul du nombre d'effectifs par classe

Initiation d'une matrice remplie de 0

Remplir cette matrice avec les effectifs par classe

Division par le nombre d'effectifs par classe

Renvoyer les moyennes conditionnelles

**Fin**

## cov\_matrix

La fonction calcule la matrice de covariance totale (V) et biaisée (Vb) entre les variables explicatives directement à partir du jeu de données.

Cette fonction a recours à la fonction [**cov()**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.cov.html) de la librairie **pandas**.

**Fonction cov\_matrix(dataset : pandas DataFrame) :**

**Var**

n = integer *# taille de l’échantillon*

V, Vb = matrices (p\*p) *# p est le nombre de variables explicatives*

**Début**

n := nombre de lignes du (dataset)

V := covariance(dataset) *# covariance est une fonction issue de la bibliothèque pandas*

Vb := (n-1)/n \* V

cov\_matrix := (V, Vb)

**Fin**

## pooled\_cov\_matrix

Cette fonction permet de calculer la matrice de covariance intra-classe totale (W) et biaisée (Wb).

D’après Ricco Rakotomalala, dans son livre sur la *Pratique de l’Analyse Discriminante Linéaire*, « en pratique, la matrice de covariance commune est estimée à l’aide de la matrice de variance covariance intra-classe » (section 1.2, p.25). Le calcul pour obtenir la matrice de covariance intra-classe est donc :

Où :

* N : taille de l’échantillon ;
* K : nombre de classes ;
* Vk : matrice de covariance pour la classe k.

**Fonction pooled\_cov\_matrix(dataset : pandas.DataFrame, className : string) :**

**Var**

n, nK, K = integer *# n = taille de l’échantillon, K = nombre de classes*

Vk, W, Wb = matrices (p\*p) *# p est le nombre de variables explicatives*

**Début**

n := nombre de lignes du (dataset)

K := nombre de valeurs uniques de dataset[classNames]

W := 0

Pour modalités dans (valeurs uniques de dataset[classNames]) :

nK := nombre de valeurs de (dataset[className] == modalités)

Vk := covariance((dataset[className] == modalités))

W := W + (nK-1)\*Vk

Fin Pour

W := W \* 1/(n-K)

Wb := (n-K)/n\*W

pooled\_cov\_matrix := (W, Wb)

**Fin**

## Wilks

La fonction calcule les valeurs du lambda de Wilks où :

Avec :

Ces deux matrices sont obtenues respectivement grâce aux fonctions **pooled\_cov\_matrix** et **cov\_matrix**, vues précédemment.

Cette fonction a recours à la fonction [**det()**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.det.html) de la librairie de calcul matriciel [**numpy.linalg**](https://numpy.org/doc/stable/reference/routines.linalg.html).

**Fonction wilks(Vb : matrice de réels, Wb : matrice de réels) :**

**Début**

detVb := det(Vb) *# obtenu grâce à la fonction det() de numpy.linalg*

detWb := det(Wb)

wilks := detWb/detVb

**Fin**

## wilks\_log

La fonction calcule les valeurs du lambda de Wilks à partir des logarithmes naturels des déterminants, où :

Cela permet de contourner les limitations des librairies utilisées dans les calculs scientifiques. En effet, si en entrée, les matrices sont trop grandes, le calcul standard avec la fonction **wilks()** ne sera pas en mesure d’aboutir.

Cette fonction a recours à la fonction [**slogdet()**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.slogdet.html) de la librairie de calcul matriciel [**numpy.linalg**](https://numpy.org/doc/stable/reference/routines.linalg.html).

**Fonction wilks\_log(Vb : matrice de réels, Wb : matrice de réels) :**

**Début**

detVb := logdet(Vb) *# obtenu grâce à la fonction slogdet() de numpy.linalg*

detWb := logdet(Wb)

wilks := exp(detWb-detVb)

**Fin**

## p\_value

La fonction calcule la p-valeur d’un test unilatéral de Fisher.

La fonction a recours à la librairie [**scipy.stats**](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.f.html).

**Fonction p\_value(F : réel, ddl1 : entier, ddl2 : entier) :**

**Début**

Si (F < 1) :

p\_value := (1.0 – f.cdf(1.0/F, ddl1, ddl2))

Fin Si

p\_value := (1.0 – f.cdf(F, ddl1, ddl2)

**Fin**

# Module : reporting.py

Ce module regroupe les différentes classes pour la création de reporting automatique, en format HTML et PDF, dont les sorties ressemblent à celles des PROC DISCRIM et STEPDISC de SAS.

## Classe HTML (Objet)

## Méthodes :

### create\_html\_head

Permet de créer le début d’un fichier HTML. La feuille de style est directement issue d’une feuille de style toute faite très populaire : [Bootstrap](https://getbootstrap.com).

Voilà ce que le fichier donne lorsqu’on lance simplement cette fonction. Notons qu’elle prend en paramètre une chaîne de caractères « proc », qui permet de changer le titre de la page avec le nom de la fonction souhaitée (STEPDISC ou DISCRIM).

<!DOCTYPE html>

<html lang="fr" dir="ltr">

  <head>

    <title>R&#233;sultats : %s</title>

    <meta charset="utf-8" />

    <style></style>

    <link

      rel="stylesheet"

      href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"

      integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"

      crossorigin="anonymous"

    />

  </head>

  <body>

    <div class="container text-center">

      <h2>Proc&#233;dure %s</h2>

### stepdisc\_html\_output

Ecriture des étapes de la sélection de variables (proc stepdisc) dans un fichier **.html**.

Les **pandas.DataFrame** sont transformés en tables HTML grâce à la fonction [**to\_html()**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_html.html).

**Fonction stepdisc\_html\_output(ProcStepdisc : objet LinearDiscriminantAnalysis,**

**fileName : chaîne)**

**Début**

Si (les 5 derniers caractères de fileName) ≠ ".html " :

fileName := fileName + ".html"

Fin Si

ouvrir(fileName)

create\_html\_head("STEPDISC")

écrire(ProcStepdisc.\_htmlStringOutput)

**Fin**

### discrim\_html\_output

Ecriture des étapes de la fonction **fit()** (proc discrim) un fichier **.html**, style basé sur **stepdisc\_html\_output**.

Les **pandas.DataFrame** sont transformés en tables HTML grâce à la fonction [**to\_html()**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_html.html).

**Fonction discrim\_html\_output(ProcStepdisc : objet LinearDiscriminantAnalysis,**

**fileName : chaîne)**

**Début**

Si (les 5 derniers caractères de fileName) ≠ ".html " :

fileName := fileName + ".html"

Fin Si

Appel des différents attributs de l’instance de la classe LinearDiscriminantAnalysis

contenant l’information qu’on a besoin de récupérer pour restituer les sorties

ouvrir(fileName)

create\_html\_head("DISCRIM")

Ecriture des différents attributs de la classe dans le fichier en format .html

**Fin**

### discrim\_html\_output\_datapane

Idem que discrim\_html\_output mais le style est différent, car basé sur la librairie [**datapane**](https://datapane.com).

**Fonction discrim\_html\_output\_datapane(ProcStepdisc : objet LinearDiscriminantAnalysis,**

**fileName : chaîne)**

**Début**

Si (les 5 derniers caractères de fileName) ≠ ".html " :

fileName := fileName + ".html"

Fin Si

Appel des différents attributs de l’instance de la classe LinearDiscriminantAnalysis

contenant l’information qu’on a besoin de récupérer pour restituer les sorties

ouvrir(fileName)

Ecriture des différents attributs de la classe dans le fichier en format .html grâce

à datapane

**Fin**

## Classe PDF(FPDF) (Objet)

Cette classe hérite la classe FPDF du package fpdf. Inspiré par le tutoriel : https://pyfpdf.readthedocs.io/en/latest/Tutorial/index.html

Méthodes :

### footer

Footer facilite l'affichage du bas de page lors d’une création d’un fichier pdf.

### discrim\_pdf\_output

Cette méthode créée un fichier pdf, dont l’utilisateur indique le nom. Il contient des sorties ressemblant aux sorties de la procédure DISCRIM de SAS : la description générale de l’échantillon (Taille d'échantillon totale, le nombre des variables et des classes, les DDL), l’information au niveau classe (les fréquences et proportions des classes), l’informations sur la matrice de covariance combinée, les statistiques de lambda de Wilks, les coefficients et l’intercept construits par le modèle.

La matrice de confusion et le taux de précision ne sont pas présentes dans cette méthode car on peut les invoquer par des méthodes séparées - confusion\_matrix() et accuracy\_score().

**Fonction discrim\_pdf\_output(ProcDiscrim : objet LinearDiscriminantAnalysis,**

**fileName : string) :**

**Début**

Si (les 4 derniers caractères de fileName) ≠ ".pdf" :

fileName := fileName + ".pdf"

Fin Si

Appel des différents attributs de l’instance de la classe LinearDiscriminantAnalysis

contenant l’information qu’on a besoin de récupérer pour restituer les sorties

Création d’une instance de la classe PDF (ouvrir(fileName))

Définition des caractères des polices pour le titre (ecriture(fileName))

Ajout du titre du paragraphe (ecriture(fileName))

Ligne rouge (ecriture(fileName))

Définition des caractères des polices pour le texte (ecriture(fileName))

Boucle : récupérant l’index et l’élément de l’info général des attributs de ProcDiscrim

Ajout des cellules de pdf contenant les informations selon l’attribut du modèle

choisi (ecriture(fileName))

Ligne rouge (ecriture(fileName))

Fin Boucle

Idem pour les statistiques des classes, l’information sur la matrice de covariance,

sur la fonction de classement, sur le lambda de Wilks (ecriture(fileName))

Sauvegarde le fichier() (fermer(fileName))

**Fin**

# Module : discriminant\_analysis.py

## Classe LinearDiscriminantAnalysis (Objet)

Ce module est le module principal et est le seul dont l’utilisateur aura besoin pour pratiquer l’Analyse Discriminante Linéaire. Après avoir chargé les données (grâce à pandas) et avoir repéré le nom de la variable catégorielle, l’utilisateur devra créer un objet issu de cette classe en spécifiant les deux paramètres.

**LDA = LinearDiscrimnantAnalysis(jeu\_de\_données, nom\_de\_la\_classe)**

À partir de là, l’utilisateur pourra directement appeler les différentes méthodes pour réaliser les calculs qu’ils souhaitent.

### Attributs

* **dataset** : matrice *# le jeu de données*
* **classEtiquette** : string *# le nom de la variable cible*
* **classNames** : liste de chaînes *# les noms des valeurs prises pour la variable cible*
* **varNames** : liste de chaînes *# les noms des variables explicatives*
* **n** : entier *# la taille de l'échantillon*
* **p** : entier *# le nombre de variables explicatives*
* **K** : entier *# le nombre de classes*
* **V** : matrice de réels *# les matrices de covariance totale*
* **Vb** : matrice de réels *# les matrices de covariance biaisée*
* **W** : matrice de réels *# la* *matrice* *de covariance intra-classe*
* **Wb** : matrice de réels *# la matrice de covariance intra-classe biaisée*
* **infoDataset** : pandas.DataFrame *# l’informations de bases sur le jeu de données*
* **infoCovMatrix** : pandas.DataFrame *# les statistiques de la matrice de covariance intra-classe*
* **infoClasses** : pandas.DataFrame *# les effectifs et fréquences relatives des classes*
* **infoWilksStats** : pandas.DataFrame *# les statistiques du Lambda de Wilks*
* **intercept**\_ : vecteur de réels *# l’intercept calculé par le modèle*
* **coef\_** : matrice de réels *# les coefficients du modèle*
* **infoFuncClassement** : pandas.DataFrame *# les valeurs de la fonction de classement*
* **confusionMatrix** : matrice de réels *# la matrice de confusion*
* **confusionMatrixGraph** : graphique *# graphique de la matrice de confusion*
* **accuracy** : réel *# le taux de précision*
* **infoWilksDecay :** liste de réels *# liste des valeurs du lambda de wilks en fonction du nombre de variables sélectionnées*
* **figWilksValue :** graphique *# courbe de la décroissance du lambda de Wilks*
* **\_htmlStringOutput :** string *# pour la sortie HTML de la sortie STEPDISC*
* **infoStepResults :** pandas.DataFrame *# lorsque l’approche de sélection de variables, affiche les différentes valeurs pour la dernière étape*
* **stepdiscSummary :** pandas.DataFrame *# résumé de l’approche de sélection de variables*

### Méthodes

Note : L’utilisation d’un **underscore** devant les méthodes à « weak internal usage » permet à ces méthodes de ne pas être convoquées lors de l’importation de la librairie   
(import \* from LinearDiscriminantAnalysis).

#### \_\_init\_\_

Création de l’objet à l’aide du constructeur python et affectation des différents attributs de base pour le bon fonctionnement des méthodes.

Prend en paramètre le jeu de données et le nom de la variable catégorielle. Il est également possible d’indiquer le nom des variables explicatives.

A la création de l’objet, on retrouvera les attributs suivants : **dataset**, **classEtiquette**, **classNames**, **varNames**, **n**, **p**, **K**, **V**, **Vb**, **W**, **Wb**.

#### \_stats\_dataset

Sert à créer l’attribut infoDataset qui représentent une matrice contenant l’information suivante : "Taille d'échantillon totale", "Variables", "Classes", "Total DDL", "DDL dans les classes", "DDL entre les classes".

#### \_stats\_pooled\_cov\_matrix

Cette méthode calcule des statistiques de la matrice de covariance intra-classes

**Fonction \_stats\_pooled\_cov\_matrix(self.W) :**

**Début**

Calcul du rang de la matrice de cov. intra-classes (self.W)

Calcul du logarithme naturel du déterminant de (self.W)

Restitution de l’attribut infoCovMatrix qui contient "Rang de la mat. de cov. intra-classes" et "Log. naturel du det. de la mat. de cov. intra-classes"

**Fin**

#### \_stats\_classes

Cette méthode calcule des statistiques des classes

**Fonction \_stats\_classes () :**

**Début**

Restitution de la variable cible

Calcul des effectifs et fréquences relatives des classes en utilisant la fonction du module calculations

Restitution de l’attribut infoClasses qui contient "Effectifs" et "Fréquences"

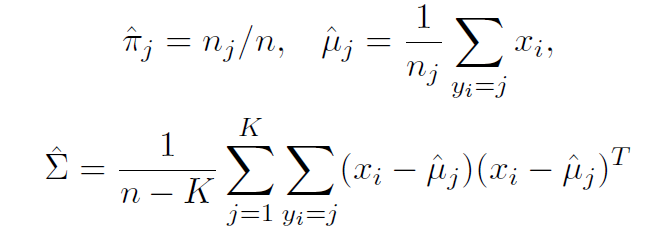
**Fin Fonction**

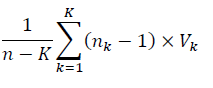
#### \_stats\_wilks

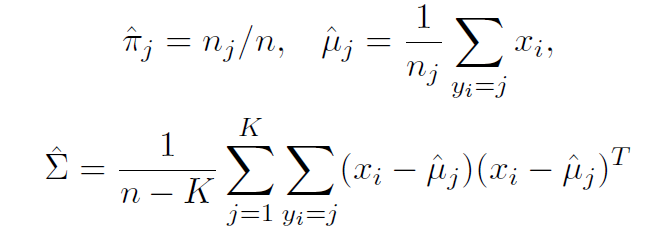
#### fit

Les valeurs nulles ne sont pas prises en compte et sont supprimées. Le message affiche le nombre d’observations supprimées. La methode renvoie les coefficients et l’intercept du modèle.

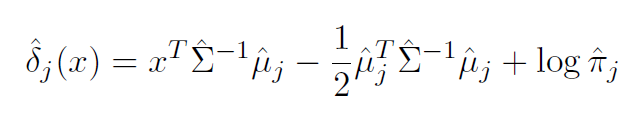
D’abord on calcule les moyennes conditionelles, les frequences relatives et la matrice de variance co-variance intra-classes[[1]](#footnote-1) :



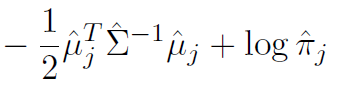




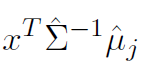
Apres on estime la fonction de classement par la formule :



Où l’intercept :



Et les coefficients :



**Fonction fit () :**

**Var**

invW : matrice

coef\_, intercept\_ : tableau

infoFuncClassement : tableau

**Début**

Sélection des valeurs des variables explicatives et cible

Vérification et suppression des valeurs nulles de dataset

Transformation des données en numpy

Calcul des effectifs et des fréquences de classes

Calcul des moyennes conditionnelles

Si le déterminant de la matrice de var-covar intra-classes (W) ≠ 0 :

invW := Calcul de la matrice inverse de W (invW)

Calcul des coefficients et de l’intercept

infoFuncClassement := Récupération des valeurs de la fonction de classement

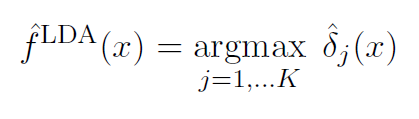
Sinon :

Afficher(erreur : la matrice n’est pas inversible)

Fin Si

**Fin**

#### predict

Pour calculer les prédictions il faut prendre le maximum parmi les scores d’appartenance à chaque classe. Ainsi on a finalement la formule pour calculer les prédictions : 

La fonction retourne le vecteur des classes prédites.

**Fonction predict(inputData : matrice des valeurs à prédire) :**

**Début**

Création d’une liste vide pour la remplir avec les prédictions

Parcours la matrice inputData

Calcul des probabilités d’appartenir aux classes

Stockage de ces valeurs dans une liste

Prédictions des classes en choisissant le maximum parmi les probabilités

**Fin**

#### confusion\_matrix

Avoir les prédictions c’est très bien, mais comment on peut savoir ou plutôt mesurer si ces prédictions sont correctes ou pas ? La matrice de confusion et le taux de précision y aident. La première « permet d’obtenir une évaluation non biaisée des performances du modèle en déploiement »[[2]](#footnote-2). Elle prend le vecteur des vraies valeurs de la variable cible qui n’étaient pas utilisées lors de l’apprentissage, et le vecteur des valeurs cibles prédites. Ainsi on peut estimer si la fonction est capable de bien prédire.

« Chaque ligne correspond à une classe réelle, chaque colonne correspond à une classe estimée. La cellule ligne L, colonne C contient le nombre d'éléments de la classe réelle L qui ont été estimés comme appartenant à la classe C »[[3]](#footnote-3).

Cette matrice permet de mesurer plusieurs d’autres indices. Mais comme cela n’est pas l’objectif de notre projet, on a retenu que la matrice de confusion et le taux de précision.

La fonction renvoie la matrice numérique.

En analyse des données c’est toujours utile de construire des graphiques ou d’avoir les données en format plus lisible. Le paramètre de la méthode graphShow est mis à True par défaut, cela nous affiche la matrice en mode plus représentatif (grâce à la librairie seaborn). Si on ne veut pas l’afficher, il faut mettre ce paramètre à False.

**Fonction confusion\_matrix(y\_true : vecteur des vrais valeurs de la cible, y\_pred : vecteur des valeurs de la cible à prédire, graphShow : booléen, True par défaut)**

**Début**

Conversion des noms des classes en numérique

Création d’une matrice zéro de taille K x K

Parcours des indices de 0 à K-1 (deux boucles imbriquées) :

Calcul des valeurs de la matrice de confusion

Si le paramètre graphShow est True :

Créer et afficher la matrice de confusion en mode graphique

Renvoyer la matrice de confusion

Fin Si

**Fin**

#### accuracy\_score

La fonction prend en entrée les vraies valeurs de la variable cible et les prédictions et renvoie la proportion des prédictions correctes.

Accuracy = (True Positive + True Negative) / (Positives + Negatives)

**Fonction accuracy\_score(y\_true : vecteur des vrais valeurs de la cible, y\_pred : vecteur des valeurs de la cible à prédire):**

**Début**

Conversion des noms des classes en numérique

Calcul du taux de précision

Renvoyer le taux de précision

**Fin**

#### wilks\_decay

Permet de calculer la valeur du lambda de Wilks en fonction du nombre de variables explicatives sélectionnées et d’afficher la courbe du décroissance du lambda de Wilks. Par exemple, s’il y a 40 variables explicatives, les valeurs du lambda seront pour {1, 2, …, 40}.

**Fonction wilks\_decay(graphShow : booléen, True par défaut) :**

**Var**

infoWilksValue : pandas.DataFrame

wilksValue, varSelection : tableau

L = réel

figWilksDecay : graphique

**Début**

Pour name dans varNames :

varSelection := varSelection + [name]

L := wilks\_log(Vb[varSelection, varSelection], Wb[varSelection, varSelection])

wilksValue := wilksValue + L

Fin Pour

infoWilksValue := pandas.DataFrame de la variable wilksValue

Si graphShow :

figWilksDecay := courbe de la décroissance de Wilks

Afficher(figWilksDecay)

Fin Si

**Fin**

#### stepdisc

Cette méthode permet de réaliser une méthode de sélection de variable. Elle est basée sur la section 4 livre sur la *Pratique de l’Analyse Discriminante* *Linéare* de Ricco Rakotomalala (pp. 87-108) et sur la PROC STEPDISC de SAS.

Elle permet à l’utilisateur d’aborder une approche ascendante, ou descendante.

**Fonction stepdisc(slentry : réel, method : chaîne)**

**Var**

\_htmlStringOutput : chaîne # pour la sortie HTML

infoStepResults, stepdiscSummary : pandas.DataFrame

varNames, colNames, enterdVarList, enteredVarSummary, infoVar : tableau de chaînes

L, L\_initial, F, R, pval : réel

ddl1, ddl2 : integer

**Début**

colnames := ["R-carre", "F-statistique", "p-value", "Lambda de Wilks"]

**Si method = "forward" :**

enteredVarList, enteredVarSummary := [], []

L\_initial := 0

**Pour q allant de 1 à p :**

**Pour name dans varNames :**

wilksVarSelect := enteredVarList + [name]

L := wilks\_log(Vb[wilksVarSelect, wilksVarSelect],

Wb[wilksVarSelect, wilksVarSelect]

ddl1 := K-1

ddl2 := n-K-q

F := ddl2/ddl1 \* (L\_initial/L-1)

R := 1-(L/L\_initial)

pval := p\_value(F, ddl1, ddl2)

infoVar := infoVar + [R, F, pval, L]

**Fin Pour**

infoStepResults = pandas.DataFrame(infoVar, index = varNames,

columns = colnames)

enteredVar := index (nom de la variable) de la valeur minimum de L de

infoStepResults

\_htmlStringOutput := \_htmlStringOutput + texte expliquant l’étape

**Si infoStepResults[enteredVar, p-valeur] > slentry :**

\_htmlStringOutput := \_htmlStringOutput + texte

expliquant l’étape

Break # fin du programme, sortie de la boucle

**Sinon :**

enteredVarList := enteredVarList + enteredVar

varNames := varNames – enteredVar

enteredVarSummary := stocke les valeurs de la valeur

entrée dans un DataFrame

L\_initial := infoStepResults(enteredVar, "Lambda de

Wilks")

\_htmlStringOutput := \_htmlStringOutput + texte

expliquant l’étape

**Fin Si**

stepdiscSummary := DataFrame de enteredVarSummary

\_htmlStringOutput := \_htmlStringOutput + texte expliquant l’étape

**Fin Pour**

**Sinon si method = “backward”:**

removedVarList, removedVarSummary := [], []

L\_initial := wilks\_log(Vb, Wb)

**Pour q allant de p à 1 :**

infoVar := []

**Pour nom dans varNames :**

wilksVarSelect := [les valeurs de varNames différentes de nom]

L := wilks\_log(Vb[wilksVarSelect, wilksVarSelect],

Wb[wilksVarSelect, wilksVarSelect])

ddl1 := K-1

ddl2 := n-K-q+1

F := ddl2/ddl1\*(L/L\_initial-1)

R := 1-(L\_initial/L)

pval := p\_value(F, ddl1, ddl2)

infoVar := infoVar + [R, F, pval, L]

infoStepResults = pandas.DataFrame(infoVar, index = varNames,

columns = colnames)

**Fin Pour**

removedVar := index (nom de la variable) de la valeur minimum de L de

infoStepResults

\_htmlStringOutput := \_htmlStringOutput + texte expliquant l’étape

**Si infoStepResults[removedVar, p-valeur] < slentry :**

\_htmlStringOutput := \_htmlStringOutput + texte expliquant

l’étape

Break # fin du programme, sortie de la boucle

**Sinon :**

removedVarList := removedVarList  + removedVar

varNames := varNames – removedVar

removedVarSummary := stocke les valeurs de la valeur entrée

dans un DataFrame

L\_initial := infoStepResults(removedVar, "Lambda de Wilks")

\_htmlStringOutput := \_htmlStringOutput + texte expliquant

l’étape

**Fin Si**

stepdiscSummary := DataFrame de enteredVarSummary

\_htmlStringOutput := \_htmlStringOutput + texte expliquant l’étape

**Fin Pour**

**Fin Si**

**Fin**

# Pseudo-code

#if we leave pseudo code under each corresponding methode, this section to be deleted

Fonction freq\_relat(y, n; taille de l'échantillon)

Fin Fonction

Fonction means\_class(X,y)

Fin Fonction

Fonction cov\_matrix(DATA)

Fin Fonction

Fonction pooled\_cov\_matrix(DATA, CLASS)

Fin Fonction

Fonction wilks(Vb, Wb)

Fin Fonction

Fonction wilks\_log(Vb, Wb)

Fin Fonction

Fonction wilks\_decay(VAR, Vb, Wb)

Fin Fonction

Fonction p\_value(F, ddl1, ddl2)

Fin Fonction

Fonction createWebFile(filename)

Fin Fonction

**class LinearDiscriminantAnalysis()**

Fonction variables\_explicatives(x)

Fin Fonction

Fonction create\_HTML()

Fin Fonction

# Table des matières

[Introduction 3](#_Toc61286401)

[Module : calculations.py 3](#_Toc61286402)

[verification\_NA 3](#_Toc61286403)

[recodification\_var\_expl : 3](#_Toc61286404)

[freq\_relat 4](#_Toc61286405)

[means\_class 4](#_Toc61286406)

[cov\_matrix 4](#_Toc61286407)

[pooled\_cov\_matrix 4](#_Toc61286408)

[Wilks 5](#_Toc61286409)

[wilks\_log 5](#_Toc61286410)

[p\_value 6](#_Toc61286411)

[Module : reporting.py 7](#_Toc61286412)

[Classe HTML (Objet) 7](#_Toc61286413)

[Méthodes : 7](#_Toc61286414)

[create\_html\_head 7](#_Toc61286415)

[stepdisc\_html\_output 7](#_Toc61286416)

[discrim\_html\_output 8](#_Toc61286417)

[discrim\_html\_output\_datapane 8](#_Toc61286418)

[Classe PDF(FPDF) (Objet) 8](#_Toc61286419)

[footer 8](#_Toc61286420)

[discrim\_pdf\_output 8](#_Toc61286421)

[Module : discriminant\_analysis.py 10](#_Toc61286422)

[Classe LinearDiscriminantAnalysis (Objet) 10](#_Toc61286423)

[Attributs 10](#_Toc61286424)

[Méthodes 11](#_Toc61286425)

[\_\_init\_\_ 11](#_Toc61286426)

[\_stats\_dataset 11](#_Toc61286427)

[\_stats\_pooled\_cov\_matrix 11](#_Toc61286428)

[\_stats\_classes 11](#_Toc61286429)

[\_stats\_wilks 11](#_Toc61286430)

[fit 11](#_Toc61286431)

[predict 13](#_Toc61286432)

[confusion\_matrix 13](#_Toc61286433)

[accuracy\_score 14](#_Toc61286434)

[wilks\_decay 14](#_Toc61286435)

[stepdisc 14](#_Toc61286436)

[Pseudo-code 16](#_Toc61286437)

1. https://www.stat.cmu.edu/~ryantibs/datamining/lectures/21-clas2.pdf [↑](#footnote-ref-1)
2. http://eric.univ-lyon2.fr/~ricco/cours/cours/Pratique\_Analyse\_Discriminante\_Lineaire.pdf [↑](#footnote-ref-2)
3. https://fr.wikipedia.org/wiki/Matrice\_de\_confusion [↑](#footnote-ref-3)