Econométrie des variables catégorielles

Théorie et application sous Python

Duvérier DJIFACK ZEBAZE

Econométrie des variables catégorielles

Théorie et application sous Python

Duvérier DJIFACK ZEBAZE

Table des matières

1 Introduction

Sommaire

Ce livre s'intéresse à la régression logistique. Il s'agit d'une technique de modélisation qui, dans sa version la plus répandue, vise à prédire et expliquer les valeurs d'une variable catégorielle binaire Y (variable à prédire, variable expliquée, variable dépendante, attribut, classe, variable endogène) à partir d'une collection de variables X continues ou binaires (variables prédictives, variables explicatives, variables indépendantes, descripteurs, variables exogènes). Elle fait partie des méthodes d'apprentissage supervisé et peut s'incrire dans lecadre de la régression linéaire généralisée. La régression logistique peut être vue comme une variante de la régression linéaire dans le cadre où la cible est qualitative.

1.1 Retour sur le modèle linéaire gaussien

1.1.1 contexte

Nous cherchons à expliquer une variable Y par p variables $X = \left(1, \mathbf{X}_1, \cdots, \mathbf{X}_p\right)'$. Pour ce faire, on dispose de n réalisations $(x_1, y_1), \cdots, (x_n, y_n)$ du couple (X, Y). Le but est de modéliser la dépendance de la variable réponse Y sur les variables explicatives $\mathbf{X}_1, \cdots, \mathbf{X}_p$.

Le modèle linéaire gaussien est donné par l'équation :

$$Y = X\beta + \varepsilon \tag{1.1}$$

qui se décline ligne à ligne (individu par individu) en

$$Y_i = \beta_0 + \sum_{j=1}^{j=p} \beta_j x_{i,j} + \varepsilon_i \tag{1.2}$$

Ce modèle a deux fonctions :

— Expliquer au mieux la variable Y comme une fonction linéaire de covariables X à laquelle s'ajoute un terme d'erreur individuelle, supposé gaussien et de variance constante ;
— Prédire la valeur de Y au vu d'un nouveau jeu de covariables (avec toutes les précautions d'usage quand on fait de la prédiction statistique).

Ce modèle est très populaire parce que son caractère gaussien se justifie souvent physiquement (via le théorème central limite), parce qu'il est assez facile à manier en pratique, et parce qu'il a des propriétés mathématiques bienvenues : notamment l'estimateur des moindres carrés ordinaires de β est égal à l'estimateur du maximum de vraisemblance.

Rappelons que ce modèle repose sur l'idée que les valeurs observées des covariables sont déterministes. Cela permet de dire, par exemple, que la loi de Y_i est gaussienne, d'espérance $\beta_0 + \sum_{j=1}^{p} \beta_j x_{i,j}$, et de variance σ^2 (hypothèse d'homoscédasticité).

Une autre approche est de dire que ces valeurs sont en fait des réalisations de variables aléatoires $X_{i,j}$. C'est un formalisme très naturel, mais qui complique un peu l'écriture : dans ce cas, c'est la loi conditionnelle de Y_i sachant $X_{i,1} = x_{i,1}, \dots, X_{i,p} = x_{i,p}$ qui est gaussienne d'espérance $\beta_0 + \sum_{j=1}^{p} \beta_j x_{i,j}$ (en faisant l'hypothèse supplémentaire que les $X_{i,j}$ sont indépendants de ε_i). Cette approche conditionnelle est souvent utilisé notamment en économétrie.

1.1.2 Limites

Parmi les hypothèses du modèle gaussien, on retient les suivantes :

— On peut **toujours** faire techniquement une régression linéaire, même quand cela n'est pas pertinent ;
— La variable à expliquer Y est gaussienne ;
— le bruit (l'erreur) est additif ;
— Les observations sont indépendantes ;
— Les observations sont de même variance (homoscédasticité) ;
— Les covariables sont supposées déterministes.

Dans le cours de modèle linéaire gaussien, il est possible de se ramener au modèle linéaire gaussien quand certaines de ces hypothèses ne sont pas vérifiées :

— transformation de variables pour se ramener à un bruit additif (typiquement : passage au logarithme dans le cas d'un bruit multiplicatif,nuages en trompette) ;
— Matrices de covariance plus générales pour ε quand la deuxième hypothèse n'est par vérifiée ;
— Transformation de la variable Y pour la rendre gaussienne.

Il se trouve que cette dernière idée est inopérante dans des cas concrets extrêmement courants, en particulier quand la réponse en Y est intrinsèquement discrète : le modèle linéaire gaussien ne peut pas être utilisé pour expliquer Y avec l'aide des covariables (qui peuvent être continues ou non).

Quelques exemples illustratifs :

1. Retour à l'emploi en moins de trois mois ($Y_i = 1$ si oui, $Y_i = 0$ si non), à expliquer par l'entrée (ou non) dans un dispositif particulier, l'âge, le sexe, la CSP, le niveau d'études, le nombre de périodes de non emploi, etc.
2. Efficacité d'un traitement médical ($Y_i = 1$ si oui, $Y_i = 0$ si non), à expliquer par l'age, le sexe,la CSP, etc.
3. Segmentation d'une clientèle en k catégories, l'affectation d'un individu dans une catégorie étant expliquer par l'âge, le sexe,la CSP, le revenu, les fréquentations,etc.
4. Caractère frauduleux d'une déclaration à l'ouverture d'un compte bancaire (régresseurs confidentiels, etc.)
5. Nombre de morts par accidentologie routière en un mois, à expliquer par l'état de la route, sa nature, etc.
6. Prêt accordé ou non, à expliquer par l'âge, le revenu, le taux d'endettement, le déficit budgétaire (pour un individu), la balance commerciale (pour un pays), etc.

Dans ces exemples, on peut s'intéresser notamment à l'effet spécifique d'un régresseur donné, par exemple le dispositif considéré dans l'exemple 1.

On peut remarquer que dans certains exemples (notamment dans l'exemple 1), c'est surtout **l'explication** Y par les covariables qui est intéressante, alors que pour d'autres c'est la partie **prédictive** qui l'est le plus (notamment dans les exemples 3 et 4). Mais à bien réfléchir, prédire une variable qui ne peut prendre que les valeurs 0 et 1 n'est pas intéressant d'un point de vue statistique, et il est intéressant d'essayer d'estimer la probabilité pour que cette variable prenne la valeur 1 (par exemple, la partie intervalle de confiance devient problématique). Quoi qu'il en soit, le modèle linéaire classique est clairement impuissant à traiter ces exemples.

1.2 Le modèle linéaire généralisé

1.2.1 La régression logistique

Sur la base des remarques évoquées ci-dessous, nous reformulons l'objet de notre étude. Quand la variable à expliquer Y est binaire, sa loi est une loi de Bernoulli, et nous nous intéressons à l'influence des covariables X sur son paramètre, c'est-à-dire la probabilité $p(x)$ qu'elle vaille 1 si le vecteur des covariables est égal à *x*.

Dans le cas de l'exemple 2 ci-dessus, il est naturel de modéliser le nombre d'accidents par une loi de Poisson conditionnellement aux covariables (quitte à valider cette hypothèse a posteriori) et l'objet d'intérêt devient le paramètre $\lambda(x)$ de cette loi pour une valeur donnée *x* des covariables.

Cette idée pourrait nous inciter à étudier des modèles du type :

$$p(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \tag{1.3}$$

et

$$\lambda(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \tag{1.4}$$

Mais cela ne peut être satisfaisant, parce que dans de tels modèles $p(x_i)$ peut être plus grand que 1 ou négatif, et $\lambda(x)$ négatif, ce qui est évidemment à proscrire.

On résout cette nouvelle difficulté par un changement de variable (cette fois-ci, c'est possible !), c'est-à-dire au moyen d'une transformation du paramètre permettant de couvrir tout $\mathbb{R}$. Voici trois exemples (non anodins) de telles transformations quand Y est une variable aléatoire de Bernoulli :

— $g(t) = \ln\left(\frac{t}{1-t}\right) = \text{logit}(t)$
— $g(t) = \Phi^{-1}(t)$ où Φ désigne la fonction de répartition de la loi normale centrée réduite ;
— $g(t) = \ln(-\ln(1-t))$ (on appelle parfois cette fonction `fonction log-log`)

Remarquons que pour le deuxième exemple, on pourrait prendre l'inverse de toute autre fonction de répartition d'une loi continue sur $\mathbb{R}$(`cauchy`, `student`,etc...). On s'intéresse alors au modèle donnée par :

$$g\left(p(x)\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \tag{1.5}$$

Dans le cas de l'intensité d'une loi de Poisson (conditionnellement aux valeurs prises par les cofacteurs), le même principe conduit à proposer le modèle

$$g\left(\lambda(x)\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \tag{1.6}$$

et $g(t) = \ln(t)$ est un choix assez naturel. Les fonctions g qui interviennent ci-dessus sont appelées `fonctions de lien`.

Définition 1.1 (Régression logistique) *Soit Y une variable à valeurs dans $\{0,1\}$ à expliquer par p variables explicatives $X = \left(1, X_1, \ldots, X_p\right)'$. Le modèle **logistique** propose une modélisation de la loi de $Y|X = x$ par une loi de Bernoulli de paramètre $p(x) = \mathbb{P}\left(Y = 1|X = x\right)$ telle que :*

$$\log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = x'\beta \tag{1.7}$$

ou encore

$$\texttt{logit}\left(p(x)\right) = x'\beta$$

`logit` désignant la fonction bijective et dérivable de $]0,1[$ dans $\mathbb{R}$: $p \to \log\left(\frac{p}{1-p}\right)$.

L'égalité 1.7 peut également s'écrire :

$$p(x) = \mathbb{P}\left(Y = 1|X = x\right) = \frac{\exp\left(x'\beta\right)}{1 + \exp\left(x'\beta\right)}$$

Remarque 1.1 *Dans un modèle logistique, deux choix sont à effectuer pour définir le modèle :*

1. *le choixe d'une loi pour* $Y|X=x$*, ici la loi de Bernoulli*
2. *le choix de la modélisation de* $\mathbb{P}(Y=1|X=x)$ *par :*

$$\texttt{logit}\left(\mathbb{P}(Y=1|X=x)\right) = x^{'}\beta$$

La fonction `logit` *est bijective et dérivable.*

Propriété 1.1 *Nous avons les propriétées suivante :*

$$\begin{cases} \mathbb{E}(Y|X=x) & = p(x) \\ V(Y|X=x) & = p(x)\left(1-p(x)\right) \end{cases}$$

Ce qui implique que la variance n'est pas constante et varie selon x*.*

1.2.2 La régression log - linéaire

Dans le moèdle logistique, la variable à expliquer est une variable binaire. Le modèle log - linéaire traite le cas d'une *variable de comptage*. On peut citer par exemple : le nombre de catastrophes aériennes sur une période donnée ; le nombre de voitures à un heu rouhe ou encore le nombre d'accidents par jour sur une autoroute.

Définition 1.2 (Régression log - linéaire) *Soit* Y *une variable de comptage à expliquer par le vecteur* $X = \left(1, X_1, \cdots, X_p\right)^{'}$*. Le modèle* ***log - linéaire*** *propose une modélisation de la loi de* $Y|X=x$ *par une loi de poisson de paramètre* $\lambda = \lambda(x)$ *telle que :*

$$\log \mathbb{E}\left[Y|X=x\right] = x^{'}\beta \tag{1.8}$$

Pour une nouvelle mesure x effectuée, le modèle log - linéaire va donc prédire la valeur $\exp\left(x^{'}\beta\right)$.

Remarque 1.2 *Deux choix sont effectués pour définir le modèle log - linéaire :*

1. *Le choix d'une loi pour* $Y|X=x$*, ici la loi de Poisson*
2. *Le choix de la modélisation de* $\mathbb{E}\left[Y|X=x\right]$ *par*

$$\log \mathbb{E}\left[Y|X=x\right] = x^{'}\beta$$

La fonction log *est bijective et dérivable.*

1.2.3 Exemples de fonctions de liens

D'autres fonctions de lien que `logit` peuvent être utilisées dans le cas où la variable à expliquer Y est binaire. On trouve notamment dans la littérature les transformations :

— `probit`, qui n'est autre que l'inverse de la fonction de répartition de la loi normale centrée réduite :

$$\forall p \in [0,1], \quad \texttt{probit}(p) = \varepsilon \quad \text{avec} \quad \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\varepsilon} \exp\left(-\frac{1}{2}u^2\right) \mathrm{d}u = p$$

.

— `log-log` définie par :

$$\forall p \in [0,1], \quad \texttt{log-log}(p) = \log(-\log(1-p))$$

.

```
# Draw link function
import numpy as np
import pandas as pd
from scipy.special import logit
import scipy.stats as st
from plotnine import *

# long - long function
def loglog(x):
    return np.log(-np.log(1-x))

p = np.linspace(0.05, 0.99,num=1000,endpoint=False)
data = pd.DataFrame({"p" : p,"logit":logit(p),"probit":st.norm.ppf(p),
                     "log-log":loglog(p)})
data_long = data.melt(id_vars=["p"],var_name="link",value_name="value")
print((ggplot(data_long,aes(x="p",y="value",color="link"))+geom_line()+
       theme(legend_position=(0.2,0.7),legend_direction="vertical")))
```

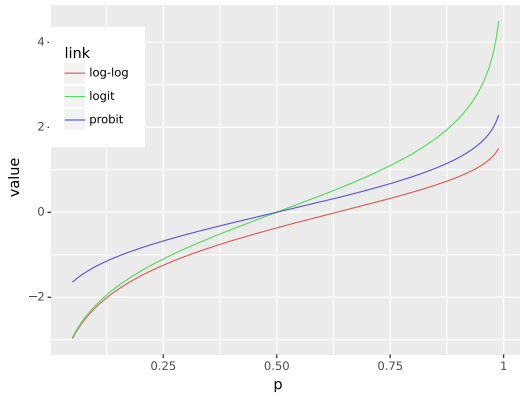


Figure 1.1 – Fonctions de liens : probit, logit, log-log

Des trois fonctions de lien présentées, la transformation `log-log` est bien appropriée aux cas où l'on souhaite modéliser les probabilités de succès de manière asymétrique. Les transformations `logit` et `probit` possèdent des propriétés identiques. Dans de nombreux cas, on préfère utiliser la transformation logistique. Plusieurs raisons motivent ce choix :

— D'un point de vue numérique, la transformation logistique est plus simple à manipuler (notamment pour l'écriture des estimateurs du maximum de vraisemblance) ;

— On a une interprétation claire des coefficients en terme d'odds ratio pour la transformation logistique ;
— Le modèle logistique est particulièrement bien adapté à un schéma d'échantillonnage rétrospectif.

Table 1.1 – Exemples de fonction de lien

Loi	Nom du lien	Fonction de lien
Bernoulli/Binomiale	lien logit	$g(t) = \text{logit}(t) = \log(t/1-t)$
Poisson	lien log	$g(t) = \log(t)$
Normale	lien identité	$g(t) = t$
Gamma	lien réciproque	$g(t) = -1/t$

1.2.3.1 Données

Autant que faire, on utilisera une base de données qui contient des données de 462 patients pour lesquels on souhaite prédire l'exposition à un infarctus. La base est disponible https ://statweb.stanford.edu/ tibs/ElemStatLearn/datasets/SAheart.data. C'est d'ailleurs un site où des données sont disponibles afin de s'entraîner sur la modélisation en employant diverses techniques.

```
# Chargement des données
donnee = pd.read_csv('./donnee/deseases.txt',sep=',',header=0,index_col=0)
print(donnee.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 462 entries, 1 to 463
## Data columns (total 10 columns):
##  #   Column     Non-Null Count  Dtype
## ---  ------     --------------  -----
##  0   sbp        462 non-null    int64
##  1   tobacco    462 non-null    float64
##  2   ldl        462 non-null    float64
##  3   adiposity  462 non-null    float64
##  4   famhist    462 non-null    object
##  5   typea      462 non-null    int64
##  6   obesity    462 non-null    float64
##  7   alcohol    462 non-null    float64
##  8   age        462 non-null    int64
##  9   chd        462 non-null    int64
## dtypes: float64(5), int64(4), object(1)
## memory usage: 39.7+ KB
## None
```

— **sbp** : pression sanguine systolique (systolic blood pressure)
— **tobacco** : la qualité de tabac consommé cumulée (cumulative tobacco (kg))
— **ldl** : taux de cholestérol dans le sang (low density lipoprotein cholesterol)
— **adiposity** : adiposité
— **famhist** : antécédents familiaux : Présent s'il y a eu des antécédants familiaux (family history of heart disease (`Present`, `Absent`))

— **typea** : comportement de type A (type-A behavior)
— **obesity** : obésité
— **alcohol** : consommation courante d'alcool (current alcohol consumption)
— **age** : age au moment de l'attaque cardiaque (age at onset)
— **chd** : variable réponse codée 1 si la maladie du coeur est présente, 0 sinon (reponse, coronary heart disease)

Pour notre modélisation, on s'intéressera à cette dernière variable. On affiche les colonnes et leurs types.

```
#liste des colonnes et leurs types
print(donnee.dtypes)
```

```
## sbp             int64
## tobacco       float64
## ldl           float64
## adiposity     float64
## famhist        object
## typea           int64
## obesity       float64
## alcohol       float64
## age             int64
## chd             int64
## dtype: object
```

Notre variable `chd` dans la base est de type `int`. Nous la transformons en variable catégorielle.

```
# transformation en variable catégorielle
donnee['chd'] =  donnee['chd'].astype('category')
print(donnee.dtypes)
```

```
## sbp             int64
## tobacco       float64
## ldl           float64
## adiposity     float64
## famhist        object
## typea           int64
## obesity       float64
## alcohol       float64
## age             int64
## chd          category
## dtype: object
```

2

Régression logistique binaire

Sommaire

En apprentissage supervisé, l'objectif est de prédire et/ou expliquer une variable Y à partir d'une collection de descripteurs $X = \left(1, X_1, \ldots, X_p\right)$. Il s'agit en quelque sort de mettre en évidence l'existence d'une liaison fonctionnelle sous - jacente de la forme :

$$Y = f\left(X, \beta\right) \tag{2.1}$$

entre ces variables. Dans le cas de la régression logistique, la variable cible Y est qualitative. La fonction $f(.)$ est le modèle de prédiction, on parle aussi de classifieur ; β est le vecteur des paramètres de la fonction, on doit estimer les valeurs à partir des données disponibles.

2.1 Un cadre bayesien pour l'apprentissage supervisée

2.1.1 Problématique - mise en oeuvre

Rappelons que dans le cadre de la discrimination binaire (*cf.* Rakotomalala (2011)), nous considérons que la variable dépendante Y ne prend que 2 modalités : positif (+) ou négatif (−). Nous cherchons à prédire correctement les valeurs de Y, mais nous pouvons également vouloir quantifier la propension (la probabilité) d'un individu à être positif (ou négatif).

Le classifieur naif bayésien repose sur l'hypothèse forte que les variables explicatives X_j sont indépendantes conditionnellement à la classe de Y. Malgré cette hypothèse simpliste d'indépendance entre les regresseurs conditionnellement aux valeurs prises par Y, le classifieur bayésien naif s'avère souvent plus performant que des modèles plus sophistiqués.

Si le nombre de régresseurs est grand, cette technique est particulière appropriée. En

effet, en grande dimension, l'estimation des fonctions de densité devient très compliquée. Avec le classifieur bayésien naif, chaque loi de probabilité des X_j conditionnellement aux valeurs de Y peut être estimée indépendamment en tant que loi de probabilité à une dimension.

Pour un individu ω, il s'agit de calculer les probabilités conditionnelles (probabilités a posteriori) :

$$\mathbb{P}\left[Y(\omega) = y_k / X(\omega)\right] \tag{2.2}$$

pour chaque modalité y_k de Y.

On affecte à l'individu la modalité la plus probable y_{k^*} c'est-à-dire :

$$y_{k^*} = \arg\max_k \mathbb{P}\left[Y(\omega) = y_k / X(\omega)\right] \tag{2.3}$$

On associe donc l'individu à la classe la plus probable compte tenu de ses caractéristiques $X(\omega)$. Cette approche est optimale au sens de l'erreur théorique, mais un problème apparaît tout de suite : `comment estimer correctement les probabilités conditionnelles?`

Exemple 2.1 *Prédire chd en fonction de famhist*

Pour nos données de la section 1.2.3.1, nous souhaitons prédire les valeurs de `chd` en fonction de `famhist`. La variable prédictive `famhist` étant binaire (et de manière plus générale catégorielle), nous pouvons utiliser les fréquences pour estimer les probabilités conditionnelles. Premièrement, on construit le tableau de contingence :

```
# Chargement des données
import pandas as pd
donnee = pd.read_csv('./donnee/deseases.txt',sep=',',header=0,index_col=0)
# Tableau de contingence
crosstab = pd.DataFrame(pd.crosstab(donnee.chd,donnee.famhist))
tab = crosstab.reset_index()
```

Table 2.1 – chd vs. famhist - Tableau de contingence

chd	Absent	Present
0	206	96
1	64	96

On calcule ensuite fréquences/probabilités conditionnelles

```
# Fréquences conditionnelles
condmean = crosstab.apply(lambda x:100*x/sum(x), axis=0)
condmean.loc['Total'] = condmean.sum(axis=0)
tab = condmean.reset_index()
```

Pour `famhist = Absent`, nous avons les fréquences conditionnelles :

— $\mathbb{P}(chd = 0 / famhist = Absent) = 0.763$

Table 2.2 – chd vs. famhist - Probabilités conditionnelles

chd	Absent	Present
0	76.2963	50
1	23.7037	50
Total	100.0000	100

— $\mathbb{P}(chd = 1/famhist = Absent) = 0.237$

En vertu du principe bayesien, nous adoptons le règle suivante :

$$\text{Si } \texttt{famhist = Absent} \text{ alors } \texttt{chd=0}\ (absent) \tag{2.4}$$

De la même manière, pour `famhist = Present`, nous calculons :

— $\mathbb{P}(chd = 0/famhist = Present) = 0.50$
— $\mathbb{P}(chd = 1/famhist = Present) = 0.50$

Nous en déduisons :

$$\text{Si } \texttt{famhist = Present} \text{ alors } \texttt{chd=1}\ (present) \tag{2.5}$$

Maintenant nous avons un modèle de prédiction $chd = f(famhist)$, il faut en évaluer les performances. Pour cela, nous confrontons les vraies valeurs de la variable dépendante avec celles prédites par le modèle.

```
# prédiction
import numpy as np
prediction = np.where(donnee.famhist=='Present',1,0)
# error rate
error = np.sum(donnee.chd!= prediction)/donnee.shape[0]
print("Erreur en resubstitution : %.4f" % (error))
```

```
## Erreur en resubstitution : 0.3463
```

Nous en déduisons le taux d'erreur 34.63% c'est-à-dire si nous classons un individu pris au hasard dans la population, nous avons 34.63% de chances de faire une prédiction erronée. A l'inverse, nous avons 65.37% de chances de faire une prédiction correcte.

Attention, il s'agit bien d'une erreur en resubstitution puisque le modèle a été élaboré (dans notre cas, les probabilités conditionnelles ont été calculées) à partir des mêmes données. Les performances annoncées sont donc sujettes à caution.

2.1.2 Limite de l'approche basée sur les fréquences

La démarche basée sur les fréquences est extrêmement séduisante par sa simplicité. Un simple comptage permet de produire les probabilités conditionnelles et déduire les règles d'affectation. Toutefois, elle n'est pas viable en situation réelle, lorsque nous avons plus d'une variable prédictive, pour différentes raisons :

1. Dans le cas où toutes les variables sont binaires, le nombre de probabilités à calculer devient rapidement prohibitif, impossible à gérer même sur les ordinateurs. Par exemple, si nous avons 20 variables, il faudrait procéder à $2 \times 2^{20} = 2.097.152$ comptages.
2. Et même si cela était possible, nous aurions la valeur 0 dans la plupart des cases de notre tableau croisé, ou tout du moins de très faibles effectifs, rendant inutilisables les estimations.
3. L'affaire se corse lorsque nous avons des descripteurs continus. Procéder par comptage global n'a plus de sens. Il faut passer par d'autres stratégies : soit en `discrétisant` ces variables (les découper en intervalles) ; soit en `estimant par comptage` les probabilités, mais localement, en se limitant au voisinage de l'observation à classer (cf. par exemple la méthode des plus proches voisins, les noyaux de Parzen, etc.).
4. Et on ne parle même pas de la situation où l'on a un mélange de variables prédictives continues et catégorielles. La solution pourrait passer par un découpage en classes des variables continues, mais il faudrait proposer des découpages pertinents, au moins en relation avec la variable à prédire, et peut être aussi en relation avec les autres variables prédictives pour tenir compte des possibles interactions.
5. Enfin, en admettant que tous les problèmes ci-dessus aient été résolus, il reste un écueil : il n'y a pas de processus de sélection de variables inhérent à la méthode. Elle ne nous indique pas quelles sont les variables pertinentes qu'il faut conserver, quelles sont les variables qui ne servent rien et que l'on peut évacuer. Pourtant, cet aspect est incontournable dès que l'on est confronté à un problème un tant soit peu réaliste. L'expert du domaine a certes une idée plus ou moins vague des « bonnes » variables, mais bien souvent il compte sur les techniques numériques pour préciser ses idées.

2.1.3 Modèle linéaire généralisé : GLM

Nous nous plaçons dans un contexte de classification binaire (*cf.* Hilbe (2016)), c'est-à-dire que nous supposons qu'il existe seulement deux groupes à discriminer.

2.1.3.1 Définition

On suppose désormais qu'on observe $p+1$ vecteurs aléatoires $(Y_i, X_{i,1}, \dots, X_{i,p})$ indépendants. On notera $X_i = (1, X_{i,1}, \dots, X_{i,p})$. On se dote par ailleurs d'un vecteur de paramètres $\beta = (\beta_0, \beta_1, \dots, \beta_p)$, et d'une fonction de lien g bijective et dérivable.

Définition 2.1 *Un modèle linéaire généralisé (GLM) établit une relation linéaire de type*

$$g\left(\mathbb{E}\left(Y_i / X_i = x_i\right)\right) = x_i \cdot \beta$$

ou, ce qui revient au même

$$g\left(\mathbb{E}\left(Y_i / X_i\right)\right) = X_i \cdot \beta$$

où le produit scalaire s'écrit $x_i \cdot \beta = \beta_0 + \beta_1 x_{i,1} + \cdots + \beta_p x_{i,p}$

On supposera tout au long de ce document que la loi conditionnelle de Y_i sachant X_i appartient à la famille exponentielle (afin de garantir les bonnes qualités des estimateurs à étudier).

2.1.3.2 Exemples de modèle GLM

Exemple 2.2 (Le modèle linéaire gaussien)

On le retrouve en prenant des Y_i de loi normale (conditionnellement aux X_i), avec la fonction de lien identité $g(t) = t$. On a alors $\mathbb{E}(Y_i/X_i = x_i) = x_i \cdot \beta$, autrement dit $\mathcal{L}(Y_i/X_i = x_i)$ est la loi normale d'espérance $x_i \cdot \beta$ et de variance σ^2 supposée indépendante de i (homoscédasticité). Cela revient à dire que $Y_i = x_i \cdot \beta + \varepsilon_i$, où ε_i est une variable aléatoire de loi normale, centrée et de variance σ^2.

Exemple 2.3 (Le modèle logistique)

Ici,les Y_i sont de loi de Bernoulli, avec la fonction de lien $g(t) = \ln(t/1-t) = \text{logit}(t)$. On a donc le modèle

$$\ln\left(\frac{p(x_i)}{1-p(x_i)}\right) = x_i \cdot \beta \tag{2.6}$$

équation qui s'inverse en

$$p(x_i) = \frac{e^{x_i \cdot \beta}}{1 + e^{x_i \cdot \beta}}$$

Exemple 2.4 (Le modèle probit)

Ici, les Y_i sont de loi de Bernoulli, avec la fonction de lien $g(t) = \Phi^{-1}(t)$ où Φ désigne la fonction de répartition de la loi normale centrée réduite. On a donc le modèle

$$\Phi^{-1}(p(x_i)) = x_i \cdot \beta \tag{2.7}$$

équation qui s'inverse en

$$p(x_i) = \Phi(x_i \cdot \beta)$$

Exemple 2.5 (Le modèle log-log (ou de Gomperz))

Ici, les Y_i sont de loi de Bernoulli, avec la fonction de lien $g(t) = \ln(-\ln(1-t))$.On a donc le modèle :

$$\ln(-\ln(1-p(x_i))) = x_i \cdot \beta \tag{2.8}$$

équation qui s'inverse en

$$p(x_i) = 1 - e^{-e^{x_i \cdot \beta}}$$

Exemple 2.6 (Le modèle log-linéaire, ou de Poisson)

Ici, les Y_i sont, conditionnellement aux X_i, de loi de Poisson, avec la fonction de lien $g(t) = \ln(t)$.On a donc le modèle

$$\ln(\lambda(x_i)) = x_i \cdot \beta \tag{2.9}$$

équation qui s'inverse en

$$\lambda(x_i) = e^{x_i \cdot \beta}$$

Remarque 2.1 *Notez que l'hypothèse selon laquelle la variable aléatoire Y_i est de loi de Poisson conditionnellement à chaque valeur possible des covariables n'entraîne pas que Y_i soit de loi (non conditionnelle) de Poisson.*

2.2 L'estimateur du maximum de vraisemblance

2.2.1 Estimation des paramètres

Il s'agit donc d'expliquer une variable binaire Y par p régresseurs $X_j, 1 \leq j \leq p$ auxquels on ajoute le régresseur constant $X_0 \equiv 1$, à l'aide du modèle de régression logistique, et ce au vu de $n-$copies (observations) indépendantes du vecteur $(Y, 1, X_1, \ldots, X_p)$. Le modèle s'écrit donc :

$$\text{logit}\left(p_\beta(X)\right) = \ln\left(\frac{p_\beta(X)}{1 - p_\beta(X)}\right) = X \cdot \beta \tag{2.10}$$

où $\beta = (\beta_0, \beta_1, \cdots, \beta_p)^{'}, p_\beta(X) = \mathbb{P}(Y = 1/X)$, et $X \cdot \beta = \beta_0 + X_1\beta_1 + \cdots + X_p\beta_p$. On prendra systématiquement des modèles contenant la constante comme régresseur d'ordre 0. L'écriture p_β signifie tout simplement que cela dépend de la valeur du paramètre choisi.

Remarque 2.2

Il est essentiel que le modèle soit identifiable ; dans le cas du modèle logistique, cela s'interprète par le fait que si on a deux valeurs différentes β et $\beta^{'}$ pour le paramètre, on doit trouver une observation x des régresseurs pour laquelle $p_\beta(x) \neq p_{\beta'}(x)$. Mathématiquement, cela se traduit par le fait que la matrice $X^{'}X$, où $X = (1, X_1, \cdots, X_p)$ est de rang $p+1$, ce qui implique en particulier que $n \geq p+1$ (on a davantage d'observations que de régresseurs).

Le calcul de la vraisemblance se fait comme dans le cas du modèle linéaire gaussien, même si formellement on calcule des vraisemblances conditionnelles : l'indépendance permet de dire que la vraisemblance conditionnelle d'un $n-$uplet d'observations est le produit des vraisemblances de chaque observation. Dans le cas du modèle logistique, cela donne :

$$\begin{aligned} L(Y_1,\cdots,Y_n,\beta/X_1=x_1,\cdots,X_n=x_n) &= \prod_{i=1}^{i=n} p_\beta(x_i)^{Y_i}(1-p_\beta(x_i))^{1-Y_i} \\ &= \prod_{i=1}^{i=n}\left(\frac{e^{x_i\beta}}{1+e^{x_i\beta}}\right)^{Y_i}\left(1-\frac{e^{x_i\beta}}{1+e^{x_i\beta}}\right)^{1-Y_i} \\ &= \prod_{i=1}^{i=n}\left(\frac{e^{x_i\beta}}{1+e^{x_i\beta}}\right)^{Y_i}\left(\frac{1}{1+e^{x_i\beta}}\right)^{1-Y_i} \\ &= \prod_{i=1}^{i=n}\left(e^{x_i\beta}\right)^{Y_i}\left(\frac{1}{1+e^{x_i\beta}}\right) \end{aligned}$$

Après passage à la log-vraisemblance, on arrive à l'expression

$$l(Y,\beta)=\sum_{i=1}^{i=n}\left(Y_i x_i\cdot\beta-\ln\left(1+e^{x_i\beta}\right)\right)$$

Le calcul de l'EMV se fait en annulant les dérivées partielles par rapport aux $\beta_j, 0\leq j\leq p$. Remarquons qu'on peut écrire $Y_i x_i\cdot\beta$ sous la forme $\sum_{k=0}^{k=p} Y_i x_{ik}\beta_k$. Cette dernière sera utile pour le calcul des dérivées partielles. Le vecteur gradient au point β défini par $\nabla\mathcal{L}_n(\beta)=\left[\frac{\partial\mathcal{L}_n}{\partial\beta_0}(\beta),\cdots,\frac{\partial\mathcal{L}_n}{\partial\beta_p}(\beta)\right]^{'}$ s'obtient par dérivation

$$\begin{aligned} \frac{\partial\mathcal{L}_n}{\partial\beta_j}(\beta) &= \sum_{i=1}^{i=n}\left[Y_i x_{ij}-\frac{x_{ij}e^{x_i\cdot\beta}}{1+e^{x_i\cdot\beta}}\right] \\ &= \sum_{i=1}^{i=n} x_{ij}\left[Y_i-\frac{e^{x_i\cdot\beta}}{1+e^{x_i\cdot\beta}}\right] \\ &= \sum_{i=1}^{i=n} x_{ij}\left(Y_i-p_\beta(x_i)\right) \end{aligned}$$

Ce qui donne en écriture matricielle

$$\begin{aligned} \nabla\mathcal{L}_n(\beta) &= \sum_{i=1}^{i=n} x_i\left(Y_i-p_\beta(x_i)\right) \\ &= X^{'}(Y-p_\beta(X)) \end{aligned}$$

L'estimateur du maximum de vraisemblance (s'il existe) est solution de l'équation (appelée équation du score) :

$$S(\beta) \;=\; \nabla\mathcal{L}_n(\beta) \;=\; X^{'}(Y-p_\beta(X)) \;=\; 0$$

On rappelle que si cette équation admet une solution en β alors l'estimateur du maximum de vraisemblance est $\widehat{\beta}=g(Y_1,\cdots,Y_n)$. Cela ne peut se résoudre explicitement et nécessite l'emploi de méthodes numériques. Ce qui explique que l'on obtienne parfois

des résultats différents d'un logiciel à l'autre : le résultat obtenu dépend de l'algorithme utilisé, du paramétrage adopté, et parfois même des choix d'implémentation de l'informaticien. Toutefois, l'existence et l'unicité de l'EMV sont assurées sous les hypothèses d'identifiabilité (voir plus haut) et de recouvrement.

Puisque $\widehat{\beta}$ est un estimateur du maximum de vraisemblance, il en possède toutes les propriétés :

— Il est asymptotiquement sans biais ;
— Il est de variance minimale ;
— Il est asymptotiquement gaussien.

Ces éléments, notamment le dernier, sont très importants pour l'inférence statistique (intervalle de confiance, test de significativité, etc.)

Exemple 2.7 *Prévision de l'occurrence d'une maladie cardiaque*

Pour illustrer nos propos, on souhaite déterminer la probabilité d'avoir une maladie du coeur compte tenu des descripteurs relevés.

$$chd = f(sbp, tobacco, age, \dots) \tag{2.11}$$

```
# Creation de formule
def create_formula(y=str,x=list[str]):
  return y + ' ~ ' + ' + '.join(x)
## Estimation par Maximum de vraisemblance
import statsmodels.formula.api as smf
formule = create_formula("chd",donnee.drop(['chd'],axis=1).columns)
fullmodel = smf.logit(formule,data=donnee).fit()
```

```
## Optimization terminated successfully.
##          Current function value: 0.510974
##          Iterations 6
```

Par défaut, l'outil s'appuie sur l'algorithme de Newton-Raphson. D'autres procédures d'optimisation sont disponibles. 6 itérations ont été nécessaires pour maximiser la log-vraisemblance.

```
# Tableau des coeffcients
coef_table = fullmodel.summary2().tables[1]
```

Le logit estimé permettant de prédire l'occurance d'une maladie cardiaque à partir de l'age, etc., s'écrit :

$$\text{logit}\left(p_{\widehat{\beta}}(X)\right) = -6.1507 + 0.0065 \times \text{sbp} + 0.0794 \times \text{tobacco} + \dots + 0.0452 \times \text{age} \tag{2.12}$$

Table 2.3 – Estimation du modèle $chd = f(sbp, tobacco, age, ...)$

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	-6.1507	1.3083	-4.7015	0.0000	-8.7149	-3.5866
famhist[T.Present]	0.9254	0.2279	4.0605	0.0000	0.4787	1.3720
sbp	0.0065	0.0057	1.1350	0.2564	-0.0047	0.0177
tobacco	0.0794	0.0266	2.9838	0.0028	0.0272	0.1315
ldl	0.1739	0.0597	2.9152	0.0036	0.0570	0.2909
adiposity	0.0186	0.0293	0.6346	0.5257	-0.0388	0.0760
typea	0.0396	0.0123	3.2138	0.0013	0.0154	0.0637
obesity	-0.0629	0.0442	-1.4218	0.1551	-0.1496	0.0238
alcohol	0.0001	0.0045	0.0271	0.9784	-0.0087	0.0089
age	0.0452	0.0121	3.7285	0.0002	0.0215	0.0690

2.2.2 Dimensions explicatives, variables explicatives

2.2.2.1 Variable explicative quantitative

Si on dispose d'une seule variable explicative X quantitative (non regroupés en classe) le modèle s'écrit :

$$\text{logit}\left(p_\beta(x)\right) = \beta_0 + \beta_1 x \tag{2.13}$$

Un seul coefficient (β_1) est alloué à X, cette variable est représentée par une seule colonne dans la matrice du design X. Sa dimension est donc égale à 1.

Exemple 2.8 *Nous souhaitons expliquer la variable Y présence (1)/absence (0) d'une maladie cardio-vasculaire (chd) par l'âge des patients. Les données sont représentées sur la figure 2.1*

```
# chd en fonction de l'âge
from plotnine import *
print((ggplot(donnee, aes(x="age",y="chd"))+geom_point()))
```

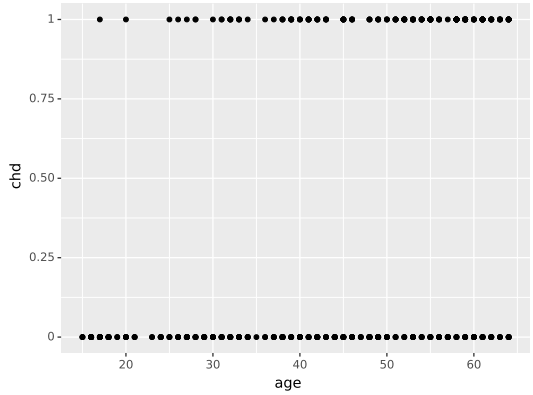


Figure 2.1 – Représentation directe de chd (Variable à expliquer Y) en fonction de l'âge (variable explicative X.

Cette figure montre qu'il est difficile de modéliser les données brutes, la variabilité de la variable chd est élevée pour tout âge. On ajuste le modèle :

$$\log\left(\frac{\mathbb{P}\left(\text{chd}=1|\text{age}\right)}{\mathbb{P}\left(\text{chd}=0|\text{age}\right)}\right)=\beta_0+\beta_1\text{age}$$

```
# on ajuste le modèle : chd ~ age
import statsmodels.formula.api as smf
age_model = smf.logit("chd~age",data=donnee).fit(disp=False)
coef_age = age_model.summary2().tables[1]
```

Table 2.4 – Estimation du modèle $chd = f(age)$

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	-3.5217	0.4160	-8.4650	0	-4.3371	-2.7063
age	0.0641	0.0085	7.5135	0	0.0474	0.0808

Le modèle ajusté s'écrit :

$$\log\left(\frac{\hat{\mathbb{P}}\left(\text{chd}=1|\text{age}\right)}{\hat{\mathbb{P}}\left(\text{chd}=0|\text{age}\right)}\right)=-3.5217+0.0641\times\text{age}$$

2.2.2.2 Variable explicative qualitative

Tout comme pour le modèle d'analyse de variance, une variable qualitative est représentée par les indicatrices associées aux différentes modalités. Considérons un modèle où la seule variable explicative est `famhist` :

$$\text{logit}p_\beta(x)=\beta_0+\beta_A 1_A(x)+\beta_P 1_P(x) \tag{2.14}$$

Mais aussi

$$\text{logit}p_\beta(x)=(\beta_0+\beta_A)+(\beta_P-\beta_A)1_P(x)$$

Exemple 2.9 *Considérons le cas où `famhist` explique `chd`*

On effectue une régression logistique sur Python :

```
# on ajuste le modèle : chd ~ c(famhist)
famhist_model = smf.logit("chd~C(famhist)",data=donnee).fit(disp=False)
coef_famhist = famhist_model.summary2().tables[1]
```

Table 2.5 – Estimation du modèle $chd = f(famhist)$

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	-1.169	0.1431	-8.1687	0	-1.4495	-0.8885
C(famhist)[T.Present]	1.169	0.2033	5.7514	0	0.7706	1.5674

Le modèle estimé s'écrit donc :

$$\text{logit}\hat{p}(famhist)=\text{logit}p_{\hat{\beta}}(famhist)=-1.1690+1.1690\times 1_{\text{famhist=Present}}$$

2.2.3 Interprétation des coefficients

Dans certains domaines, l'explication est bien plus importante que la prédiction. On souhaite comprendre les phénomènes de causalité, mettre à jour les relations de cause à effet. Sur la figure 2.2, nous avons représenté l'allure de la courbe représentative de la fonction $x \mapsto \dfrac{\exp(x\beta)}{1+\exp(x\beta)}$ pour différentes valeurs du paramètre β.

```
# Calcul des valeurs
x = np.linspace(-5,5,50)
def f(t,beta):
    return((np.exp(t*beta))/(1+np.exp(t*beta)))
# Représentation graphique
fig = plt.figure(figsize=(16,8))
for i,b,t in zip(range(4),[0,0.5,2,10],[[0.3,0.7],[0.2,0.8],[0.0,1.0],[0.0,1.0]]):
  ax = fig.add_subplot(2,2,i+1)
  ax.plot(f(x,b));
  ax.set(ylim=[-0.1,1.1],title=r"$\beta$ = "+str(b),xticks=[],yticks=t,
         yticklabels=[str(x) for x in t]);
  ax.grid(True);
plt.tight_layout();
plt.show()
```

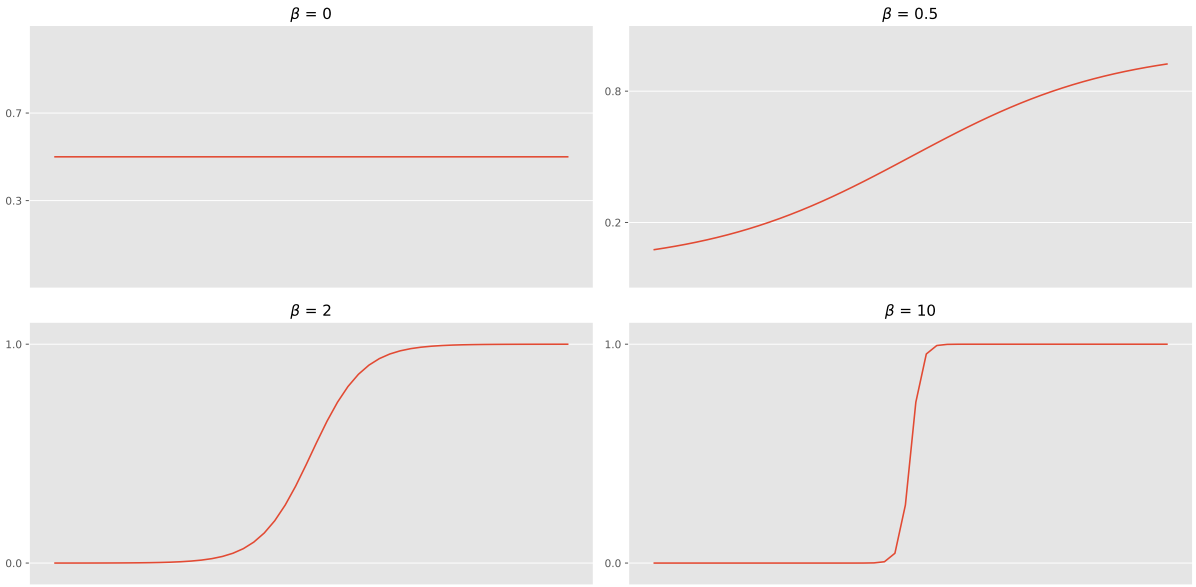


Figure 2.2 – $\mathbb{P}_\beta(Y=1|X=x)$ pour différentes valeurs de β.

On remarque que :

— pour de faibles valeurs de β, on a une large plage de valeurs de x pour lesquelles la fonction se situe aux alentours de 0.5(la fonction est même constante (0.5) dans le cas extrême $\beta = 0$). Pour ces valeurs $p_\beta(x) = \mathbb{P}_\beta(Y=1|X=x)$ sera proche de 0.5 et on peut donc penser qu'il sera difficile de discriminer ;

— Lorsque β augmente, la zone où la fonction est proche de 0.5 diminue et la fonction est proche de 0 ou de 1 pour un grand nombre de valeurs de x. Par conséquent, $\mathbb{P}_\beta(Y=1|X=x)$ sera souvent proche de 1 ou 0, ce qui risque de minimiser d'éventuelles erreurs de prévisions.

On peut interpréter ainsi : *plus β est grand, mieux on discrimine*. Cependant, une telle interprétation dépend des valeurs que x prend (de son échelle). C'est pourquoi en général l'interprétation des coefficients β s'effectue en termes d'*odds ratio*.Les odds ratio sont des outils souvent appréciés dans le domaine de l'épidémiologie (mais pas toujours bien utilisés !).

Les odds ratio servent à mesurer l'effet d'une variable quantitative ou le contraste entre les effets d'une variables qualitative. L'idée générale est de raisonner en termes de probabilités ou de rapport de cotes (odds). Si on a, par exemple, une probabilité $p = 1/4$ de gagner à un jeu, cela signifie que sur 4 personnes une gagne et les trois autres perdent, soit un rapport de 1 gagnant sur trois perdants, c'est-à-dire $p/(1-p) = 1/3$. Ce rapport $p/(1-p)$ varie entre 0 (0 gagnant) et l'infini (que des gagnants) en passant par 1 (un gagnant pour un perdant).

2.2.4 Risque relatif, odds, odds ratio

Soit le tableau de contingence suivant mettant en liaison la variable cible Y binaire et une variable explicative catégorielle (binaire) :

Table 2.6 – Tableau de contingence - Croisement Y vs X

Y \ X	0	1	Total
0	a	b	a+b
1	c	d	c+d
Total	a+c	b+d	n

2.2.4.1 Risque relatif

Définition 2.2 (Risque relatif) *On appelle **risque relatif**, le surcroît de chances d'être positif du groupe exposé par rapport au groupe témoin.*

$$RR = \frac{\mathbb{P}(Y = 1/X = 1)}{\mathbb{P}(Y = 1/X = 0)} = \frac{\frac{d}{b+d}}{\frac{c}{a+c}} \tag{2.15}$$

Pour illustrer nos propos, nous considérons les données du tableau 2.1. Ce tableau croise la variable dépendante `chd` (avoir une maladie cardiaque ou pas) avec la variable explicative `famhist` (antécédents familiaux).

```
# Croisement chd vs famhist
contingence = crosstab.copy()
contingence.loc['Total'] = contingence.sum(axis=0)
contingence.loc[:,'Total'] = contingence.sum(axis=1)
tab = contingence.reset_index()
```

On calcule le risque relatif :

$$RR = \frac{96/192}{206/270} = 0.66$$

Table 2.7 – chd vs. famhist - Tableau de contingence

chd	Absent	Present	Total
0	206	96	302
1	64	96	160
Total	270	192	462

```
# Risque relatif
num_rr = contingence.values[1,1]/contingence.values[2,1]
deno_rr = contingence.values[0,0]/contingence.values[2,0]
risque_relatif = num_rr/deno_rr
print('Risque relatif : %.2f' %(risque_relatif))
```

```
## Risque relatif : 0.66
```

Les persones qui ont des antécédents familiaux de la maladie ont 0.66 fois plus de chances que les autres (ceux qui n'en ont pas) de développer une maladie cardique. Il Caractérise un lien entre l'apparition de la maladie et l'oocurence des antécédents familiaux. Lorsque $RR = 1$, cela veut dire que les antécédents familiaux n'ont pas d'incidence sur la maladie.

2.2.4.2 Odds

Définition 2.3 (Odds) *L'**odds** ou* rapport de chances *est défini comme un rapport de probabilités dans un groupe. Par exemple, dans le groupe exposé, il s'écrit :*

$$odds(1) = \frac{\mathbb{P}(Y = 1/X = 1)}{\mathbb{P}(Y = 0/X = 1)} = \frac{\frac{d}{b+d}}{\frac{b}{b+d}} = \frac{d}{b} \tag{2.16}$$

Pour nos données, nous avons :

$$\text{odds}(1) = \frac{\mathbb{P}\left(\text{chd} = 1 | \text{famhist} = \text{Present}\right)}{\mathbb{P}\left(\text{chd} = 0 | \text{famhist} = \text{Present}\right)} = \frac{96}{96} = 1$$

```
# odds
odds = contingence.values[1,1]/contingence.values[0,1]
print('odds(1) : %.2f' % (odds))
```

```
## odds(1) : 1.00
```

2.2.4.3 Odds - ratio

Définition 2.4 (Odds-ratio) *L'**odds ratio** est égal au rapport entre l'odds du groupe exposé et l'odds du groupe témoin.*

$$OR = \frac{odds(1)}{odds(0)} = \frac{\frac{d}{b}}{\frac{c}{a}} = \frac{d \times a}{b \times c} \tag{2.17}$$

Pour nos données, nous avons :

$$OR = \frac{96 \times 206}{64 \times 96} = 3.22$$

```
# odds ratio
num_or = contingence.values[1,1]*contingence.values[0,0]
deno_or = contingence.values[1,0]*contingence.values[0,1]
odds_ratio = num_or/deno_or
print('odds ratio : %.2f' % (odds_ratio))
```

```
## odds ratio : 3.22
```

L'odds - ration indique à peu près la même chose que le risque relation, à savoir : dans le groupe des personnes ayant des antécédents familiaux, on a 3.22 fois plus de chances d'avoir la maladie que dans le groupe des personnes n'ayant pas d'antécédents familiaux.

2.2.4.4 Log odds - ratio

Définition 2.5 (log odds-ratio) *Il s'agit simplement du logarithme de l'odds-ratio. Développons son expression, nous verrons ainsi le rapport avec la régression logistique :*

$$\begin{aligned}
\ln(OR) &= \ln\left(\frac{odds(1)}{odds(0)}\right) = \ln(odds(1)) - \ln(odds(0)) \\
&= \ln\left(\frac{\mathbb{P}(Y=1/X=1)}{\mathbb{P}(Y=0/X=1)}\right) - \ln\left(\frac{\mathbb{P}(Y=1/X=0)}{\mathbb{P}(Y=0/X=0)}\right) \\
&= \ln\left(\frac{\mathbb{P}(Y=1/X=1)}{1-\mathbb{P}(Y=1/X=1)}\right) - \ln\left(\frac{\mathbb{P}(Y=1/X=0)}{1-\mathbb{P}(Y=1/X=0)}\right) \\
&= logit(1) - logit(0)
\end{aligned}$$

On constate que le log-odds ratio peut s'interpréter comme un écart entre 2 logit.

2.2.4.5 Cas de la régression simple

On suppose que notre variable cible Y est expliquée par une seule variable explicative X. On a le modèle suivant :

$$\text{logit}\left(p_\beta(X)\right) = \beta_0 + \beta_1 X \tag{2.18}$$

L'interprétation des coefficients dépend du type de la variable explicative X ;

Cas 1 : Variable explicative binaire

Ce cas est en relation directe avec le tableau 2.6. Dans cette configuration, le coefficient β_1 correspond au logarithme de l'odds-ratio calculé à partir du tableau de contingence 2.6. L'idée est relativement simple :

$$\begin{array}{rcccccccc} X & = & 1 & \longrightarrow & \text{logit}(1) & = & \beta_0 + \beta_1 \times 1 & = & \beta_0 + \beta_1 \\ X & = & 0 & \longrightarrow & \text{logit}(0) & = & \beta_0 + \beta_1 \times 0 & = & \beta_0 \end{array}$$

Ce qui implique :

$$\ln(OR) = \text{logit}(1) - \text{logit}(0) = \beta_1 \quad \Longrightarrow \quad OR = e^{\beta_1}$$

En reprenant l'exemple croisant `chd` et `famhist`, l'odds-ratio était égal à 3.22. En utilisant, le modèle de la section 2.2.2.2, nous obtenons $\widehat{\beta}_1 = 1.1690$. En prenant l'exponentielle, nous obtenons :

```
# famhist coefs et odds - ratios
famhist_coefs = pd.DataFrame({
  'name': famhist_model.params.index,'coef':famhist_model.params.values.round(4),
  'odds ratio': np.exp(famhist_model.params.values).round(4)})
```

Table 2.8 – Coefficients du modèle $chd = f(famhist)$

name	coef	odds ratio
Intercept	-1.169	0.3107
C(famhist)[T.Present]	1.169	3.2188

Ainsi, la régression logistique nous permet de mesurer directement le surcroît de risque associé à un facteur explicatif binaire :

— Si $\widehat{\beta}_j < 0 \quad \longrightarrow \quad OR < 1$: il y a une diminution du risque ;
— Si $\widehat{\beta}_j > 0 \quad \longrightarrow \quad OR > 1$: il y a une augmentation du risque ;

Cas 2 : Variable explicative quantitative

Pour comprendre l'interprétation des coefficients dans le cas d'une variable explicative quantitative, voyons l'évolution du `logit` lorsqu'on fait varier X d'une unité.

$$\begin{array}{rcl} \text{logit}(X+1) & = & \beta_0 + \beta_1 \times (X+1) = \beta_0 = \beta_1 X + \beta_1 \\ \text{logit}(X) & = & \beta_0 + \beta_1 X \end{array}$$

Ce qui implique que :

$$\text{logit}(X+1) - \text{logit}(X) = \beta_1 \tag{2.19}$$

Dans ce cas, la quantité e^{β_1} s'interprète comme l'odds ratio consécutif à l'augmentation d'une unité de la variable explicative. Par conséquent, si l'on augmente de b unités la variable explicative, l'odds-ratio devient alors $\exp(b \times \beta_1)$.

En reprenant l'exemple de la section 2.2.2.1 où nous avons essayer de prédire `chd` en fonction de `age`, nous avons obtenu $\widehat{\beta}_1 = 0.0641$ et par conséquent :

```
# age coefs
age_coefs = pd.DataFrame({
  'name': age_model.params.index,'coef': age_model.params.values.round(4),
  'odds ratio': np.exp(age_model.params.values).round(4)})
```

Table 2.9 – odds - ratio du modèle chd=f(age)

name	coef	odds ratio
Intercept	-3.5217	0.0295
age	0.0641	1.0662

Lorsque l'age augmente d'une unité, les individus ont 1.0662 fois plus de chances de développer une maladie cardiaque.

2.2.5 Coefficients standardisés en régression logistique

Lorsque les explicatives sont exclusivement quantitatives, il peut être intéressant de comparer leur impact sur la variable dépendante. Quelle est celle qui joue le rôle le plus important ? Dans quel sens ?.

Comparer les odds-ratio paraît une solution immédiate. Mais comme les explicatives ne sont pas exprimées sur une même échelle, la variation d'une unité n'a absolument pas la même signification d'une variable à l'autre. Les odds ne sont pas comparable en l'état. La solution la plus simple est de centrer et réduire les explicatives. Ainsi, nous pouvons mieux jauger leur influence et, de plus, nous pouvons disposer d'interprétations sous forme de variations d'écarts-type.

Dans cette section, nous souhaitons mettre en place un dispositif qui permet de :

— Comparer les influences respectives des variables explicatives ;
— Mesurer l'impact de la variation d'un écart-type d'une variable explicative sur le logit, soit en termes **absolus** c'est-à-dire écarts absolus entre logit (l'exponentielle de l'écart entre deux logit est un odds-ratio), soit en termes **relatifs**, c'est-à-dire variation en écarts-type du logit.

2.2.5.1 Modélisation avec uniquement les variables explicatives quantitatives

```
# Selection des variables explicatives quantitatives
donnee.famhist = donnee.famhist.astype('category')
Xquant = donnee.drop('chd', axis=1).select_dtypes(np.number)
print(Xquant.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 462 entries, 1 to 463
## Data columns (total 8 columns):
##  #   Column     Non-Null Count  Dtype
## ---  ------     --------------  -----
##  0   sbp        462 non-null    int64
##  1   tobacco    462 non-null    float64
##  2   ldl        462 non-null    float64
##  3   adiposity  462 non-null    float64
##  4   typea      462 non-null    int64
##  5   obesity    462 non-null    float64
##  6   alcohol    462 non-null    float64
##  7   age        462 non-null    int64
```

```
## dtypes: float64(5), int64(3)
## memory usage: 32.5 KB
## None
```

On garde les 8 variables explicatives quantitatives. On entraîne notre modèle

```
# on ajuste le modèle
fromule = create_formula("chd",Xquant.columns)
fullmodel2 = smf.logit(formule,data=donnee).fit(disp=False)
coef2_table = fullmodel2.summary2().tables[1]
```

Table 2.10 – Coefficients du modèle $chd = f(.)$

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	-6.1507	1.3083	-4.7015	0.0000	-8.7149	-3.5866
famhist[T.Present]	0.9254	0.2279	4.0605	0.0000	0.4787	1.3720
sbp	0.0065	0.0057	1.1350	0.2564	-0.0047	0.0177
tobacco	0.0794	0.0266	2.9838	0.0028	0.0272	0.1315
ldl	0.1739	0.0597	2.9152	0.0036	0.0570	0.2909
adiposity	0.0186	0.0293	0.6346	0.5257	-0.0388	0.0760
typea	0.0396	0.0123	3.2138	0.0013	0.0154	0.0637
obesity	-0.0629	0.0442	-1.4218	0.1551	-0.1496	0.0238
alcohol	0.0001	0.0045	0.0271	0.9784	-0.0087	0.0089
age	0.0452	0.0121	3.7285	0.0002	0.0215	0.0690

Cas 1 : Standardisation sur les explicatives seulement

Nous calculons le coefficient standardisé de la manière suivante :

$$\hat{\beta}_j^{\text{std.1}} = \hat{\beta}_j \times \hat{\sigma}_j \tag{2.20}$$

où $\hat{\sigma}_j$ est l'écart type empirique lié à la variable explicative quantitative X_j.

```
# Coefficients standardisés
params_std1 = pd.DataFrame({"Coef." : fullmodel2.params[1:],
                    "ecart-type " : Xquant.std(),
                    "coef. stand." :fullmodel2.params[1:]*Xquant.std(ddof=0)})
```

‘

Table 2.11 – Coefficients standardisés par l'écart - type de la variable

params	Coef.	ecart-type	coef. stand.
adiposity	0.0186	7.7807	0.1445
age	0.0452	14.6090	0.6600
alcohol	0.0001	24.4811	0.0030
famhist[T.Present]	0.9254		
ldl	0.1739	2.0709	0.3598
obesity	-0.0629	4.2137	-0.2648
sbp	0.0065	20.4963	0.1332
tobacco	0.0794	4.5930	0.3642
typea	0.0396	9.8175	0.3883

Plusieurs informations apparaissent :

— Les ceoffcients mesurent la variation absolue du logit consécutive à une augmentation de 1 écart - type des variables :

$$\hat{\beta}_j^{\text{std.1}} = \Delta_{logit}\left(\hat{\sigma}_j\right) \tag{2.21}$$

— Comme nous mesurons l'impact sur le logit des variations en écarts - type des explicatives, nous pouvons comparer leur poids relatif dans la regression. Manifestement, « alcohol » (consommation courante d'alcool) a un impact plus élevé (écart absolu du logit), suivi de « sbp » (pression sanguine systolique).
— Nous ne disposons pas d'informations sur la variation relative.

Cas 2 : Standardisation sur les explicatives et l'écart-type du logit

Une autre standardisation est proposée dans la littérature :

$$\hat{\beta}_j^{std.2} = \hat{\beta}_j \times \frac{\hat{\sigma}_j}{\hat{\sigma}_{\text{logit}}} \tag{2.22}$$

```
# Coefficients standardisés
params_std2 = params_std1.copy()
params_std2["coef. stand."] = (
              params_std2["coef. stand."]/fullmodel2.fittedvalues.std(ddof=0))
```

‘

Table 2.12 – Coefficients standardisés par l'écart - type du logit

params	Coef.	ecart-type	coef. stand.
adiposity	0.0186	7.7807	0.1076
age	0.0452	14.6090	0.4914
alcohol	0.0001	24.4811	0.0022
famhist[T.Present]	0.9254		
ldl	0.1739	2.0709	0.2679
obesity	-0.0629	4.2137	-0.1972
sbp	0.0065	20.4963	0.0992
tobacco	0.0794	4.5930	0.2712
typea	0.0396	9.8175	0.2891

Quelques commentaires :

— Les nouveaux coeficients mesurent la variation relative du logit lorsqu'on augment de 1 écart - type l'explicative c'est - à - dire :

$$\hat{\beta}_j^{\text{std.2}} = \delta_{logit}\left(\hat{\sigma}_j\right) \tag{2.23}$$

— Ils permettent aussi de comparer l'impact des explicatives.
— Enfin, dernier commentaire important, cette standardisation nous fournit directement **les coefficients que l'on aurait obtenu si on avait lancé la régression sur les données centrées réduites**

Cas 3 : Standardisation sur les variables explicatives et l'écart-type théorique de la loi de répartition logistique

La dernière standardisation vaut surtout parce qu'elle est proposée dans le logiciel SAS.

$$\hat{\beta}_j^{std.1} = \hat{\beta}_j \times \frac{\hat{\sigma}_j}{\hat{\sigma}_{\text{théorique}}} \tag{2.24}$$

avec $\sigma_{\text{théorique}}$ est l'écart - type théorique de la loi de distribution logistique standard (*cf.* Scherrer (2007)).

Remarque 2.3 (Loi logistique standard) *La loi logistique standard est la loi logistique de paramètres* 0 *et* 1*. Sa fonction de répartition est la sigmoide :*

$$F(x) = \frac{1}{1 + e^{-x}}$$

Son espérance vaut alors 0 *et sa variance* $\frac{\pi^2}{3}$.

```
# Coefficients standardisés
params_std3 = params_std1.copy()
params_std3["coef. stand."] = (
              params_std1["coef. stand."]/(np.pi/np.sqrt(3)))
```

‘

Table 2.13 – Coefficients standardisés par l'écart - type théorique

params	Coef.	ecart-type	coef. stand.
adiposity	0.0186	7.7807	0.0796
age	0.0452	14.6090	0.3639
alcohol	0.0001	24.4811	0.0016
famhist[T.Present]	0.9254		
ldl	0.1739	2.0709	0.1984
obesity	-0.0629	4.2137	-0.1460
sbp	0.0065	20.4963	0.0734
tobacco	0.0794	4.5930	0.2008
typea	0.0396	9.8175	0.2141

Comme pour toutes les autres standardisation, les coefficients permettent de comparer l'impact des explicatives. Mais elles ne s'interprètent pas en termes de variation du logit.

2.3 Précision des estimateurs

2.3.1 Loi asymptotique des estimateurs

Le comportement asymptotique de l'estimateur du maximum de vraisemblance $\hat{\beta}$ est :

$$\sqrt{n}(\hat{\beta} - \beta) \xrightarrow{\mathcal{L}} \mathcal{N}(0, \mathcal{I}(\beta)^{-1}) \tag{2.25}$$

où $\mathcal{I}$ est la matrice d'information de Fisher au point β. On déduit :

$$\left(\hat{\beta}-\beta\right)^{'} n\mathcal{I}(\beta)(\hat{\beta}-\beta) \xrightarrow{\mathcal{L}} \chi^2_{p+1}$$

Un tel résultat n'est pas utilisable tel quel puisque la matrice $\mathcal{I}(\beta)$ est inconnue. On remarque que d'après la loi des grands nombres

$$\begin{aligned}\hat{\mathcal{I}}_{kl} &= -\frac{1}{n}\sum_{i=1}^{n}\frac{\partial^2}{\partial\beta_k\partial\beta_l}\mathcal{L}_1(Y_i;\beta) = \frac{1}{n}\frac{\partial^2}{\partial\beta_k\partial\beta_l}\sum_{i=1}^{n}\mathcal{L}_1(Y_i;\beta) = \frac{1}{n}\frac{\partial^2}{\partial\beta_k\partial\beta_l}\sum_{i=1}^{n}\mathcal{L}_n(Y_1,\dots,Y_n;\beta)\\ &= \frac{1}{n}(X^{'}W_{\beta}X)_{kl}\end{aligned}$$

converge presque sûrement vers $\mathcal{I}(\beta)_{kl}$.

Comme $\hat{\beta}$ converge faiblement vers β, on obtient grâce au théorème de Slutsky et aux opérations classiques sur la convergence en loi

$$(\hat{\beta}-\beta)^{'}\hat{\Sigma}(\hat{\beta}-\beta) \xrightarrow{\mathcal{L}} \chi^2_{p+1} \tag{2.26}$$

avec $\hat{\Sigma} = (X^{'}W_{\hat{\beta}}X)$

2.3.2 Intervalle de confiance

On déduit du paragraphe précédent qu'un estimateur de la variance de $\hat{\beta}_j$ est donné par le $j^{\text{ème}}$ terme de la diagonale de $\hat{\Sigma}^{-1}$.Notons par $\hat{\sigma}^2_j$ cet estimateur. Il vient :

$$\frac{(\hat{\beta}_j-\beta_j)^2}{\hat{\sigma}^2_j} \xrightarrow{\mathcal{L}} \mathcal{X}^2_1 \quad \text{ou encore} \quad \frac{\hat{\beta}_j-\beta_j}{\hat{\sigma}_j} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1) \tag{2.27}$$

Un intervalle de confiance (asymptotique) de niveau $1-\alpha$ pour β_j est donné par :

$$IC_{1-\alpha}(\beta_j) = \left[\hat{\beta}_j - z_{1-\alpha/2}\times\hat{\sigma}_j; \hat{\beta}_j + z_{1-\alpha/2}\times\hat{\sigma}_j\right] \tag{2.28}$$

où $z_{1-\alpha/2}$ représente le quantile de niveau $1-\dfrac{\alpha}{2}$ de la loi normale $\mathcal{N}(0,1)$.

Pour obtenir des intervalles de confiance à 95% des coefficients de notre modèle global, nous faisons appel à la méthode conf_int() de l'objet résultat fourni par fit().

```
# intervalle de confiance des coefficients à 95%
confint = fullmodel.conf_int(alpha=0.05)
confint.columns = ["[0.025","0.975]"]
```

Remarque 2.4 *La validité de ces intervalles est relative puisqu'il s'agit d'une approximation valable aymptotiquement. Il est toujours possible de compléter cette étude par un bootstrap afin d'obtenir d'autres intervalles de confiance dans le cas ou ceux-ci sont parfaitement importants. Cela dit, en pratique, on se contente de l'intervalle de confiance bâti grâce à la matrice d'information de Fisher.*

Table 2.14 – Intervalle de confiance

	[0.025	0.975]
Intercept	-8.7149	-3.5866
famhist[T.Present]	0.4787	1.3720
sbp	-0.0047	0.0177
tobacco	0.0272	0.1315
ldl	0.0570	0.2909
adiposity	-0.0388	0.0760
typea	0.0154	0.0637
obesity	-0.1496	0.0238
alcohol	-0.0087	0.0089
age	0.0215	0.0690

2.3.3 Test de Student de significativité d'un coefficient

On déduit également de l'équation 2.27 les tests de nullité des coefficients du modèle. On note :

$$H_0 : \beta_j = 0 \quad \text{contre} \quad H_1 : \beta_j \neq 0 \tag{2.29}$$

Alors sous H_0 :

$$\frac{\hat{\beta}_j}{\hat{\sigma}_j} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1) \tag{2.30}$$

On rejettera H_0 si la valeur observée de $\dfrac{\hat{\beta}_j}{\hat{\sigma}_j}$ dépasse en valeur absolue le quantile d'ordre $1-\alpha/2$ de la loi $\mathcal{N}(0,1)$

```
#Test de significativité des paramètres
ttest = pd.concat([fullmodel.tvalues,fullmodel.pvalues],axis=1)
ttest.columns = ['t','P>|t|']
```

Table 2.15 – Test T-student de significativité des paramètres

	t	P>|t|
Intercept	-4.7015	0.0000
famhist[T.Present]	4.0605	0.0000
sbp	1.1350	0.2564
tobacco	2.9838	0.0028
ldl	2.9152	0.0036
adiposity	0.6346	0.5257
typea	3.2138	0.0013
obesity	-1.4218	0.1551
alcohol	0.0271	0.9784
age	3.7285	0.0002

P>|t| est appelée probabilité critique ou « p-value ». Elle représente la probabilité, pour la statistique de test sous H_0, de dépasser la valeur estimée.

```
# Règle
ttest["conclusion"] = np.where(ttest["P>|t|"] < 0.05,"accept","rejet")
```

Table 2.16 – Test T-student de significativité des paramètres avec règle de décision

	t	P>|t|	conclusion
Intercept	-4.7014512	0.0000026	accept
famhist[T.Present]	4.0605297	0.0000490	accept
sbp	1.1350027	0.2563742	rejet
tobacco	2.9837580	0.0028473	accept
ldl	2.9151664	0.0035550	accept
adiposity	0.6345832	0.5257003	rejet
typea	3.2138226	0.0013098	accept
obesity	-1.4217645	0.1550946	rejet
alcohol	0.0271373	0.9783502	rejet
age	3.7284643	0.0001927	accept

2.3.4 Test de nullité de *q* coefficients libres

La théorie du maximum de vraisemblance nous donnant la loi (asymptotique) des estimateurs, il est possible de teste la significativité des variables explicatives. Pour cela, trois tests sont généralement utilisé :

— Le test de Wald ;
— Le test du rapport de vraisemblance ou de la déviance.
— Le test du score.

Les hypothèses s'écrivent :

$$H_0 : \beta_{j1} = \beta_{j2} = \cdots = \beta_{jq} = 0 \quad \text{contre} \quad H_1 : \exists k \in 1, \dots, q : \beta_{jk} \neq 0 \tag{2.31}$$

Pour alléger les notations, nous supposons sans perte de généralité que nous testons la nullité des q premiers coefficients du modèle :

$$H_0 : \beta_0 = \beta_1 = \cdots = \beta_{q-1} = 0 \quad \text{contre} \quad H_1 : \exists k \in 0, \dots, q-1 : \beta_k \neq 0 \tag{2.32}$$

Pour ces 3 tests, on rejette l'hypothèse nulle si la valeur observée de la statistique de test dépasse le quantile d'ordre $1 - \alpha$ de la loi de $\mathcal{X}^2_q$.

2.3.4.1 Test de Wald

Il est basé sur l'équation 2.26. On note $\beta_{0,\dots,q-1}$ le vecteur composé des q premières composantes de β et $\widehat{\Sigma}^{-1}_{0,\dots,q}$ la matrice bloc composée des q premières lignes et colonnes de $\widehat{\Sigma}^{-1}$. Il est facile de voir que sous H_0 :

$$\hat{\beta}'_{0,\dots,q-1} \widehat{\Sigma}^{-1}_{0,\dots,q} \hat{\beta}_{0,\dots,q-1} \xrightarrow{\mathcal{L}} \chi^2_q \tag{2.33}$$

Les coefficients non - significativement différents de 0 au risque 5% - dont la p-value est supérieure à 0.05 - sont ceux des variables : sbp, adiposity, obesity, alcohol. Tout d'abord nous affichons la matrice des variances - covariances des coefficients estimés.

```
# Matrice - covariance des coefficients estimés
cov_params = fullmodel.cov_params()
```

Table 2.17 – Matrice de variance - covariance des coefficients

	Intercept	famhist[T.Present]	sbp	tobacco	ldl	adiposity	typea	obesity	alcohol	age
Intercept	1.7115	-0.0167	-0.0036	-1e-04	-0.0063	0.0120	-0.0088	-0.0277	-1e-04	-0.0057
famhist[T.Present]	-0.0167	0.0519	0.0001	6e-04	-0.0006	0.0001	0.0000	-0.0005	-1e-04	-0.0002
sbp	-0.0036	0.0001	0.0000	0e+00	0.0000	0.0000	0.0000	0.0000	0e+00	0.0000
tobacco	-0.0001	0.0006	0.0000	7e-04	0.0000	0.0000	0.0000	0.0000	0e+00	-0.0001
ldl	-0.0063	-0.0006	0.0000	0e+00	0.0036	-0.0004	0.0000	-0.0001	0e+00	0.0000
adiposity	0.0120	0.0001	0.0000	0e+00	-0.0004	0.0009	0.0000	-0.0010	0e+00	-0.0002
typea	-0.0088	0.0000	0.0000	0e+00	0.0000	0.0000	0.0002	-0.0001	0e+00	0.0000
obesity	-0.0277	-0.0005	0.0000	0e+00	-0.0001	-0.0010	-0.0001	0.0020	0e+00	0.0002
alcohol	-0.0001	-0.0001	0.0000	0e+00	0.0000	0.0000	0.0000	0.0000	0e+00	0.0000
age	-0.0057	-0.0002	0.0000	-1e-04	0.0000	-0.0002	0.0000	0.0002	0e+00	0.0001

Nous affichons ensuite les écarts-types des coefficients estimés.

```
# Ecarts-types des coefficients estimés
std_params = np.sqrt(np.diag(cov_params))
print(std_params)
```

```
## [1.30826006 0.22789401 0.0057304  0.02660284 0.05966174 0.02928941
##  0.01232023 0.04424774 0.00448322 0.01212975]
```

Statsmodels renvoie les écarts-types des coefficients estimés grâce à l'outil bse.

```
# Ecart-type des coefficients estimés
params_std = fullmodel.bse.to_frame("std.")
```

Table 2.18 – Ecarts - type des coefficients

	std.
Intercept	1.3083
famhist[T.Present]	0.2279
sbp	0.0057
tobacco	0.0266
ldl	0.0597
adiposity	0.0293
typea	0.0123
obesity	0.0442
alcohol	0.0045
age	0.0121

Nous récupérons ensuite la sous-partie de la matrice de variance-covariance qui nous concerne :

```
# indice des coefficients concernés
index = [1,4,7,8]
# Sous-matrice
cov_subparams = cov_params.iloc[index,index]
```

Nous inversons cette sous-matrice :

```
# Inversion de cette matrice
inv_cov_subparams = np.linalg.inv(cov_subparams)
```

Nous récupérons également le sous-vecteur des coefficients estimés :

Table 2.19 – Matrice de variance - covariance des coefficients (sous matrice)

	famhist[T.Present]	ldl	obesity	alcohol
famhist[T.Present]	0.0519	-0.0006	-5e-04	-1e-04
ldl	-0.0006	0.0036	-1e-04	0e+00
obesity	-0.0005	-0.0001	2e-03	0e+00
alcohol	-0.0001	0.0000	0e+00	0e+00

Table 2.20 – Inverse de la matrice de variance - covariance des coefficients (sous matrice)

	famhist[T.Present]	ldl	obesity	alcohol
famhist[T.Present]	19.4263	2.8836	4.4244	67.9350
ldl	2.8836	284.2508	13.4756	-354.1309
obesity	4.4244	13.4756	513.1211	-197.2982
alcohol	67.9350	-354.1309	-197.2982	50533.4052

```
# Coefficients estimés
sub_params = fullmodel.params[index]
```

Table 2.21 – Paramètres à tester

famhist[T.Present]	ldl	obesity	alcohol
0.9254	0.1739	-0.0629	1e-04

Nous calculons la forme quadratique correspondant à la statistique de test ainsi que le p-value :

```
# Statistique de test (Forme quadratique)
import scipy.stats as st
wald_stat = np.dot(sub_params, np.dot(inv_cov_subparams, sub_params))
pvalue = 1.0 - st.chi2.cdf(wald_stat, len(index))
wald_test = pd.DataFrame({"statistic" : wald_stat, "df" : len(index),
                          "pvalue" : pvalue},index=["Wald test"])
```

Table 2.22 – Test de Wald

	statistic	df	pvalue
Wald test	27.3864	4	0

Statsmodels propose l'outil wald_test() pour réaliser les tests généralisés. L'enjeu réside alors dans la définition de la matrice **R** permettant de spécifier les coefficients à tester :

— sbd est en position 1 ;
— adiposity en position 4 ;
— obesity est en position 7 ;
— alcohol est en position 8.

```
# Matrice des coefficients à tester
R = np.array([[0,1,0,0,0,0,0,0,0,0],
              [0,0,0,0,1,0,0,0,0,0],
```

```
              [0,0,0,0,0,0,0,1,0,0],
              [0,0,0,0,0,0,0,0,1,0]])
# Wald test
wald = fullmodel.wald_test(R)
wald_test = pd.DataFrame({"statistic" : float(wald.statistic),"df" : wald.df_denom,
                          "pvalue": float(wald.pvalue)},index=["Wald test"])
```

Table 2.23 – Test de Wald

	statistic	df	pvalue
Wald test	27.3864	4	0

Les résultats sont complètement cohérents.

2.3.4.2 Test du rapport de vraisemblance ou de la déviance

La statistique de test est basée sur la différence des rapports de vraisemblance entre le modèle complet et le modèle sous H_0. On note $\hat{\beta}_{H_0}$ l'estimateur du maximum de vraisemblance contraint par H_0 (il s'obtient en supprimant les q premières variables du modèle). On a alors sous H_0 :

$$2\left(\mathcal{L}_n(\hat{\beta}) - \mathcal{L}_n(\hat{\beta}_{H_0})\right) \xrightarrow{\mathcal{L}} \chi^2_q \tag{2.34}$$

Exemple 2.10 *Application*

On définit une nouvelle matrice X sans les variables sbp, adiposity, obesity et alcohol.

```
# Matrice X sans les variables sbp, adiposity, obesity et alcohol
Xbis = (donnee.drop("chd",axis=1)
              .drop(['sbp','adiposity','obesity','alcohol'],axis=1))
print(Xbis.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 462 entries, 1 to 463
## Data columns (total 5 columns):
##  #   Column   Non-Null Count  Dtype
## ---  ------   --------------  -----
##  0   tobacco  462 non-null    float64
##  1   ldl      462 non-null    float64
##  2   famhist  462 non-null    category
##  3   typea    462 non-null    int64
##  4   age      462 non-null    int64
## dtypes: category(1), float64(2), int64(2)
## memory usage: 18.6 KB
## None
```

Nous réalisons ensuite la régression sans les 4 variables.

```
# Régression sans les 4 variables
# on ajuste le modèle
formule = create_formula("chd",Xbis.columns)
fullmodel3 = smf.logit(formule,data=donnee).fit(disp=False)
coef3_table = fullmodel3.summary2().tables[1]
```

Table 2.24 – Coefficients du modèle

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	-6.4464	0.9209	-7.0004	0.0000	-8.2513	-4.6416
famhist[T.Present]	0.9082	0.2258	4.0228	0.0001	0.4657	1.3507
tobacco	0.0804	0.0259	3.1057	0.0019	0.0297	0.1311
ldl	0.1620	0.0550	2.9470	0.0032	0.0543	0.2697
typea	0.0371	0.0122	3.0505	0.0023	0.0133	0.0610
age	0.0505	0.0102	4.9442	0.0000	0.0305	0.0705

Nous pouvons par la suite réaliser le test en calculant la statistique du rapport de vraisemblance.

```
# Statistique de test
deviance_test = 2*(fullmodel.llf - fullmodel3.llf)
# Degré de liberté
ddl = fullmodel3.df_resid - fullmodel.df_resid
# p-value associée
pvalue = 1 - st.chi2.cdf(deviance_test, ddl)
lrtest = pd.DataFrame({"statistic": deviance_test,"df" : ddl,"pvalue":pvalue},
                      index=["likelihood ratio test"])
```

Table 2.25 – Test du rapport de vraisemblance

	statistic	df	pvalue
likelihood ratio test	3.5455	4	0.471

Nous pouvons également confronter des déviances du modèle complet et du null modèle (modèle trival composé exclusivement de la constante). Statsmodels renvoie llr pour la statistique du rapport de vraisemblance et llr_pvalue.

```
# Test du rapport de vraisemblance : full model vs. null model
lr_test = pd.DataFrame({"statistic":fullmodel.llr,"pvalue":fullmodel.llr_pvalue},
                       index=["LR test"])
```

Table 2.26 – Test du rapport de vraisemblance - full model vs. null model

	statistic	pvalue
LR test	123.9684	0

Au risque 5%, nous pouvons rejeter l'hypothèse de nullité des coefficients.

2.3.4.3 Test du score

On cherche ici à vérifier si la fonction de score (gradient de la log-vraisemblance) est proche de 0 sous H_0. Sous H_0, on a :

$$S(\hat{\beta}_{H_0})' \hat{\Sigma}_{H_0}^{-1} S(\hat{\beta}_{H_0}) \xrightarrow{\mathcal{L}} \chi_q^2$$

où $\widehat{\Sigma}_{H_0} = X' W_{\widehat{\beta}_{H_0}} X$.

On rejette l'hypothèse nulle si la valeur observée de la statistique de test dépasse le quantile d'ordre $1-\alpha$ de la loi de χ_q^2.

Exemple 2.11 *Un chef d'entreprise souhaite vérifier la qualité de ces machines en fonction de l'âge et de la marque des moteurs. Il dispose :*

- *d'une variable binaire* `Y` *(*1 *si le moteur a déjà connu une panne,* 0 *sinon)*
- *d'une variable quantitative* `age` *représentant l'âge du moteur ;*
- *d'une variable qualitaive à* 3 *modalités* `marque` *représentant la marque du moteur.*

On souhaite expliquer la variable Y *à partir des deux autres variables. On construit un modèle logistique permettant d'expliquer* Y *par l'âge du moteur et sa marque. Notre modèle s'écrit :*

$$logit\left(p_\beta(x)\right) = \beta_0 + \beta_1 age + \beta_2 1_{marque=1} + \beta_3 1_{marque=3} \tag{2.35}$$

```
# Chargement des données
import os
import xlrd
workbook = xlrd.open_workbook("./donnee/panne.xls",logfile=open(os.devnull,"w"))
panne = pd.read_excel(workbook)
model = smf.logit("panne~age+C(marque)",data=panne).fit(disp=False)
# Tableau des coeffcients
coef_model = model.summary2().tables[1]
```

Table 2.27 – Estimation du modèle $Y = f(age, marque)$

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	0.4781	0.8330	0.5739	0.5660	-1.1546	2.1108
C(marque)[T.1]	-0.4194	0.8143	-0.5151	0.6065	-2.0154	1.1765
C(marque)[T.3]	-1.4561	1.0536	-1.3820	0.1670	-3.5211	0.6089
age	0.0139	0.0940	0.1477	0.8826	-0.1703	0.1981

La modalité 0 de la variable `marque` est prise comme modalité de référence. On effectue le test du score pour :

$$H_0 : \beta_1 = \beta_2 = \beta_3 = 0 \quad \text{versus} \quad H_1 : \exists j \in \{1,2,3\}, \beta_j \neq 0$$

Le calcul de la statistique de test et la probabilité critique s'obtiennent avec les commandes :

```
# Test du score
modelH0 = smf.logit("panne~1",data=panne).fit(disp=False)
prevH0 = modelH0.predict()
X2 = panne.age.astype("float")
X3 = pd.get_dummies(panne.marque,drop_first=True)
```

```
X = np.c_[np.tile(1,panne.shape[0]),X2,X3]
scoreH0 = X.T.dot(panne.panne.astype("float") - prevH0)
WH0 = np.diag(prevH0*(1-prevH0))
SigmaH0 = X.T.dot(WH0).dot(X)
statscore = scoreH0.T.dot(np.linalg.inv(SigmaH0)).dot(scoreH0)
pcscore = st.chi2.sf(statscore,3)
score_test = pd.DataFrame({"statistic" : statscore,"df":3,"pvalue":pcscore},
                          index=["Score test"])
```

Table 2.28 – Test du score - full model vs. null model - Donnee Panne

	statistic	df	pvalue
Score test	2.163	3	0.5393

Le test du score accepte l'hypothèse nulle.

La figure 2.3 permet de visualiser les trois tests. Le test du score revient à tester la nullité de la pente en $\hat{\beta}_{H_0}$ ($\hat{\beta}$ sous H_0), Le test de Wald la nullité de la distance entre $\hat{\beta}$ et $\hat{\beta}_{H_0}$ et Le test du rapport de vraisemblance la nullité de la différence entre les vraisemblances en ces deux points.

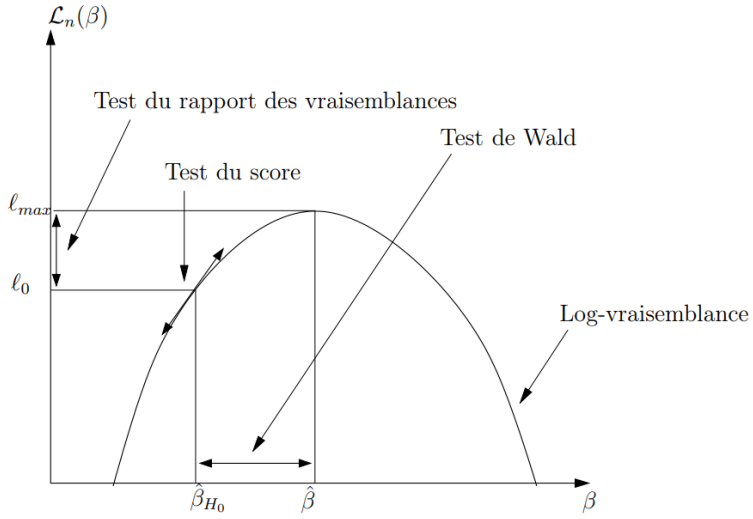


Figure 2.3 – Rapport de vraisemblance, score, test de Wald

2.4 Les pseudo-R^2

Une question cruciale est de pouvoir déterminer si le modèle obtenu est « intéressant » ou non.

2.4.1 Estimation de la constante et de la déviance du modèle trivial

Rappelons que dans le cadre binaire, pour un individu donné, sa probabilité a priori d'être positif s'écrit $\mathbb{P}[Y(\omega) = +] = \pi(\omega)$. Lorsqu'il ne peut y avoir d'ambiguïtés, nous la noterons simple π.

Lorsque l'échantillon est issu d'un tirage aléatoire dans la population, sans distinction des classes d'appartenance, si n_+ est le nombre d'observations positives dans Ω, π peut être estimée par $\frac{n_+}{n_-}$.On parle alors de **schéma de mélange**.

On rappelle également que la **probabilité a posteriori** d'un individu ω d'être positif c'est-à-dire sachant les valeurs prises par les descripteurs est notée $\mathbb{P}\left[Y(\omega) = +/X(\omega)\right] = p(\omega)$. Sans confusion, nous avons noté p.

Le modèle trivial est réduit à la seule constante c'est-à-dire :

$$\text{logit}(M_0) = \ln\left(\frac{p}{1-p}\right) = \beta_0 \tag{2.36}$$

Nous ne tenons pas compte des variables explicativesX_j. De fait :

$$\frac{p}{1-p} = \frac{\pi}{1-\pi} \times \frac{\mathbb{P}(X/Y=+)}{\mathbb{P}(X/Y=-)} = \frac{\pi}{1-\pi} \tag{2.37}$$

On devine aisément **l'estimateur** $\hat{\beta}_0$ de la régression :

$$\hat{\beta}_0 = \ln\left(\frac{\hat{\pi}}{1-\hat{\pi}}\right) = \ln\left(\frac{n_+}{n_-}\right) \tag{2.38}$$

Le nombre de positifs n_+ et négatifs n_- dans l'échantillon suffit pour estimer le paramètre du modèle trivial. Pour prédire la probabilité a posteriori pour un individu d'être positif $\hat{p}(\omega)$, nous utilisons simplement la proportion des positifs $\hat{\pi} = \frac{n_+}{n_-}$ dans la base, soit :

$$\hat{p}(\omega) = \hat{\pi}, \quad \forall\omega \tag{2.39}$$

Réécrivons la log-vraisemblance du modèle trivial, nous obtenons :

$$\begin{aligned}
\mathcal{L}_0 &= \sum_{\omega} \left[Y \times \ln(\hat{\pi}) + (1-Y) \times \ln(1-\hat{\pi})\right] \\
&= \sum_{\omega} Y \times \ln(\hat{\pi}) + \sum_{\omega} (1-Y) \times \ln(1-\hat{\pi}) \\
&= n_+ \times \ln(\hat{\pi}) + n_- \times \ln(1-\hat{\pi}) \\
&= n \times \ln(1-\hat{\pi}) + n_+ \ln\left(\frac{\hat{\pi}}{1-\hat{\pi}}\right)
\end{aligned}$$

Nous pouvons en déduire la `déviance` du modèle trivial :

$$D_0 = -2 \times \mathcal{L}_0 \tag{2.40}$$

2.4.1.1 Estimation directe

Reprenons l'exemple de notre base de données :

```
# Effectifs théoriques
print(donnee.chd.value_counts())
```

```
## chd
## 0    302
## 1    160
## Name: count, dtype: int64
```

Nous y retrouvons $n_+ = 160$ observations positives parmi $n = 462$. Nous obtenons directement :

- Le nombre de négatifs $n_- = n - n_+ = 302$
- La proportion de positifs $\hat{\pi} = \dfrac{160}{462} = 0.3463$
- L'estimation de la constante $\beta_0 = \ln\left(\dfrac{n_+}{n_-}\right) = \ln\left(\dfrac{160}{302}\right) = -0.6353$
- La log-vraisemblance $\mathcal{L}_0 = 462 \times \ln(1-0.3463) + 160 \times \ln\left(\dfrac{0.3463}{1-0.3463}\right) = -298.05$
- La déviance $D_0 = -2 \times \mathcal{L}_0 = -2 \times (-298.05) = 596.1$

2.4.1.2 Estimation usuelle

Par curiosité, nous souhaitons vérifier si les résultats de l'estimation directe concordent avec ceux de la procédure usuelle en utilisant Python.

```
# Importation de la classe de calcul
import statsmodels.formula.api as smf
# on ajuste le modèle
nullmodel = smf.logit("chd~1",data=donnee).fit(disp=False)
coef_null = nullmodel.summary2().tables[1]
```

Table 2.29 – Coefficients du modèle null

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	-0.6353	0.0978	-6.4966	0	-0.8269	-0.4436

Les résultats sont totalement cohérents avec l'approche directe : l'estimation $\hat{\beta}_0$ = -0.6353 et la log-vraisemblance $\mathcal{L}_0$ = -298.05. Ce qui est plutôt encourageant. Le calcul direct nous épargne une optimisation compliquée.

2.4.2 Quelques pseudo-R^2

Les pseudo-R^2 sont des indicateurs similaires dans l'esprit au R^2 de la régression linéaire. Ils sont basés sur la confrontation entre la log-vraisemblance du modèle et de celle du modèle trivial composé uniquement de la constante (`null model`). *Grosso modo*, il s'agit de varifier si notre modèle fait mieux que le modèle trivial c'est-à-dire s'il présente une vraisemblance ou une log-vraisemblance plus favorable.

L'analogie avec le coefficient de détermination R^2 de la régression linéaire multiple est tout à fait intéressante. En effet, il est usuellement interprété comme `la part de la variance expliquée par la modèle`. Mais il peut être également compris comme une confrontation entre les performances du modèle analysé (traduite par la somme des carrés des résidus $SCR = \sum_{\omega}(Y - \widehat{Y})^2$) et celles du modèle par défaut réduite à la simple constante (dans ce cas, la constante est estimée par la moyenne de l'endogène $\bar{Y}$, la somme des carrés totaux correspond donc à la somme des carrés des résidus du modèle réduit à la simple constante $SCT = \sum_{\omega}(Y - \overline{Y})^2$. N'oublions pas que $R^2 = \dfrac{SCT - SCR}{SCT} = 1 - \dfrac{SCR}{SCT}$. Sa définition répond exactement à la notion d'efficacité prédictive. Plusieurs formes de pseudo-R^2 sont proposés dans la littérature.

2.4.2.1 R^2 de McFadden

Sa formule est donnée par :

$$R^2_{MF} = 1 - \frac{\mathcal{L}_n}{\mathcal{L}_0} \tag{2.41}$$

où $\mathcal{L}_n$ et $\mathcal{L}_0$ représentent respectivement la log-vraisemblance du modèle étudié et du modèle trivial.

Remarque 2.5 — $R^2_{MF} = 0$ *si* $\mathcal{L}_n = \mathcal{L}_0$*, on ne fait pas mieux que le modèle trivial ;*
— $R^2_{MF} = 1$ *si* $\mathcal{L}_n = 0$*, notre modèle est parfait ;*
— *L'analogie avec le* R^2 *de la régression linéaire multiple est totale.*

```
# Log - vraisemblance des modèles
llf = pd.DataFrame({"Modèle complet" : fullmodel.llf,
                    "Modèle nul" : fullmodel.llnull},index=["Log vraisemblance"])
```

Table 2.30 – Log - vraisemblance

	Modèle complet	Modèle nul
Log vraisemblance	-236.07	-298.0542

On applique la formule de calcul :

```
#R2 de McFadden
r2Mcfadden = 1 - fullmodel.llf / fullmodel.llnull
print("R2 de McFadden : %.4f" % (r2Mcfadden))
```

```
## R2 de McFadden : 0.2080
```

L'outil fourni directement la valeur du R^2 de McFadden.

```
#qui est founi directement par l'outil
print("R2 de McFadden : %.4f" % (fullmodel.prsquared))
```

```
## R2 de McFadden : 0.2080
```

2.4.2.2 R^2 de Cox and Snell

Sa formule de calcul est donnée par :

$$R^2_{CS} = 1 - \left(\frac{L_0}{L_n}\right)^{\frac{2}{n}} \tag{2.42}$$

où L_n et L_0 représentent respectivement la vraisemblance du modèle étudié et du modèle trivial.Sa valeur minimale est égale à 0 et sa valeur maximale est atteinte lorsque L_n vaut 1, c'est-à-dire $\max\left[R^2_{CS}\right] = 1 - L_0^{\frac{2}{n}}$. L'indicateur n'est pas normalisé, ce qui est un peu gênant.

```
# Vraisemblance du modèle null
L0 = np.exp(fullmodel.llnull)
# Vraisemblance du modèle étudié
Ln = np.exp(fullmodel.llf)
# R2 de Cox and Snell
r2CoxSnell = 1.0 - (L0/Ln)**(2.0/donnee.shape[0])
print("R2 de Cox - Snell : %.4f" % (r2CoxSnell))
```

```
## R2 de Cox - Snell : 0.2353
```

2.4.2.3 R^2 de Nagelkerke

Sa formule de calcul est :

$$R^2_N = \frac{R^2_{CS}}{\max\left[R^2_{CS}\right]} = \frac{R^2_{CS}}{1 - L_0^{\frac{2}{n}}} \tag{2.43}$$

C'est un indicateur compris entre 0 et 1. Le R^2 de Nagelkerke est une simple normalisation du R^2 de Cox and Snell.

```
# max du R2 de Cox and Snell
maxR2CoxSnell = 1.0 - (L0)**(2.0/donnee.shape[0])
# R2 de Nagelkerke
r2Nagelkerke = r2CoxSnell / maxR2CoxSnell
print("R2 de Nagelkerke : %.4f" % (r2Nagelkerke))
```

```
## R2 de Nagelkerke : 0.3247
```

3

Sélection et évaluation de la régression logistique binaire

Sommaire

Dans ce chapitre, nous allons traiter les questions de sélection et d'évaluation des modèles à travers l'exemple du modèle logistique. Nous noterons M_β le modèle logistique défini par le vecteur de paramètre β. L'ensemble des méthodes présentées peut s'étendre à d'autres problématiques de sélection - validation de modèles.

3.1 Sélection ou choix de modèle

Si on se restreint à des modèles logistiques, sélectionner un modèle revient à choisir les variables qui vont constituer le modèle.

3.1.1 La déviance

Bien souvent, on utilise la quantité

$$D_M = -2\mathcal{L}_n(\hat{\beta}) \tag{3.1}$$

est appelée **déviance** (ou déviance résiduelle, en anglais `residual deviance`). Contrairement à la log-vraisemblance, elle est positive. L'objectif de l'algorithme d'optimisation est de minimiser cette déviance. On peut faire le parallèle avec la `somme des carrés des résidus` de la régression linéaire multiple. La `null deviance` D_0 calculée sur le modèle uniquement composée de la `constante` correspondrait à la `somme des carrés totaux`.

Exemple 3.1 *Calcul de déviance*

Pour illustrer nos propos, reprenons notre modèle du chapitre précédent. Notre modèle s'écrit :

$$chd = f(sbp, tobacco, age, \dots)$$

```
# Chargement des données
import pandas as pd
donnee = pd.read_csv('./donnee/deseases.txt',sep=',',header=0,index_col=0)
## Estimation par Maximum de vraisemblance
import statsmodels.formula.api as smf
donnee.famhist = donnee.famhist.astype('category')
formule = create_formula("chd",donnee.drop(['chd'],axis=1).columns)
fullmodel = smf.logit(formule,data=donnee).fit(disp=False)
```

La log - vraisemblance du modèle vaut -236.07. On applique ensuite la formule de la déviance :

```
# Déviance
deviance =  -2*fullmodel.llf
print('Deviance : %.2f' % (deviance))


## Deviance : 472.14
```

La déviance du modèle est 472.14.

Remarque 3.1

Dans certains ouvrages, on définit la déviance D de manière plus générique :

$$\begin{aligned} D &= 2 \times \ln\left[\frac{L(\text{Modèle saturé})}{L(\text{Modèle étudié})}\right] \\ &= -2 \times \mathcal{L}(\text{Modèle étudé}) - [-2 \times \mathcal{L}(\text{Modèle saturé})] \\ &= D_M - [-2 \times \mathcal{L}(\text{Modèle saturé})] \end{aligned}$$

Un modèle saturé pour des données individuelles est un modèle reconstituant parfaitement les valeurs de la variable dépendante. Sa vraisemblance est égale à 1 et sa log-vraisemblance 0. Dans ce contexte, $D = D_M$.

3.1.2 Test de déviance entre 2 modèles emboîtés

Rappelons que par définition un modèle emboîté dans un autre plus général (ou plus grand) lorsqu'il est un cas particulier de ce modèle en général.

Exemple 3.2 *Les modèles logistiques définis par*

$$\textit{logit}\left(p_\beta(x)\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \tag{3.2}$$

et

$$\textit{logit}\left(p_\beta(x)\right) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 \tag{3.3}$$

sont emboîtés l'un dans l'autre.

En effet, il est facile de voir que faire un test entre modèles emboîtés est équivalent à tester la nullité de certains coefficients du grand modèle. On peut ainsi, les tests de Wald, du maximum de vraisemblance ou du score pour tester deux modèles emboîtés. Si par exemple $\mathcal{M}_1 \subset \mathcal{M}_2$, alors on a :

$$-2\left(\mathcal{L}_n(\widehat{\mathcal{M}_1}) - \mathcal{L}_n(\widehat{\mathcal{M}_2})\right) \xrightarrow{\mathcal{L}} \chi^2_{p_2-p_1} \tag{3.4}$$

où p_j désigne la dimension du modèle M_j.

Remarque 3.2 *La statistique de test peut s'écrire $D_{M_1} - D_{M_2}$, c'est pourquoi ce test est également appelé test de déviance.*

3.1.3 Critère de choix de modèle

Le test que nous venons d'étudier permet de sélectionner un modèle parmi deux modèles emboîtés. Or, à partir de p variables explicatives, il est possible de définir un grand nombre de modèles logistiques qui ne sont pas forcément emboîtés. L'utilisation d'un simple test de déviance se révèle alors insuffisante. On a recours à des critères de choix de modèles qui permettent de comparer des modèles qui ne sont pas forcément emboîtés les uns dans les autres.

Les critères AIC et BIC sont les plus utilisés.Ces critères sont basés sur la philosophie suivante : *plus la vraisemblance est grande, plus grande est donc la log-vraisemblance et meilleur est le modèle (en terme d'ajustement).* Cependant la vraisemblance augmente avec la complexité du modèle, et choisir le modèle qui maximise la vraisemblance revient à choisir le modèle saturé. Ce modèle est clairement sur-paramétré, il « sur-ajuste » les données (*overfitting*).

Pour choisir des modèles plus parcimonieux, une stratégie consiste à pénaliser la vraisemblance par une fonction du nombre de paramètres.

— Par définition l'AIC (Akaike Informative Criterion) pour un modèle $\mathcal{M}$ de dimension p est défini par

$$AIC(\mathcal{M}) = -2\mathcal{L}_n\left(\hat{\mathcal{M}}\right) + 2p$$

```
# AIC
aic = -2*fullmodel.llf + 2*len(fullmodel.params)
print('AIC : %.2f' %(aic))
```

```
## AIC : 492.14
```

Par défaut, Statsmodels calcul l'AIC et le retourne grâce à l'outil aic.

```
# AIC
print('AIC : %.2f' % (fullmodel.aic))
```

```
## AIC : 492.14
```

— Le critère de choix de modèle le BIC (Bayesian Informative Criterion) pour un modèle $\mathcal{M}$ de dimension p est défini par :

$$BIC(\mathcal{M}) = -2\mathcal{L}_n\left(\widehat{\mathcal{M}}\right) + p\log(n)$$

```
# BIC
import numpy as np
bic = -2*fullmodel.llf + len(fullmodel.params)*np.log(donnee.shape[0])
print('BIC : %.2f' % (bic))
```

```
## BIC : 533.50
```

Par défaut, Statsmodels calcul le BIC et le retourne grâce à l'outil bic.

```
# AIC
print('BIC : %.2f' % (fullmodel.bic))
```

```
## BIC : 533.50
```

On choisira le modèle qui possède le plus petit AIC ou BIC. L'utilisation de ces critères est simple. Pour chaque modèle concurrent, le critère de modèles est calculé et le modèle qui présente le plus faible est sélectionné.

Remarque 3.3 — *Remarquons que certains logiciels utilisent $-AIC$ et $-BIC$ il est donc prudent de bien vérifier dans quel sens doivent être optimisés ces critères (maximisation ou minimisation). Ceci peut être fait aisément en comparant un modèle très mauvais (composé uniquement de la constante par exemple) à un bon modèle et de vérifier dans quel sens varient les critères de choix.*

— *Les pénalités ne sont pas choisies au hasard... Le critère BIC est issue de la théorie bayésienne. En deux mots, les modèles candidats sont vus comme des variables aléatoires sur lesquels on met une loi de probabilité a priori. Dans ce contexte, choisir le modèle qui possède le plus petit BIC revient à choisir le modèle qui maximise la probabilité a posteriori.*

3.1.4 Subdivision apprentissage - test

Ce critère mesure la performance d'un modèle en terme de prévision. Pour rappel, nous avons construits nos modèle précédents sur l'ensemble de nos données. Or dans ce contexte, les classifieurs plus complexes ayant tendance à « coller » aux données laissent à penser, à tort, qu'ils présentent de meilleures performances. En règle générale, plus une observation pèse sur son propre classement en généralisation, plus optimiste sera le taux d'erreur en resubstitution. Bref, le taux d'erreur en resubstitution est totalement inutilisable dès lors que l'on souhaite comparer les performances de modèles de complexité différente (ou reposant sur des représentations différentes ex. arbre de décision vs. régression logistique).

Parmi les solutions envisageables, la plus simple consiste à évaluer le classifeur sur des données à part qui n'ont pas participé au processus d'apprentissage. Nous procédons de la manière suivante lorsque l'on dispose d'un échantillon Ω de taille n :

1. Nous tirons au hasard n_1 individus parmi n, il s'agit de **l'échantillon d'apprentissage**, nous les utilisons pour construire le modèle de prédiction M_1. On dédie généralement 70% des données à l'apprentissage.
2. Sur les $n_2 = n - n_1$ observations restantes, **l'échantillon test**, nous appliquons le modèle M_1 afin d'avoir les valeurs prédictes les confronter aux valeurs observées. Habituellement $\frac{n_2}{n} = 1 - \frac{n_1}{n} = 30\%$.

Principal atout de cette approche, les indicateurs ainsi obtenus sont non-biaisés. Ils permettent de comparer les mérites respectifs de plusieurs modèles, même s'ils sont de complexité différente, même s'ils ne reposent pas sur des systèmes de représentation identiques (ex. un classifeur linéaire vs. un classifeur non linéaire). C'est la démarche à privilégier si l'on dispose de suffisamment d'observations.

Et c'est bien là le principal défaut de cette démarche. Lorsque nous travaillons sur un petit échantillon, en réserver une partie pour l'évaluation pénalise la construction du modèle, sans pour autant que l'on ait une évaluation fiable des performances puisque l'effectif est trop faible. Nous sommes face à 2 exigences contradictoires :

— Réserver une grande partie des données à l'apprentissage favorise la construction d'un modèle de bonne qualité. En revanche, l'échantillon test sera trop réduit pour espérer obtenir une estimation viable des performances en prédiction.
— Réserver une fraction plus forte au test permet certes d'obtenir une évaluation fiable. Mais dans ce cas nous nous tirons une balle dans le pied (aïe !) car le modèle élaboré peut être dégradé faute d'informations (d'observations) suffisantes.

Bref, les proportions habituellement mises en avant (70% vs. 30%) ne doivent pas être prises au pied de la lettre. Tout est affaire de compromis : il en faut suffisamment pour l'apprentissage afin de produire un modèle consistant ; il en faut suffisamment pour le test afin d'obtenir une évaluation fiable des performances. Les « bonnes » proportions dépendent souvent des caractéristiques du classifieur et des données analysées (rapport entre le nombre d'observations et le nombre de variables, degré de difficulté du concept à apprendre, etc.).

Pour trouver le meilleur compromis **biais - variance**, il est recommandé de subdiviser notre dataset en deux parties :

— Une partie qui sera utilisée pour entraîner le modèle avec différents niveaux de complexité et différentes valeurs d'hyperparamètres qu'on appelle **échantillon d'apprentissage** ou (*train set* en anglais).
— Une partie pour mesurer l'erreur commise par la meilleure règle de prédiction et la comparer à l'erreur commise par des règles de familles différentes sur des données qui n'ont pas du tout participé à la construction des règles : C'est **l'échantillon test** ou (*test set* en anglais).

Sous scikit-learn, on utilise la fonction train_test_split du module model_selection qui permet de séparer le dataset en train set et en test set. Cette méthode mélange le dataset de façon aléatoire avant d'effectuer le tirage. La part des observations consacrée à test set dépend de la taille totale du dataset. Avec un test set trop grand, on peut nuire à la qualité de l'apprentissage. En général, on met 70% de notre dataset dans le train set et 30% sur le test set.

```
# Modélisation sur le train set
from sklearn.model_selection import train_test_split
DTrain, DTest = train_test_split(donnee,test_size=0.3,random_state=123)
model = smf.logit(formule,data=DTrain).fit(disp=False)
coef_model = model.summary2().tables[1]
```

Table 3.1 – Coefficients du modèle $chd = f(.)$ sur données d'apprentissage

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	-5.6845	1.6495	-3.4461	0.0006	-8.9176	-2.4515
famhist[T.Present]	0.6379	0.2803	2.2756	0.0229	0.0885	1.1873
sbp	-0.0006	0.0074	-0.0815	0.9350	-0.0151	0.0139
tobacco	0.1098	0.0351	3.1281	0.0018	0.0410	0.1785
ldl	0.1817	0.0726	2.5042	0.0123	0.0395	0.3240
adiposity	0.0264	0.0360	0.7341	0.4629	-0.0442	0.0970
typea	0.0464	0.0164	2.8307	0.0046	0.0143	0.0786
obesity	-0.0865	0.0543	-1.5934	0.1111	-0.1929	0.0199
alcohol	0.0011	0.0056	0.1989	0.8424	-0.0098	0.0120
age	0.0562	0.0156	3.6112	0.0003	0.0257	0.0867

3.1.5 Validation croisée

Lorsque les effectifs sont très faibles, nous avons intérêt à construire le modèle $\mathcal{M}$ sur la totalité des données, puis à utiliser des techniques de ré-échantillonnage pour en mesurer les performances (ex. **la validation croisée, le bootstrap**). L'intérêt est double. Nous utilisons la totalité des données (la totalité de l'information disponible) pour construire le classifieur. Et nous pouvons obtenir une évaluation (plus ou moins) faiblement biaisée de son erreur de prédiction.

Le principe de la validation croisée est de **moyenner** le pourcentage de mal classés à l'aide de plusieurs découpages de l'échantillon. Plus précisément, on divise l'échantillon initial en K sous-échantillons E_k de même taille et on effectue K procédures d'apprentissage-validation pour lesquelles :

— L'échantillon test sera constitué d'une division E_k ;
— L'échantillon d'apprentissage sera constituée de l'ensemble des autres divisions $E - E_k$.

On obtient ainsi une prévision pour chaque individu de la division E_k et une fois les K procédures apprentissage-validation effectuées, on a une prévision pour tous les individus de l'échantillon. Il suffit alors de comparer ces prévisions aux valeurs observées pour obtenir une estimation de la probabilité d'erreur de la règle. Le modèle retenu sera le modèle qui conduit à l'estimation minimale.

Bien entendu le choix du nombre K de parties n'est pas anodin :

— Plus K est faible, plus la capacité de prévision sera évaluée dans de nombreux cas puisque le nombre d'observations dans la validation sera élevé, mais moins l'estimation sera précise puisque moins de données seront utilisées pour estimer les paramètres du modèle ;
— Au contraire, un K élevé conduit à peu d'observations dans la validation et donc à une plus grande variance dans l'estimation de la probabilité d'erreur.

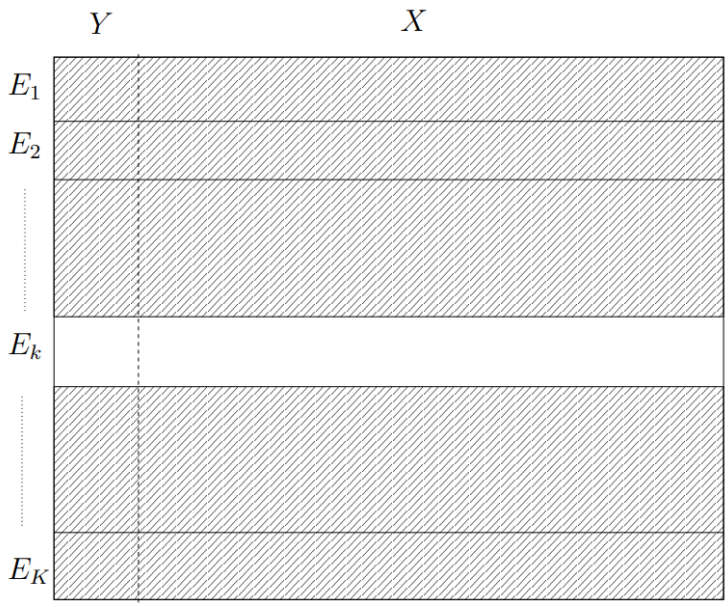


Figure 3.1 – Découpage de l'échantillon pour la validation croisée. L'échantillon d'apprentissage correspond à la partie harchurée.

Remarque 3.4 *Sous Python, le package scikit-learn à travers le module model_selection propose plusieurs outils permettant d'effectuer une validation croisée. Cependant, avec Statsmodels, il est impossible d'effectuer une validation croisée.*

3.1.6 Sélection automatique

Les procédures que nous venons d'étudier permettent de sélectionner un modèle à partir d'une famille de modèles donnée. Une autre approche de la sélection de modèle consiste à chercher parmi les variables $X_1, \ldots, X_p$, celles qui « expliquent le mieux » Y . Par exemple, pour la régression logistique, nous pourrions nous poser le problème de chercher le meilleur sous-ensemble des p variables explicatives pour un critère C donné (AIC, BIC...). Le nombre de sous ensembles de p variables étant 2^p, nous serions en présence de 2^p modèles logistiques possibles, c'est-à-dire 2^p modèles différents. Bien entendu, nous sélectionnerions le modèle qui optimiserait le critère C. Cependant, dans de nombreuses situations, p est grand et par conséquent le nombre de modèles considérés est « très grand ». Les algorithmes d'optimisation du critère C deviennent très coûteux en temps de calcul. On préfère alors souvent utiliser des méthodes de recherche *pas à pas*.

3.1.6.1 Forward selection

A chaque pas, une variable est ajoutée au modèle.

— Si la méthode ascendante utilise un test de déviance, nous rajoutons la variable X_j dont la valeur p (probabilité critique) associée à la statistique de test de déviance qui compare les 2 modèles est minimale. Nous nous arrêtons lorsque toutes les variables sont intégrées ou lorsque la valeur p est plus grande qu'une valeur seuil.
— Si la méthode ascendante utilise un critère de choix, nous ajoutons la variable X_j dont l'ajout au modèle conduit à l'optimisation la plus grande du critère de

choix. Nous nous arrêtons lorsque toutes les variables sont intégrées ou lorsque qu'aucune variable ne permet l'optimisation du critère de choix.

3.1.6.2 Backward selection

A la première étape toutes les variables sont intégrées au modèle.

— Si la méthode descendante utilise un test de déviance, nous éliminons ensuite la variable X_j dont la valeur p associée à la statistique de test de déviance est la plus grande. Nous nous arrêtons lorsque toutes les variables sont retirées du modèle ou lorsque la valeur p est plus petite qu'une valeur seuil.
— Si la méthode descendante utilise un critère de choix, nous retirons la variable X_j dont le retrait du modèle conduit à l'augmentation la plus grande du critère de choix. Nous nous arrêtons lorsque toutes les variables sont retirées ou lorsque qu'aucune variable ne permet l'augmentation du critère de choix.

```
# Backward Selection based on p-value
import statsmodels.api as sm
from statsmodels.genmod.generalized_linear_model import GLM
from statsmodels.genmod import families

def backward_selected(X,y,
                      threshold_out = 0.05,
                      verbose=True):
    included=list(X.columns)
    while True:
        changed=False
        model = GLM(y,sm.add_constant(pd.DataFrame(X[included])),
                    family=families.Binomial()).fit(attach_wls=True, atol=1e-10)
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.idxmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature,
                                                             worst_pval))
        if not changed:
            break
    model = GLM(y,sm.add_constant(pd.DataFrame(X[included])),
                family=families.Binomial()).fit(attach_wls=True, atol=1e-10)
    return model.summary2()

backward = backward_selected(XTrain, yTrain)
print(backward)
```

3.1.6.3 Stepwise selection

Idem que l'ascendante, sauf que l'on peut éliminer des variables déjà introduites. En effet, il peut arriver que des variables introduites au début de l'algorithme ne soient plus significatives après introduction de nouvelles variables. Remarquons qu'en général la variable « constante » est toujours présente dans le modèle.

Maintenant que nous avons construit un modèle de prédiction, il faut en évaluer l'efficacité.

3.2 Évaluation de la régression logistique binaire

Dans ce section, nous nous consacrons à ce que l'on appellerait des méthodes d'évaluation externes basées sur les prédictions $\widehat{Y}(\omega)$ et/ou les probabilités a posteriori $\widehat{p}(\omega)$ fournies par le classifieur. Nos indicateurs calculés dans le cadre de ce chapitre sont issus des résultats du modèle entraîné sur les données du train set.

3.2.1 Matrice de confusion

Pour évaluer la capacité à bien classer du modèle, il est judicieux de construire ce que l'on appelle une **matrice de confusion**. Elle confronte toujours les *valeurs observées* de la variable dépendante avec les *variables prédictes*, puis comptabilise les bonnes et les mauvaises prédictions. Son intérêt est qu'elle permet à la fois d'appréhender la quantité d'erreur (le taux d'erreur) et de rendre compte de la structure de l'erreur (la manière de se tromper du modèle). Le tableau 3.2 présente un exemple de matrice de confusion :

Table 3.2 – Matrice de confusion - Forme générique

Y \ $\widehat{Y}$	$\widehat{-}$	$\widehat{+}$	Total
$-$	a	b	a+b
$+$	c	d	c+d
Total	a + c	b + d	n=a+b+c+c

Sous Python, l'attribut fittedvalues correspond aux valeurs du logit calculées sur les données d'apprentissage.

```
# valeurs estimées par la régression en resubstitution
print(model.fittedvalues)
```

```
## row.names
## 274   -2.214521
## 308    0.878742
## 391   -0.316382
## 319   -2.798634
## 161   -1.298120
##          ...
## 231    0.092591
## 99     1.299708
## 324   -0.818047
```

```
## 384    0.820044
## 367   -2.777325
## Length: 323, dtype: float64
```

Nous pouvons reproduire ces valeurs à partir des coefficients de la régression et de la matrice des descripteurs en appliquant la formule :

$$\text{logit} p_{\widehat{\beta}}(X) = X \cdot \widehat{\beta} \tag{3.5}$$

Pour obtenir la probabilité d'appartenance à une classe, on applique la formule :

$$p_{\widehat{\beta}}(X) = \frac{1}{1 + e^{-X \cdot \widehat{\beta}}} \tag{3.6}$$

```
yprobTrain = model.fittedvalues.apply(lambda x : 1/(1+np.exp(-x)))
yprobTrain
```

```
## row.names
## 274    0.098454
## 308    0.706561
## 391    0.421558
## 319    0.057398
## 161    0.214482
##           ...
## 231    0.523131
## 99     0.785786
## 324    0.306178
## 384    0.694246
## 367    0.058562
## Length: 323, dtype: float64
```

L'outil predict de Statsmodels nous renvoie cette probabilité

```
# Probabilité d'appartenance - Statsmodels
yprobTrain2 = model.predict(DTrain)
# 10 premières valeurs
print(yprobTrain2[:10])
```

```
## row.names
## 274    0.098454
## 308    0.706561
## 391    0.421558
## 319    0.057398
## 161    0.214482
## 218    0.047225
## 173    0.213685
## 297    0.168588
## 186    0.257632
## 263    0.107823
## dtype: float64
```

Nous calculons la probabilité d'appartenance à une classe sur l'échantillon test

```
# valeurs prédictes sur le test set
yprobTest = model.predict(DTest)
# 10 premières valeurs
print(yprobTest[:10])
```

```
## row.names
## 50     0.108350
## 86     0.138589
## 35     0.379416
## 232    0.754659
## 237    0.480198
## 402    0.339740
## 108    0.363170
## 94     0.268040
## 310    0.117324
## 12     0.747615
## dtype: float64
```

Nous pouvons déduire du logit la prédiction en utilisant la règle d'affectation simple :

$$\widehat{Y} = \begin{cases} 1 & \text{si} \quad \text{logit}\left(p_{\widehat{\beta}}(X)\right) \geq 0 \\ 0 & \text{si} \quad \text{logit}\left(p_{\widehat{\beta}}(X)\right) < 0 \end{cases} \tag{3.7}$$

Ceci est équivalent à

$$\widehat{Y} = \begin{cases} 1 & \text{si} \quad p_{\widehat{\beta}}(X) \geq 0.5 \\ 0 & \text{si} \quad p_{\widehat{\beta}}(X) < 0 \end{cases} \tag{3.8}$$

```
# valeurs chd prédictes en resubstitution
ypredTrain = np.where(yprobTrain> 0.5,1, 0)
# 1à premières valeurs
print(ypredTrain[:10])
```

```
## [0 1 0 0 0 0 0 0 0 0]
```

Via un tableau croisé entre les classes observées et prédictes, nous obtenons la matrice de confusion sur l'échantillon d'apprentissage suivante :

```
# Matrice de confusion - Train set
cmTrain = pd.DataFrame(pd.crosstab(DTrain.chd, ypredTrain))
cmTrain.loc['Total'] = cmTrain.sum(axis=0)
cmTrain.loc[:,'Total'] = cmTrain.sum(axis=1)
tab = cmTrain.reset_index()
```

Statsmodels retourne la matrice de confusion calculée sur l'échantillon d'apprentissage grâce à l'attribut pred_table() :

Table 3.3 – Matrice de confusion - Train set

chd	0	1	Total
0	183	32	215
1	50	58	108
Total	233	90	323

```
# matrice de confusion fourni par l'outil - train set
print(model.pred_table(threshold=0.5))
```

```
## [[183.  32.]
##  [ 50.  58.]]
```

Dans un problème de classification binaire de type (+ vs -), à partir de la forme générique de la matrice de confusion (tableau 3.2), plusieurs indicateurs peuvent être déduits pour rendre compte de la concordance entre les valeurs observées et les valeurs prédictes.

Nous sauvegardons dans l'objet `cmTest` notre matrice de confusion construite sur l'échantillon test. Elle nous servira pour le calcul des indicateurs ci-dessous

On calcule la matrice de confusion sur l'échantillon test :

```
# valeurs chd prédictes en resubstitution
ypredTest = np.where(yprobTest> 0.5,1, 0)
# Matrice de confusion - Test set
cmTest = pd.DataFrame(pd.crosstab(DTest.chd, ypredTest))
cmTest.loc['Total'] = cmTest.sum(axis=0)
cmTest.loc[:,'Total'] = cmTest.sum(axis=1)
tab = cmTest.reset_index()
```

Table 3.4 – Matrice de confusion - Test set

chd	0	1	Total
0	71	16	87
1	25	27	52
Total	96	43	139

Sur les 139 individus en test, 96 sont associés à la classe « 0 », 43 à « 1 ».

3.2.1.1 Les vrais positifs

Ce sont les observations qui ont été classées positives et qui le sont réellement. Sa valeur dans le tableau 3.2 est d.

```
# Vrais positifs
vrai_positif  = cmTest.iloc[1,1]
print('Vrais positifs : %.i' % (vrai_positif))
```

```
## Vrais positifs : 27
```

3.2.1.2 Les faux positifs

Ce sont les individus classés positifs et qui sont en réalité des négatifs. Sa valeur dans le tableau 3.2 est b.

```
# Faux positifs
faux_positif = cmTest.iloc[0,1]
print('Faux positifs : %.i' % (faux_positif))
```

```
## Faux positifs : 16
```

3.2.1.3 Les faux négatifs

Ce sont les individus classés négatifs et qui sont en réalité des positifs. Sa valeur dans le tableau 3.2 est c.

```
# Faux négatifs
faux_negatif = cmTest.iloc[1,0]
print('Faux négatifs : %.i' % (faux_negatif))
```

```
## Faux négatifs : 25
```

3.2.1.4 Les vrais négatifs

Ce sont les individus qui ont été classés négatifs et qui le sont réellement. Sa valeur dans le tableau 3.2 est a.

```
# Vrai négatifs
vrai_negatif = cmTest.iloc[0,0]
print('Vrais négatifs : %.i' % (vrai_negatif))
```

```
## Vrais négatifs : 71
```

Remarque 3.5 *Ces termes sont peu utilisés en pratique car les positifs et les négatifs n'ont pas le même statut dans la majorité des études.*

3.2.1.5 Le taux d'erreur

C'est égal au nombre de mauvais classement rapporté à l'effectif total c'est-à-dire

$$\varepsilon = \frac{b+c}{n} = 1 - \frac{a+b}{n} = \frac{\text{faux positifs} + \text{faux négatifs}}{n} \tag{3.9}$$

Il estime la probabilité de mauvais classement du modèle.

```
# Taux d'erreur
error_rate = (faux_positif + faux_negatif)/len(ypredTest)
print('Error rate : %.2f' % (error_rate))
```

```
## Error rate : 0.29
```

3.2.1.6 Le taux de succès

Il correspond à la probabilité de bon classement. C'est le complémentaire à 1 du taux d'erreur :

$$\theta = \frac{a+d}{n} = 1 - \varepsilon = \frac{\text{vrais positifs} + \text{vrais négatifs}}{n} \tag{3.10}$$

```
# Taux de succès
accuracy = (vrai_positif + vrai_negatif)/len(ypredTest)
print('Accuracy : %.2f' %(accuracy))
```

```
## Accuracy : 0.71
```

3.2.1.7 La sensibilité

Encore appelée **rappel** ou **taux de vrais positifs (TVP)**, la sensibilité indique la capacité du modèle à retrouver les positifs.

$$\text{Sensibilité} = \frac{d}{c+d} = \frac{\text{vrais positifs}}{\text{faux négatifs} + \text{vrais positifs}} \tag{3.11}$$

```
# Sensibilité
sensibilite = vrai_positif/(faux_negatif+vrai_positif)
print('Sensibilité : %.2f' % (sensibilite))
```

```
## Sensibilité : 0.52
```

3.2.1.8 La précision

Elle indique la proportion de vrais positifs parmi les individus qui ont été classés positifs

$$\text{précision} = \frac{d}{b+d} = \frac{\text{vrais positifs}}{\text{faux positifs} + \text{vrais positifs}} \tag{3.12}$$

Elle estime la probabilité d'un individu d'être réellement positif lorsque le modèle le classe comme tel. Dans certains domaines, on parle de **valeur prédicte positive (VPP)**.

```
# Précision
precision = vrai_positif/(faux_positif+vrai_positif)
print('Précision : %.2f' % (precision))
```

```
## Précision : 0.63
```

3.2.1.9 La spécificité

A l'inverse de la sensibilité, la spécificité indique la proportion de négatifs détectés

$$\text{Spécificité} = \frac{a}{a+b} = \frac{\text{vrais négatifs}}{\text{vrais négatifs} + \text{faux positifs}} \tag{3.13}$$

```
# Spécificité
specificite = vrai_negatif/(vrai_negatif+faux_positif)
print('Specificité : %.2f' % (specificite))
```

```
## Specificité : 0.82
```

3.2.1.10 Le taux de faux positifs (TFP)

Il correspond à la proportion de négatifs qui ont été classés positifs, c'est-à-dire

$$\text{TFP} = \frac{b}{a+b} = \frac{\text{faux positifs}}{\text{vrais négatifs} + \text{faux positifs}} = 1 - \text{Spécificité} \tag{3.14}$$

```
# Taux de faux positifs (TFP)
taux_faux_positif = faux_positif/(vrai_negatif + faux_positif)
print('Taux de faux positifs : %.2f' % (taux_faux_positif))
```

```
## Taux de faux positifs : 0.18
```

3.2.1.11 la F-Mesure

Indicateur très utilisé en recherche d'information, la F-Mesure synthétise (moyennes harmonique) le rappel et la précision. L'importance accordée à l'une ou à l'autre est paramétrable avec β.

$$F_\beta = \frac{(1+\beta)^2 \times \text{rappel} \times \text{précision}}{\beta^2 \times \text{précision} + \text{rappel}} \tag{3.15}$$

Lorsque

— $\beta = 1$, on accorde la même importance au rappel et à la précision, la F-Mesure devient :

$$F_{\beta=1} = \frac{2 \times \text{rappel} \times \text{précision}}{\text{précision} + \text{rappel}} \tag{3.16}$$

— $\beta < 1$, on accorde plus d'importance à la précision par rapport au rappel. Une valeur fréquemment utilisée est $\beta = 0.5$, on accorde deux fois plus d'importance à la précision.

— $\beta > 1$, on accorde plus d'importance au rappel par rapport à la précision. Une valeur fréquemment rencontrée est $\beta = 2$.

Remarque 3.6 *La F-Mesure est une moyenne harmonique entre le rappel et la précision. En effet, nous pouvons l'écrire sous la forme suivante :*

$$F = \frac{1}{\alpha \frac{1}{\textit{précision}} + (1-\alpha)\frac{1}{\textit{rappel}}} \quad \textit{où} \quad \beta^2 = \frac{1-\alpha}{\alpha} \tag{3.17}$$

```
# F-mesure avec beta=1
f_mesure = 2*sensibilite*precision/(precision+sensibilite)
print('F-Mesure : %.2f' % (f_mesure))
```

```
## F-Mesure : 0.57
```

3.2.1.12 Indice de Youden

L'indice de Youden est bien connu en biostatistique, moins en apprentissage supervisé. Il s'écrit :

$$IY = \text{Sensibilité} + \text{Spécificité} - 1 \tag{3.18}$$

Son mérite est de caractériser le classifieur selon la sensibilité et la spécificité. Il prend la valeur maximum 1 lorsque le modèle est parfait. En effet, dans ce cas Sensibilité = 1 et Spécificité $= 1$. Il peut être utilisé pour comparer les performances de plusieurs modèles.

Son interprétation n'est pas très évidente en revanche. C'est le principal frein à son utilisation.

```
# Indice de Youden
indiceYouden = sensibilite + specificite - 1
print('Indice de Youden : %.2f' % (indiceYouden))
```

```
## Indice de Youden : 0.34
```

3.2.1.13 Rapport de vraisemblance

Le rapport de vraisemblance décrit le surcroît de chances des positifs (par rapport aux négatifs) d'être classés positifs. Sa formule est la suivante :

$$L = \frac{\text{Sensibilité}}{1 - \text{Spécificité}} \tag{3.19}$$

Le rapport de vraisemblance ne dépend pas de la proportion des positifs. Il donne donc des indications valables même si l'échantillon n'est pas représentatif. Plus grande est sa valeur, meilleur est le modèle.

```
# Rapport de vraisemblance
rapport_vraisemblance = sensibilite/(1-specificite)
print('Rapport de vraisemblance : %.2f' % (rapport_vraisemblance))
```

```
## Rapport de vraisemblance : 2.82
```

Sous Python, le module metrics du package sklearn propose des outils pour le calcul de certains de ces indicateurs.

Remarque 3.7 (Inconvénients de la matrice de confusion) *Aussi intéressante qu'elle soit, la matrice de confusion présente une faiblesse importante : elle repose essentiellement sur les prédictions (seuil de séparation fixé sur la probabilité estimée d'avoir $Y = 1$). Également, la matrice de confusion et le taux d'erreur sont sensibles à l'importance relative des groupes, c'est-à-dire que le classement dans le groupe le plus important pèse davantage.*

3.2.2 Diagramme de fiabilité

Contrairement à certaines méthodes d'apprentissage supervisé (*ex.* support vector machine, classifieur bayesien naïf), la régression logistique produit une bonne approximation de la quantité $\widehat{p}(\omega)$. Lapremière idée qui vient à l'esprit est de confronter les probabilités estimées par le modèle et celles observées dans le fichier de données. On construit pour cela le **diagramme de fiabilité** ou *reliability diagram*[1].

Ici également, si nous en avons la possibilité, nous avons tout intérêt à construire lediagramme de fiabilité à partir des données tests n'ayant pas participé à l'élaboration du classifieur. Les indications obtenues n'en seront que plus crédibles (*cf.* Rakotomalala (2011)).

Voici les étapes de la construction du diagramme de fiabilité :

1. Appliquer le classifieur sur les données pour obtenir le score $\widehat{p}(\omega)$;
2. Trier le fichier selon le score croissant ;
3. Sur la base du score, subdiviser les données en intervalles (ex. $0.0-0.2, 0.2-0.4$, etc.) ;
4. Dans chaque intervalle, calculer la proportion de positifs ;
5. Dans le même temps, toujours dans chaque intervalle, calculer la moyenne des scores ;
6. Si les chiffres concordent dans chaque intervalle, les scores sont bien calibrés, le classifieur est de bonne qualité ;
7. Nous pouvons résumer l'information dans un graphique *nuage de points* appelé **diagramme de fiabilité**, avec en abscisse la moyenne des scores, en ordonnée la proportion de `positifs` ;
8. **Si les scores sont bien calibrés, les points devraient être alignés sur une droite** `la première bissectrice` ;
9. Les points s'écartant sensiblement de la première bissectrice doivent attirer notre attention.

Exemple 3.3 *Diagramme de fiabilité*

Nous reprenons notre exemple de détection des maladies du cœur. Nous avons calculés les probabilités d'appartenance estimées sur l'échantillon d'apprentissage. On crée

1. https ://towardsdatascience.com/introduction-to-reliability-diagrams-for-probability-calibration-ed785b3f5d44

un dataframe incorporant la variable `score` et la variable cible `chd`. Puis on applique l'étape 3.

```
# dataframe temporarire
frame = pd.DataFrame({'chd':DTrain.chd,'score':yprobTrain}, index=DTrain.index)
# 5 intervalles de largeur égales
frame['bins'] = pd.cut(frame.score,bins=5,include_lowest=True)
```

A partir de ce dataset, nous pouvons calculer la moyenne des scores estimés dans chaque groupe délimité par les individus (`étape 5`) :

```
# moyenne des scores par groupe
mean_score = frame.pivot_table(index='bins',values='score',aggfunc='mean')
tab = mean_score.round(4).reset_index()
print(tab)
```

```
##                bins   score
## 0  (0.00205, 0.191]  0.0960
## 1    (0.191, 0.377]  0.2767
## 2    (0.377, 0.564]  0.4742
## 3     (0.564, 0.75]  0.6620
## 4     (0.75, 0.937]  0.8279
```

Puis la proportion des observations (scores observés) positives dans les mêmes groupes.

```
# Proportion des positifs dans chaque groupe
df = frame.groupby(["bins","chd"]).size().reset_index()
df.columns = ["groupe","chd","nb"]
df = df.pivot(index="groupe", columns="chd", values="nb")
df["prop. pos."] = df.iloc[:,1]/df.sum(axis=1)
tab = df.round(4).reset_index()
print(tab)
```

```
## chd            groupe    0   1  prop. pos.
## 0    (0.00205, 0.191]  114  12      0.0952
## 1      (0.191, 0.377]   51  23      0.3108
## 2      (0.377, 0.564]   29  25      0.4630
## 3       (0.564, 0.75]   15  29      0.6591
## 4       (0.75, 0.937]    6  19      0.7600
```

Le diagramme de fiabilite est un graphique nuage de points opposant les scores estimés et observés. S'ils forment une droite, nous pouvons considérer que la modélisation est pertinente car le modèle arrive à approcher de manière satisfaisante l'appartenance aux classes des individus.

```
# Diagramme de fiabilité
from plotnine import *
data = pd.DataFrame({"x" : np.arange(0,1,0.1),"y":np.arange(0,1,0.1)})
```

```
data2 = pd.DataFrame(np.c_[mean_score,df["prop. pos."]],columns=["x","y"])
print((ggplot(data,aes(x="x",y="y"))+geom_line(color="blue")+
          geom_point(data2,aes(x="x",y="y"),colour="green")+
          geom_line(data2,aes(x="x",y="y"),colour="green",linetype="dashed")+
          labs(x='Moyenne des scores',y='Proportion des "+"',
               title='Diagramme de fiabilité')))
```

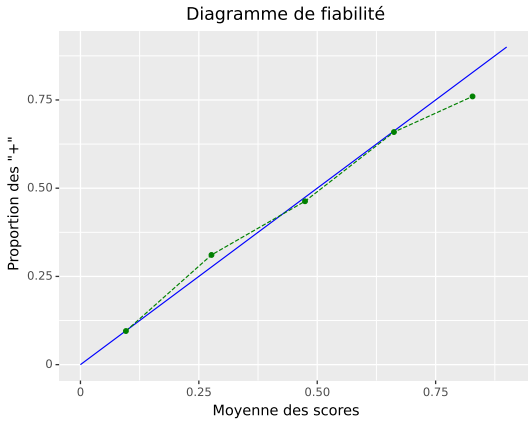


Figure 3.2 – Diagramme de fiabilite

Pour ce qui est de notre exemple, la modélisation semble relativement intéressante, même si nous notons ici ou là des écarts par rapport à la première bissectrice.

3.2.3 Quelques tests statistiques

3.2.3.1 Test de Hosmer-Lemeshow

Le test de Hosmer - Lemeshow (*cf.* Lemeshow, Sturdivant, and Hosmer Jr (2013)) relève à peu près de la même logique que le diagramme de fiabilité. A la différence qu'au lieu de se baser simplement sur une impression visuelle, on extrait du tableau un indicateur statistique qui permet de quantifier la qualité des estimations $\widehat{p}(\omega)$.

Concrètement, nous procédons de la manière suivante :

1. Appliquer le classifieur sur les données pour obtenir les estimations $\widehat{p}(\omega)$;
2. Trier les données selon le score croissant ;
3. Subdivisez les données en K groupes en se basant sur les quantiles (les auteurs proposent prioritairement les déciles (K=10)) (ex. les quantiles d'ordre 4 correspondent aux quartiles, les quantiles d'ordre 10 aux déciles, etc.). Les auteurs proposent prioritairement les déciles $(K = 10)$. Il semble par ailleurs plus judicieux d'utiliser les quantiles plutôt que les seuils sur les scores comme cela a été fait pour le diagramme de fiabilité. L'approximation de la loi de distribution de la statistique du test sous H_0 est de meilleure qualité.
4. Dans chaque groupe $k(k = 1, \dots, K)$ d'effectif n_k, nous devons calculer plusieurs quantités :

   — n_k^+ : le nombre de positifs observés ;
   — n_k^- : le nombre de négatifs observés ;

— $\hat{n}_k^+ = \sum_{\omega \in k} \hat{p}(\omega)$: la somme des scores des observations situées dans le groupe k. On la désigne comme la fréquence théorique des positifs dans le groupe ;

— $\hat{p}_k^+ = \dfrac{\hat{n}_k^+}{n_k}$:la moyenne des scores observés dans le groupe k ;

— $\hat{n}_k^- = n_k - \hat{n}_k^+$: la fréquence théorique des négatifs.

5. Nous calculons alors la statistique de Hosmer et Lemeshow en utilisant une des formules suivantes :

$$\chi^2_K = \sum_k \left[\frac{(n_k^+ - \hat{n}_k^+)^2}{\hat{n}_k^+} + \frac{(n_k^- - \hat{n}_k^-)^2}{\hat{n}_k^-} \right] \tag{3.20}$$

$$= \sum_{k=1}^{k=K} \frac{n_k (n_k^+ - \hat{n}_{k1})^2}{\hat{n}_k^+ (n_k - \hat{n}_k^+)} \tag{3.21}$$

$$= \sum_{k=1}^{k=K} \frac{(n_k^+ - \hat{n}_k^+)^2}{\hat{n}_k^+ (1 - \hat{p}_k^+)} \tag{3.22}$$

6. Lorsque le modèle est correct (H_0), la statistique χ^2_K suit approximativement une loi de χ^2 à $K-2$ degrés de liberté.
7. Lorsque la probabilité critique du test (`p-value`) est plus grande que le risque choisie, le modèle issu de la régression logistique est accepté.

Remarque 3.8 (Hosmer et Lemeshow sur un échantillon test) *Tout comme pour la matrice de confusion, nous pouvons subdiviser les données en deux parties : la première pour construire le modèle, la seconde pour l'évaluer. La procédure de Hosmer et Lemeshow peut être élaborée sur ce second échantillon. La statistique de test reste identique, les degrés de libertés en revanche sont modifiés puisqu'aucun paramètre n'a été estimé sur ces données.*

Nous reprenons notre exemple de détection des problèmes de cœur.

```
# Data frame
frame2 = pd.DataFrame({'chd':DTrain.chd,'score':yprobTrain},index=DTrain.index)
# Coupure en déciles
frame2['bins'] = pd.qcut(frame2.score, q=10)
```

Nous calculons le nombre d'observations par groupe.

```
# Effectifs par groupe
eff_total = frame2.groupby("bins").size().reset_index()
eff_total.columns = ["bins","effectif"]
print(eff_total)
```

```
##                              bins  effectif
## 0  (0.0029899999999999996, 0.0504]        33
## 1                  (0.0504, 0.0919]        32
## 2                   (0.0919, 0.153]        32
```

```
## 3                    (0.153, 0.198]        32
## 4                     (0.198, 0.27]        33
## 5                     (0.27, 0.354]        32
## 6                    (0.354, 0.462]        32
## 7                    (0.462, 0.587]        32
## 8                    (0.587, 0.717]        32
## 9                    (0.717, 0.937]        33
```

Nous calculons également le score par groupe

```
# Somme des scores par groupe
sum_score = frame2.pivot_table(index='bins',values='score',aggfunc = 'sum')
print(sum_score)
```

```
##                                        score
## bins
## (0.0029899999999999996, 0.0504]     1.027218
## (0.0504, 0.0919]                    2.213967
## (0.0919, 0.153]                     3.826084
## (0.153, 0.198]                      5.611816
## (0.198, 0.27]                       7.690688
## (0.27, 0.354]                      10.022808
## (0.354, 0.462]                     13.237102
## (0.462, 0.587]                     16.829485
## (0.587, 0.717]                     20.996406
## (0.717, 0.937]                     26.544427
```

Nous calculons le nombre de positif et de négatif par groupe :

```
# Proportion des positifs dans chaque groupe
eff_pos_neg = frame2.groupby(["bins","chd"]).size().reset_index()
eff_pos_neg.columns = ["groupe","chd","nb"]
eff_pos_neg = eff_pos_neg.pivot(index="groupe", columns="chd", values="nb")
tab = eff_pos_neg.round(4).reset_index()
print(tab)
```

```
## chd                            groupe   0   1
## 0    (0.0029899999999999996, 0.0504]  33   0
## 1                    (0.0504, 0.0919]  30   2
## 2                     (0.0919, 0.153]  32   0
## 3                      (0.153, 0.198]  22  10
## 4                       (0.198, 0.27]  23  10
## 5                       (0.27, 0.354]  22  10
## 6                      (0.354, 0.462]  20  12
## 7                      (0.462, 0.587]  15  17
## 8                      (0.587, 0.717]  11  21
## 9                      (0.717, 0.937]   7  26
```

```
# Effectifs
n_tot, n_pos = eff_total.effectif.values, eff_pos_neg.iloc[:,1].values
s_score = np.r_[sum_score.score.values]
# bloc 1
chi1 = np.sum((n_pos-s_score)**2/s_score)
# # bloc 2
chi2 = np.sum((n_pos - (n_tot-s_score))**2/((n_tot-s_score)))
# # Statistique de Hosmer-Lemeshow
HL = chi1 + chi2
# pvalue
import scipy.stats as st
pvalue =  1.0 - st.chi2.cdf(HL,len(n_tot)-2)
hl_test=pd.DataFrame({"stat":HL,"pvalue":pvalue},index=["Hosmer-Lemeshow test"])
```

Table 3.5 – Test de Hosmer et Lemeshow

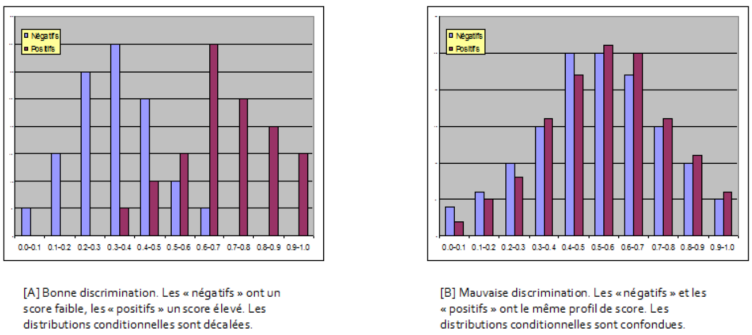
	stat	pvalue
Hosmer-Lemeshow test	192.0708	0

Au risque 5%, nous rejetons l'hypothèse compatilité de notre modèle avec les données

3.2.3.2 Test de Mann-Whitney

Le test de Mann-Whitney est de test non paramétrique de comparaison de populations (*cf.* Tallarida and Murray (1987)).

La discrimination sera d'autant meilleure que les positifs ont un score $\widehat{p}(\omega)$ élevé et les négatifs un score faible. Dans les tableaux où l'on trie les observations selon un score croissant, les négatifs seraient agglutinés en haut, les positifs en bas. On peut illustrer ce point de vue en comparant les distributions des scores conditionnellement aux classes d'appartenance. Lorsque le modèle est de bonne qualité, les distributions conditionnelles des scores sont bien différenciées (*cf.* Figure 3.3, A) ; dans le cas contraire, elles sont confondues (*cf.* Figure 3.3, B).



[A] Bonne discrimination. Les « négatifs » ont un score faible, les « positifs » un score élevé. Les distributions conditionnelles sont décalées.

[B] Mauvaise discrimination. Les « négatifs » et les « positifs » ont le même profil de score. Les distributions conditionnelles sont confondues.

Figure 3.3 – Distributions types des scores conditionnellement aux classes (source : @rakotomalala2011pratique)

Il faut pouvoir quantifier cette impression visuelle. Pour ce faire, un test de comparaison de populations semble approprié. L'objectif est de répondre à la question : « est-ce que les positifs ont des scores (significativement) plus élevés que les négatifs ? ». Le test non paramétrique de Mann-Whitney est celui que l'on retient le plus souvent dans la littérature. Dans le cadre de l'apprentissage supervisé, il convient surtout parce qu'il

est en relation avec le critère AUC (Area Under Curve) associé à la courbe ROC. Les formules associées à ce test sont les suivantes :

1. A partir des scores $\widehat{p}(\omega)$, nous calculons le rang $r(\omega)$ des individus dans l'ensemble de l'échantillon, sans distinction de classes ;
2. Nous calculons alors les scores conditionnelles de rang :
   — Pour les positifs :
   $$r_+ = \sum_{\omega : y(\omega)=1} r(\omega)$$
   — Pour les négatifs :
   $$r_- = \sum_{\omega : y(\omega)=0} r(\omega)$$
3. Nous en déduisons les statistiques :
   $$\begin{cases} U_+ &= r_+ - \dfrac{n_+(n_+ + 1)}{2} \\ U_- &= r_- - \dfrac{n_-(n_- + 1)}{2} \end{cases}$$
4. La statistique de Mann-Whitney correspond au minimum de ces deux quantités, soit
   $$U = \min\left(U_+, U_-\right) \tag{3.23}$$
5. Sous H_0, les distributions sont confondues, la statistique centrée et réduite Z suit une loi normale $\mathcal{N}(0,1)$
   $$Z = \frac{U - \dfrac{n_+ n_-}{2}}{\sqrt{\dfrac{1}{12}\left(n_+ \times n_- + 1\right) n_+ n_-}}$$
6. Il s'agit usuellement d'un test bilatéral. Mais en réalité on imagine mal que les positifs puissent présenter des scores significativement plus faibles que les négatifs. Ou alors, il faudrait prendre le complémentaire à 1 des valeurs produites par le classifieur.

Remarque 3.9 ***Deux types de corrections*** *peuvent être introduites pour préciser les résultats dans certaines circonstances :*

— *Une correction de continuité lorsque les effectifs sont faibles ;*
— *Une correction du dénominateur de la statistique centrée et réduite lorsqu'ils y a des ex-aequo, on utilise habituellement les rangs moyens.*

```
# effectif par groupe
eff = DTrain.chd.value_counts()
n_moins, n_plus = eff[0], eff[1]

# frame
frame3 = pd.DataFrame({'chd':DTrain.chd,'score':yprobTrain},index=DTrain.index)
```

```
# rang des observations suivant le score
frame3["rang"] = frame3.score.rank()
print(frame3)
```

```
##             chd     score   rang
## row.names
## 274           0  0.098454   69.0
## 308           0  0.706561  289.0
## 391           1  0.421558  210.0
## 319           0  0.057398   39.0
## 161           0  0.214482  137.0
## ...         ...       ...    ...
## 231           1  0.523131  244.0
## 99            1  0.785786  308.0
## 324           0  0.306178  175.0
## 384           1  0.694246  285.0
## 367           0  0.058562   41.0
##
## [323 rows x 3 columns]
```

```
# somme des rangs de chaque classe
rang_sum = frame3.groupby('chd')['rang'].sum()
# Effectif par groupe
r_moins,r_plus = rang_sum[0], rang_sum[1]
# Statistiques
U_moins = r_moins - (n_moins*(n_moins+1)/2)
U_plus = r_plus - (n_plus*(n_plus+1)/2)
U = min(U_moins, U_plus)
# Statistique de Mann-Whitney
num_mn = U - (n_plus*n_moins)/2
deno_mn = np.sqrt((1/12)*(n_moins*n_plus+1)*n_moins*n_plus)
# Statistique de Mann-Whitney
MN = num_mn/deno_mn
# p-value - two-sided
pvalue = 2*st.norm.sf(abs(MN))
mn_test = pd.DataFrame({"statistic" : MN, "pvalue" : pvalue},
                       index=["Mann - Whitney test"])
```

Table 3.6 – Test de Mann - Whitney

	statistic	pvalue
Mann - Whitney test	-1.096047	0.2730583

Nous obtenons la probabilité critique du test avec la loi de répartition normale centrée et réduite pvalue = 0.2731. Au risque usuel de 5%, nous concluons que les distributions conditionnelles des scores ne sont pas décalées.

3.2.4 Quelques courbes d'évaluation

3.2.4.1 La courbe ROC

La courbe ROC est un outil d'évaluation et de comparaison des modèles, dans le cas où la variable d'intérêt Y est qualitative binaire. Cette courbe permet de savoir si un modèle M1 sera toujours meilleur qu'un modèle M2 quelle que soit la matrice de coût. La courbe ROC est opérationnelle même dans le cas des distributions très déséquilibrées. C'est un outil graphique qui permet de visualiser les performances. Un seul coup d'oeil doit permettre de voir le(s) modèle(s) susceptible(s) de nous intéresser. Un indicateur synthétique associé = **aire sous la courbe ROC**.

$P(Y = 1|X = x) \geq P(Y = 0|X = x)$ équivaut à une règle d'affectation $P(Y = 1|X = x) \geq 0.5$ (seuil = 0.5). Cette règle d'affectation fournit une matrice de confusion $MC_{0.5}$, et donc 2 indicateurs $TVP_{0.5}$ et $TFP_{0.5}$. Si nous choisissons un autre seuil (0.6 par exemple), nous obtiendrons une matrice de confusion $MC_{0.6}$ et donc $TVP_{0.6}$ et $TFP_{0.6}$...etc.

L'idée de la courbe ROC est de faire varier le seuil de 1 à 0 et, pour chaque cas, calculer le TVP et le TFP que l'on reporte dans un graphique : en abscisse le TFP, en ordonnée le TVP. Modèle au hasard = diagonale Modèle parfait = passe par les points (0,0), (0,1) et (1,1) car les scores des individus avec $Y = 1$ sont tous supérieurs aux scores des individus avec $Y = 0$.

Dans de nombreuses applications, la courbe ROC fournit des informations plus intéressantes sur la qualité de l'apprentissage que le simple taux d'erreur. C'est surtout vrai lorsque les classes sont très déséquilibrées, et lorsque le coût de mauvaise affectation est susceptible de modifications. Il faut néanmoins que l'on ait une classe cible (par exemple Y = 1) clairement identifiée et que la méthode d'apprentissage puisse fournir un `score` proportionnel à $P(Y = 1|X)$.

Il existe plusieurs façons de faire la courbe ROC sous python. On peut utiliser la fonction plot_roc_curve qui se trouve dans le module metrics de scikit-learn ou faire des manipulations avec la librairie plotnine. C'est cette deuxième approche qui sera utilisée.

La courbe ROC est un peu lourde à mettre en place. Dans la pratique, il n'est pas nécessaire de construire explicitement la matrice de confusion, nous procédons de la manière suivante :

1. Calculer le score de chaque individu à l'aide du modèle de prédiction ;
2. Trier le fichier selon un `score décroissant` ;
3. Considérons qu'il n'y a pas d'ex-aequo. Chaque valeur du score être potentiellement un seuil s. Pour toutes les observations dont le score est supérieur ou égal à s, les individus dans la partie haute du tableau, nous pouvons comptabiliser le nombre de positifs $n_+(s)$ et le nombre de négatifs $n_-(s)$. Nous en déduisons $TVP = \dfrac{n_+(s)}{n_+}$ et $TFP = \dfrac{n_-(s)}{n_-}$
4. La courbe ROC correspond au graphique nuage des points qui relie les couples (TVP, TFP). Le premier point est forcément (0,0). Le dernier est (1,1).

Remarque 3.10 *Deux situations extrêmes peuvent survenir :*

— *La discrimination est parfaite. Tous les positifs sont situés devant les négatifs, la courbe ROC est collée aux extrémités Ouest et Nord du repère.*

— *Les scores sont totalement inopérants, le classifieur attribue des valeurs au hasard, dans ce cas les positifs et les négatifs sont mélangés. La courbe ROC se confond avec la première bissectrice.*

```
# False positive rate - True positive rate
from sklearn import metrics as mt
fpr, tpr, threshold1 = mt.roc_curve(DTest.chd,yprobTest,pos_label=1)
roc_df = pd.DataFrame({"fpr" : fpr,"tpr" : tpr})
print((ggplot(roc_df,aes(x="fpr",y="tpr"))+geom_line(color="blue")+
       geom_abline(intercept = 0, slope = 1,color="red")+
       labs(x="Taux de Faux positifs",y="Taux de vrais positifs",
           title="Courbe ROC")))
```

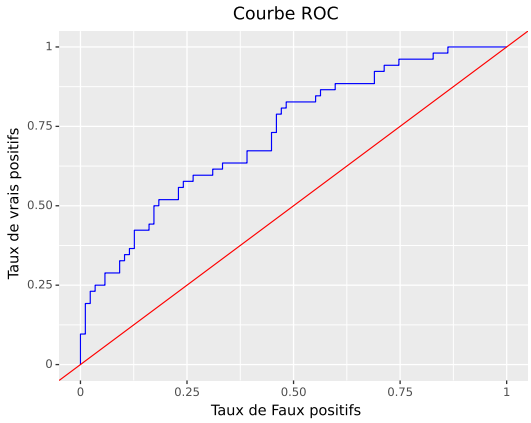


Figure 3.4 – Courbe ROC sur l'échantillon test

Il est possible de caractériser numériquement la courbe ROC en calculant la surface située sous la courbe : c'est **le critère AUC**. Elle exprime `la probabilité de placer un individu positif devant un négatif`. Ainsi, dans le cas d'une discrimination parfaite, les positifs sont sûrs d'être placés devant les négatifs, nous avons $AUC = 1$. A contrario, si le classifieur attribue des scores au hasard, il y a autant de chances de placer un positif devant un négatif que l'inverse, la courbe ROC se confond avec la première bissectrice, nous avons $AUC = 0.5$. C'est la situation de référence, notre classifieur doit faire mieux. On propose généralement différents paliers pour donner un ordre d'idée sur la qualité de la discrimination (*cf.* Tableau 3.7}.

Table 3.7 – Interprétation des valeurs du critère AUC

Valeur de l'AUC	$AUC = 0.5$	$0.7 \leq AUC < 0.8$	$0.8 \leq AUC < 0.9$	$AUC \geq 0.9$
Commentaire	Pas de discrimination	Discrimination acceptable	Discrimination excellente	Discrimination exceptionnelle

Elle indique la probabilité pour que la fonction SCORE place un positif devant un négatif (dans le meilleur des cas AUC = 1 pour le modèle parfait). Si SCORE classe au hasard les individus (c'est-à-dire le modèle de prédiction ne sert à rien), AUC = 0.5 symbolisé par la diagonale principale dans le graphique.

Dans scikit-learn, on peut utiliser la fonction **auc** du module metrics et y introduire le taux de faux positif et le taux de vrai positif.

```
# Aire sous la courbe
area = mt.auc(fpr,tpr)
print('ROC AUC score : %.2f' %(area))
```

```
## ROC AUC score : 0.73
```

ou encore en utilisant la fonction **roc_auc_score** en y introduisant l'échantillon test et les probabilités estimées.

```
# Aire sous la courbe
from sklearn.metrics import roc_auc_score
area2 = mt.roc_auc_score(DTest.chd,yprobTest)
print('ROC AUC score : %.2f' % (area2))
```

```
## ROC AUC score : 0.73
```

Nous avons 73% de chances de placer un positif devant un négatif en « scorant » avec notre classifieur, à comparer avec les 50% de la situation de référence.

Remarque 3.11 (Critère AUC et statistique de Mann-Withney) *Il existe une relation entre la statistique U_+ de Mann-Withney et le critère AUC. La meilleure justification est certainement du côté de l'interprétation de ces quantités sous l'angle des comparaisons par paires. La relation est la suivante :*

$$AUC = \frac{U_+}{n_- \times n_+} \tag{3.24}$$

3.2.4.2 La courbe LIFT

Encore appelée **courbe de gain**, cette courbe représente la proportion de vrais positifs en fonction des individus sélectionnés, lorsqu'on fait varier le seuil. Sa forme dépend du taux de positif a priori. Elle a même ordonnée que la courbe ROC, mais une abscisse généralement plus grande. La courbe LIFT est généralement sous la courbe ROC.

On peut, soit utiliser la fonction plot_cumulative_gain de la librairie **scikitplot** de python afin de pouvoir avoir une une représentation graphique de la courbe LIFT, soit construire la courbe à l'aide des librairies numpy, pandas et plotnine. C'est la deuxième approche qui sera employée. L'inconvénient de cette approche est qu'elle est longue et nécessite plusieurs manipulations.

Tout d'abord, on transforme en 0 et 1 la cible de l'échantillon test et on récupère les éléments de la deuxième colonne qui correspond à l'indice 1.

```
# Transformation en 0 et 1 # Eléments de la 2ème colonne
pos = pd.get_dummies(DTest.chd).values[:,1]
# 10 premières valeurs
print(pos[:10])
```

```
## [False False False False False False  True  True False  True]
```

On calcule le nombre total de positif.

```
# Nombre total de positif
n_pos = np.sum(pos)
print(n_pos)
```

```
## 52
```

On tri selon le score croissant puis on inverse selon le score décroissant.

```
# Index pour tri selon le score croissant
index = np.argsort(yprobTest)[::-1]
print(index[:10])
```

```
## row.names
## 191     60
## 7       91
## 279     46
## 192    120
## 349     76
## 176     40
## 376    102
## 459    108
## 348     18
## 148     57
## dtype: int64
```

On tri les individus.

```
# Trie des individus
sort_pos = pos[index]
print(sort_pos[:10])
```

```
## [ True  True  True  True  True False  True  True  True  True]
```

On fait la somme cumulée

```
# Somme cumulée
cum_pos = np.cumsum(sort_pos)
print(cum_pos[:10])
```

```
## [1 2 3 4 5 5 6 7 8 9]
```

On calcule le rappel

```
# Rappel
rappel = cum_pos/n_pos
print(rappel[:10])
```

```
## [0.01923077 0.03846154 0.05769231 0.07692308 0.09615385 0.09615385
##  0.11538462 0.13461538 0.15384615 0.17307692]
```

On définit la taille de l'échantillon test et celle de la cible.

```
# Taille d'échantillon test
n = DTest.shape[0]
# Taille de cible - sequence de valeurs
taille =  np.arange(1,n+1,1)
```

On fait passer le rappel en proportion

```
# Passer en proportion
taille = taille/n
print(taille[:10])
```

```
## [0.00719424 0.01438849 0.02158273 0.02877698 0.03597122 0.04316547
##  0.05035971 0.05755396 0.0647482  0.07194245]
```

pour finalement avoir le graphique suivant :

```
lift_df = pd.DataFrame({"taille":taille,"rappel" : rappel})
print((ggplot(lift_df,aes(x="taille",y="rappel"))+geom_line(color="green")+
       geom_abline(intercept = 0, slope = 1,color="red")+
       labs(x="Taille de la cible en %",y = 'Proportion de "+" retrouvés en %',
            title="Courbe LIFT")))
```

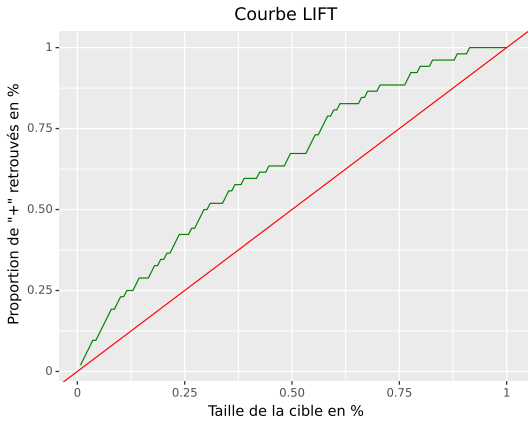


Figure 3.5 – Courbe LIFT sur l'échantillon test.

Remarque 3.12 (Aire sous la courbe LIFT) *Il existe un lien entre les aires sous les courbes LIFT et ROC. L'aire sous la courbe LIFT, notée AUL, s'exprime simplement à priori par :*

$$AUL = \frac{p}{2} + (1-p)AUC \tag{3.25}$$

où p probabilité a priori de l'évènement Y = 1 dans la population.

```
# proportion des positifs
prop = n_pos/len(pos)
# Aire AUL
area_lift = (prop/2)+(1-prop)*area
print('LIFT AUL score : %.2f' % (area_lift))
```

```
## LIFT AUL score : 0.64
```

3.2.4.3 La courbe rappel-prévision

La courbe rappel-précision est très utilisée en recherche d'information (en anglais, *information retrieval*). Suite à une requête, nous obtenons un ensemble d'individus que nous appellerons la « cible », nous sommes face à deux exigences contradictoires : nous aimerions retrouver une fraction élevée des positifs potentiels (`rappel`) ; nous aimerions que la cible ne contienne que des positifs (`précision`). La courbe traduit l'arbitrage entre ces deux critères lorsque l'on fait varier le seuil d'affectation s.

Elle est conceptuellement proche de la courbe ROC. Pour chaque valeur de s, nous formons (virtuellement) la matrice de confusion et nous calculons les deux indicateurs. Il y a quand même une différence très importante. La précision étant un « profil-colonne » de la matrice de confusion, il faut donc travailler sur un échantillon représentatif (la proportion des positifs $\frac{n_+}{n}$ doit être le reflet de la probabilité d'être positif p) pour pouvoir l'exploiter convenablement. Si cette condition est respectée, elle paraît plus adaptée que la courbe ROC lorsque les classes sont très déséquilibrées (la proportion des positifs est très faible), notamment pour différencier le comportement des algorithmes d'apprentissage supervisé.

Pour élaborer la courbe rappel-précision, nous procédons comme suit :

1. Calculer le score de chaque individu ;
2. Trier les données selon un score décroissant ;
3. Admettons qu'il n'y a pas d'ex-aequo, chaque valeur du score est un outil potentiel s. Pour les individus situés dans la partie haute du tableau c'est-à-dire dont le score est supérieur ou égal à s, il s'agit de la cible, nous comptabilisons le nombre de positifs $n_+(s)$ et le nombre total d'observations $n(s)$.
4. Nous en déduisons le $rappel(s) = \frac{n_+(s)}{n_+}$ et la $prcision(s) = \frac{n_+(s)}{n(s)}$

Remarque 3.13 *Dans les parties hautes du tableau, lorsque le seuil est élevé, la taille de la cible sera réduite. La précision sera forte, dans la cible ne seront présents que des positifs ; mais le rappel sera faible, une faible fraction de l'ensemble des positifs y sont inclus. A mesure que s diminue, la taille de la cible augmente, elle sera de plus en plus polluée (la précision diminue) mais intégrera une plus grande fraction des positifs (le rappel augmente). La courbe est donc globalement décroissante, mais elle n'est pas forcément monotone.*

On construit le graphique suivant :

```
# Precsion - recall curve
precision, recall, threshold2 = mt.precision_recall_curve(DTest.chd,yprobTest)
pr_df = pd.DataFrame({"precision" : precision, "rappel" : recall})
print((ggplot(pr_df,aes(x="precision",y="rappel"))+geom_line(color="blue")+
        geom_point(color="blue")+
        labs(x="Rappel",y="Précision",title="Recall vs Precision Curve")))
```

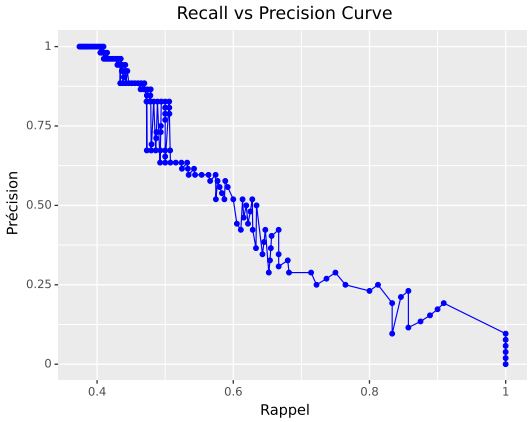


Figure 3.6 – Courbe rappel - précision

4 Analyse des résidus

Sommaire

L'analyse des résidus permet de diagnostiquer la qualité de la régression. Plusieurs questions se posent à l'issue du processus de modélisation, nous devons y apporter des réponses :

- Déterminer les points qui « clochent » dans les données, qui s'écartent fortement des autres dans l'espace de représentation. On parle de données « atypiques ».
- Déterminer les points qui sont mal modélisés (mal expliqués) par la régression logistique. On parle de résidus.
- Déterminer les points qui pèsent fortement dans la régression. On parle de points « leviers ».
- Déterminer les points qui pèsent exagérément sur les résultats. Si on les retirait de l'ensemble d'apprentissage, le modèle obtenu serait très différent. On parle de points « influents ».

On se donne M_β un modèle logistique. Pour simplifier les notations, on écrira $p_i = p_\beta(x_i)$ et $\hat{p}_i = p_{\widehat{\beta}}(x_i)$.

Pour illustrer nos propos, reprenons l'exemple de détection de la maladie cardiaque :

```
# load dataset
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
donnee = pd.read_csv('./donnee/deseases.txt',sep=',', header=0,index_col=0)
donnee["famhist"] = donnee["famhist"].astype('category')
formule = create_formula("chd",donnee.drop(['chd'],axis=1).columns)
model = smf.logit(formule,data=donnee).fit(disp=False)
coef_table = model.summary2().tables[1]
```

Table 4.1 – Estimation du modèle $chd = f(sbp, tobacco, age, ...)$

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	-6.1507	1.3083	-4.7015	0.0000	-8.7149	-3.5866
famhist[T.Present]	0.9254	0.2279	4.0605	0.0000	0.4787	1.3720
sbp	0.0065	0.0057	1.1350	0.2564	-0.0047	0.0177
tobacco	0.0794	0.0266	2.9838	0.0028	0.0272	0.1315
ldl	0.1739	0.0597	2.9152	0.0036	0.0570	0.2909
adiposity	0.0186	0.0293	0.6346	0.5257	-0.0388	0.0760
typea	0.0396	0.0123	3.2138	0.0013	0.0154	0.0637
obesity	-0.0629	0.0442	-1.4218	0.1551	-0.1496	0.0238
alcohol	0.0001	0.0045	0.0271	0.9784	-0.0087	0.0089
age	0.0452	0.0121	3.7285	0.0002	0.0215	0.0690

4.1 Les différents types de résidus

A l'image de la régression linéaire plusieurs types de résidus sont proposés par les logiciels. Le premier, le plus simple à calculer est tout simplement $Y_i - \hat{p}_i$. Ces résidus sont appelés *résidus bruts*. Ils permettent de mesurer l'ajustement du modèle sur chaque observation. Ces résidus n'ayant pas la même variance, ils sont difficiles à comparer. En effet, on rappelle que $\mathrm{V}_\beta(Y|X = x_i) = p_i(1 - p_i)$. Par conséquent, la variance de tels résidus risquent d'être élevées pour des valeurs de p_i proches de $1/2$. Un moyen de pallier à cette difficulté est de considérer les **résidus de Pearson** :

$$RP_i = \frac{y_i - \hat{p}_i}{\sqrt{\hat{p}_i(1 - \hat{p}_i)}} \tag{4.1}$$

```
# Résidu de Pearson
def pearson_resid(y,p):
  return (y - p)/np.sqrt(p*(1-p))
# Application
yprob = model.predict(donnee)
resid_p = pearson_resid(donnee.chd,yprob).to_frame("r.pears")
# Représentation graphique
from plotnine import *
print((ggplot(resid_p,aes(x=donnee.index,y="r.pears"))+
       geom_point(color="black")+labs(x="index",y="Residu de Pearson")))
```

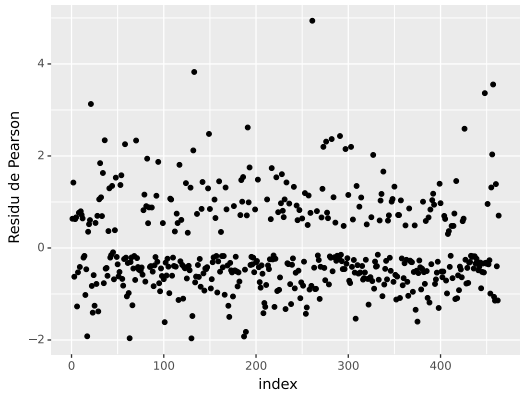


Figure 4.1 – Résidus de Pearson

Par défaut, l'outil de Statsmodels nous renvoie les résidus de Pearson grâce à l'attribut model.resid_pearson.

Par définition, on standardise les résidus par la variance théorique de Y_i. Cependant, comme $\widehat{p}_i$ est aléatoire, il est évident que $V_\beta(Y_i - p_i) \neq V_\beta(Y_i - \widehat{p}_i)$. En effet, en notant

$$\begin{cases} \varepsilon_i &= y_i - p_i \\ \widehat{\varepsilon}_i &= y_i - \widehat{p}_i \end{cases}$$

On a :

Hypothèses	Réalité
$\mathbb{E}(\varepsilon_i) = 0$	$\mathbb{E}(\widehat{\varepsilon}_i) \simeq 0$
$V(\varepsilon) = p_i(1-p_i)$	$V(\widehat{\varepsilon}) \simeq p_i(1-p_i)(1-h_{ii})$

où h_{ii} est l'élément de la $i^{\text{ème}}$ ligne et de la $i^{\text{ème}}$ colonne de la matrice $H = X(X^{'}W_{\widehat{\beta}}X)^{-1}X^{'}W_{\widehat{\beta}}$.

Il est par conséquent intéressant de considérer la version standardisée des résidus de Pearson :

$$rp_i^{stand} = \frac{y_i - \widehat{p}_i}{\sqrt{\widehat{p}_i(1-\widehat{p}_i)(1-h_{ii})}} \tag{4.2}$$

Ces résidus seront en effet plus facile à analyser (leur distribution étant « presque » centrée réduite).

```
# Residus de Pearsonb standardisés
def spearson_resid(y,p,h):
  return pearson_resid(y,p)/np.sqrt((1-h))
# Application
infl = model.get_influence()
hii = infl.hat_matrix_diag
resid_sp = spearson_resid(donnee.chd,yprob,hii).to_frame("r.spearson")
# Représentation graphique
print((ggplot(resid_sp, aes(x=donnee.index,y="r.spearson"))+
   geom_point(color="black")+labs(x="index",y="Residu de Pearson standardisé")))
```

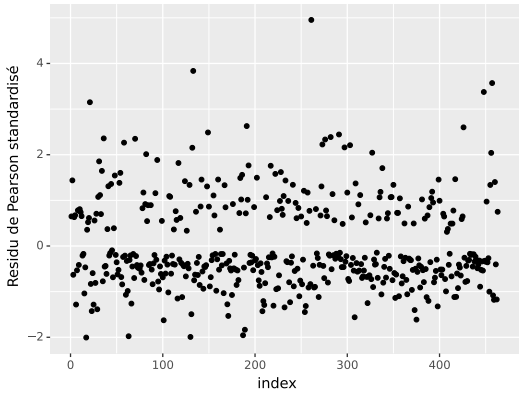


Figure 4.2 – Résidu de Pearson standarddisé

Les **résidus de déviance** sont définis par :

$$rd_i = \text{signe}(y_i - \hat{p}_i)\sqrt{2\mathcal{L}_1(Y_i, \hat{\beta})} = \text{signe}(Y_i - \hat{p}_i)\sqrt{2\left(Y_i \log \hat{p}_i + (1 - Y_i)\log(1 - \hat{p}_i)\right)} \tag{4.3}$$

Ceci est équivalent à :

$$rd_i = \begin{cases} +\sqrt{2 \times |\ln(\hat{p}_i)|} & \text{si} \quad y_i = 1 \\ -\sqrt{2 \times |\ln(1 - \hat{p}_i)|} & \text{si} \quad y_i = 0 \end{cases} \tag{4.4}$$

Statsmodels nous renvoie les résidus de dévisance grâce à l'attribut model.resid_dev

```
# Résidus de déviance
resid_dev = model.resid_dev.to_frame("r.dev")
# Représentation graphique
print((ggplot(resid_dev, aes(x=donnee.index,y="r.dev"))+
              geom_point(color="black")+
              labs(x="index",y="Residu de déviance")))
```

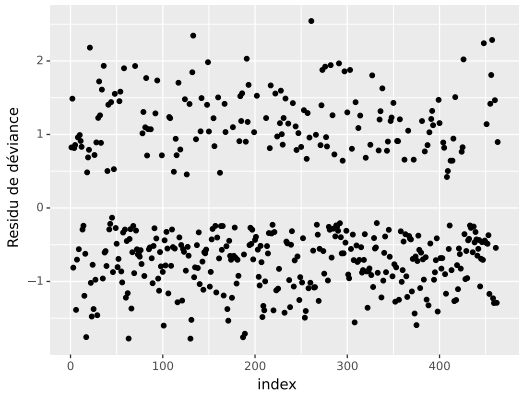


Figure 4.3 – Résidu de déviance

Nous pouvons en déduire la statistique D appelée **déviance** :

$$D = \sum_{i=1}^{i=n} rd_i^2 \tag{4.5}$$

Sur les données individuelles, la déviance ainsi calculée coïncide avec la déviance du modèle M_β.

```
# Déviance
D = np.sum(np.square(resid_dev))
print("La déviance vaut %.2f"%(D))
```

```
## La déviance vaut 472.14
```

Là encore pour tenir compte de la variabilité ces résidus sont standardisés :

$$rd_i^{stand} \frac{rd_i}{\sqrt{1-h_{ii}}} \tag{4.6}$$

```
# Résidus de déviance standardisé
resid_sdev = (resid_dev["r.dev"]/np.sqrt((1-hii))).to_frame("r.sdev")
# Représentation graphique
print((ggplot(resid_sdev, aes(x=donnee.index,y="r.sdev"))+
               geom_point(color="black")+
               labs(x="index",y="Residu de déviance standardisé")))
```

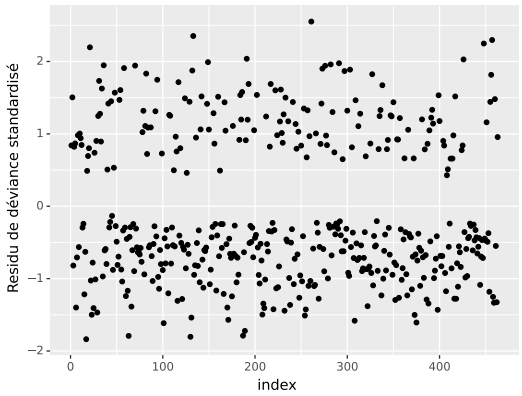


Figure 4.4 – Résidu de déviance standardisé

Ces deux types de résidus de déviance sont ceux qui sont en général conseillés.

4.2 Examen des résidus

4.2.1 Index plot

Pour la régression logistique, les résidus de déviance sont souvent préférés. De nombreuses études expérimentales ont montré qu'ils approchent mieux la loi normale que les résidus de Pearson. Pour cette raison, ces résidus prennent généralement des valeurs qui varient entre -2 et 2. Nous pourront construire un *index plot* pour détecter des valeurs aberrantes. Ce graphique ordonne les résidus en fonction du numéro de leur observation. Les points pour lesquels on observe un résidu élevé (hors de [-2,2] par exemple) devront faire l'objet d'une étude approfondie.

```
# Résidus de déviance
resid_dev = model.resid_dev.to_frame("r.dev")
# Représentation graphique
print((ggplot(resid_dev, aes(x=donnee.index,y="r.dev"))+
               geom_point(color="black")+geom_hline(yintercept=(-2,2),color="red")+
               labs(x="index",y="Residu de déviance")))
```

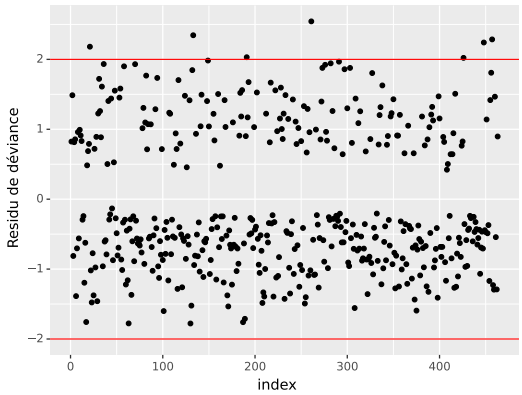


Figure 4.5 – Index plot

4.2.2 Graphique prédiction linéaire-résidus

Ce graphique qui représente $X'\hat{\beta}$ en abscisse et $\hat{\varepsilon}$ en ordonné permet de détecter les valeurs aberrantes mais aussi les structurations suspectes. Si une structuration suspecte apparaît, il sera peut être adéquat d'ajouter une nouvelle variable afin de prendre en compte cette structuration. Dans le cas des données individuelles, ce type de graphique donne toujours des structurations (*cf.* Figure 4.6) et n'est donc pas conseiller.

```
# Donnee
data = pd.DataFrame({"logit":model.fittedvalues,"rstudent":model.resid_pearson})
# Représentation graphique
print((ggplot(data, aes(x="logit",y="rstudent"))+
              geom_point(color="black")+
              labs(x="prévision linéaire",y="Résidus studentisés par VC")))
```

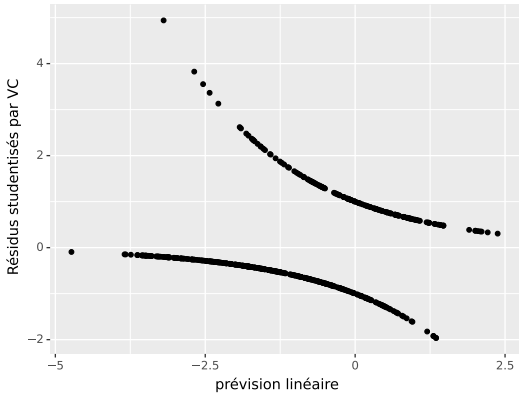


Figure 4.6 – Graphique prédiction linéaire/résidus

4.2.3 Résidus partiels

Les résidus partiels sont définis par :

$$\hat{\varepsilon}_j^P = \frac{y_i - \hat{p}_i}{\hat{p}_i(1-\hat{p}_i)} + \hat{\beta}_j X_j \tag{4.7}$$

L'analyse consiste à tracer pour toutes les variables j les points avec en abscisse la variable j et en ordonnée les résidus partiels. Si le tracé est linéaire alors tout est normal. Si par contre une tendance non linéaire se dégage, il faut remplacer la variable j par une fonction de celle ci donnant la même tendance que celle observée.

```
# Résidus partiels
from statsmodels.nonparametric.smoothers_lowess import lowess

def partial_resid_plot(res,p,var=str,ax=None,color="blue"):
  exog = pd.DataFrame(res.model.exog,columns=res.model.exog_names)
  resid = (res.resid_response/p*(1-p))+res.params[var]*exog[var].values
  # Représentation graphique
  if ax is None:
    ax = plt.gca()
  x, y = exog[var].values, resid.values
  filtered = lowess(y,x)  # Lowess regression
  ax.scatter(x,y,color=color);
  ax.plot(filtered[:,0],filtered[:,1],color='black');
  ax.set(xlabel=var,ylabel=f"Residual + beta*"+str(var));
# Application
columns = donnee.drop(columns=["chd","famhist"],axis=1).columns
fig = plt.figure(figsize=(16,8))
for i, col in enumerate(columns):
  ax = fig.add_subplot(2,4,i+1)
  partial_resid_plot(model,yprob,col,ax)
fig.tight_layout()
plt.show()
```

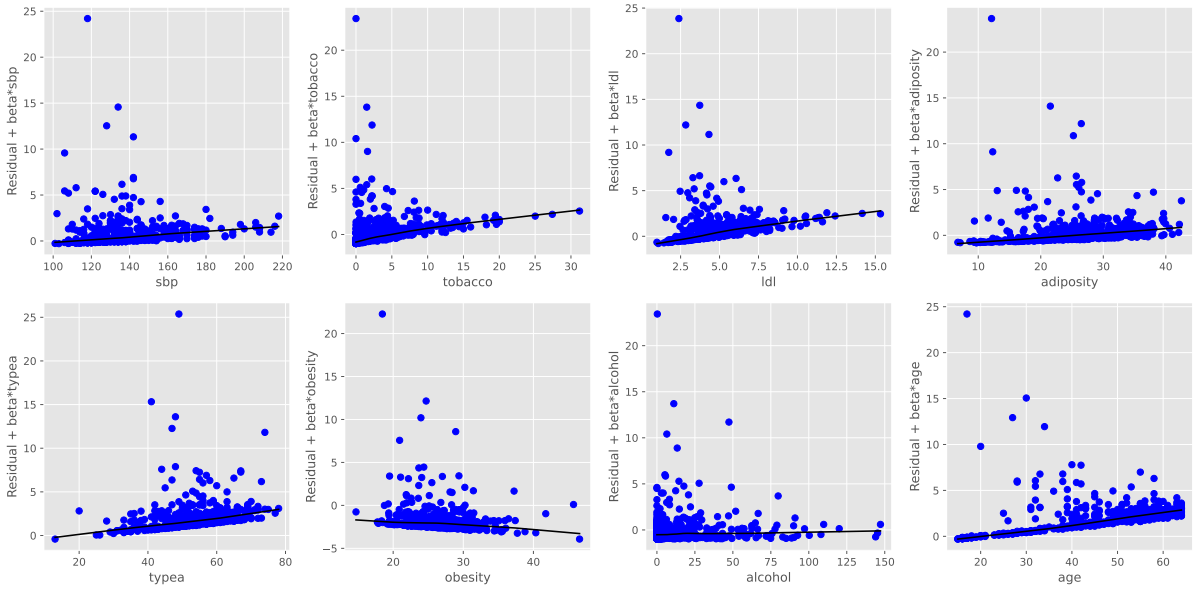


Figure 4.7 – Résidus partiels sur les variables quantitatives.

Dans la figure 4.7 Le trait noir représente le résumé lissé des données par l'estimateur loess.

Mallows (1986) propose d'utiliser les résidus partiels augmentés qui dans certaines situations permettent de mieux dégager cette tendance. Les résidus partiels augmentés pour la $j^{\text{ème}}$ variable nécessitent un nouveau modèle logistique identique mis à part le fait qu'une variable explicative supplémentaire est ajoutée : $X_{p+1} = X_j^2$ la $j^{\text{ème}}$ variable élevée au carré. Le nouveau vecteur de coefficient β du modèle est estimé et les résidus partiels sont alors définis comme :

$$\hat{\varepsilon}_{.j}^{PA} = \frac{Y_i - \hat{p}_i}{\hat{p}_i(1-\hat{p}_i)} + \hat{\beta}_j X_j + \hat{\beta}_{p+1} X_j^2$$

L'analyse des diagrammes est identique à ceux des résidus partiels.

4.3 Points leviers et points influents

Ces notions sont analogues à celles du modèle linéaire (*cf.* Cornillon et al. (2019), chapitre 4).

4.3.1 Points leviers

Par définition les points leviers sont les points du design qui déterminent très fortement leur propre estimation. Nous avons vu que l'algorithme d'estimation des paramètres effectue à chaque étape une régression linéaire et s'arrête lorsque le processus devient stationnaire :

$$\hat{\beta} = (X' W_{\hat{\beta}} X)^{-1} X' W_{\hat{\beta}} Z$$

et la prédiction linéaire est :

$$X\hat{\beta} = X(X' W_{\hat{\beta}} X)^{-1} X' W_{\hat{\beta}} Z = HZ$$

où H est une matrice de projection selon la métrique $W_{\hat{\beta}}$. Comme nous transformons $X\hat{\beta}$ par une fonction monotone, des $X\hat{\beta}$ extrêmes entraînent des valeurs de $\hat{p}$ extrêmes. Nous allons donc utiliser la même méthode de diagnostic que celle de la régression simple avec une nouvelle matrice de projection H. Pour la $i^{\text{ème}}$ prédiction linéaire nous avons :

$$[X\hat{\beta}]_i = H_{ii} Z_i + \sum_{j \neq i} H_{ij} Z_j$$

Si H_{ii} est grand relativement aux H_{ij}, $j \neq i$ alors la $i^{\text{ème}}$ observation contribue fortement à la construction de $[X\hat{\beta}]_i$. On dira que le *poids* de l'observation i sur sa propre estimation vaut H_{ii}.

Comme H est un projecteur nous savons que $0 \leq H_{ii} \leq 1$. Nous avons les cas extrêmes suivants :

— Si $H_{ii} = 1$, $\hat{p}_i$ est entièrement déterminé par Y_i car $H_{ij} = 0$ pour tout *j*.
— Si $H_{ii} = 0$, Y_i n'a pas d'influence sur $\hat{p}_i$.

La trace d'un projecteur étant égale à la dimension du sous espace dans lequel on projette, on a $tr(H) = \sum_i H_{ii} = p$. Donc en moyenne H_{ii} vaut p/n. Pour dire que la valeur de H_{ii} contribue trop fortement à la construction de $\hat{p}_i$, il faut un seuil au delà duquel le point est un point levier. Par habitude, si $H_{ii} > 2p/n$ ou si $H_{ii} > 3p/n$ alors le $i^{\text{ème}}$ point est déclaré comme un point levier.

En pratique un tracé de H_{ii} est effectué et l'on cherche les points dont le H_{ii} est supérieur à $3p/n$ ou $2p/n$. Ces points sont `leviers` et leur valeur influe fortement sur leur propre prévision.

```
# Points leviers
n, p = donnee.shape
data = pd.DataFrame({"index" : donnee.index,"h":hii})
print((ggplot(data,aes(x="index",y="h"))+
               geom_bar(stat="identity",color="black",width=0.005)+
               geom_hline(yintercept=(2*p/n,3*p/n),color=("green","blue"))+
               labs(x="index",y="$h_{ii}$")))
```

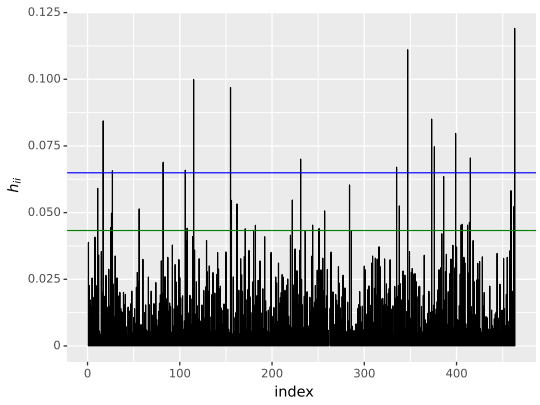


Figure 4.8 – Points leviers

4.3.2 Points influents

Les points influents sont des points qui influent sur le modèle de telle sorte que si on les enlève, alors l'estimation des coefficients sera fortement changée. La mesure la plus classique d'influence est `la distance de Cook`. Il s'agit d'une distance entre le coefficient estimé avec toutes les observations et celui estimé avec toutes les observations sauf une. La distance de Cook pour l'individu i est définie par :

$$D_i = \frac{1}{p}(\hat{\beta}_{(i)} - \hat{\beta})' X' W_{\hat{\beta}} X (\hat{\beta}_{(i)} - \hat{\beta}) \approx \frac{r_{P_i}^2 H_{ii}}{p(1 - H_{ii})^2}$$

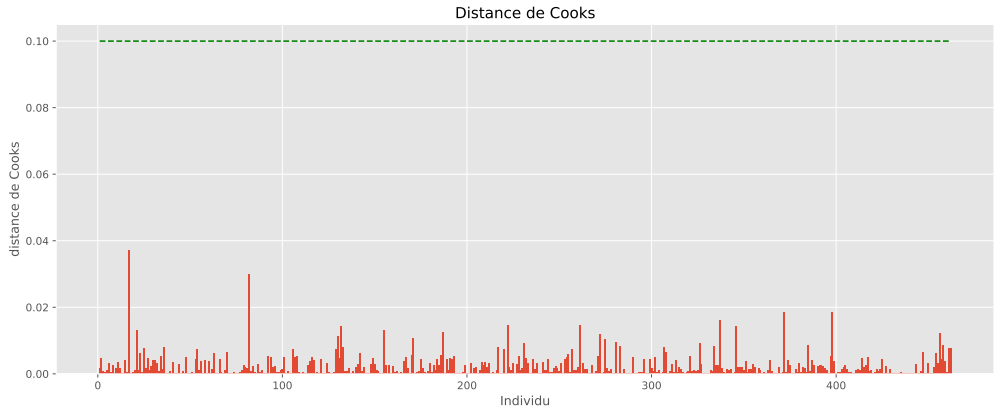
où r_{P_i} est le résidu de Pearson pour le $i^{\text{ème}}$ individu.

Les distances de Cook sont généralement représentées comme sur la figure **??**. Si une distance se révèle grande par rapport aux autres, alors ce point sera considéré comme influent. Il convient alors de comprendre pourquoi il est influent, soit :

— il est levier ;
— il est aberrant ;
— (les deux !)

Dans tous les cas il convient de comprendre si une erreur de mesure, une différence dans la population des individus est à l'origine de ce phénomène. Eventuellement pour obtenir des conclusions robustes il sera bon de refaire l'analyse sans ce(s) point(s).

```
# Distance de Cooks
x = np.linspace(1,n,n)
yok = np.repeat(0.1,n)
cd, pval = infl.cooks_distance
fig, ax = plt.subplots(figsize = (16,6))
ax.bar(x,cd);
ax.plot(x,yok,linestyle='dashed',color='green');
ax.set(xlabel="Individu",ylabel="distance de Cooks",title="Distance de Cooks");
plt.show()
```



5 Régression logistique multinomiale

Sommaire

Lorsque la variable dépendante prend $K\,(K > 2)$ modalités, nous sommes dans le cadre de la régression logistique polytomique. Dans ce chapitre, nous considérons qu'elle est nominale c'est - à - dire il n'y a pas de relation d'ordre entre les modalités, on parle de régression logistique multinomiale. Elle peut être vue comme une généralisation de la régression logistique binaire.

5.1 Le modèle polytomique nominal

5.1.1 Modèle polytomique nominal

C'est le modèle utilisé par défaut lorsque les différentes modalités de Y n'ont pas de hiérarchie entre elles. On s'inspire du modèle logistique binaire :

$$\begin{cases} \mathbb{P}(Y=0|X) = \dfrac{1}{1+e^{\beta X}} \\ \mathbb{P}(Y=1|X) = \dfrac{e^{\beta X}}{1+e^{\beta X}} \end{cases}$$

d'où l'on tire :

$$\mathbb{P}(Y=1|X) = p(X) = e^{\beta X}\mathbb{P}(Y=0|X) \tag{5.1}$$

On peut étendre ce modèle au cas où Y est à valeurs dans $\{0, 1, \dots, K\}$ en prenant (par exemple) $Y = 0$ comme modalité de référence. On aboutit au modèle spécifié par :

$$\mathbb{P}(Y=k|X) = p_{\beta}^{(k)}(X) = e^{\beta^{(k)}X}\mathbb{P}(Y=0|X) \tag{5.2}$$

où $\beta^{(k)}$ est le vecteur des $p+1$ coefficients des régresseurs pour la modalité $k \geq 1$ de l'output Y. Comme la somme des probabilités de sortie vaut 1, on trouve :

$$\begin{cases} \mathbb{P}(Y=0|X) = p_\beta^{(0)}(X) = \dfrac{1}{\sum_{r=0}^{r=K} e^{\beta^{(r)}X}} \\ \mathbb{P}(Y=k|X) = p_\beta^{(k)}(X) = \dfrac{e^{\beta^{(k)}X}}{\sum_{r=0}^{r=K} e^{\beta^{(r)}X}} \end{cases}$$

Remarque 5.1 — $K=1$ *ramène au modèle logistique binaire.*

— *La première équation se ramène à la seconde en posant* $\beta^{(0)}=0$ *(vecteur nul). En d'autres termes,* $Y=0$ *a été pris comme modalité de référence.*

On en déduit que ce choix modifie les valeurs des paramètres du modèle, mais pas la nature du modèle en lui-même. En particulier, les valeurs estimées de $\mathbb{P}(Y=k|X)$ ne dépendent pas du choix de la modalité de référence.

Remarquons que pour deux modalités j et k avec $j \neq k$, on déduit l'écriture du modèle que :

$$\frac{\mathbb{P}(Y=k/X)}{\mathbb{P}(Y=j/X)} = e^{(\beta^{(k)}-\beta^{(j)})X}$$

Il y a $\dfrac{p+1}{K}$ paramètres à estimer dans ce modèle, ce qui peut vite faire beaucoup.

Les exemples d'application des modèles de régression logistique multinomiale sont nombreuses :

1. Les choix professionnels des gens peuvent être influencés par la profession de leurs parents et par leur propre niveau d'éducation. Nous pouvons étudier la relation entre le choix professionnel d'une personne, son niveau d'éducation et la profession du père. Les choix professionnels seront la variable de réponse qui se compose de catégories de professions.
2. Un biologiste peut être intéressé par les choix alimentaires que font les alligators. Les alligators adultes peuvent avoir des préférences différentes de celles des jeunes. La variable de résultat ici sera les types de nourriture, et les variables prédictives pourraient être la taille des alligators et d'autres variables environnementales.
3. Les étudiants qui entrent au secondaire font des choix de programme parmi un programme général, un programme professionnel et un programme académique. Leur choix pourrait être modélisé en fonction de leur score d'écriture et de leur statut socio-économique.

Dans ce chapitre, nous travaillerons sur une version de la base WAVEFORM où la variable cible « classe » comporte 3 modalités $\{A, B, C\}$. Ce jeu de données a été utilisé par Ricco Rakatomalala comme travaux dirigés [1] sur la régression logistique multinomiale.

1. https ://eric.univ-lyon2.fr/ricco/cours/regression_logistique/Regression_Logistique_TD_8.html

```
# Chargement des données
import pandas as pd
waveform = pd.read_excel("./donnee/waveform.xlsx")
```

Table 5.1 – 10 premières observations - Données WAVEFORM

V05	V07	V10	V11	V12	V15	V18	classe
1.54	4.48	1.51	1.09	1.32	2.25	0.53	A
0.95	2.71	2.09	1.50	1.09	2.85	1.87	A
2.48	3.16	2.06	2.24	2.87	3.42	1.58	A
4.35	6.64	3.57	1.47	0.95	1.13	-1.43	A
4.04	4.34	2.61	0.62	1.81	-0.14	-0.77	A
3.89	3.67	2.62	2.73	1.61	2.21	0.20	A
2.70	3.66	3.00	3.70	3.19	4.26	1.95	A
2.53	3.51	2.12	2.23	2.81	2.92	3.11	A
2.68	3.67	2.91	1.45	1.18	1.24	0.51	A
1.44	2.59	2.11	1.58	-0.72	3.66	0.36	A

Nous affichons les informations sur nos colonnes :

```
# Informations sur les colonnes
print(waveform.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 500 entries, 0 to 499
## Data columns (total 8 columns):
##  #   Column  Non-Null Count  Dtype
## ---  ------  --------------  -----
##  0   V05     500 non-null    float64
##  1   V07     500 non-null    float64
##  2   V10     500 non-null    float64
##  3   V11     500 non-null    float64
##  4   V12     500 non-null    float64
##  5   V15     500 non-null    float64
##  6   V18     500 non-null    float64
##  7   classe  500 non-null    object
## dtypes: float64(7), object(1)
## memory usage: 31.4+ KB
## None
```

Le tableau 5.2 donne les distributions de notre variable cible « origin » en termes d'effectifs et de proportions (ou probabilité a priori) :

```
# Distribution
waveform["classe"] = waveform["classe"].astype("category")
n_k = waveform.classe.value_counts()
p_k = waveform.classe.value_counts(normalize=True)
stats = pd.concat([n_k,p_k],axis=1).reset_index()
```

5.1.2 Estimation des paramètres

Les paramètres $\beta^k, k = 1, \ldots, K-1$ sont estimés par maximum de vraisemblance. La log-vraisemblance s'écrit :

Table 5.2 – Distribution de la variable cible - Données WAVEFORM

classe	count	proportion
A	173	0.346
C	170	0.340
B	157	0.314

$$\ln L(Y,\beta) = \sum_{i=1}^{i=n}\sum_{k=0}^{K} 1_{\{Y_i=k\}}\left(\beta^{(r)}X_i - \ln\sum_{r=0}^{K} e^{\beta^{(r)}X_i}\right) \tag{5.3}$$

On en déduit l'EMV des $\beta_j^{(k)}$ par des méthodes numériques. L'EMV a les mêmes propriétés que le modèle logistique (Intervalle de confiance, test de significativité et adéquation analogues).

Exemple 5.1 *Estimation de la classe des wave*

Pour notre analyse, nous souhaitons travailler sur des données centrer et réduite.

```
# Centrer - réduire
from sklearn.preprocessing import StandardScaler
y,x= waveform.classe, waveform[waveform.columns[:-1]]
sc = StandardScaler(with_mean=True,with_std=True)
sc.set_output(transform="pandas");
xc = sc.fit_transform(x)
donnee = pd.concat([y,xc],axis=1)
```

Nous pouvons à présent estimer les paramètres de notre modèle :

```
# Importation des classes de calcul
import statsmodels.api as sm
model = sm.MNLogit(y,sm.add_constant(xc)).fit(disp=False)
print(model.summary2())
```

```
##                          Results: MNLogit
## ================================================================
## Model:              MNLogit          Method:           MLE
## Dependent Variable: classe           Pseudo R-squared: 0.674
## Date:               2023-09-22 09:38 AIC:              389.6253
## No. Observations:   500              BIC:              457.0590
## Df Model:           14               Log-Likelihood:   -178.81
## Df Residuals:       484              LL-Null:          -548.87
## Converged:          1.0000           LLR p-value:      7.0183e-149
## No. Iterations:     8.0000           Scale:            1.0000
## ----------------------------------------------------------------
## classe = 0   Coef.   Std.Err.     t     P>|t|    [0.025   0.975]
## ----------------------------------------------------------------
##    const    -0.6740    0.3789  -1.7785  0.0753  -1.4167   0.0688
##      V05    -0.2867    0.3152  -0.9096  0.3630  -0.9044   0.3310
##      V07     0.7064    0.3278   2.1548  0.0312   0.0639   1.3490
```

```
##      V10     1.3435    0.3010   4.4641  0.0000   0.7536   1.9334
##      V11     1.7895    0.3077   5.8156  0.0000   1.1864   2.3925
##      V12     1.0512    0.3225   3.2594  0.0011   0.4191   1.6832
##      V15    -1.7309    0.3591  -4.8205  0.0000  -2.4347  -1.0271
##      V18    -1.0404    0.2762  -3.7664  0.0002  -1.5818  -0.4990
## ------------------------------------------------------------------
## classe = 1   Coef.   Std.Err.      t     P>|t|    [0.025   0.975]
## ------------------------------------------------------------------
##    const    -0.5643    0.3713  -1.5196  0.1286  -1.2920   0.1635
##      V05    -2.1808    0.3355  -6.5006  0.0000  -2.8383  -1.5233
##      V07    -1.2247    0.3544  -3.4557  0.0005  -1.9193  -0.5301
##      V10     0.8134    0.2810   2.8943  0.0038   0.2626   1.3642
##      V11     1.3957    0.2990   4.6678  0.0000   0.8097   1.9817
##      V12     1.3660    0.2985   4.5756  0.0000   0.7809   1.9511
##      V15    -0.0709    0.3145  -0.2255  0.8216  -0.6873   0.5455
##      V18    -0.2276    0.2324  -0.9792  0.3275  -0.6831   0.2280
## ==================================================================
```

Remarque 5.2 *Par défaut, Python prend comme modalité de référence la première lettre suivant l'ordre alphabétique (ou la plus petite valeur suivant l'ordre numérique). Par conséquent, la modalité $Y = A$ est prise comme modalité de référence. Notre modèle à estimer s'écrit :*

$$\begin{cases} \ln\left(\dfrac{\mathbb{P}(Y=B|X=x)}{\mathbb{P}(Y=A|X=x)}\right) = & \beta_0^B + \beta_1^B \times V05 + \cdots + \beta_6^B \times V15 \\ \ln\left(\dfrac{\mathbb{P}(Y=C|X=x)}{\mathbb{P}(Y=A|X=x)}\right) = & \beta_0^C + \beta_1^C \times V05 + \cdots + \beta_6^C \times V15 \end{cases} \tag{5.4}$$

Nous avons les modèles (logit) estimés suivants :

$$\begin{cases} \ln\left(\dfrac{\mathbb{P}(Y=B|X=x)}{\mathbb{P}(Y=A|X=x)}\right) = & -0.6741 - 0.2870 \times V05 + \cdots - 1.737 \times V15 \\ \ln\left(\dfrac{\mathbb{P}(Y=C|X=x)}{\mathbb{P}(Y=A|X=x)}\right) = & -0.5643 - 2.1830 \times V05 + \cdots - 0.0710 \times V15 \end{cases}$$

Pour vérifier la concordance de nos résultats, nous affichons les résultats obtenus avec le logiciel R et la librairie nnet.

```
# Multinomial logistic regression ----------------------------------------
library(nnet)
mydata = py$donnee
mydata$classe <- relevel(mydata$classe, ref = 'A') # modalité de ref
model <- multinom(formula = 'classe~.', data=mydata)
```

```
## # weights:  27 (16 variable)
## initial  value 549.306144
## iter  10 value 206.201665
## iter  20 value 179.334041
## final  value 178.812650
## converged
```

```
print(summary(model))
```

```
## Call:
## multinom(formula = "classe~.", data = mydata)
##
## Coefficients:
##   (Intercept)        V05        V07       V10      V11      V12         V15
## B  -0.6740562 -0.2866797  0.7064434 1.3435203 1.789443 1.051182 -1.73091968
## C  -0.5643315 -2.1807949 -1.2247084 0.8133882 1.395664 1.365960 -0.07089012
##          V18
## B -1.0403662
## C -0.2275803
##
## Std. Errors:
##   (Intercept)       V05       V07       V10       V11       V12       V15
## B   0.3789445 0.3151585 0.3278410 0.3009606 0.3076991 0.3224957 0.3590727
## C   0.3713247 0.3354743 0.3544015 0.2810331 0.2989993 0.2985299 0.3145032
##         V18
## B 0.2762305
## C 0.2324199
##
## Residual Deviance: 357.6253
## AIC: 389.6253
```

On calcule la valeur de z et la p-value associée :

```
# 2-tailled Z-test ----------------------------------------------------------
z <- summary(model)$coefficients/summary(model)$standard.errors
p <- (1-pnorm(abs(z),0,1))*2
print(round(p,3))
```

```
##   (Intercept)   V05   V07   V10 V11   V12   V15   V18
## B       0.075 0.363 0.031 0.000   0 0.001 0.000 0.000
## C       0.129 0.000 0.001 0.004   0 0.000 0.822 0.327
```

Les résultats sont identiques.

5.1.3 Odds et odds ratios

Les odds ratios n'apparaissent généralement pas dans les sorties logiciels pour le modèle multinomial : il faut donc les calculer à la main en prenant garde du codage particulier des variables explicatives qualitatives.

5.1.3.1 Odds

On rappelle que l'odds d'un évènement $Y = k$ sachant $X = x$ est égal au rapport $\frac{\mathbb{P}(Y = k|X = x)}{\mathbb{P}(Y \neq k|X = x)}$. Dans le cas du mutinomial, on définit l'odds d'un évènement $Y = k$ contre un évènement $Y = l$ par :

$$\texttt{odds}(x, Y = k \text{ vs } Y = l) = \frac{\mathbb{P}(Y = k|X = x)}{\mathbb{P}(Y = l|X = x)} = \exp\left((\beta^k - \beta^l)' x\right) \tag{5.5}$$

Dans le cas où $Y = l$ est défini comme modalité de référence ($l = 0$), on définit l'odds de la $Y = k$ sachant $X = x$ par :

$$\texttt{odds}(x, Y = k \text{ vs } Y = 0) = \frac{\mathbb{P}(Y = k|X = x)}{\mathbb{P}(Y = 0|X = x)} = \exp(\beta^k x) \tag{5.6}$$

Notons que cet odds est défini relativement à la modalité de référence et il s'agit en fait d'un rapport de probabilités, et non à proprement parler d'une cote (odds). Il faudra donc être extrêmement vigilant quant aux interprétations données aux résultats.

5.1.3.2 Odds ratios

Pour deux observations x_1 et x_2, on définit alors l'odds ratio par :

$$\texttt{OR}(x_1, x_2, Y = k \text{ vs } Y = l) = \frac{\dfrac{\mathbb{P}_\beta(Y = k|X = x_1)}{\mathbb{P}_\beta(Y = l|X = x_1)}}{\dfrac{\mathbb{P}_\beta(Y = k/X = x_2)}{\mathbb{P}_\beta(Y = l|X = x_2)}} = \exp\left((\beta^k - \beta^l)'(x_1 - x_2)\right) \tag{5.7}$$

Ainsi, l'odds-ratio de $Y = k$ par rapport à $Y = 0$ pour deux valeurs possibles x_1 et x_2 des régresseurs est :

$$\texttt{OR}(x_1, x_2, Y = k \text{ vs } Y = 0) = \frac{\dfrac{\mathbb{P}(Y = k|X = x_1)}{\mathbb{P}(Y = 0|X = x_1)}}{\dfrac{\mathbb{P}(Y = k|X = x_2)}{\mathbb{P}(Y = 0|X = x_2)}} = \exp\left(\beta^k(x_1 - x_2)\right) \tag{5.8}$$

Cette valeur dépend de la modalité de référence. Son interprétation est assez délicate, notamment quand il n'y a pas de modalité de référence « naturelle ».

On extrait les coefficients d'estimation

```
# coefficients d'estimation
mnlogit_coef = model.params.T.round(4)
mnlogit_coef.index = ['B','C']
```

Table 5.3 – Coefficients du modèle multinomial

	const	V05	V07	V10	V11	V12	V15	V18
B	-0.6740	-0.2867	0.7064	1.3435	1.7895	1.0512	-1.7309	-1.0404
C	-0.5643	-2.1808	-1.2247	0.8134	1.3957	1.3660	-0.0709	-0.2276

Ensuite on applique l'exponentielle pour obtenir les odds-ratios.

```
# odds ratio
import numpy as np
odds_ratio = np.exp(mnlogit_coef)
```

Table 5.4 – Odds ratios du modèle multinomial

	const	V05	V07	V10	V11	V12	V15	V18
B	0.5097	0.7507	2.0267	3.8324	5.9865	2.8611	0.1771	0.3533
C	0.5688	0.1130	0.2938	2.2556	4.0378	3.9196	0.9316	0.7964

Remarque 5.3 (Effets marginaux) *Statsmodels, grâce à l'attribut get_margeff permet d'avoir les effects marginaux des variables explicatives par rapport à chaque logit.*

```
# Effets marginaux
model_margeff = model.get_margeff()
print(model_margeff.summary())
```

```
##         MNLogit Marginal Effects
## ====================================
## Dep. Variable:                 classe
## Method:                           dydx
## At:                           overall
## ==============================================================================
##   classe=A      dy/dx    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## V05            0.0997      0.018      5.554      0.000       0.064       0.135
## V07            0.0253      0.021      1.220      0.222      -0.015       0.066
## V10           -0.0808      0.018     -4.388      0.000      -0.117      -0.045
## V11           -0.1205      0.017     -7.269      0.000      -0.153      -0.088
## V12           -0.0932      0.019     -4.901      0.000      -0.130      -0.056
## V15            0.0640      0.020      3.163      0.002       0.024       0.104
## V18            0.0461      0.016      2.910      0.004       0.015       0.077
## ------------------------------------------------------------------------------
##   classe=B      dy/dx    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## V05            0.0476      0.017      2.727      0.006       0.013       0.082
## V07            0.0838      0.019      4.393      0.000       0.046       0.121
## V10            0.0636      0.016      3.994      0.000       0.032       0.095
## V11            0.0752      0.015      4.994      0.000       0.046       0.105
## V12            0.0275      0.017      1.585      0.113      -0.007       0.062
## V15           -0.1117      0.018     -6.124      0.000      -0.148      -0.076
## V18           -0.0615      0.015     -4.151      0.000      -0.091      -0.032
## ------------------------------------------------------------------------------
##   classe=C      dy/dx    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## V05           -0.1473      0.016     -9.121      0.000      -0.179      -0.116
## V07           -0.1092      0.022     -5.061      0.000      -0.151      -0.067
## V10            0.0172      0.017      1.030      0.303      -0.016       0.050
## V11            0.0453      0.017      2.641      0.008       0.012       0.079
```

```
## V12            0.0657      0.017      3.962      0.000       0.033       0.098
## V15            0.0477      0.019      2.513      0.012       0.010       0.085
## V18            0.0154      0.014      1.099      0.272      -0.012       0.043
## ==============================================================================
```

5.1.4 Quelques statistiques de test

5.1.4.1 Pseudo-R^2 de McFadden

Notons par LL_M la vraisemblance du modèle étudié, le pseudo-R^2 de McFadden est défini de la même manière que pour la régression binaire, à savoir :

$$R^2_{MF} = 1 - \frac{LL_M}{LL_0} \tag{5.9}$$

Le pseudo-R^2 de McFadden varie entre 0 (modèle pas meilleur que le trivial) et 1 (modèle parfait).

```
#R2 de McFadden
r2Mcfadden = 1 - model.llf / model.llnull
print("R2 de McFadden : %.2f" % (r2Mcfadden))
```

```
## R2 de McFadden : 0.67
```

Statsmodels retourne la valeur du pseudo-R^2 de McFadden :

```
#qui est founi directement par l'outil
print("R2 de McFadden : %.2f" % (model.prsquared))
```

```
## R2 de McFadden : 0.67
```

5.1.4.2 Test du rapport de vraisemblance

Le test du rapport de vraisemblance revient à tester les hypothèses suivantes :

$$H_0 : \beta_1^k = \cdots = \beta_6^k \quad \text{contre} \quad H_1 : \exists (k,l) \in \{B,C\} \times \{1,\dots,6\},\ \beta_l^k \neq 0$$

Il consiste à comparer 2 déviances. Pour l'évaluation globale, il s'agit de confronter celles du modèle étudié et du modèle trivial. La statistique du test s'écrit :

$$LR = D_0 - D_M \tag{5.10}$$

Cette statistique suit une loi de χ^2 à $ddl = (K-1) \times p$ degrés de liberté. La région critique du test au risque α correspond aux grandes valeurs de la statistique de test, c'est-à-dire :

$$LR \geq \chi^2_{1-\alpha}(ddl) \tag{5.11}$$

Nous pouvons aussi décider via la `p-value`. Si elle est plus petite que α, le modèle est globalement significatif.

```
import scipy.stats as st
# Déviance du modèle étudié
DM = -2*model.llf
# Deviance du modèle trivial
D0 = -2*model.llnull
# Test du rapport
LR = D0 - DM
#
K,p = len(np.unique(y)),len(x.columns)
# degré de liberté
ddl = (K-1)*p
# p-value associée
pvalue = 1.0-st.chi.cdf(LR,ddl)
lr_test = pd.DataFrame({"statistic":LR,"ddl":ddl,"pvalue":pvalue},
                       index=["Likelihood Test"])
```

Table 5.5 – Test du rapport de vraisemblance

	statistic	ddl	pvalue
Likelihood Test	740.1113	14	0

La p-value est nulle, par conséquent on rejette l'hypothèse nulle. Donc le modèle est globalement très significatif.

5.2 Métriques multi - classe

Comment évaluer un modèle qui prédit trois évènements possibles ou davantage ? Le plus souvent, on évalue des modèles prédisant deux alternatives (santé/maladie, transaction frauduleuse/normale, etc.), c'est - à - dire des modèles de « classification binaire ». Pour évaluer ces modèles, la matrice de confusion à deux classes est bien connue et l'ensemble des métriques qui en découlent aussi (Précision, Recall, Specificity, etc.).

Alors comment adapter les méthodes de la classification binaire lorsqu'il s'agit de prédire plus de deux issues possibles ? Par exemple dans notre modèle qui indique si un wage sera A, B ou C ?

Nous expliquons dans cette section la différence entre classification binaire (*binary classification*) et multi-classe (*multi-class classification*), et présentons trois approches pour évaluer un modèle multi - classe, avec leurs avantages et leurs limites.

5.2.1 Différence entre classification binaire et multi-classe

Nous savons que les problèmes de classification peuvent se séparer en deux catégories :

— La classification binaire, par exemple la prédiction de la présence ou de l'absence de la maladie du cœur.
— La classification multi - classe où l'objectif est de prédire correctement une classe parmi K. Par exemple, la prédiction d'une catégorie de wave : A, B ou C.

Dans les chapitres précédents, nous avons considéré le cas binaire. Le cas multi - classe diffère du cas binaire sur les points suivants :

Table 5.6 – Différence entre classification binaire et multi - classe

	Binaire	Multi - classe
Probabilités prédictes	Le modèle prédit une **probabilité d'être positif**. Exemple : chaque individu a une certaine probabilité de posséder la maladie	Une probabilité est prédicte **pour chaque classe**. Exemple : chaque wave a une certaine probabilité d'être A, une probabilité d'être B et une probabilité d'être C.
Labels prédits	Si la probabilité dêtre positif dépasse un seuil s alors l'individu est prédit possitif. s est appelé le **seuil de classification** est le plus souvent choisi par les experts du domaine.	Le label prédit est celui dont **la probabilité est la plus élevée**

En quoi la prédiction des labels multi - classes diffèrent - t - elle de la prédiction binaire ? La principale différence vient du fait qu'**on ne peut pas appliquer un seuil de classification aux probabilités multi - classes** pour prédire les labels comme on le fait dans le cas binaire. De plus, choisir un seuil de classification n'aurait pas de sens pour la prédiction de labels dans la situation multi - classe.

Pour résumer, la différence majeure entre laprédiction binaire et la prédiction multi - classe est la suivante :

— Dans le cas binaire : les prédictions dépendent d'un seuil de classification (qui vaut par défaut 50%) ;
— Dans le cas multi - classe : la prédiction correspond directement au label de probabilité prédite la plus élevée sans l'utilisation intermédiaire d'un seuil.

5.2.2 Matrice de confusion multi- classe standard

La matrice de confusion confronte les valeurs observées de Y sur l'échantillon et les valeurs prédictes par le modèle. Comme la prédiction multi - classe ne repose pas sur un seuil de classification, elle possède une **unique matrice de confusion** standard du modèle, à la différence du cas binaire où de nombreuses matrice de confusion sont possibles (selon la valeur du seuil). Cette matrice est organisée de la même façon que la matrice de confusion binaire, avec en ligne les labels prédits et en colonne les labels réels. Seulement, au lieu d'avoir 2 lignes et 2 colonnes (classes 0 et 1), la matrice de confusion multi - classe a K lignes et K colonnes avec $K > 2$.

Table 5.7 – Matrice de confusion multi-classes ($K > 2$)

	Observé					
	$Y = 0$	$\cdots$	$Y = k$	$\cdots$	$Y = K-1$	Total
$\widehat{Y} = 0$	n_{00}	$\cdots$	n_{0k}	$\cdots$	n_{0K}	$n_{1\bullet} = n_0$
$\cdots$						$\cdots$
$\widehat{Y} = k$	n_{k0}	$\cdots$	n_{kk}	$\cdots$	n_{kK}	$n_{k\bullet} = n_k$
$\cdots$						$\cdots$
$\widehat{Y} = K-1$	$n_{K-1,0}$	$\cdots$	$n_{K-1,k}$	$\cdots$	$n_{K-1,K-1}$	$n_{K-1\bullet} = n_{K-1}$
Total	$n_{\bullet 0}$		$n_{\bullet k}$		$n_{\bullet K-1}$	n

L'effectif de la case (k, l) est égal au nombre d'individus appartenant à la catégorie $Y = l$ qui ont été affectés à $Y = k$, c'est-à-dire :

$$n_{kl} = \text{card}\left\{\omega \in \Omega, \widehat{Y}(\omega) = k \text{ et } Y(\omega) = l\right\} \tag{5.12}$$

Sous forme de métrique, le tableau 5.7 peut être exprimé comme suit (*cf.* Figure 5.1) :

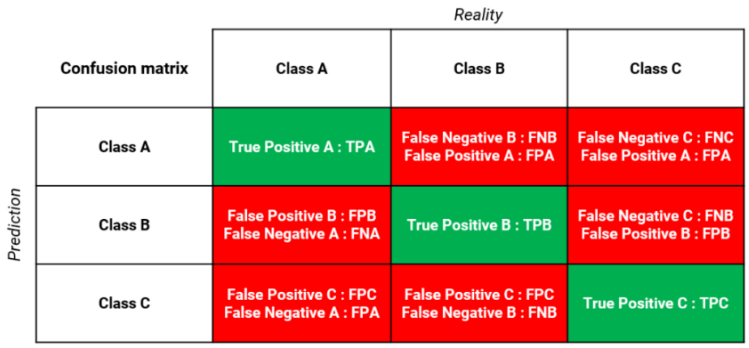


Figure 5.1 – Matrice de confusion multi - classe théorique pour 3 classes

Les faux négatifs se trouvent dans les colonnes (réalité) associées à chaque classe et le sfaux positifs se truvent dans les lignes (prédiction) associées à chaque classe. Un individu dont la classe réelle est A sera soit un Vrai Positif A soit un Faux Négatif A. Dans le cas d'une Faux Négatif A, selon la prédiction cet individu sera aussi un Faux positif B ou un Faux Positif C.

Nous effectuons les prédictions sur les données test. Nous devons tout d'aborder centrer et réduire les données de test en utilisant les paramètres (moyennes et écarts - type) calculés sur l'échantillon d'apprentissage parce que les individus de l'échantillon test représentent la population de déploiement, ils doivent être traités individuellement et non pas collectivement.

```
# Chargement des données de test
DTest = pd.read_excel("./donnee/waveform.xlsx",sheet_name=1)
xtest = DTest.drop("classe",axis=1)
# Centrage et réduction
xtest = (xtest - sc.mean_)/sc.scale_
XTest = sm.add_constant(xtest)
```

On calcule les scores prédits par le modèle :

```
# Probabilités prédictes
XTest = XTest.astype(float)
yprob = model.predict(XTest)
yprob.columns = ["A","B","C"]
# 5 premières observations
yprob.head()
```

```
##           A         B         C
## 0  0.736641  0.000252  0.263107
## 1  0.808590  0.189190  0.002220
## 2  0.969069  0.030515  0.000417
## 3  0.001529  0.955748  0.042723
## 4  0.014177  0.948026  0.037797
```

Par la suite, on calcule la classe prédicte. La règle de prédiction consiste à affecter un individu à la classe pour laquelle il possède la plus grande probabilité. Si on note $\hat{\pi}_i^k$ $(k = A, B, C)$, la probabilité d'appartenir à la classe k pour l'individu i. Pour le premier

individu, on a les probabilités d'appartenance aux classes suivantes : $\hat{\pi}_1^A = 0.74$, $\hat{\pi}_1^B = 0$ et $\hat{\pi}_1^C = 0.26$. On constate que $\hat{\pi}_1^A > \hat{\pi}_1^C > \hat{\pi}_1^B$. On affecte l'individu 1 à la classe A. On fait le même raisonnement pour les autres individus.

```
# Classe prédicte
DTest["pred"] = np.asarray(yprob).argmax(1)
# Mapping des modalités
DTest["pred"] = DTest["pred"].map({0:"A",1:"B",2:"C"})
DTest.head()
```

```
##      V05   V07   V10   V11   V12   V15   V18 classe pred
## 0   0.60  0.85  0.89  1.08  4.20  4.59  3.32      C    A
## 1   2.30  5.52  2.22  2.81  1.61  1.88  1.41      B    A
## 2   2.42  4.94  2.07  0.51  1.45  1.41  0.62      A    A
## 3   1.13  2.37  4.84  4.65  4.05  1.24 -1.43      B    B
## 4   2.66  2.69  3.53  4.82  4.79  1.73  0.13      B    B
```

Nous définissons notre matrice de confusion à partir des données de test.

```
# Matrice de confusion échantillon de test
cm = pd.DataFrame(pd.crosstab(DTest.pred,DTest.classe),columns=['A','B','C'],
                  index = ['A','B','C'])
tab = cm.copy()
tab.loc[:,'Total'] = tab.sum(axis=1)
tab.loc['Total',:] = tab.sum(axis=0)
```

Table 5.8 – Matrice de confusion sur l'échantillon de test - Donnee WAVEFORM *

	A	B	C	Total
A	1323	143	189	1655
B	199	1379	143	1721
C	135	125	1364	1624
Total	1657	1647	1696	5000

* En lignes, les prédictions du modèle et en colonnes, la réalité.

Le taux d'erreur est l'estimation de la probabilité de mal classer, il correspond au rapport entre le nombre total d'observations mal classées et l'effectif total dans le jeu de données :

$$\varepsilon = \frac{\sum_k \sum_{j \neq k} n_{kj}}{n} = 1 - \frac{\sum_k n_{kk}}{n} \tag{5.13}$$

Si le modèle classe parfaitement les observations, nous avons $\varepsilon = 0$.

```
# Taux d'erreur (en resubtitution)
error_rate = 1 - np.trace(cm)/DTest.shape[0]
print(f"Taux d'erreur : %.4f" % (error_rate))
```

```
## Taux d'erreur : 0.1868
```

Le taux de mauvais classement est 18.68%.

Le taux de succès θ est toujours le complément à 1 du taux d'erreur. Il indique la probabilité de bien classer :

$$\theta = 1 - \varepsilon = \frac{\sum_k n_{kk}}{n} \tag{5.14}$$

```
# Accuracy
accuracy = 1 - error_rate
print(f'Taux de succès : %.4f' % (accuracy))
```

```
## Taux de succès : 0.8132
```

Le taux de succès vaut 81.32%.

5.2.3 Métriques

Les métriques de classification usuelles sont définies dans le cas binaire à partir de la matrice de confusion binaire. Comment utiliser la matrice de confusion multi - classes pour généraliser les métriques binaires ? Nous allons voir les 3 méthodes classiques :

— L'approche « per class », pour laquelle on s'intéresse à K problèmes de classification binaire indépendamment. Cette approche permet de s'intéresser à chaque classe individuellement mais il est difficile d'en tier une synthèse car on obtient une métrique par classe.
— L'approche « macro » qui permet de résumer les métriques obtenues avec l'approche « per class ». Il existe 2 variantes « macro » : une variante classique robuste au déséquilibre de classe et une variante pondérée, plus représentative des données.
— L'approche « micro », qui utilise directement les valeurs de la matrice de confusion multi - classe pour produire une métrique qui résume la performance du modèle. Cette approche présente des limites.

5.2.3.1 Métriques « per class »

L'approche « per class » consiste à diviser une classification multi - classe en K classification binaire. On évalue ensuite chaque classification binaire séparément.

Pour bien comprendre, reprenons notre exemple et calculons le recall par classe. Pour ce faire, nous commençons par établir chaque matrice de confusion en repartant de la matrice de confusion multi - classe standard.

Table 5.9 – Passage de la matrice de confusion multi - classe à la matrice de confusion binaire

(a) Matrice de confusion multi - class

	A	B	B
A	1323	143	189
B	199	1379	143
C	135	125	1364

(b) Matrice de confusion : **classe C**

	0 : Non C	1 : C
0 : Non C	VN = 3044	FN = 332
1 : C	FP = 260	VP = 1364

Pour construire la matrice de confusion « classe C », on identifie dans la colonne et la ligne « classe C » les vrais positifs (VP), les faux négatifs (FN) et les faux positifs (FP).

En appliquant, cette transformation pour les classes « classe A » et « classe B », on obtient les 3 matrices de confusion binaires supplémentaires :

Table 5.10 – Matrices de confusion des classes A et B

(a) Matrice de confusion : **classe A**

	0 : Non A	1 : A
0 : Non A	VN=3011	FN = 334
1 : A	FP = 332	VP = 1323

(b) Matrice de confusion : **classe B**

	0 : Non B	1 : B
0 : Non B	VN = 3011	FN = 268
1 : B	FP = 342	VP = 1379

Maintenant, définissons ces 3 matrices sous Python. Pour le faire, on utilise la fonction multilabel_confusion_matrix se klearn.

```
# Mtrice de confusion binaire
import sklearn.metrics as mt
bin_cm = mt.multilabel_confusion_matrix(DTest.classe,DTest.pred,
                                        labels=["A","B","C"])
print(bin_cm)
```

```
## [[[3011  332]
##   [ 334 1323]]
##
##  [[3011  342]
##   [ 268 1379]]
##
##  [[3044  260]
##   [ 332 1364]]]
```

Remarquons que cette matrice contient en lignes les vraies valeurs et en colonnes les prédictions.

```
# Stockage des matrices
cmA, cmB, cmC = bin_cm[0], bin_cm[1], bin_cm[2]
```

On s'est donc ramené à trois problèmes de classification binaire, pour lesquels on peut calculer toutes les métriques dérivées de la matrice de confusion. On obtient pour chaque métriques 3 valeurs, correspondant aux 3 classes.

```
def multiclassmetrics(tab,idx):
  tn, fp, fn, tp = tab.ravel()
  false_positive_rate = fp / (fp + tn)
  false_negative_rate = fn / (tp + fn)
  true_negative_rate = tn / (tn + fp) # specificity
  negative_predictive_value = tn/ (tn + fn)
  false_discovery_rate = fp/ (tp + fp)
  recall = tp / (tp+fn) # true positive rate/sensitivity
  precision = tp/ (tp + fp) # Positive predictive value
  accuracy = (tp + tn) / (tp + fp + fn + tn)
  f1_score = (2*precision*recall)/(precision+recall)
  # Matthews Correlation Coefficient MCC
```

```
  mcc = (tp*tn - fp*fn)/((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
  res = pd.DataFrame({
    "Sensitivity" : recall,
    "Specificity" : true_negative_rate,
    "False positive rate" : false_positive_rate,
    "False negative rate" : false_negative_rate,
    "True negative rate" : true_negative_rate,
    "Negative predictive value" : negative_predictive_value,
    "False discovery rate" : false_discovery_rate,
    "Precision":precision,
    "accuracy" : accuracy,
    "F1 score" : f1_score,
    "Matthews Correlation Coefficient MCC" : mcc
   },index = [idx]).T
  return res
mat = [cmA,cmB,cmC]
label = ["class : A", "class: B", "class: C"]
mnlogit_metrics = pd.concat(map(multiclassmetrics,mat,label),axis=1)
```

Table 5.11 – Métriques per class

	class : A	class : B	class : C
Sensitivity	0.7984	0.8373	0.8042
Specificity	0.9007	0.8980	0.9213
False positive rate	0.0993	0.1020	0.0787
False negative rate	0.2016	0.1627	0.1958
True negative rate	0.9007	0.8980	0.9213
Negative predictive value	0.9001	0.9183	0.9017
False discovery rate	0.2006	0.1987	0.1601
Precision	0.7994	0.8013	0.8399
accuracy	0.8668	0.8780	0.8816
F1 score	0.7989	0.8189	0.8217
Matthews Correlation Coefficient MCC	0.0000	0.0000	0.0000

5.2.3.2 Rappel précision par catégorie

Le rappel et la précision sont des indicateurs très populaires car leurs interprétations sont simples à appréhender. Le premier indique la capacité du modèle à retrouver les positifs, le second, la capacité à les prédire (désigner) avec justesse. Nous pouvons les associer aux catégories dans le cadre multi-classes, pour le rappel de $Y = k$:

$$\text{rappel}_k = \frac{n_{kk}}{n_{\bullet k}} \tag{5.15}$$

et la précision de $Y = k$:

$$\text{précision}_k = \frac{n_{kk}}{n_{k\bullet}} \tag{5.16}$$

```
# rappel et précision
rappel = np.diagonal(cm)/np.sum(cm,axis=0)
precision = np.diagonal(cm)/np.sum(cm,axis=1)
```

```
rp = pd.DataFrame(np.c_[rappel,precision],columns = ['rappel', 'precision'],
                  index = ['class: A','class: B','class: C']).T
```

Table 5.12 – Rappel et précision par classe

	class : A	class : B	class : C
rappel	0.7984	0.8373	0.8042
precision	0.7994	0.8013	0.8399

Nous pouvons nous servir des fonctions définies par sklearn :

```
# Métriques avec sklearn
rappelk = mt.recall_score(DTest.classe,DTest.pred,average=None)
precisionk = mt.precision_score(DTest.classe,DTest.pred,average=None)
f1_scorek = mt.f1_score(DTest.classe,DTest.pred,average=None)
rp2 = pd.DataFrame(np.c_[rappelk,precisionk,f1_scorek],
                   index= ["class: A","class : B","class: C"],
                   columns=["Rappel","Precision","F1 score"]).T
```

Table 5.13 – Rappel, précision et f1-score par classe

	class : A	class : B	class : C
Rappel	0.7984	0.8373	0.8042
Precision	0.7994	0.8013	0.8399
F1 score	0.7989	0.8189	0.8217

Une façon plus efficace est d'utiliser la fonction classification_report :

```
# Classification report
print(mt.classification_report(DTest.classe,DTest.pred,
                               target_names=["class: A","class: B","class: C"]))
```

```
##               precision    recall  f1-score   support
##
##     class: A       0.80      0.80      0.80      1657
##     class: B       0.80      0.84      0.82      1647
##     class: C       0.84      0.80      0.82      1696
##
##     accuracy                           0.81      5000
##    macro avg       0.81      0.81      0.81      5000
## weighted avg       0.81      0.81      0.81      5000
```

5.2.3.3 Microaveraging et macroaveraging

La *microaveraging* (micro-moyenne) est une moyenne pondérée où les catégories pèsent selon leur effectif dans le tableau de contingence. On accorde le même poids aux observations. Il est produit directement via la matrice de confusion (Tableau 5.1).

La *macroaveraging* (macro-moyenne) est une moyenne non-pondérée où l'on accorde le même poids aux catégories. Nous pouvons le produire directement via les rappels

et précisions obtenues pour les catégories. Lorsque les prévalences des modalités de la variable dépendante sont très différentes, ces deux ratios peuvent diverger assez fortement. A nous de choisir le bon selon les objectifs de l'étude. La micro-moyenne met l'accent sur les modalités fréquentes, la macro-moyenne accorde plus d'importance à celles qui sont peu fréquentes.

Table 5.14 – Microaveraging et macroaveraging

	Microaveraging	Macroaveraging
Rappel	$\mu_{\text{rappel}} = \frac{\sum_{k=1}^{k=K} n_{kk}}{\sum_{k=1}^{k=K} n_{k\bullet}}$	$\rho_{\text{rappel}} = \frac{\sum_{k=1}^{k=K} \text{rappel}_k}{K}$
Précision	$\mu_{\text{précision}} = \frac{\sum_{k=1}^{k=K} n_{kk}}{\sum_{k=1}^{k=K} n_{\bullet k}}$	$\rho_{\text{rappel}} = \frac{\sum_{k=1}^{k=K} \text{précision}_k}{K}$

On rappelle que $\sum_{k=1}^{K} n_{k\bullet} = \sum_{k=1}^{K} n_{\bullet k} = n$, ce qui implique $\mu_{\text{rappel}} = \mu_{\text{précision}} =$ taux de succès.

```
# Microaveraging et macroaveraging
mu_rappel = mu_precision = np.trace(cm)/DTest.shape[0]
rho_rappel = np.sum(rappel)/len(np.unique(DTest.classe))
rho_precision = np.sum(precision)/len(np.unique(DTest.classe))
mat2 = np.array([[mu_rappel,rho_rappel],[mu_precision, rho_precision]])
mima = pd.DataFrame(mat2,columns = ["microaveraging","macroaveraging"],
                    index = ['rappel','précision'])
```

Table 5.15 – Microaverage et macroaverage

	microaveraging	macroaveraging
rappel	0.8132	0.8133
précision	0.8132	0.8135

Utilisons sklearn pour comparer nos résultats

```
# Mico et macroaverage
micro_rappel = mt.recall_score(DTest.classe,DTest.pred,average="micro")
macro_rappel = mt.recall_score(DTest.classe,DTest.pred,average="macro")
micro_precision = mt.precision_score(DTest.classe,DTest.pred,average="micro")
macro_precision = mt.precision_score(DTest.classe,DTest.pred,average="macro")
micro_f1 = mt.f1_score(DTest.classe,DTest.pred,average="micro")
macro_f1 = mt.f1_score(DTest.classe,DTest.pred,average="macro")
micro_macro = pd.DataFrame(np.array([[micro_rappel, macro_rappel],
                                     [micro_precision, macro_precision],
                                     [micro_f1, macro_f1]]),
                           columns=["Microaverage","Macroaverage"],
                           index= ["Rappel","Precision","F1 score"])
```

On retrouve les mêmes résultats. Ce qui est rassurant.

Table 5.16 – Microaverage et macroaverage : Rappel - Precision - F1 score

	Microaverage	Macroaverage
Rappel	0.8132	0.8133
Precision	0.8132	0.8135
F1 score	0.8132	0.8132

5.2.3.4 Taux d'erreur et échantillon non représentatif

Lorsque l'échantillon de test n'est pas représentatif, le taux d'erreur n'est pas transposable à la population. Il ne correspond pas à la probabilité de mauvais classement du modèle. il nous faut le corriger en utilisant les "vraies" prévalences p_k des catégories dans la population. La formulation est simple :

$$\varepsilon = \sum_{k=1}^{k=K} p_k \times (1 - \text{rappel}_k) \tag{5.17}$$

```
# Taux d'erreur corrigé
error_rate2 = sum(p_k*(1-rappel))
print(f"error rate modified : %.4f" % (error_rate2))
```

```
## error rate modified : 0.1874
```

Remarque 5.4 *Lorsque l'échantillon est représentatif, nous pouvons estimer p_k par $\hat{p}_k = \frac{n_{\bullet k}}{n}$, ce qui revient à :*

$$\begin{aligned}
\varepsilon &= \sum_{k=1}^{k=K} \hat{p}_k \times (1 - \textit{rappel}_k) &&= \sum_{k=1}^{k=K} \frac{n_{\bullet k}}{n} \times \left(1 - \frac{n_{kk}}{n_{\bullet k}}\right) \\
&= \sum_{k=1}^{k=K} \left(\frac{n_{\bullet k}}{n} - \frac{n_{kk}}{n}\right) &&= \sum_{k=1}^{k=K} \frac{n_{\bullet k}}{n} - \sum_{k=1}^{k=K} \frac{n_{kk}}{n} \\
&= 1 - \sum_{k=1}^{k=K} \frac{n_{kk}}{n}
\end{aligned}$$

5.2.4 Courbe ROC multi-class et AUC

Dans la littérature, nous retrouvons de nombreuses approches pour la généralisation de la courbe ROC en multi-classes. Nous distinguons deux grandes approches : une classe face à une autre et une classe face à toutes les autres, qui permettent de déterminer le nombre de dimension de l'hyper - espace dans lequel sera dessiné la représentation ROC multi-classes.

5.2.4.1 Un contre un

Cette approche propose de manipuler une classe face à une autre et considère un coût de mauvaise classification différent pour chaque couple de la classe. On cherche à affiner et à pondérer précisément le type d'erreur. Dans ces conditions, l'espace d'évaluation des performances est à $K(K-1)$ dimensions, ce qui revient à utiliser

les éléments hors de la diagonale principale, les $n_{k,l}$ avec $k \neq l$, de la matrice de confusion 5.7. Cette extension théorique pose néanmoins certains problèmes tels que la complexité des calculs et la représentation des performances. Par exemple, pour tois classes, nous obtenons un espace à 6 dimensions difficilement représentable. Afin de réduire la complexité des calculs, une autre approche a été formulée consistant cette fois à étudier une classe face à toutes les autres.

Nous illustrons cette approche en utilisant nos données. Nous considérons les 6 cas suivants (*cf.* Tableau 5.17)

Table 5.17 – Différents cas possibles pour la courbe ROC - Un contre un

Classe	cas 1	Cas 2	Cas 3	Cas 4	Cas 5	Cas 6
Positive	A	A	B	B	C	C
Négative	B	C	A	C	A	B

```
# Permutations des éléments
import statsmodels.api as sm
from itertools import permutations

perm = list(permutations(["A","B","C"],2))
colors = ["green","red","blue","cyan","violet","yellow"]
fig, ax = plt.subplots(figsize=(14,7))
auc_ovo = [0]*len(colors)

for i, p in enumerate(perm):
  perm = list(p)
  df = donnee[donnee.classe.isin(perm)]
  df.classe = df.classe.map({perm[0]:1,perm[1] : 0})
  x = df.drop("classe",axis=1)
  clf = sm.Logit(df.classe,sm.add_constant(x)).fit(disp=False)
  # Echantillon test
  test = pd.concat([DTest.classe,XTest],axis=1)
  Dtest =  test[test.classe.isin(perm)]
  Dtest.classe = Dtest.classe.map({perm[0]:1,perm[1] : 0})
  Xtest = Dtest.drop("classe",axis=1)
  # Prediction
  fpr, tpr, thresholds = mt.roc_curve(Dtest.classe, clf.predict(Xtest))
  auc_val = mt.auc(fpr, tpr)
  auc_ovo[i] = auc_val
  ax.plot(fpr, tpr,color=colors[i],
   label = f"ROC curve for {perm[0]} vs. {perm[1]} (AUC = {round(auc_val,2)})");
  ax.set(xlabel='False Positive Rate',ylabel='True Positive Rate');
ax.plot([0,1],[0,1],color="black",linestyle="--");
ax.legend();
plt.show()
```

Sous Python, cette approche est implémentée dans la fonction OneVsOneClassifier de la librairie sklearn.

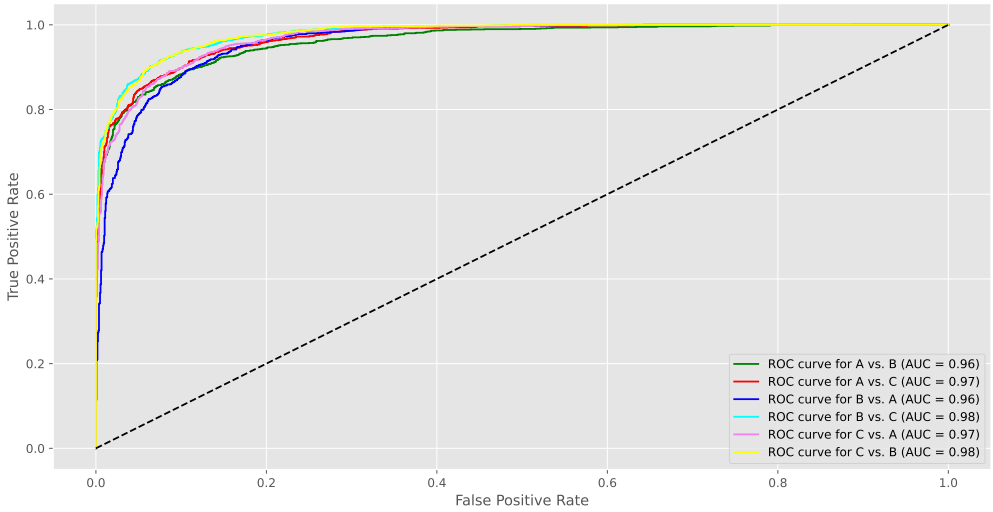


Figure 5.2 – Courbe ROC multi - class - Approche One versus One

5.2.4.2 Un contre tous

Cette approche propose de manipuler les K classes en générant K courbes ROC, une pour chaque classe. Sur l'ensemble de toutes les classes, la $k^{\text{ème}}$ courbe ROC correspond à l'évaluation des performances utilisant la classe k comme classe positive et toutes les autres classes comme négative. Le coût de mauvaise classification est, pour cette approche, fixe pour chaque classe car in ne cherche pas à différencier les erreurs. Dans ces conditions, l'espace d'évaluation des performances est à K dimensions, ce qui revient à n'utiliser que les éléments de la diagonale principale, les $n_{k,l}$, de la matrice de confusion.

Nous illustrons cette approche en utilisant nos données. Nous considérons les 3 cas suivants (*cf.* Tableau 5.18)

Table 5.18 – Différents cas possibles pour la courbe ROC - Un contre tous

Classe	cas 1	Cas 2	Cas 3
Positive	A	B	C
Négative	B et C	A et C	A et B

```
# Semi régression logistique - One versus Rest
target_train = pd.get_dummies(donnee.classe)
target_test = pd.get_dummies(DTest.classe)
labels = ["A vs. (B & C)","B vs. (A & C)","C vs. (A & B)"]
# Représentation graphique
fig, ax = plt.subplots(figsize=(14,7))
colors = ["green", "red", "blue"]
auc_ovr = [0]*len(colors)
for i, label in enumerate(["A","B","C"]):
  clf = sm.Logit(target_train[label],sm.add_constant(xc)).fit(disp=False)
  fpr, tpr,_ = mt.roc_curve(target_test.values[:,i],clf.predict(XTest))
  auc_ovr[i] = mt.auc(fpr, tpr)
  mt.RocCurveDisplay.from_predictions(target_test.values[:,i],clf.predict(XTest),
                        name=f"ROC curve for {labels[i]}",color=colors[i],ax=ax);
```

```
ax.plot([0,1],[0,1],color="black",linestyle="--");
plt.show()
```

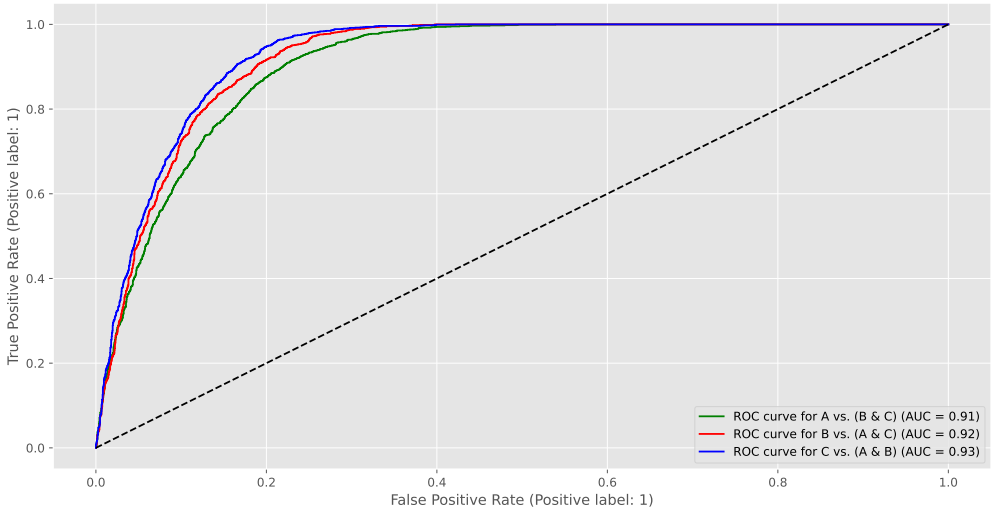


Figure 5.3 – Courbe ROC multi - class - Approche One versus Rest

Sous Python, cette approche est implémentée par la fonction OneVsRestClassifier de la librairie sklearn.

5.2.4.3 VUS

En deux classes, nous utilisions l'AUC comme critère d'évaluation des performances d'un classifieur. Maintenant que nous disposions d'un système à K classes, le critère d'évaluation que nous allons utiliser est le « Volume sous l'hyper-surface ROC ». Le VUS [2] est l'adaptation des méthodes de calcul de l'AUC aux problèmes multi - classes.

L'idée consiste à tracer un ensemble de courbes ROC, une classe contre une autre (*one versus one*), à partir desquelles les AUC sont calculées. Ainsi, nous obtenons $K(K-1)$ aires et le critère d'évaluation des performances est soit la moyenne des AUC ; soit la somme des AUC.

```
# Moyennes des auc
auc_ovo_mean = np.mean(auc_ovo)
print("Moyenne des AUC (One vs One) : %.2f"%(auc_ovo_mean))
```

```
## Moyenne des AUC (One vs One) : 0.97
```

```
# Moyennes des auc
auc_ovr_mean = np.mean(auc_ovr)
print("Moyenne des AUC (One vs Rest) : %.2f"%(auc_ovr_mean))
```

```
## Moyenne des AUC (One vs Rest) : 0.92
```

2. https ://citeseerx.ist.psu.edu/viewdoc/download ?doi=10.1.1.14.2427&rep=rep1&type=pdf

Le problème de cette généralisation est qu'elle ne correspond pas à l'extension théorique de l'AUC. En effet, aucun calcul de l'hyper - volume (VUS) n'est mis en œuvre et cette méthode compromet l'indépendance vis - à - vis des probabilités a priori et des coûts de classification. En revanche, elle permet d'évaluer des systèmes ayant un nombre de classes très grand [3].

5.3 Test sur les coefficients de la régression multinomiale

Les tests sur les coefficients consistent avant tout à éprouver leur significativité. Par rapport à la régression logistique binaire, l'analyse dans la régression logistique multinomiale est plus compliquée car nous pouvons multiplier les possibilités : tester la nullité de q coefficients dans un logit, dans un ensemble de logit ou dans les $K-1$ logit. Les conséquences ne sont les mêmes. Si une variable n'est pas significative dans l'ensemble des logit, nous pouvons l'exclure de l'étude. Si elle est significative dans un logit au moins, son rôle est avéré dans la caractérisation d'une des modalités de la variable dépendante. La variable ne peut pas être exclue.

Comme pour la régression binaire, nous disposons de deux outils pour réaliser les tests : la statistique du rapport de vraisemblance qui correspond toujours à la comparaison des déviances des régressions sous H_0 et H_1 et la statistique de Wald qui exploite la normalité asymptotique des estimateurs du maximum de vraisemblance.

5.3.1 Estimation de la matrice de variance covariance

La matrice de variance covariance est une pièce essentielle de la statistique inférentielle. Concernant la régression logistique, elle nous permettra de mettre en place les tests de Wald. Nous pourrions en tirer parti également pour la production des intervalles de confiance des coefficients et des prédictions.

La matrice de variance covariance $\widehat{\Sigma}$ correspond à l'inverse de la matrice hessienne. Elle est aussi symétrique par blocs. Il faut bien faire attention pour discerner les informations importantes qu'elles comportent : nous avons la variance des coefficients pour chaque équation logit, les covariances entre coefficients de la même équation logit, et les covariances des coefficients relatives à des équations logit différentes. On peut s'y perdre rapidement.

```
# Matrice de variances-covariances des coefficients
cov_mat = model.cov_params()
tab = cov_mat.round(4).reset_index()
```

C'est une matrice de taille $[(K-1)\times(p+1),(K-1)\times(p+1)]$, soit 16×16.

— Les variances des coefficients pour chaque équation logit sont lues sur le diagonale principale de la matrice. En prenant la racine carrée, nous obtenons les écarts - types.
— Dans les blocs situés sur la diagonale principale, nous avons les covariances des coefficients intra - logit : Par exemple, $\widehat{\text{Cov}}\left(\hat{\beta}_1^B, \hat{\beta}_2^B\right) = -0.0295$.

3. http ://dmip.webs.upv.es/ROCML2005/papers/fieldsend2CRC.pdf

Table 5.19 – Matrice de variances-covariances des coefficients

		B								C							
classe	level_1	B const	V05	V07	V10	V11	V12	V15	V18	C const	V05	V07	V10	V11	V12	V15	V18
B	const	0.1436	-0.0235	-0.0325	-0.0051	-0.0103	0.0312	0.0231	0.0163	0.0517	-0.0142	-0.0066	0.0127	0.0260	0.0307	-0.0195	-0.0098
B	V05	-0.0235	0.0993	-0.0295	0.0045	0.0083	0.0173	0.0286	-0.0004	-0.0170	0.0461	-0.0126	-0.0033	-0.0018	0.0041	0.0202	0.0027
B	V07	-0.0325	-0.0295	0.1075	0.0040	0.0111	0.0183	0.0267	0.0012	-0.0021	-0.0144	0.0393	0.0012	0.0016	0.0031	0.0210	0.0046
B	V10	-0.0051	0.0045	0.0040	0.0906	-0.0127	0.0048	0.0095	0.0072	0.0118	-0.0087	-0.0046	0.0455	-0.0036	0.0063	0.0126	0.0016
B	V11	-0.0103	0.0083	0.0111	-0.0127	0.0947	-0.0175	-0.0156	-0.0108	0.0230	0.0000	0.0058	-0.0054	0.0443	-0.0011	-0.0021	-0.0003
B	V12	0.0312	0.0173	0.0183	0.0048	-0.0175	0.1040	-0.0108	-0.0038	0.0292	-0.0008	0.0001	0.0078	0.0045	0.0547	-0.0094	-0.0101
B	V15	0.0231	0.0286	0.0267	0.0095	-0.0156	-0.0108	0.1289	-0.0132	-0.0168	0.0200	0.0167	0.0090	-0.0038	-0.0070	0.0512	-0.0069
B	V18	0.0163	-0.0004	0.0012	0.0072	-0.0108	-0.0038	-0.0132	0.0763	-0.0107	0.0012	0.0025	0.0012	0.0011	-0.0078	-0.0088	0.0305
C	const	0.0517	-0.0170	-0.0021	0.0118	0.0230	0.0292	-0.0168	-0.0107	0.1379	0.0302	0.0455	0.0065	0.0088	0.0110	-0.0304	-0.0141
C	V05	-0.0142	0.0461	-0.0144	-0.0087	0.0000	-0.0008	0.0200	0.0012	0.0302	0.1125	0.0034	-0.0050	-0.0037	0.0008	0.0192	0.0042
C	V07	-0.0066	-0.0126	0.0393	-0.0046	0.0058	0.0001	0.0167	0.0025	0.0455	0.0034	0.1256	-0.0189	-0.0024	0.0058	0.0353	0.0073
C	V10	0.0127	-0.0033	0.0012	0.0455	-0.0054	0.0078	0.0090	0.0012	0.0065	-0.0050	-0.0189	0.0790	-0.0172	-0.0005	0.0228	0.0059
C	V11	0.0260	-0.0018	0.0016	-0.0036	0.0443	0.0045	-0.0038	0.0011	0.0088	-0.0037	-0.0024	-0.0172	0.0894	-0.0073	0.0026	0.0072
C	V12	0.0307	0.0041	0.0031	0.0063	-0.0011	0.0547	-0.0070	-0.0078	0.0110	0.0008	0.0058	-0.0005	-0.0073	0.0891	-0.0038	-0.0070
C	V15	-0.0195	0.0202	0.0210	0.0126	-0.0021	-0.0094	0.0512	-0.0088	-0.0304	0.0192	0.0353	0.0228	0.0026	-0.0038	0.0989	-0.0101
C	V18	-0.0098	0.0027	0.0046	0.0016	-0.0003	-0.0101	-0.0069	0.0305	-0.0141	0.0042	0.0073	0.0059	0.0072	-0.0070	-0.0101	0.0540

— Dans les blocs hors diagonale, nous avons les covariances des coefficients inter - logit.Par exemple $\widehat{\text{Cov}}\left(\hat{\beta}_1^B, \hat{\beta}_2^C\right) = -0.0126$, qui est différent de $\widehat{\text{Cov}}\left(\hat{\beta}_1^C, \hat{\beta}_2^B\right) = -0.0144$.

Afin d'obtenir les écarts types des coefficients tel que décrit par le modèle, nous devons diviser notre matrice en $K-1$ matrices de taille $(p+1)\times(p+1)$. Nous avons la matrice 1 relative au logit $\log\left(\dfrac{\mathbb{P}(Y=B|X=x)}{\mathbb{P}(Y=A|X=x)}\right)$

```
# Bloc 1
cov_mat1= cov_mat.iloc[:8,:8]
cov1 = cov_mat1.round(4).reset_index()
```

Table 5.20 – Matrice de variances-covariances des coefficients du modèle $\log\left(\dfrac{\mathbb{P}(Y=B|X=x)}{\mathbb{P}(Y=A|X=x)}\right)$

classe	level_1	B const	V05	V07	V10	V11	V12	V15	V18
B	const	0.1436	-0.0235	-0.0325	-0.0051	-0.0103	0.0312	0.0231	0.0163
B	V05	-0.0235	0.0993	-0.0295	0.0045	0.0083	0.0173	0.0286	-0.0004
B	V07	-0.0325	-0.0295	0.1075	0.0040	0.0111	0.0183	0.0267	0.0012
B	V10	-0.0051	0.0045	0.0040	0.0906	-0.0127	0.0048	0.0095	0.0072
B	V11	-0.0103	0.0083	0.0111	-0.0127	0.0947	-0.0175	-0.0156	-0.0108
B	V12	0.0312	0.0173	0.0183	0.0048	-0.0175	0.1040	-0.0108	-0.0038
B	V15	0.0231	0.0286	0.0267	0.0095	-0.0156	-0.0108	0.1289	-0.0132
B	V18	0.0163	-0.0004	0.0012	0.0072	-0.0108	-0.0038	-0.0132	0.0763

et la matrice 2 relative au logit $\log\left(\dfrac{\mathbb{P}(Y=C|X=x)}{\mathbb{P}(Y=A|X=x)}\right)$

```
# Bloc 2
cov_mat2 = cov_mat.iloc[8:,8:]
cov2 = cov_mat2.round(4).reset_index()
```

Ensuite on applique la racine carré sur les éléments diagonaux de chacune d'elles.

```
# Ecarts - types des coeffcients
sigma1 = np.sqrt(np.diagonal(cov_mat1))
sigma2 = np.sqrt(np.diagonal(cov_mat2))
sigma_params = pd.DataFrame(np.c_[sigma1,sigma2],columns = ['B','C'],
                            index = model.params.index)
```

Table 5.21 – Matrice de variances-covariances des coefficients $\log\left(\frac{\mathbb{P}(Y=C|X=x)}{\mathbb{P}(Y=A|X=x)}\right)$

classe	level_1	C const	V05	V07	V10	V11	V12	V15	V18
C	const	0.1379	0.0302	0.0455	0.0065	0.0088	0.0110	-0.0304	-0.0141
C	V05	0.0302	0.1125	0.0034	-0.0050	-0.0037	0.0008	0.0192	0.0042
C	V07	0.0455	0.0034	0.1256	-0.0189	-0.0024	0.0058	0.0353	0.0073
C	V10	0.0065	-0.0050	-0.0189	0.0790	-0.0172	-0.0005	0.0228	0.0059
C	V11	0.0088	-0.0037	-0.0024	-0.0172	0.0894	-0.0073	0.0026	0.0072
C	V12	0.0110	0.0008	0.0058	-0.0005	-0.0073	0.0891	-0.0038	-0.0070
C	V15	-0.0304	0.0192	0.0353	0.0228	0.0026	-0.0038	0.0989	-0.0101
C	V18	-0.0141	0.0042	0.0073	0.0059	0.0072	-0.0070	-0.0101	0.0540

Table 5.22 – Ecarts - types des coefficients - calcul manuel

	B	C
const	0.3789	0.3713
V05	0.3152	0.3355
V07	0.3278	0.3544
V10	0.3010	0.2810
V11	0.3077	0.2990
V12	0.3225	0.2985
V15	0.3591	0.3145
V18	0.2762	0.2324

Statsmodels renvoie l'écart-type des coefficients estimés grâce à l'outil bse.

```
# standard errors
sigma_coef = model.bse
sigma_coef.columns = ['B','C']
```

Table 5.23 – Ecarts - types des coefficients - Statsmodels

	B	C
const	0.3789	0.3713
V05	0.3152	0.3355
V07	0.3278	0.3544
V10	0.3010	0.2810
V11	0.3077	0.2990
V12	0.3225	0.2985
V15	0.3591	0.3145
V18	0.2762	0.2324

5.3.2 Significativité d'un coefficient dans un logit

L'hypothèse nulle de ce est s'écrit :

$$H_0 : \beta_p^k = 0$$

Un coefficient dans un des logit est-il significatif ? Si la réponse est non, il ne l'est pas alors nous pouvons supprimer la variable associée dans le logit concerné. Cependant, nous ne pouvons rien conclure concernant les autres logit : Nous ne pouvons donc pas exclure la variable de l'étude.

5.3.2.1 Test du rapport de vraisemblance

Pour ce test, il s'agit d'optimiser la vraisemblance en forçant $\beta_p^k = 0$. Nous obtenons le modèle contraint (modèle sous H_0), d'en extraire la déviance D_{H_0} que l'on comparera à celle du modèle complet D_M. La statistique de test s'écrit :

$$LR = D_{H_0} - D_M \tag{5.18}$$

La statistique LR suit une χ^2 à 1 degré de liberté sous H_0.

5.3.2.2 Test de Wald

La statistique de Wald est formée par le rapport entre le carré du coefficient est sa variance.

$$W_p^k = \left(\frac{\hat{\beta}_p^k}{\hat{\sigma}_{\hat{\beta}_p^k}} \right)^2 \tag{5.19}$$

La statistique W_p^k suit une χ^2 à 1 degré de liberté sous H_0.

Dans notre exemple, nous souhaitons tester la significativité du coefficient associé à la variable « V18 » dans la seconde équation logit (C vs. A).

```
# Test de Wald sur << V18 >> sur le deuxième logit
wald_V18 = (model.params.iloc[7,1]/sigma_coef.iloc[7,1])**2
# p-value
pvalue = 1.0 - st.chi2.cdf(wald_V18,1)
wtest=pd.DataFrame({"statistic":wald_V18,"pvalue":pvalue},index=["Wald test"])
```

Table 5.24 – Test de Wald de la variable V18 sur le second logit

	statistic	pvalue
Wald test	0.9588	0.3275

Avec un $\chi^2(1)$, nous avons une p - value de 0.3275. Nous acceptons l'hypothèse nulle au risque 5%. La variable « V18 » n'est pas significative sur le deuxième logit.

5.3.3 Significativité d'un coefficient dans tous les logit

L'hypothèse nulle du test s'écrit :

$$H_0 : \beta_p^k = 0 \quad \forall k \tag{5.20}$$

Il va plus loin que le précédent. Il cherche à savoir si les coefficients d'une variable explicative sont simultanément nuls dans l'ensemble des logit. Si les données sont compatibles avec H_0, cela veut dire que la variable a le même impact dans tous les logit. Il n'est pas question en revanche de la supprimer de la régression si elle est par ailleurs significative : son impact est le même, mais il n'est pas nul.

5.3.3.1 Test du rapport de vraisemblance

Le principe est toujours le même, nous calculons la déviance du modèle contraint et nous la comparons à celle du modèle complet. La statistique de test est :

$$LR = D_{H_0} - D_M \tag{5.21}$$

Cette statistique suit une loi du χ^2 à $K-1$ degrés de liberté sous H_0.

Nous voulons tester la pertinence de chaque variable au modèle. Cela revient à effectuer un test de nullité des coefficients de la variable dans les différentes équations du modèle.

```
# Test du rapport de vraisemblance
def univariatelrtest(fullmodel,data,target,remove):
  y = data[target]
  x = data.drop([target,remove],axis=1)
  rmodel = sm.MNLogit(y,sm.add_constant(x)).fit(disp=False)
  # Deviance
  LR = -2*(rmodel.llf - fullmodel.llf)
  # Degré de liberté
  ddl = rmodel.df_resid - fullmodel.df_resid
  # p - value
  pvalue = 1.0 - st.chi.cdf(LR,ddl)
  return pd.DataFrame({"lr stat":LR,"ddl":ddl,"pvalue":pvalue},index=[remove])
# Application
ulrtest = pd.concat(map(lambda x : univariatelrtest(model,donnee,"classe",x),
                    x.columns),axis=0)
```

Table 5.25 – Test du rapport de vraisemblance entre un modèle et le modèle complet

	lr stat	ddl	pvalue
V05	63.7690	2	0
V07	29.8018	2	0
V10	23.1885	2	0
V11	51.3059	2	0
V12	26.3236	2	0
V15	34.1744	2	0
V18	16.4646	2	0

Au risque de 5%, toutes les pvalues sont nulles. Donc, les variables sont significatives.

5.3.3.2 Test de Wald

La statistique de test s'écrit :

$$W_p = \hat{\beta}_p' \widehat{\Sigma}_j^{-1} \hat{\beta}_p \tag{5.22}$$

Sous H_0, cette statistique suit une loi du χ^2 à $K-1$ degrés de liberté. $\hat{\beta}_p$ est le vecteur des coefficients à évaluer, de dimension $(K-1) \times 1$, $\widehat{\Sigma}_j$ est leur matrice de variance

covariance [4]

Pour notre jeu de données, nous souhaitons tester la nullité du coefficient « V18 » sur les tous les logit. Tout d'abord, nous récupérons la sous-partie de la matrice de variance covariance qui nous concerne :

```
# indice des coefficients concernés
index = [7,15]
# Sous-matrice
cov_V18 = np.zeros(shape=(2,2))
for i in range(len(index)):
    for j in range(len(index)):
        cov_V18[i,j] = cov_mat.iloc[index[i],index[j]]
# Vérification
print(cov_V18)
```

```
## [[0.07630312 0.03053078]
##  [0.03053078 0.05401898]]
```

Nous inversions cette sous-matrice :

```
# Inversion de cette matrice
inv_cov_V18 = np.linalg.inv(cov_V18)
print(inv_cov_V18)
```

```
## [[16.93550397 -9.57171266]
##  [-9.57171266 23.92181179]]
```

Nous récupérons également le sous-vecteur des coefficients estimés.

```
# Coefficients à tester
coef_V18 = model.params.iloc[7,:]
print(coef_V18)
```

```
## 0   -1.040381
## 1   -0.227579
## Name: V18, dtype: float64
```

Nous calculons la forme quadratique correspondant à la statistique de test ainsi que le p-value associée

```
# Statistique de test (Forme quadratique)
wald_V18 = np.dot(coef_V18, np.dot(inv_cov_V18,coef_V18))
# p-value avec ddl=len(index)
pvalue = 1.0 - st.chi2.cdf(wald_V18, len(index))
wald_test=pd.DataFrame({"statistic":wald_V18,"ddl" : len(index),
                        "pvalue":pvalue},index=["Wald test"])
```

4. Tout l'enjeu est de savoir lire correctement la matrice de variance covariance globale $\widehat{\Sigma}$ et y « piocher » les valeurs de $\widehat{\Sigma}_j$.

Table 5.26 – Test de Wald de nullité sur tous les logit de la variable V18

	statistic	ddl	pvalue
Wald test	15.0372	2	5e-04

Avec un $\chi^2(2)$, nous avons une p - value de 5×10^{-4}. Nous rejetons l'hypothèse nulle au risque 5%.

Ce résultat doit nous interpéller. En effet, tester individuellemnt dans chaque équation logit, le coefficient de « V18 » dans le second logit n'est pas significatif. En revanche, tester simultanément, nous rejetons l'hypothèse nulle.Un test simultané ne peut pas être réduit en une succession de tests individuels.

Statsmodels propose l'outil wald_test() pour réaliser les tests généralisés. L'enjeu réside alors dans la définition de la matrice **r_matrix** permettant de spécifier les coefficients à tester.

5.3.4 Test d'égalité d'un coefficient dans tous les logit

Nous souhaitons savoir si les coefficients d'une variable X_p ont identiques d'un logit à l'autre. L'hypothèse nulle s'écrit :

$$H_0 : \beta_p^1 = \cdots = \beta_p^{K-1}$$

Lorsqu'elle est compatible avec les données, cela veut dire que la variable a le même impact dans tous les logit. Il n'est pas question en revanche de la supprimer de la régression si elle est par ailleurs significative : son impact est le même, mais il n'est pas nul.

5.3.4.1 Test de rapport de vraisemblance

Définir le modèle contraint dans les logiciels de statistique n'est pas très facile.

5.3.4.2 Test de Wald - Calcul direct

Nous illustrons cette approche à travers notre exemple. On souhaite tester l'égalité des coefficients de la variable « V18 » sur les deux logit. L'hypothèse nulle du test est :

$$H_0 : \beta_6^B = \beta_6^C$$

qui peut écrire

$$H_0 : \delta_6 = \beta_6^B - \beta_6^C = 0$$

La statistique de test est :

$$\hat{\delta}_6 = \hat{\beta}_6^B - \hat{\beta}_6^C \tag{5.23}$$

Sous H_0, cette statistique est d'espérance nulle et de variance (estimée) :

$$\hat{\mathrm{V}}(\hat{\delta}_6) = \hat{\mathrm{V}}(\hat{\beta}_6^B) + \hat{\mathrm{V}}(\hat{\beta}_6^C) - 2 \times \mathrm{Cov}(\hat{\beta}_6^B, \hat{\beta}_6^C) \tag{5.24}$$

Sous H_0, la statistique de test

$$\frac{\delta_6^2}{\hat{\mathrm{V}}(\hat{\delta}_6)} \tag{5.25}$$

suit une loi de χ^2 à 1 degré de liberté.

```
# Différence des coefficients
delta_V18 = model.params.iloc[7,0]-model.params.iloc[7,1]
# Variance
var_delta_V18 = np.sum(np.diagonal(cov_V18))-2*cov_V18[1,0]
# Statistique de Wald
wald_V18 = delta_V18**2/var_delta_V18
# p-value
pvalue = 1.0 - st.chi2.cdf(wald_V18,1)
wald_test=pd.DataFrame({"statistic":wald_V18,"pvalue":pvalue},index=["Wald test"])
```

Table 5.27 – Test de Wald d'égalité sur tous les logits de la variable V18 - Calcul direct

	statistic	pvalue
Wald test	9.5386	0.002

Avec un $\chi^2(1)$, nous avons une p - value de 0.002. Nous rejetons l'hypothèse nulle au risque 5%.

5.3.4.3 Test de Wald - Calcul générique

Lorsque le nombre d'équations logit est supérieur à 2, l'affaire devient plus compliquée. Il paraît plus judicieux de passer par l'écriture générique des tests en écrivant correctement la matrice **R**. Elle est de format m lignes et $l = (K-1) \times (p+1)$ colonnes. m $(m \leq p+1)$ est le nombre de variables (y compris la constante) à tester. Si on pose $\hat{\alpha} = R\hat{\beta}$, la statistique de test s'écrit :

$$W_{(R)} = \hat{\alpha}' \left[R\widehat{\Sigma}R' \right]^{-1} \alpha \tag{5.26}$$

Cette statistique suit une χ^2 à m degrés de liberté.

Pour notre exemple, grâce à la librairie numpy, nous créons une matrice nulle de format $(1, 16)$. Ensuite nous remplaçons les valeurs des indices 7 et 15 [5] par 1 et -1 respectivement.

```
# Matrice R
R = np.zeros(shape=(1,16))
R[:,7]=1
R[:,15]=-1
print(R)
```

5. Rappel : Python débute le comptage à partir de 0.

```
## [[ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0. -1.]]
```

On peut à présent effectuer les calculs de la statistique du test de Wald.

```
# coefficient beta_hat
beta = np.r_[model.params.iloc[:,0],model.params.iloc[:,1]]
# alpha
alpha = np.dot(R,beta)
# Matrice V
V = np.dot(R,np.dot(cov_mat,R.T))
# Wald_test
wald_test = alpha*np.linalg.inv(V)*alpha
print(f'Wald test : %.4f' % (wald_test))
```

```
## Wald test : 9.5386
```

Nous retrouvons exactement la même valeur qu'avec l'approche directe. Les conclusions sont identiques.

5.3.5 Modèle multinomial et régresseurs catégorielles

Cette section sera illustrée à travers un exemple.

5.3.5.1 Problématique

On suppose que nos données sont les suivantes :

- Y : variable à expliquer avec plusieurs catégories $(1, 2, 3)$
- X : une variable catégorielle à trois modalités (A,B,C)
- Z : une variable catégorielle à deux modalités (E,F)

On veut construire le modèle suivant :

$$\ln\left(\frac{\mathbb{P}(Y=k)}{\mathbb{P}(Y=\text{ref})}\right) = \beta_0^k + \beta_A^k 1_{Z=A} + \beta_B^k 1_{Z=B} + \beta_C^k 1_{Z=C} + \beta_E^k 1_{X=E} + \beta_F^k 1_{X=F} \quad (5.27)$$

et une contrainte sur les coefficients β_A^k, β_B^k et β_C^k et sur les coefficients β_E^k et β_F^k pour que le modèle soit identifiable, soit $\forall k$:

$$\begin{cases} \beta_A^k + \beta_B^k + \beta_C^k & = 0 \\ \beta_E^k + \beta_F^k & = 0 \end{cases}$$

Notons que la base de données est en général sous la forme :

Exemple 5.2

On a posé à 195 étudiants la question : si vous trouvez un portefeuille dans la rue contenant de l'argent et des papiers :

Table 5.28 – Tableau de données - Régresseurs catégoriels

individu	X	Z	Y
ind 1	E	A	1
ind 2	F	B	2
ind 3	F	C	1
ind 4	E	C	3
...			

— vous gardez tout (réponse 1) ;
— vous gardez l'argent et rendez le portefeuille (réponse 2) ;
— vous rendez tout (réponse 3).

On construit alors la variable `WALLET` telle que :

— `WALLET=1` si l'étudiant répond 1 ;
— `WALLET=2` si l'étudiant répond 2 ;
— `WALLET=3` si l'étudiant répond 3 ;

Ppour chaque étudiant, on note :

— Le sexe (variable `MALE=1` si homme,0 si femme) ;
— La nature des études suivies (variable `BUSINESS=1` pour les écoles de commerce, 0 pour les autres écoles) ;
— L'existence de punitions passées (variable `PUNISH=1` si puni seulement à l'école primaire, 2 si puni seulement à l'école primaire te secondaire et 3 si puni seulement à l'école primaire, secondaire et supérieur) ;
— L'explication ou pas par les parents des punitions reçues dans l'enfance (variable `EXPLAIN=1` si les parents expliquaient, 0 sinon).

On cherche à expliquer la variable `WALLET` par les autres variables.

```
# loda dataset
portefeuille = pd.read_sas('./donnee/portefeuille.sas7bdat')
print(portefeuille.head())
```

```
##    wallet  male  business  punish  explain
## 0     2.0   0.0       0.0     2.0      0.0
## 1     2.0   0.0       0.0     2.0      1.0
## 2     3.0   0.0       0.0     1.0      1.0
## 3     3.0   0.0       0.0     2.0      0.0
## 4     1.0   1.0       0.0     1.0      1.0
```

On voit nos données possèdent des virgules, ce qui traduit qu'ils sont de type `float`.

```
# structure
print(portefeuille.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 195 entries, 0 to 194
## Data columns (total 5 columns):
##  #   Column    Non-Null Count  Dtype
```

```
## ---  ------    --------------  -----
##  0   wallet    195 non-null    float64
##  1   male      195 non-null    float64
##  2   business  195 non-null    float64
##  3   punish    195 non-null    float64
##  4   explain   195 non-null    float64
## dtypes: float64(5)
## memory usage: 7.7 KB
## None
```

On effectue une double conversion de données : d'abord sous forme de `int` afin d'annuler l'effet float et ensuite sous forme de variable catégorielle.

```
# conversion
portefeuille = portefeuille.astype('int').astype('category')
print(portefeuille.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 195 entries, 0 to 194
## Data columns (total 5 columns):
##  #   Column    Non-Null Count  Dtype
## ---  ------    --------------  -----
##  0   wallet    195 non-null    category
##  1   male      195 non-null    category
##  2   business  195 non-null    category
##  3   punish    195 non-null    category
##  4   explain   195 non-null    category
## dtypes: category(5)
## memory usage: 1.6 KB
## None
```

Le modèle polytomique multinomial permettant d'expliquer la variable `WALLET` par les autres variables (en prenant comme modalité de référence $Y = 1$) :

$$\ln\left(\frac{p_\beta^k(x)}{p_\beta^1(x)}\right) = \beta_0^k + \beta_1^k 1_{\{x_1=1\}} + \beta_2^k 1_{\{x_2=1\}} + \beta_3^k 1_{\{x_3=2\}} + \beta_4^k 1_{\{x_3=3\}} + \beta_5^k 1_{\{x_4=1\}} \quad \text{avec} \quad k = 2, 3 \tag{5.28}$$

On entraîne notre modèle :

```
# Importation de la classe de calcul
import statsmodels.formula.api as smf

portefeuille.wallet = portefeuille.wallet.astype('int')
formula = 'wallet~C(male)+C(business)+C(punish)+C(explain)'
cat_mnlogit = smf.mnlogit(formula=formula,data = portefeuille).fit(disp=False)
print(cat_mnlogit.summary().tables[1])
```

```
## ==============================================================================
```

```
##         wallet=2       coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------------
## Intercept            1.2994      0.663      1.961      0.050       0.001       2.598
## C(male)[T.1]        -0.0956      0.584     -0.164      0.870      -1.240       1.049
## C(business)[T.1]    -0.7635      0.562     -1.358      0.174      -1.865       0.338
## C(punish)[T.2]      -0.8959      0.656     -1.366      0.172      -2.182       0.390
## C(punish)[T.3]      -1.7880      0.713     -2.506      0.012      -3.186      -0.390
## C(explain)[T.1]      0.7957      0.565      1.409      0.159      -0.311       1.902
## ------------------------------------------------------------------------------------
##         wallet=3       coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------------
## Intercept            2.4062      0.624      3.854      0.000       1.183       3.630
## C(male)[T.1]        -1.2672      0.554     -2.287      0.022      -2.353      -0.181
## C(business)[T.1]    -1.1791      0.549     -2.149      0.032      -2.254      -0.104
## C(punish)[T.2]      -1.1451      0.633     -1.809      0.070      -2.386       0.096
## C(punish)[T.3]      -2.1412      0.681     -3.144      0.002      -3.476      -0.806
## C(explain)[T.1]      1.5935      0.549      2.902      0.004       0.517       2.670
## ====================================================================================
```

On affiche les coefficients d'estimation des différents logit :

```
# coefficients de régression
cat_coef = cat_mnlogit.params
cat_coef.columns = ['2','3']
```

Table 5.29 – Paramètres du modèle multinomial avec régresseurs catégoriels

	2	3
Intercept	1.2994	2.4062
C(male)[T.1]	-0.0956	-1.2672
C(business)[T.1]	-0.7635	-1.1791
C(punish)[T.2]	-0.8959	-1.1451
C(punish)[T.3]	-1.7880	-2.1412
C(explain)[T.1]	0.7957	1.5935

Tout comme pour les autres variables, on peut tester la nullité d'un sous ensemble de coefficients à l'aide des statistiques de Wald, du rapport de vraisemblance et du score. Par exemple, on obtient la probabilité critique du test du rapport de vraisemblance pour le test :

$$H_0 : \beta_1^k = \cdots = \beta_5^k = 0 \quad \text{contre} \quad H_1 : \exists (k,l) \in \{2,3\} \times \{1,\dots,5\}$$

avec les commandes

```
# Modèle null
cat_nullmnlogit = smf.mnlogit(formula='wallet~1',data=portefeuille).fit(disp=False)
# Test du rapport de vraisemblance
statRV = -2*(cat_nullmnlogit.llf - cat_mnlogit.llf)
# degré de liberté
ddl = cat_nullmnlogit.df_resid - cat_mnlogit.df_resid
# p-value
pvalue = 1.0 - st.chi2.cdf(statRV,ddl)
lr_test=pd.DataFrame({"statistic":statRV,"pvalue":pvalue},index=["LR test"])
```

Table 5.30 – Test du rapport de vraisemblance de nullité des coefficients

	statistic	pvalue
LR test	50.5793	0

Au risque de 5%, nous avons une pvalue nulle, donc on rejette l'hypothèse H_0 de nullité des coeffcients. Toutes les variables sont significatives.

On veut calculer l'odds ratio de la variable punish pour les modalités 2 et 3 $\texttt{OR}(2, 3, Y = 2 \text{ vs } Y = 3)$. On calcule d'abord $\texttt{odds}(2, Y = 2 \text{ vs } Y = 3)$ et $\texttt{odds}(3, Y = 2 \text{ vs } Y = 3)$

```
# coefficients
coef = cat_mnlogit.params.values
# odd de 2
odd_223 = np.exp(coef[0,0]+coef[3,0]-coef[0,1]-coef[3,1])
# odds de 3
odd_323 = np.exp(coef[0,0]+coef[4,0]-coef[0,1]-coef[4,1])
# odds ratio
OR_23 = odd_223/odd_323
print(f'OR(2,3,Y=2 vs Y=3) : %.2f' % (OR_23))
```

```
## OR(2,3,Y=2 vs Y=3) : 0.90
```

Remarque 5.5 *Certains packages proposent de changer la forme du tableau tel que présenté au tableau 5.28. Il est alors recommandé d'avoir la base de données sous la forme :*

X	Z	frequence(Y=1)	frequence(Y=2)	frequence(Y=3)
E	A	3	2	0
E	B	0	3	15
E	C	1	12	7
F	A	0	0	2
...				

où l'on calcule le nombre de fois que l'on observe $Y = 1$, $Y = 2$ et $Y = 3$ pour chaque combinaison possible de X et Z. Cette nouvelle base contient au maximum autant de lignes que de combinaisons possibles de X et Z : $2 \times 3 = 6$ lignes maximum pour le jeu de données de la table 5.28.

Le modèle s'écrit :

$$\begin{aligned}\ln\left(\frac{p_\beta^k(x)}{p_\beta^1(x)}\right) &= \beta_0^k + \beta_1^k 1_{\{x_1=0\}} + \beta_2^k 1_{\{x_1=1\}} + \beta_3^k 1_{\{x_2=0\}} + \beta_4^k 1_{\{x_2=1\}} + \beta_5^k 1_{\{X_3=1\}} \\ &\quad + \beta_6^k 1_{\{X_3=2\}} + \beta_7^k 1_{\{X_3=3\}} + \beta_8^k 1_{\{X_4=0\}} + \beta_9^k 1_{\{X_4=1\}} \quad k = 2, 3\end{aligned}$$

Et on pose les contraintes d'identifiabilité du modèle :

$$\beta_1^k + \beta_2^k = 0, \quad \beta_3^k + \beta_4^k = 0, \quad \beta_5^k + \beta_6^k + \beta_7^k = 0, \quad \beta_8^k + \beta_9^k = 0 \quad \text{avec} \quad k = 2, 3$$

Nous illustrons cela sous R avec le package VGAM. On transforme notre tableau initial.

```
##load dataset
library(haven)
portefeuille <- read_sas("./donnee/portefeuille.sas7bdat")
# transformation
library(tidyverse)
dat.freq1 <- portefeuille %>%
  group_by(male, business, punish, explain) %>%
  summarize(freq.wallet.1 = sum(wallet == 1),
            freq.wallet.2 = sum(wallet == 2),
            freq.wallet.3 = sum(wallet == 3))
dat.freq <- dat.freq1 %>%
  ungroup(male, business, punish, explain) %>%
  mutate(male = as.factor(male),
         business = as.factor(business),
         punish = as.factor(punish),
         explain = as.factor(explain))
# Affichage
print(dat.freq)
```

```
## # A tibble: 23 x 7
##    male  business punish explain freq.wallet.1 freq.wallet.2 freq.wallet.3
##    <fct> <fct>    <fct>  <fct>           <int>         <int>         <int>
##  1 0     0        1      0                   1             3             8
##  2 0     0        1      1                   0             5            45
##  3 0     0        2      0                   0             2             5
##  4 0     0        2      1                   0             2             5
##  5 0     0        3      0                   3             1             1
##  6 0     0        3      1                   0             0             3
##  7 0     1        1      0                   0             0             1
##  8 0     1        1      1                   1             2             2
##  9 0     1        2      0                   1             0             0
## 10 0     1        2      1                   0             1             2
## # i 13 more rows
```

On entraîne le modèle :

```
### Entraînement du modèle
library(VGAM)
model <- vglm(cbind(dat.freq$freq.wallet.1,dat.freq$freq.wallet.2,
                    dat.freq$freq.wallet.3) ~ male + business +
                punish + explain, family = multinomial, data = dat.freq)
# Affichage
print(summary(model))
```

```
##
## Call:
## vglm(formula = cbind(dat.freq$freq.wallet.1, dat.freq$freq.wallet.2,
##     dat.freq$freq.wallet.3) ~ male + business + punish + explain,
##     family = multinomial, data = dat.freq)
##
```

```
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept):1  -2.4062     0.6243  -3.854 0.000116 ***
## (Intercept):2  -1.1068     0.4282  -2.585 0.009750 **
## male1:1         1.2672     0.5542   2.287 0.022216 *
## male1:2         1.1716     0.3717   3.152 0.001623 **
## business1:1     1.1791     0.5486   2.149 0.031623 *
## business1:2     0.4156     0.4234   0.981 0.326389
## punish2:1       1.1451     0.6330   1.809 0.070473 .
## punish2:2       0.2492     0.4823   0.517 0.605448
## punish3:1       2.1412     0.6811   3.144 0.001669 **
## punish3:2       0.3532     0.6398   0.552 0.580964
## explain1:1     -1.5935     0.5490  -2.902 0.003703 **
## explain1:2     -0.7978     0.4060  -1.965 0.049421 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Names of linear predictors: log(mu[,1]/mu[,3]), log(mu[,2]/mu[,3])
##
## Residual deviance: 31.9197 on 34 degrees of freedom
##
## Log-likelihood: -46.3233 on 34 degrees of freedom
##
## Number of Fisher scoring iterations: 4
##
## No Hauck-Donner effect found in any of the estimates
##
##
## Reference group is level  3  of the response
```

La fonction vglm de ce package prend la classe $Y = 3$ comme référence.

6

Régression logistique ordinale

Sommaire

Dans ce chapitre, on présente une généralisation de la régression logistique pour une variable réponse prenant plus de deux modalités qui peuvent être ordonnées et dont on souhaite tenir compte de l'ordre. En effet, notre objectif est de modéliser la dépendance d'une certaine variable réponse sur des variables explicatives dans un cas où la variable réponse prend plusieurs modalités ordonnées. C'est par exemple le cas si notre variable réponse correspond aux différents stades d'un cancer, à différents niveaux de douleurs ou encore lorsqu'on peut regrouper en classes ordonnées les valeurs d'une variable. Plus généralement, on suppose que les modalités de la variable réponse Y sont hiérarchisées. Dans ce cas, il y a une liaison intuitive entre ces modalités. En particulier, les questions qui se posent naturellement sont donc plus du type $Y = k$ versus $Y = k+1$ ou $Y = k-1$, ou $Y \leq k$ versus $Y > k$ que de comparer toutes les modalités à une modalité de référence. On s'intéresse ici à mesurer l'influence de chacun des prédicteurs sur la variable réponse et donc à déterminer quels prédicteurs sont significatifs.

6.1 Type de modèle polytomique ordonnée

Une variable est dite ordinale si l'ordre des modalités la constituant a son importance. L'utilisation de la régression logistique multinomiale dans le cas où la variable réponse est ordonnée ne tient pas compte de ce caractère ordinal. Ce qui se traduit sur l'estimation des coefficients de régression et par conséquent les rapports de cotes ne reflètent pas cet aspect dans l'interprétation.

La principale différence entre les modèles multinomial et ordinal est le fait que dans le modèle multinomial on compare chaque catégorie à une catégorie de référence, tandis que, dans le modèle ordinal, nous devons préciser le type de comparaison souhaitée. Il existe beaucoup de modèle qui répondent à ce type de problématique parmi lesquels on peut citer : le modèle à catégories adjacentes,le modèle à rapport continu et modèles à odds proportionnels ou logit cumulatif.

Dans l'écriture du modèle de cette partie, on a toujours écrit implicitement (et génériquement) $X = (1, X_1, ..., X_p)$ pour le vecteur des régresseurs, et $\beta = (\beta_0, \beta_1, ..., \beta_p)$ pour le vecteur des coefficients à estimer. Dans le même cadre, on notera génériquement $X^* = (X_1, ..., X_p)$ et $\beta^* = (\beta_1, ..., \beta_p)$ (on a donc exclu le régresseur constant).

6.1.1 Le modèle des catégories adjacentes

Dans le cas du modèle des catégories adjacentes (en anglais, *Adjacentes categories logits*), on compare une catégorie de la variable dépendante Y à la prochaine plus grande catégori, par exemple $Y = k$ avec $Y = k + 1$. En reprenant les notations du modèle nominal, le modèle à catégories adjacentes cherche à expliquer le passage d'une catégorie à la catégorie suivante selon une équation du type :

$$\frac{\mathbb{P}(Y = k+1|X)}{\mathbb{P}(Y = k|X)} = e^{\beta_0^{(k+1)} + \beta^* X^*} \quad 0 \leq k \leq K-1 \tag{6.1}$$

ou encore

$$\log\left(\frac{\mathbb{P}(Y = k+1|X)}{\mathbb{P}(Y = k|X)}\right) = \beta_0^{(k+1)} + \beta^* X^* \quad 0 \leq k \leq K-1 \tag{6.2}$$

où (ce qui est l'essentiel), β^* ne dépend pas de la classe k.

Ce modèle est un cas particulier du modèle nominal ; en particulier, l'hypothèse selon laquelle β^* ne dépend pas de la classe k peut (et doit !) être testée dans ce cadre.

Un des intérêts de ce modèle est qu'on a sensiblement réduit le nombre de paramètres à estimer : il y en a désormais $K + p$. Par ailleurs, il peut aussi constituer un outil de backtesting des classes en Y (amenant à aménager ou concaténer ces classes si c'est possible pour pouvoir valablement utiliser ce modèle).

6.1.2 Le modèle séquentiel

On parle de modèle séquentiel ou à rapport contion (en anglais, *continuation - ratio logits*). Ce modèle est utilisé pour comparer une modalité $Y = k$ aux autres modalités inférieures $(Y < k)$. Mais aussi, pour comparer la catégorie $Y = k$ aux catégories supérieures $(Y > k)$. Dans ce modèle, la contrainte de l'égalité des coefficients dans tous les logits n'est pas prise en considération, le logit s'écrira donc :

$$\log\left(\frac{\mathbb{P}\left(Y = k|X\right)}{\mathbb{P}\left(Y < k|X\right)}\right) = \beta_0^k + \beta^* X^* \tag{6.3}$$

ou

$$\log\left(\frac{\mathbb{P}\left(Y = k|X\right)}{\mathbb{P}\left(Y > k|X\right)}\right) = \alpha_0^k + \beta^* X^* \tag{6.4}$$

6.2 Le modèle à odds proportionnels

Connu en anglais sous le nom de *proportional odds model*, ce modèle est le plus répandu dans la régression logistique ordinale.

6.2.1 Présentation du modèle

6.2.1.1 Formulation générale

Le modèle des cotes proportionnelles consiste en la comparaison, en ce qui concerne les rapports de cotes, entre deux catégories à partir d'un point de coupure établi selon l'objectif k. Le principe du modèle à odds proportionnels est de construire un modèle pseudo-logique où la sortie d'intérêt serait l'évènement $Y \leq k$. Mathématiquement, un tel modèle s'écrit :

$$\frac{\mathbb{P}(Y \leq k|X)}{\mathbb{P}(Y > k|X)} = \exp\left(\beta_0^k + \beta^* X^*\right) \quad 0 \leq k \leq K-1 \tag{6.5}$$

ou encore

$$\texttt{logit}\left(\mathbb{P}(Y \leq k|X)\right) = \beta_0^k + \beta^* X^* \quad 0 \leq k \leq K-1 \tag{6.6}$$

Dans ce modèle, l'effet principal d'une variable explicative X_j est le même à travers les $K-1$ logits de Y, ce qui veut dire que les graphes $\left(X_j, logit\right)$ sont parallèles entre eux pour les différentes valeurs de X_j. En effet, pour chaque variable explicative X_j, les $K-1$ logit ont les mêmes pentes mais différentes constantes $\beta_0^{(k+1)}$, ces droites sont donc parallèles entre elles et décalées suivant les valeurs de la constante $\beta_0^{(k+1)}$. Cette supposition constitue l'hypothèse H_0 pour l'utilisation du modèle des cotes proportionnelles. Si H_0 est rejetée par le test de χ^2, ceci veut dire que les coefficients ne sont pas les mêmes dans tous les logits, alors le modèle de la régression logistique ordinale ne peut pas ajuster correctement les données. Dès lors, le modèle multinomial constitue une alternative.

Remarque 6.1 — *Il n'est pas évident de prime abord que ces équations soient compatibles...*
- — *...mais on peut montrer qu'elles le sont.*
- — *Si $K = 1$ on retrouve un modèle logistique inversé (en permutant les rôles de $Y = 0$ et $Y = 1$).*
- — *Il y a $K + p$ paramètres à estimer dans ce modèle.*
- — *C'est le modèle le plus utilisé...*
- — *...mais certains logiciels le spécifient différemment (se méfier...)*
- — *Les procédures habituelles (IC, tests...) s'adaptent bien.*

Remarque 6.2 (Pourquoi « odds proportionnels » ?) *L'écriture*

$$\frac{\mathbb{P}(Y \leq k|X)}{\mathbb{P}(Y > k|X)} = exp\left(\beta_0^k + \beta^* X^*\right) \quad 0 \leq k \leq K-1 \tag{6.7}$$

dit que la cote (l'odds) de $Y \leq k$ sachant X vaut exp $(\beta_0^k + \beta^* X^*)$. Du coup, une expression extrêmement simple de l'odds ratio correspond à :

$$OR(x_1, x_2, Y \leq X \text{ vs } Y > k) = exp\left(\beta * (x_1^* - x_2^*)\right) \tag{6.8}$$

Il est évidemment remarquable que cet odds -ratio est indépendant du niveau k choisi, le nom du modèle (odds proportions est tiré de cette propriété).

6.2.1.2 Autre formulation

Nous revenons d'abord au cas où Y est binaire (0 ou 1) et supposons, sans perte de généralité, que nous somme en présence d'une seule variable explicative X. On introduit ε une variable aléatoire centrée et une variable latente (non observée) $Y^* = \tilde{\beta}_0 + \beta_1 x + \varepsilon$ telle que $Y|X = x$ vaut 1 lorsque la variable latente Y^* est grande (supérieure à un seuil s) et 0 sinon. Nous obtenons :

$$\mathbb{P}\left(Y = 1|X = x\right) = \mathbb{P}\left(\tilde{\beta}_0 + \beta_1 x + \varepsilon > s\right) = \mathbb{P}\left(-\varepsilon < -s + \tilde{\beta}_0 + \beta_1 x\right) = F\left(\beta_0 + \beta_1 x\right)$$

où F est la fonction de répartition de la variable $-\varepsilon$ et $\beta_0 = -s + \tilde{\beta}_0$. Pour finir de spécifier le modèle, il reste à choisir la fonction de répartition F. Si On choisit

$$F(x) = \frac{1}{1 + e^{-x}}, \quad \forall x \in \mathbb{R} \tag{6.9}$$

on obtient le modèle logistique. Si F est la fonction de répartition associée à la loi normale centrée

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} \mathrm{d}x, \quad \forall x \in \mathbb{R} \tag{6.10}$$

nous obtenons alors le modèle probit.

La figure 6.1 montre les fonctions de répartion des lois normale et logistique :

```
# Représentation graphique
import numpy as np
import pandas as pd
import scipy.stats as st
from plotnine import *
def logit(x):return 1/(1+np.exp(-x))
pas =np.linspace(-4,4,100)
fn = pd.DataFrame({"pas": pas, "logit":logit(pas),"probit":st.norm.cdf(pas)})
fn_long = fn.melt(id_vars=["pas"],var_name="link",value_name="value")
print((ggplot(fn_long,aes(x="pas",y="value",group="link"))+
       geom_line(aes(linetype="link", color="link"))+
       theme(legend_position=(0.2,0.75),legend_direction="vertical")))
```

Plus généralement, le modèle polytomique peut être présenté comme une généralisation du modèle dichotomique avec cette fois Y prenant K modalités ordonnées. On se

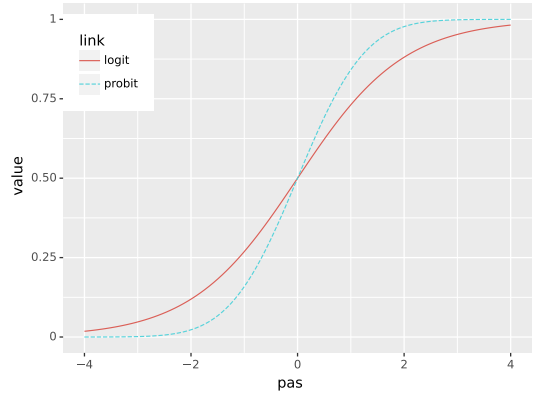


Figure 6.1 – Fonctions de répartition des loins normale (tirets) et logistique (trait plein)

place toujours dans le cas d'une seule variable explicative X. On introduit non lus un seul seuil, mais plusieurs seuils $\alpha_1, \dots, \alpha_{K-1}$ déterministes tels que :

$$(Y|X=x) = \begin{cases} 1 & \text{si} \quad Y^* < \alpha_1 \\ j & \text{si} \quad \alpha_{j-1} \leq Y^* < \alpha_j, j = 2, \dots, K-1 \\ k & \text{si} \quad Y^* \geq \alpha_{K-1} \end{cases} \quad \text{avec} \quad Y^* = \beta_1 x + \varepsilon$$

Le choix de la fonction de répartition logistique (6.9) conduit au modèle :

$$\mathbb{P}_\beta\left(Y \leq j|x\right) = F\left(\alpha_j - \beta_1 x\right), \quad j = 1, \dots, K-1$$

où encore

$$\texttt{logi}\left(p_\beta^j(x)\right) = \alpha_j - \beta_1 x, \quad j = 1, \dots, K-1 \tag{6.11}$$

Si on est en présence de p variables explicatives, le modèle devient :

$$\texttt{logit}\left(p_\beta^j(x)\right) = \alpha_j - \beta_1 x_1 - \cdots - \beta_p x_p, \quad j = 1, \dots, K-1 \tag{6.12}$$

ou encore

$$p_\beta^j(x) = \frac{\exp\left(\alpha_j - \beta_1 x_1 - \cdots - \beta_p x_p\right)}{1 + \exp\left(\alpha_j - \beta_1 x_1 - \cdots - \beta_p x_p\right)}$$

Dans ce modèle, seule la constante diffère suivant les différents niveaux de Y. Ce modèle nécessite l'estimation de $K + p - 1$ coeffcients (p pentes et $K - 1$ constantes car $\sum_{j=1}^{k=K} \mathbb{P}\left(Y = j|X = x\right) = 1$).

Remarque 6.3 *Suivant le logiciel, les coefficients estimés peuvent différer. La procédure LOGISTIC de SAS estime par exemple les pentes $b_j = -\beta_j$. Sous R, les fonctions polr et vglm des librairies MASS et VGAM permettent de construire des modèles logistiques pour expliquer une variable qualitative ordinale. Il en est de même pour la fonction OrderedModel de la librairie Statsmodels de Python.*

6.2.2 Estimateur du maximum de vraisemblance et odds ratio

6.2.2.1 Estimateur du maximum de vraisemblance

Les $p + K - 1$ paramètres sont estimés par maximum de vraisemblance. La log-vraisemblance pour un couple (x_i, y_i) s'écrit :

$$\ln \mathcal{L}(\alpha, \beta | x_i, y_i) = \sum_{j=1}^{j=K-1} 1_{y_i=j} \log \left[F\left(\alpha_j - \beta x\right) - F\left(\alpha_{j-1} - \beta x\right) \right] \tag{6.13}$$

Ainsi, pour toute le jeu de données, nous avons :

$$\log \mathcal{L}(\alpha, \beta | x_i, y_i) = \sum_{i=1}^{i=n} \ln \mathcal{L}(\alpha, \beta | x_i, y_i) \tag{6.14}$$

Dans ces différentes expressions, F représente la fonction de répartition de la loi logistique.

On en déduit l'estimateur du maximum de vraisemblance par des méthodes numériques. Il a les mêmes propriétés que le modèle logistique (Intervalle de confiance, test de significativité et adéquation analogues).

Exemple 6.1 *Données de portefeuille*

Nous allons utiliser le jeu de données du portefeuille. Nous supposons que qu'il y a un ordre dans les réponses des étudiants. On cherche à expliquer la variable `WALLET` par les autres variables. On note $Y =$ `WALLET`, $X_1 =$ `MALE`, $X_2 =$ `BUSINESS`, $X_3 =$ `PUNISH` et $X_4 =$ `EXPLAIN`. Le modèle s'écrit :

$$\texttt{logit}\left(p_\beta^j\right) = \alpha_j - \beta_1 1_{x_1=1} - \beta_2 1_{x_2=1} - \beta_3 1_{x_3=2} - \beta_4 1_{x_3=3} - \beta_5 1_{x_4=1}$$

```
# chargement des données
from statsmodels.miscmodels.ordinal_model import OrderedModel
portefeuille = pd.read_sas('./donnee/portefeuille.sas7bdat')
portefeuille = portefeuille.astype(int).astype('category')
portefeuille.wallet = pd.Categorical(portefeuille.wallet,ordered=True,
                                     categories=[1,2,3])
formula = 'wallet~C(male)+C(business)+C(punish)+C(explain)'
ordlogit = (OrderedModel.from_formula(formula,portefeuille,distr='logit')
                        .fit(method='bfgs', disp=False))
ologit_coef =  pd.concat([ordlogit.params, ordlogit.bse,ordlogit.tvalues,
                        ordlogit.pvalues,ordlogit.conf_int(alpha=0.05)],axis=1)
```

6.2.2.2 Odds ratio

Dans le cas où la variable Y est ordonnée, on définit l'odds relativement à l'évènement $\{Y \leq k\}$ par

Table 6.1 – Coefficients du modèle polytomique ordonnée - Données portefeuille

	coef	std err	t	P>|t|	[0.025	0.975]
C(male)[T.1]	-1.0598	0.3274	-3.2366	0.0012	-1.7016	-0.4180
C(business)[T.1]	-0.7387	0.3556	-2.0773	0.0378	-1.4357	-0.0417
C(punish)[T.2]	-0.6278	0.4048	-1.5509	0.1209	-1.4212	0.1656
C(punish)[T.3]	-1.4031	0.4823	-2.9093	0.0036	-2.3483	-0.4579
C(explain)[T.1]	1.0519	0.3408	3.0866	0.0020	0.3840	1.7199
1/2	-2.5677	0.4190	-6.1285	0.0000	-3.3889	-1.7466
2/3	0.5759	0.1326	4.3427	0.0000	0.3160	0.8359

$$\texttt{odds}(x_i) \quad = \quad \frac{\mathbb{P}_\beta(Y \leq k|X=x)}{1-\mathbb{P}_\beta(Y \leq k|X=x)} \quad = \quad \exp\left(\beta^k + \beta^* x^*\right)$$

L'odds ratio relativement à l'évènement $\{Y \leq k\}$ s'écrit donc :

$$\texttt{OR}(x_1, x_2) \quad = \quad \exp\left((x_1 - x_2)' \beta^*\right)$$

Cet odds ratio ne dépend pas de la modalité k, on dit que l'hypothèse des seuils aléatoires se traduit par une « hypothèse de proportionnalité des odds ratio ».

```
# Odds ratio
OR = (pd.concat([ordlogit.params[:-2],ordlogit.conf_int(alpha=0.05)],axis=1)
        .apply(lambda x : np.exp(x),axis=0))
```

Table 6.2 – Odds ratio du modèle polytomique ordonnée - Données portefeuille

	OR	[0.025	0.975]
C(male)[T.1]	0.3465	0.1824	0.6583
C(business)[T.1]	0.4777	0.2380	0.9592
C(punish)[T.2]	0.5338	0.2414	1.1801
C(punish)[T.3]	0.2458	0.0955	0.6326
C(explain)[T.1]	2.8632	1.4681	5.5840

6.2.3 Égalité des pentes

L'écriture

$$\texttt{logit}\left(\mathbb{P}(Y \leq k|X)\right) = \beta_0^k + \beta^* X^* \quad 0 \leq k \leq K-1 \tag{6.15}$$

dit que les droites représentant la fonction $x_j \mapsto \texttt{logit}\left(\mathbb{P}(Y \leq k|X=x_j\right)$ sont parallèles (de pente β_j)suivant les modalités k.

Supposons pour simplifier que l'on dispose d'une seule variable explicative X et considérons le modèle suivant :

$$\texttt{logit}\left(\mathbb{P}(Y \leq k|X)\right) = \beta^k + \beta^* X^* \tag{6.16}$$

On constate que seule la constante diffère selon k, c'est pourquoi on parle d'égalité des pentes.

```
 # Graphique
from plotnine import *
print((ggplot()+geom_line(aes(x=[0,10],y=[0,.5]),color = 'blue')+
         geom_line(aes(x=[0,10],y=[0.15,0.65]),color = 'blue')+
         geom_line(aes(x=[0,10],y=[0.3,0.8]),color = 'blue')+
         geom_text(aes(x=[10.1,10.1,10.1],y=[0.5,0.65,0.8],
                       label=['k=1','k=2','k=3']))+
         ylim((0.0,1.0))+labs(x="x",y='$p_k(x)$')))
```

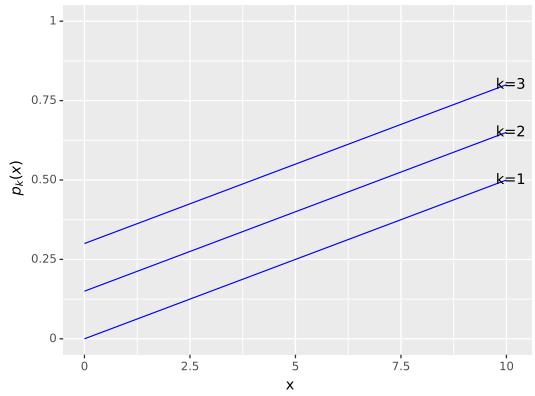


Figure 6.2 – Représentation du modèle $p_\beta^k(x) = \beta^k - \beta^* x = \eta_k(x)$

Si on envisage des valeurs différentes pour les paramètres de pentes $\beta_k (k = 1, K-1)$ alors les droites de la figure 6.2 vont se couper. Il est facile de voir que ceci remet en cause le caractère ordonné des modalités de la variable expliquée. En effet, supposons qu'au delà d'une valeur x_0, la première droite ($k = 1$) se situe au-dessus de la seconde ($k = 2$), on a alors :

$$\forall x > x_0, \quad \mathbb{P}_\beta(Y \leq 1 | X = x) > \mathbb{P}_\beta(Y \leq 2 | X = x)$$

Un tel résultat est évidemment gênant : il remet en cause la modélisation retenue, qui distribue les différentes modalités de Y sur un même axe ordonné. Il faut dans ce cas se tourner vers un modèle plus général (modèle logistique multinomial).

6.3 Tests sur la régression logistique ordonnée

6.3.1 Tests sur les coefficients

Comme pour la régression binaire et multinomial, nous disposons de deux outils pour réaliser les tests : la statistique du rapport de vraisemblance qui correspond toujours à la comparaison des déviances des régressions sous H_0 et H_1 et la statistique de Wald qui exploite la normalité asymptotique des estimateurs du maximum de vraisemblance.

6.3.1.1 Test du rapport de vraisemblance

Pour ce test, il s'agit d'optimiser la vraisemblance en forçant $\beta_j = 0$. Nous obtenons le modèle contraint (modèle sous H_0), d'en extraire la déviance D_{H_0} que l'on comparera à celle du modèle complet D_M. La statistique de test s'écrit :

$$LR = D_{H_0} - D_M \tag{6.17}$$

La statistique LR suit une χ^2 à 1 degré de liberté sous H_0.

Pour notre exemple, Nous souhaitons tester l'hypothèse nulle

$$H_0 : \beta_1 = \cdots = \beta_5 = 0$$

contre l'hypothèse alternative :

$$H_1 : \exists j \in \{1, \ldots, 5\}, \beta_j \neq 0$$

```
## Test du rapport de vraisemblance
lr_test = pd.DataFrame({"statistic":ordlogit.llr,"pvalue":ordlogit.llr_pvalue},
                       index=["Likelihood ratio test"])
```

Table 6.3 – Test du rapport de vraisemblance

	statistic	pvalue
Likelihood ratio test	44.8047	0

Au risque de 5%, la pvalue est égale à 1.5897328×10^{-8}. Nous rejetons l'hypothèse H_0 de nullité des coefficients.

6.3.1.2 Test de Wald

La statistique de Wald est formée par le rapport entre le carré du coefficient est sa variance.

$$W_p^k = \left(\frac{\widehat{\beta}_p^k}{\widehat{\sigma}_{\widehat{\beta}_p^k}} \right)^2 \tag{6.18}$$

La statistique W_p^k suit une χ^2 à 1 degré de liberté sous H_0.

Pour effectuer le test de Wald, nous avons besoin de la matrice de variance et covariance des coeffcients

```
# Matrice de variance - covariance des coeffcients
cov_params = ordlogit.cov_params()
```

Nous effectuons le test de Wald en faisant attention de bien choisir les indices des variables pour lesquelles nous souhaitons effectuer ce test.

Table 6.4 – Matrice de variance et covariance des coefficients

	C(male)[T.1]	C(business)[T.1]	C(punish)[T.2]	C(punish)[T.3]	C(explain)[T.1]	1/2	2/3
C(male)[T.1]	0.1072	-0.0227	0.0151	0.0133	-0.0053	0.0668	-0.0049
C(business)[T.1]	-0.0227	0.1265	-0.0103	-0.0117	-0.0097	0.0176	-0.0044
C(punish)[T.2]	0.0151	-0.0103	0.1639	0.0513	0.0317	0.0726	-0.0038
C(punish)[T.3]	0.0133	-0.0117	0.0513	0.2326	0.0325	0.0820	-0.0105
C(explain)[T.1]	-0.0053	-0.0097	0.0317	0.0325	0.1162	0.0697	0.0066
1/2	0.0668	0.0176	0.0726	0.0820	0.0697	0.1755	-0.0263
2/3	-0.0049	-0.0044	-0.0038	-0.0105	0.0066	-0.0263	0.0176

```
# Indice des variables
index = [i for i in range(5)]
# Statistique de Wald
coef_W = ordlogit.params.iloc[index]
cov_W  = cov_params.iloc[index,index]
statW = np.dot(coef_W, np.dot(np.linalg.inv(cov_W),coef_W))
# p-value avec ddl=len(index)
pvalue = 1.0 - st.chi2.cdf(statW,len(index))
wald_test=pd.DataFrame({"statistic":statW,"ddl":len(index),"pvalue":pvalue},
                       index=["Wald test"])
```

Table 6.5 – Test de Wald de nullité de tous les coefficients - Calcul directe

	statistic	ddl	pvalue
Wald test	38.6492	5	0

Avec un $\chi^2(5)$, nous avons une p - value de 2.7939863×10^{-7}. Nous rejetons l'hypothèse nulle au risque 5%.

Statsmodels propose l'outil wald_test() pour réaliser les tests généralisés. L'enjeu réside alors dans la définition de la matrice **r_matrix** permettant de spécifier les coefficients à tester.

```
# Test de Wald
R = np.eye(len(ordlogit.params))
R[5,5] = R[6,6] = 0
# hypotheses = '(male=0),(business=0),(punish=0),(explain=0)'
Wald = ordlogit.wald_test(R)
Wtest = pd.DataFrame({"statistic" : Wald.statistic[0],"ddl" : Wald.df_denom,
                      "pvalue" : Wald.pvalue},index=["Wald test"])
```

Table 6.6 – Test de Wald de nullité de tous les coeffcients - Statsmodels

	statistic	ddl	pvalue
Wald test	38.6492	5	0

Les résultats sont identiques, par conséquent, les conclusions le sont également.

6.3.2 Test d'égalité des pentes

On se pose la question de vérifier si la modélisation en terme de seuil aléatoire est « raisonnable » vis à vis de nos données. Nous avons vu dans la partie précédente que cette

hypothèse se traduit par une hypothèse d'égalité des pentes ou de proportionnalité des odds ratio. Un moyen de vérifier si cette hypothèse est raisonnable consiste à tester l'hypothèse d'égalité des pentes. Cette approche consiste tout simplement à comparer le modèle en question (avec égalité des pentes) à un modèle où on lève l'égalité des pentes. Il s'agit de considérer les hypothèses :

$$H_0 : \beta_j^1 = \cdots = \beta_j^{K-1}, \quad \forall j = 1, \ldots, p$$

contre

$$H_1 : \exists j \in \{1, \ldots, p\}, \exists\, (k, l) \in \left(\{1, \ldots, K-1\}\right)^2 \quad \text{tels que} \quad \beta_j^k \neq \beta_j^l$$

pour le modèle

$$\texttt{logit}\left(p_\beta^k(x)\right) = \alpha_k - \sum_{j=1}^{j=p} \beta_j^k x_j, \quad k = 1, \ldots, K-1 \tag{6.19}$$

Sous H_0 (égalité des pentes), nous retrouvons le modèle polytomique ordinal. Ce test peut être mené à l'aide des statistiques de Wald, du rapport de vraisemblance et du score. Par exemple, la `PROC LOGISTIC` de `SAS` effectue le test du score, « *Score Test for Proportional Odds Assumption* ». Si H_0 est vérfiée, la pente de la log - vraisemblance pour l'estimateur du maximum de varisemblance contraint $\hat{\gamma} = \left(\hat{\alpha}, \ldots, \hat{\alpha}_k, \hat{\beta}, \ldots, \hat{\beta}_p\right)$ doit être proche de 0. On utilise le fait que sous H_0 la statistique du score

$$\nabla \mathcal{L}_n\left(\hat{\gamma}\right) \hat{\mathcal{J}}_{H_0}^{-1} \nabla \mathcal{L}_n\left(\hat{\gamma}\right) \tag{6.20}$$

converge en loi vers une loi de χ^2 à $p \times (K-2)$ degrés de libertés ($\hat{\mathcal{J}}_{H_0}$ est un estimateur de la matrice d'information de Fisher du modèle). Le nombre de degrés de liberté s'obtient en faisant la différence du modèle 6.19 $(K - 1 + p(K-1))$ et celle du modèle sous H_0 $(K - 1 + p)$.

La probabilité critique du test d'égalité des pentes pour la statistique de rapport de vraisemblance s'obtient faciblement sous Python. Nous comparons le modèle de régression polytomique ordonnée au modèle logistique multinomial.

```
# Modèle logistique multinomial
import statsmodels.formula.api as smf
df = portefeuille.copy()
df.wallet = df.wallet.astype('int')
mnlogit = smf.mnlogit(formula,data=df).fit(disp=False)
# Statistique du rapport
statRV = - 2*(ordlogit.llf - mnlogit.llf)
# Degré de liberté
ddl = ordlogit.df_resid - mnlogit.df_resid
# Pvalue
pvalue = 1.0 - st.chi.cdf(statRV,ddl)
lr_test = pd.DataFrame({"statistic" : statRV, "ddl" : ddl, "pvalue" : pvalue},
                       index = ["likelihood test"])
```

Avec un χ^2 à 5 degrés de liberté, nous avons une p - value de 3.2128463×10^{-6}. Nous rejetons l'hypothèse nulle au risque 5%.

Table 6.7 – Test d'égalité des pentes - Test du rapport de vraisemblance

	statistic	ddl	pvalue
likelihood test	5.7746	5	0

6.3.3 Evaluation des classifieurs

6.3.3.1 Prédiction

Les prédictions peuvent se faire sur les mêmes données qui ont permis de construire le modèle ou sur des nouvelles données. Le principe des prédictions est très similaire à celui utilisé pour la régression logistique. Seulement comme il y a un nombre fini de combinaisons possibles, tous les individus avec une même combinaison des variables explicatives auront la même probabilité d'être $Y = 1$, $Y = 2$ ou $Y = 3$. Sous Python, on obtient directement les probabilités d'être $Y = 1$, $Y = 2$ et $Y = 3$ pour chaque combinaison en faisant :

```
# probabilités d'appartenance à la classe
yprob = ordlogit.predict(portefeuille)
yprob.columns = [1,2,3]
# 5 premières observations
yprob.head()
```

```
##           1         2         3
## 0  0.125653  0.334146  0.540201
## 1  0.047794  0.181361  0.770845
## 2  0.026092  0.110855  0.863052
## 3  0.125653  0.334146  0.540201
## 4  0.071767  0.242326  0.685907
```

Par la suite, on calcule la classe prédicte. La règle de prédiction consiste à affecter un individu à la classe pour laquelle il possède la plus grande probabilité. Si on note $\hat{\pi}_i^k$ $(k = 1, 2, 3)$, la probabilité d'appartenir à la classe k pour l'individu i. Pour le premier individu, on a les probabilités d'appartenance aux classes suivantes : $\hat{\pi}_1^1 = 0.13$, $\hat{\pi}_1^2 = 0.33$ et $\hat{\pi}_1^3 = 0.54$. On constate que $\hat{\pi}_1^3 > \hat{\pi}_1^2 > \hat{\pi}_1^1$. On affecte l'individu 1 à la classe 3. On fait le même raisonnement pour les autres individus.

```
# Classe prédicte
D = portefeuille.copy()
D["pred"] = np.asarray(yprob).argmax(1)
# Mapping des modalités
D["pred"] = D["pred"].map({0:1,1:2,2:3})
D.head()
```

```
##   wallet male business punish explain  pred
## 0      2    0        0      2       0     3
## 1      2    0        0      2       1     3
## 2      3    0        0      1       1     3
## 3      3    0        0      2       0     3
## 4      1    1        0      1       1     3
```

6.3.3.2 Matrice de confusion

Etant donnée que nous sommes dans un cadre de régression logistique où $K > 2$, la matrice de confusion du modèle polytomique ordonnée est identique à celle d'un modèle multinomial. Les indicateurs de performance sont les mêmes et s'interprètent de la même façon.

Nous définissons notre matrice de confusion à partir des données « portefeuile ».

```
# Matrice de confusion - donnée portefeuille
cm = pd.DataFrame(pd.crosstab(D.wallet,D.pred),columns=[1,2,3],index = [1,2,3])
tab = cm.copy()
tab.loc[:,'Total'] = tab.sum(axis=1)
tab.loc['Total',:] = tab.sum(axis=0)
tab = tab.reset_index()
```

Table 6.8 – Matrice de confusion- Donnee Portefeuille*

index	1	2	3	Total
1	4	11	9	24
2	3	9	38	50
3	0	12	109	121
Total	7	32	156	195

* Les lignes correspondent à la réalité et les colonnes correspondent aux prédictions du modèle.

A l'aide cette matrice de confusion, nous pouvons en déduire le taux de mauvais classement :

```
# Taux d'erreur (en resubtitution)
error_rate = 1 - np.trace(cm)/D.shape[0]
print(f"Taux d'erreur : %.4f" % (error_rate))
```

```
## Taux d'erreur : 0.3744
```

Le taux de mauvais classement est 37.44%.

Nous calculons le rappel et la précision par classe :

```
# rappel et précision
rappel = np.diagonal(cm)/np.sum(cm,axis=1)
precision = np.diagonal(cm)/np.sum(cm,axis=0)
rp = pd.DataFrame(np.c_[rappel,precision],columns = ['rappel', 'precision'],
                  index = [1,2,3])
rp = rp.reset_index().rename(columns={"index":"classe"})
```

6.3.3.3 Métrique « per class »

Nous avons vu au chapitre précédent les différentes métriques « per class ». Nous les calculons ici :

Table 6.9 – Rappel et précision par classe

classe	rappel	precision
1	0.1667	0.5714
2	0.1800	0.2812
3	0.9008	0.6987

```
# Métrique << per class >>
import sklearn.metrics as mt
cm1,cm2,cm3 = mt.multilabel_confusion_matrix(D.wallet,D.pred)
mat = [cm1,cm2,cm3]
label = ["class : 1", "class: 2", "class: 3"]
ologit_metrics = pd.concat(map(multiclassmetrics,mat,label),axis=1)
```

Table 6.10 – Métriques per class

	class : 1	class : 2	class : 3
Sensitivity	0.1667	0.1800	0.9008
Specificity	0.9825	0.8414	0.3649
False positive rate	0.0175	0.1586	0.6351
False negative rate	0.8333	0.8200	0.0992
True negative rate	0.9825	0.8414	0.3649
Negative predictive value	0.8936	0.7485	0.6923
False discovery rate	0.4286	0.7188	0.3013
Precision	0.5714	0.2812	0.6987
accuracy	0.8821	0.6718	0.6974
F1 score	0.2581	0.2195	0.7870
Matthews Correlation Coefficient MCC	0.0001	0.0000	0.0000

6.3.3.4 Métrique « macro » classique

La métrique « macro » classique revient à faire la moyenne des métrique « per class ». En termes mathématiques, on obtient la formule suivante pour une métrique

$$\text{macro-metric} = \frac{1}{K}\sum_{k=1}^{k=K} \text{metric}_k \tag{6.21}$$

La métrique macro classique accorde autant d'importance à chacune des classes, epu importe la proportion d'individus qu'elles contiennent.Cela permet de ne pas négliger une classe qui serait sous - représentée dans nos données et rend cette métrique robuste au déséquilibre des classes.

6.3.3.5 Métrique « macro » pondérée

La variante « macro » pondérée revient à faire la moyenne pondérée des métriques « per class ». Chaque métrique a une importance pproportionnelle à la proportion d'individus dans la classe correspondante. Mathématiquement, cela se traduit par la formule suivante pour une métrique :

$$\texttt{macro-weighted-metric} = \sum_{k=1}^{k=K} p_k \text{metric}_k \tag{6.22}$$

avec $p_k = \frac{n_k}{n}$, où n_k désigne le nombre d'individus dans la classe k.

Le **macro-weighted-recall** est la **proportion d'individus correctement détectés**.

La métrique macro - weighted accorde une importance plus grande aux classes les plus nombreuses dans les données, ce qui la rend plus représentative. C'est un avantage lorsque les classes jouent le même rôle. Cependant si une classe a un rôle particulier et qu'elle est sous-représentée dans les données, une métrique macro-weighted ne permettra pas de détecter des changements de performance sur cette classe particulière.

Remarque 6.4 (Quelle variante macro faut - il privilégier ?) *Le paramètre qui détermine le choix entre macro classique et macro - weighted est le déséquilibre des données :*

- *Pour les données équilibrées, la proportion de chaque classe est environ* $\frac{1}{K}$ *et les deux variantes sont équivalentes. Pour les* 3 *classes, ces proportions valent environ toutes* 33.33%.
- *Pour les données déséquilibrées, les deux variantes ne seront pas équivalentes.*

6.3.3.6 Métrique micro

L'approche « micro » utilise directement la métrice de confusion mulri-classe pour faire une matrice de confusion « synthétique ». Elle reprend les termes de la matrice de confusion binaire : True Positive (TP), True Negative (TN), False Positive (FP) et False Negative. Ces termes sont calculés de la manière suivante :

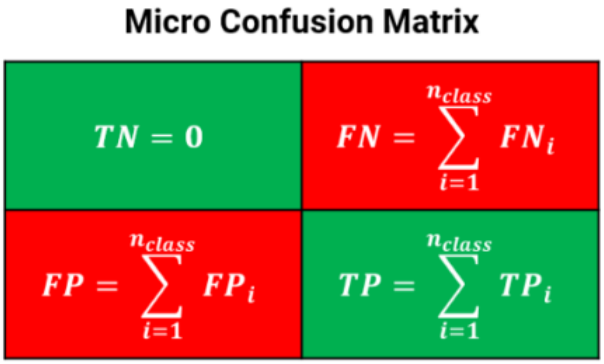


Figure 6.3 – Valeurs utilisées pour le calcul des métriques micro

Remarque 6.5 *En classification multi - classe, il n'y a pas de classe « négative » et par conséquent le terme des vrais négatifs est nul ::*

$$TN = 0 \tag{6.23}$$

Dans un problème multi - classe, toutes les prédictions fausses (cases rouges) sont à la fois un faux positif et un faux négatif. Par conséquent

$$FP = \sum_{k=1}^{k=K} FP_k = \sum_{k=1}^{k=K} FN_k = FN \tag{6.24}$$

On fera donc attention à ne pas sommer FP et FN lorsqu'on calcule les métriques.

Nous retrouvons à nouveau une matrice de confusion binaire à partir de laquelle on peut calculer les différentes métriques de classification [1].

L'approche « micro » s'intéresse à des proportions d'individus, car chaque somme se fait en nombre d'individus. L'approche macro-weighted, qui pondère par le nombre d'individus dans chaque classe, donne aussi des métriques représentatives de la proportion d'individus même si celles-ci diffèrent, on pourra retrouver des points communs entre ces deux approches. Nous avons donc les formules suivantes :

— micro - precision :

$$\text{mico-precision} = \frac{TP}{TP + FP}$$

— micro - recall :

$$\text{micro-recall} = \frac{TP}{TP + FN}$$

— micro - accuracy :

$$\text{micro-accuracy} = \frac{TP}{TP + FP(+FN) + TN}$$

— micro-f1-score

$$\text{micro-f1-score} = \mathcal{H}\left(\text{micro-recall}, \text{micro-recall}\right)$$

où $\mathcal{H}$ est l'opérateur moyenne harmonique et $(+FN)$ signifie que l'on ne prend pas en compte FN dans la somme, car c'est un doublon de FP.

Remarque 6.6 (Limites de la métrique micro) *L'approche micro étant représentative des proportions d'individus comme les métriques macro-weighted, on retrouve en présence de données déséquilibrées la limite qui a déjà été développée lors de la comparaison entre macro classique et macro-weighted. L'approche micro accorde une faible importance aux classes minoritaires.*

La deuxième limite de l'approche micro vient du fait que chaque individu dont la prédiction est fausse est à la fois un faux positif de sa classe prédite et un faux négatif de sa classe réelle. Par conséquent, $FP = FN$ et l'approche micro ne différencie pas les métriques. La micro-precision, le micro-recall, la micro-accuracy et le micro-F1-score sont tous égaux.

```
# Métrique multiclasses
def mnlogitmetrics(ytrue,ypred,average):
  recall = mt.recall_score(y_true=ytrue,y_pred=ypred,average=average)
  precision = mt.precision_score(y_true=ytrue,y_pred=ypred,average=average)
  f1score = mt.f1_score(y_true=ytrue,y_pred=ypred,average=average)
  res =pd.DataFrame({"Recall" : recall, "Precision" : precision,
                     "F1 score" : f1score},index=[average])
```

1. https ://kobia.fr/category/classification-metrics/

```
  return res
# Application
lst = ["weighted","macro","micro"]
mc_met = pd.concat(map(lambda x : mnlogitmetrics(D.wallet,D.pred,x),lst),axis=0)
```

Table 6.11 – Métriques multi - classes

	Recall	Precision	F1 score
weighted	0.6256	0.5760	0.5764
macro	0.4158	0.5171	0.4215
micro	0.6256	0.6256	0.6256

6.3.4 Exemple des données universitaires

Une étude examine les facteurs qui influencent la décision de postuler ou non aux études supérieures. On demande aux juniors universitaires s'ils sont peu probables (unlikely), assez probables (somewhat likely) ou très susceptibles (very likely) de postuler à des études supérieures. Par conséquent, notre variable de résultat comporte trois catégories. Des données sur le statut d'éducation des parents, que l'établissement de premier cycle soit public ou privé, et la moyenne cumulative actuelle (gpa) sont également collectées. Les chercheurs ont des raisons de croire que les « distances » entre ces trois points ne sont pas égales. Par exemple, la distance entre « unlikely » et « somewhat likely » peut être plus courte que la distance entre « somewhat likely » et « very likely ». Ces données sont accessibles sur https ://stats.idre.ucla.edu/stat/data/ologit.dta.

```
# Chargement des données
ologit = pd.read_stata("https://stats.idre.ucla.edu/stat/data/ologit.dta")
print(ologit.head())
```

```
##              apply  pared  public   gpa
## 0      very likely      0       0  3.26
## 1  somewhat likely      1       0  3.21
## 2         unlikely      1       1  3.94
## 3  somewhat likely      0       0  2.81
## 4  somewhat likely      0       0  2.53
```

Ce jeu de données comporte une variable à trois niveaux appelée `apply`, avec les niveaux « unlikely », « somewhat likely » et « very likely », codés 1, 2 et 3, respectivement, que nous utiliserons comme variable de résultat. Nous avons également trois variables que nous utiliserons comme prédicteurs : `pared`, qui est une variable 0/1 indiquant si au moins un parent a un diplôme d'études supérieures ; `public`, qui est une variable 0/1 où 1 indique que l'établissement de premier cycle est public et 0 privé, et gpa, qui est la moyenne pondérée cumulative de l'étudiant.

Nous ajustons le modèle de régression logistique ordinal suivant.

$$\texttt{logit}\left(p_\beta^j\right) = \alpha_k - \beta_1 1_{\texttt{pared}=1} - \beta_2 1_{\texttt{public}=1} - \beta_3 \texttt{gpa} \tag{6.25}$$

```
# Régression polytomique ordinale
mdl_ologit = OrderedModel.from_formula('apply~C(pared)+C(public)+gpa',data=ologit,
                                       distr='logit').fit(method='bfgs',disp=False)
mdl_coef =  pd.concat([mdl_ologit.params,mdl_ologit.bse,mdl_ologit.tvalues,
                       mdl_ologit.pvalues,mdl_ologit.conf_int(alpha=0.05)],axis=1)
```

Table 6.12 – Coefficients du modèle polytomique ordonnée - Données universitaires

	coef	std err	t	P>|t|	[0.025	0.975]
C(pared)[T.1]	1.0476	0.2658	3.9416	0.0001	0.5267	1.5686
C(public)[T.1]	-0.0586	0.2979	-0.1968	0.8440	-0.6424	0.5252
gpa	0.6158	0.2606	2.3628	0.0181	0.1050	1.1266
unlikely/somewhat likely	2.2035	0.7795	2.8267	0.0047	0.6757	3.7314
somewhat likely/very likely	0.7398	0.0801	9.2363	0.0000	0.5828	0.8967

Le modèle estimé peut s'écrire sous la forme :

$$\begin{cases} \texttt{logit}\left(\widehat{\mathbb{P}}_\beta(Y \leq 1)\right) = & 2.2035 - 1.0476 \times 1_{\text{pared}=1} - (-0.0586) \times 1_{\text{public}=1} - 0.6158 \times \text{gpa} \\ \texttt{logit}\left(\widehat{\mathbb{P}}_\beta(Y \leq 2)\right) = & 0.7398 - 1.0476 \times 1_{\text{pared}=1} - (-0.0586) \times 1_{\text{public}=1} - 0.6158 \times \text{gpa} \end{cases}$$

Pour plus d'informations et d'interprétations, cliquez https ://stats.idre.ucla.edu/r/dae/ordinal-logistic-regression/.

7 Régression de Poisson

Sommaire

En statistique, la régression de Poisson est un modèle linéaire généralisé utilisé pour les données de comptage et les tableaux de contingence. Cette régression suppose que la variable réponse Y suit une loi de Poisson et que le logarithme de son espérance peut être modélisé par une combinaision linéaire de paramètre inconnus.

7.1 Rappel sur la loi de Poisson

7.1.1 Distribution de Poisson

En théorie des probabilités et en statistiques, la loi de Poisson est une loi de probabilité discrète qui décrit le comportement du nombre d'événements se produisant dans un intervalle de temps fixé, si ces événements se produisent avec une fréquence moyenne ou espérance connue, et indépendamment du temps écoulé depuis l'événement précédent.

7.1.1.1 Définition

Supposons que la réponse Y représente le nombre d'observations d'un événement dans un intervalle donné (intervalle de temps, de longueur, de surface, etc.). Supposons de plus que ces événements sont indépendants, c'est - à - dire que l'observation d'un premier événement n'influence pas la probabilité d'en observer ou non un deuxième.

Définition 7.1 *La variable Y suit une distribution de Poisson, avec un paramètre λ représentant le taux moyen d'observations par intervalle. La probabilité d'une certaine valeur y de Y en fonction de λ est donnée par l'équation suivante :*

$$\mathbb{P}(Y = y|\lambda) = e^{-\lambda}\frac{\lambda^y}{y!} \tag{7.1}$$

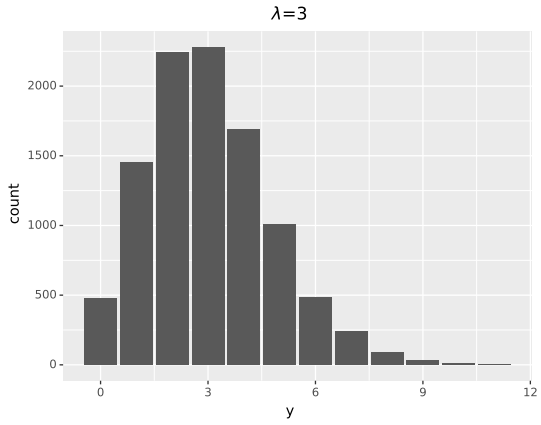
On dit alors que Y suit la loi de Poisson de paramètre λ, noté $Y \sim \mathcal{P}(\lambda)$

Par exemple, si un certain type d'événements se produit en moyenne 4 fois par minute, pour étudier le nombre d'événements se produisant dans un laps de temps de 10 minutes, on choisit comme modèle une loi de Poisson de paramètre $\lambda = 10 \times 4 = 40$.

Sous Python, la fonction numpy.random.poisson de la librairie numpy permet de générer des données suivant une distribution de Poisson.

```
# Diagramme à barres de 10000 valeurs aléatoires tirées
# de la distribution de Poisson avec lambda = 3
import numpy as np
import pandas as pd
from plotnine import *

data = pd.DataFrame(np.random.poisson(3,10000),columns=["value"])
print((ggplot(data,aes(x="value"))+geom_bar()+
          labs(title="$\lambda$=3",y="count",x="y")))
```



Le fonction scipy.stats.poisson de la librairie scipy permet de déterminer la probabilité telle que définie par la formule (7.1) :

```
# Probabilité d'obtenir Y=1 si lambda = 3
import scipy.stats as st
print(st.poisson.pmf(1,3))
```

```
## 0.14936120510359185
```

Propriété 7.1 *Le paramètre λ mesure à la fois l'espérance mathématique et la variance d'une variable aalétoire suivant une distribution de Poisson.*

7.1.2 Comparaison avec d'autres lois de probabilités

7.1.2.1 Comparaison avec la distribution binomiale

Pour la distribution binomiale, nous avions une réponse positive ou négative pour chaque individu (ou unité d'échantillonnage) et nous essayions de prédire la probabilité d'une réponse positive p.

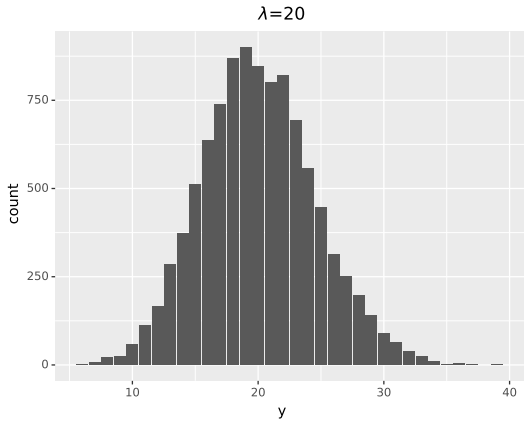
Dans la distribution de Poisson, la réponse est le nombre d'observations par unité d'échantillonage et nous essayons de prédire la moyenne de ce nombre.

Si on a une réponse binomiale qui est rare et au niveau individuel (p est petit) et que la population n est grande, alors on peut modéliser le nombre de cas au niveau de la population par une disctribution de Poisson avec $\lambda = np$.

7.1.2.2 Comparaison avec la distribution normale

Lorsque λ est assez grand, la distribution de Poisson devient plus symétrique et tend vers une distribution normale.

```
# Simulation d'une loi de Poisson avec lambda = 20
data['normale'] =np.random.poisson(20,10000)
print((ggplot(data,aes(x="normale"))+geom_bar()+
          labs(title="$\lambda$=20",y="count",x="y")))
```



Autrement dit, si le nombre moyen est assez grand, le nombre d'observations se comporte presque comme une variable continue suivant une distribution normale avec $\mu = \lambda$.

Dans ce cas, la régression linéaire pourrait s'appliquer. Toutefois, il faut se rappeler que les deux modèles font différentes suppositions au sujet de la variance de la réponse. Dans le modèe de régression linéaire, la moyenne μ dépend des prédicteurs, mais la variance σ^2 est constante. Dans le modèle de régresion de Poisson, comme on va le voir par la suite, la moyenne et la variance dépendent des prédicteurs, parce qu'elles sont toutes deux égales à λ. Il faudra donc observer les graphiques de résidus pour déterminer quel modèle est le plus approprié.

7.2 Régression de Poisson

Enore appelée régression de log - linéaire, il s'agit d'un modèle où on observe des données de comptage, par exemple des tables de contingence, chaque case correspondant à un croisement x de modalités de variables explicatives. Dans ce cas on s'intéresse au nombre Y d'observations faites dans la case du tableau correspondant à ce croisement de modalités. Le modèle Log-linéaire suppose que chaque Y est de loi de Poisson de paramètre $\lambda(x)$.

7.2.1 Présentation du modèle

La régression de Poisson est un modèle linéaire généralisé où la réponse $Y = y$ suit une distribution de Poisson :

$$Y \sim \lambda(x)$$

Définition 7.2 (Régression log - linéaire) *Soit Y uen variable de comptage à expliquer par le vecteur $X = \left(1, X_1, \dots, X_p\right)'$. Le modèle log - linéaire propose une modélisation de la loi de $Y|X = x$ par une loi de Poisson de paramètre $\lambda = \lambda(x)$ telle que :*

$$\log \mathbb{E}\left[Y|X = x\right] = \beta_0 + \sum_{j=1}^{j=p} \beta_j x_j = x\beta \tag{7.2}$$

On sait que $\mathbb{E}\left[Y|X = x\right] = \lambda(x)$, ce qui implique :

$$\log \lambda(x) = \eta(x) = x\beta \tag{7.3}$$

Selon les propriétés de la fonction logarithmique, pour tout x, une valeur de 0 du prédicteur η correspond à $\lambda = 1$, une valeur positive de η correspond à $\lambda > 1$ et une valeur négative à $\lambda < 1$.

```
# Représentation graphique
pas = np.linspace(-2,2,100)
df = pd.DataFrame({"eta" : pas, "lamb" : np.exp(pas)})
print((ggplot(df,aes(x="eta",y="lamb"))+geom_line(color="black")+
         geom_segment(aes(x = 0, y = 0, xend = 0, yend = 1),
                      linetype="dashed",color="blue")+
         geom_segment(aes(x = -2, y = 1, xend = 0, yend = 1),
                      linetype="dashed",color="blue")+
         labs(x= "$\eta$", y = "$\lambda$")))
```

En inversant le logarithme, on obtient une relation exponentielle entre la réponse moyenne λ et les prédicteurs

$$\lambda(x) = \exp\left\{\beta_0 + \sum_{j=1}^{j=p} \beta_j x_j\right\} \tag{7.4}$$

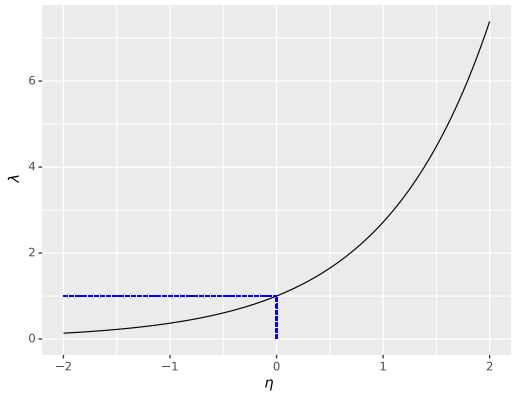


Figure 7.1 – Représentation de l'équation $\log(\lambda) = \eta$.

Puisque l'addition de puissances correspond à une multiplication, ce modèle correspond à des effets multiplicatifs des prédicteurs sur la réponse :

$$\lambda(x) = e^{\beta_0} e^{\beta_1 x_1} \cdots e^{\beta_p x_p}$$

Le logarithme est la fonction de lien par défaut pour la régression de Poisson. Pour bien interpréter les résultats de la régression, il faut se rappeler d'une transformation logarithmique est appliquée à la réponse moyenne.

7.2.2 Estimation des paramètres

Comme indiqué plus haut, à partir d'un paramètre β et d'un vecteur $X \in \mathbb{R}^{n+1}$, la variable de sortie Y suit une loi de Poisson de paramètre :

$$\lambda(x) = \mathbb{E}\left(Y|X = x\right) = \exp\left(x\beta\right) \tag{7.5}$$

La fonction de masse de cette loi de Poisson est alors :

$$\mathbb{P}\left(Y = y|X = x; \beta\right) = \exp\left(-\lambda(x)\right) \frac{\lambda(x)^y}{y!}$$

Supposons que l'on ait accès à une collection de n couples indépendants : $(X_i, y_i)_{1 \leq i \leq n}$. Alors, pour un vecteur β donné, la fonction de vraisemblance (c'est - à - dire la probabilité d'obtenir cet ensemble de données particulier) s'écrit :

$$\mathcal{L}\left(\beta|X, y\right) = \prod_{i=1}^{i=n} \mathbb{P}\left(Y = y_i|X = x_i; \beta\right) = \prod_{i=1}^{i=n} \frac{e^{-\exp(x_i\beta)[\exp(x_i\beta)]^{y_i}}}{y_i!} \tag{7.6}$$

L'estimateur du maximum de vraisemblance, renvoie la valeur de β qui maximise la vraisemblance des données. Pour ce faire, puisqu'il est difficile d'optimiser une fonction écrite comme un produit dont tous les termes sont positifs, on minimise la négative log - vraisemblance :

$$l(\beta|X,y) = -\log\left(\mathcal{L}\left(\beta|X,y\right)\right) = \sum_{i=1}^{i=n}\left(-y_i \times x_i\beta + \exp\left(x_i\beta\right) + \log\left(y_i!\right)\right) \tag{7.7}$$

On peut remarquer que le terme $\log\left(y_i!\right)$ ne dépend pas de β. Puisque l'on cherche à trouver le β qui minimise cette négative log - vraisemblance, on peut la simlifier à une constante additive près. Par abus de langage, on identifie la véritable négative log - vraisemblance et la version à une constante additive près :

$$l(\beta|X,y) = \sum_{i=1}^{i=n}\left(-y_i \times x_i\beta + \exp\left(x_i\beta\right)\right)$$

Pour trouver le minimum de cette négative log - vraisemblance, on résout l'équation $\dfrac{\partial l(\beta|X,y)}{\partial\beta} = 0$, qui n'a pas de solution explicite. Nous présentons ici deux algorithme d'optimisation.

7.2.2.1 Algorithme IRLS

La méthode IRLS(*Iterated Reweighted Least Squared*) s'appuie sur la régression pondérée. On se fixe une valeur de départ et on itère jusqu'à convergence. On parle aussi de **Fisher Scoring**.

A l'étape t de l'algorithme, le coefficient estimé s'écrit :

$$\widehat{\beta}^{(t)} = \left(X'WX\right)' X'Wz \tag{7.8}$$

avec W une matrice diagonale d'ordre n dont les éléments w_{ii} correspondent à

$$w_{ii} = \widehat{\lambda}_i = \exp\left[x_i\widehat{\beta}^{t-1}\right]$$

Il s'agit de l'espérance estimée de la cible pour l'observation i. La variable z est égale à :

$$z = \ln\left(\widehat{\lambda}_i\right) + \frac{y_i - \widehat{\lambda}_i}{\widehat{\lambda}_i}$$

On doit faire attention à l'inversion de matrice. En effet, on peut faire face à un problème potentiel en cas de colinéarité.

7.2.2.2 Algorithme Newton - Raphson

C'est un algorithme d'optimisation itératif également qui s'appuie sur deux informations supplémentaires : le vecteur gradient et la matrice hessienne.

A l'étape t de l'algorithme, le coefficient estimé s'écrit :

$$\widehat{\beta}^{(t+1)} = \widehat{\beta}^{(t)} - H^{-1} \times g \tag{7.9}$$

H est la matrice Hessienne (courbure locale de la fonction objectif) définie pour deux variables x_j et x_k par :

$$H(j,k) = \sum_{i=1}^{i=n} x_{i,j} x_{i,k} \hat{\lambda}_i \quad \Rightarrow \quad H = H' W X$$

Son inverse, H^{-1}, correspond à la matrice de variance covariance des coefficients [1] :

$$\mathrm{V}\left(\hat{\beta}\right) = H^{-1} \tag{7.10}$$

g est le vecteur gradient (pente locale de la fonction objectif). Pour une variable j, on a :

$$g_j = \sum_{i=1}^{i=n} \left(y_i - \hat{\lambda}_i\right) x_{i,j} \quad \Rightarrow \quad g = X'$$

A l'optimum (lorsque la solution $\hat{\beta}$ a été trouvée), $g = 0$ [2].

Pour illustrer ce chapitre, on considère les données affairs accessibles sur la page de Ricco Rakotomalala [3].

```
# Chargement des données
D = pd.read_excel("./donnee/affairs.xlsx")
```

Table 7.1 – Données affairs

Affairs	Gender	Age	YearsMarried	Children	Religiousness	Education	Occupation	RatingMarriage
0	0	42	15.000	1	3	12	1	4
0	0	27	7.000	0	2	17	5	4
0	1	27	4.000	0	2	18	6	3
0	1	37	10.000	1	2	18	6	4
0	0	22	1.500	0	3	16	5	5
0	0	22	0.125	0	2	16	6	3
1	0	37	15.000	1	4	14	4	2
1	1	32	4.000	0	4	20	6	4
2	1	57	15.000	1	1	17	4	4
3	0	32	10.000	1	4	14	1	4
3	0	42	15.000	1	4	16	5	4
7	0	37	10.000	0	1	20	5	3
7	1	37	10.000	1	1	18	5	3
7	0	32	15.000	1	3	14	3	2
7	1	27	10.000	1	2	20	6	4
12	1	37	15.000	1	5	17	5	2
12	1	37	10.000	1	2	20	6	2
1	1	32	1.500	1	2	14	2	4
1	1	27	10.000	1	5	20	6	5
1	0	22	1.500	1	2	14	1	5

Nous disposons de 20 observations et 3 descripteurs : Gender (1 : homme, 0 : femme), `YearsMarried` : le nombre d'années de mariage et `RatingMarrage` : satisfaction dans

1. Attention colinéarité !
2. Différentes approches sont possibles pour définir la convergence.
3. voir : https ://eric.univ-lyon2.fr/ricco/cours/slides/regression_poisson.pdf

l'union (1 : faible ; 5 très satisfait(e)). La variable cible « `Affairs` » est le nombre d'infidélités d'une personne (sur 1année). On souhaite modéliser le nombre d'infidélités (`Y=Affairs`) d'une personne sur l'année passée.

```
# Régression log - linéaire  - Estimation des paramètres
import statsmodels.formula.api as smf
model = smf.poisson("Affairs~C(Gender)+YearsMarried+RatingMarriage",
                    data=D).fit(disp=0)
coef_model = model.summary2().tables[1]
```

Table 7.2 – Coefficients du modèle de Poisson

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	1.8404	0.7366	2.4984	0.0125	0.3966	3.2841
C(Gender)[T.1]	0.7508	0.2676	2.8059	0.0050	0.2264	1.2753
YearsMarried	0.0763	0.0342	2.2330	0.0256	0.0093	0.1433
RatingMarriage	-0.5951	0.1444	-4.1224	0.0000	-0.8781	-0.3122

7.2.3 Qualité de l'ajustement

7.2.3.1 Statistique de Pearson

La statistique de Pearson est définie par :

$$\chi^2_c = \sum_{i=1}^{i=n} \frac{\left(y_i - \hat{\lambda}_i\right)^2}{\hat{\lambda}_i} \tag{7.11}$$

Cette statistique suit une loi de χ^2 à $(n-p-1)$ degrés de libertés. Si on rejette H_0 (pvalue $< \alpha$), alors les valeurs prédites s'écartent significativement des valeurs observées. Le modèle est mal ajusté.

L'attribut model.fittedvalues retourne les valeurs $\ln\left(\hat{\lambda}_i\right) = X_i\beta$. Pour avoir les valeurs estimées, $\hat{\lambda}_i$, on passe à l'exponentiel :

$$\hat{\lambda}_i = \exp\left(x_i\hat{\beta}\right)$$

```
# Statistique de Pearson
import scipy.stats as st
fitval = np.exp(model.fittedvalues)
khi2 = np.sum(((D.Affairs - fitval)**2)/fitval)
# Probabilité critique
pvalue = st.chi2.sf(khi2,model.df_resid)
chitest = pd.DataFrame({"statistic":khi2,"ddl":model.df_resid,"pvalue":pvalue},
                       index=["Pearson test"])
```

Nous avons une pvalue égale à 9.5832814×10^{-4}, donc notre modèle n'est pas bien ajusté.

Table 7.3 – Statistique de Pearson

	statistic	ddl	pvalue
Pearson test	39.38	16	0.001

7.2.3.2 Statistique déviance

La statistique de déviance est donnée par[4] :

$$D = 2\sum_{i=1}^{i=n} y_i \ln\left(\frac{y_i}{\hat{\lambda}_i}\right) - \left(y_i - \hat{\lambda}_i\right) \tag{7.12}$$

Sous l'hypothèse nulle, cette statistique suit une loi de χ^2 à $n-p-1$ degrés de liberté. Si on rejette H_0 (pvalue < 0), le modèle est mal ajusté.

```
# Valeurs pour statistique de dev
rd = np.where(D.Affairs==0,0,D.Affairs*np.log(D.Affairs/fitval))-(D.Affairs-fitval)
# Statistique deviance
DS = 2*np.sum(rd)
# probabilité critique
pvalue = st.chi2.sf(DS,model.df_resid)
devtest = pd.DataFrame({"statistic":DS,"ddl":model.df_resid,"pvalue":pvalue},
                       index=["Deviance test"])
```

Table 7.4 – Statistique de Deviance

	statistic	ddl	pvalue
Deviance test	44.5875	16	2e-04

Notre modèle n'est pas bien ajusté.

7.2.3.3 Pseudo - R^2

Il consiste à tester le modèle nul, c'est - à - dire le modèle réduit à la constante (ou modèle trivial) et la modèle complet. L'hypothèse nulle correspond au modèle trivial. Sous cette hypothèse, la statistique de deviance s'écrit :

$$D_0 = 2\sum_{i=1}^{i=n} y_i \ln(y_i) - y_i\hat{\beta}_0 - \left(y_i - e^{\widehat{\beta}_0}\right) \tag{7.13}$$

avec $\hat{\beta}_0 = \ln(\overline{y})$.

La statistique du pesudo - R^2 s'écrit :

$$R^2 = \frac{D_0 - D}{D_0} = 1 - \frac{D}{D_0}$$

4. Avec $0 \times \ln 0 = 0$.

Comme dans le cas de la régression linéaire, le pseudo - R^2 est compris entre 0 et 1 : si 1, le modèle est parfait, si 0, ne fait pas mieux que le modèle trivial.

```
# Estimation de la constante
beta0 = np.log(np.mean(D.Affairs))
# Résidu dévance simplifié
rd0 = (np.where(D.Affairs==0,0,D.Affairs*np.log(D.Affairs)) - D.Affairs*beta0 -
                    (D.Affairs - np.exp(beta0)))
# Statistique deviance
D0 = 2*np.sum(rd0)
# Pseudo - R2
pR2 = 1 - (DS/D0)
print("Le pseudo - r2 vaut %.4f"%(pR2))
```

```
## Le pseudo - r2 vaut 0.5099
```

Pas terrible, vraiment.

7.2.4 Inférence statistique

7.2.4.1 Test du rapport de vraisemblance

Le test du rapport de vraisemblance revient à tester les hypothèses suivantes :

$$H_0 : \beta_1 = \cdots = \beta_p \quad \text{contre} \quad H_1 : \exists j \in \{1, \dots, p\}, \ \beta_j \neq 0$$

Il consiste à comparer 2 déviances. Pour l'évaluation globale, il s'agit de confronter celles du modèle étudié et du modèle trivial. La statistique du test s'écrit :

$$LR = D_{H_0} - D_M \tag{7.14}$$

Cette statistique suit une loi de χ^2 à p degrés de liberté. La région critique du test au risque α correspond aux grandes valeurs de la statistique de test, c'est-à-dire :

$$LR \geq \chi^2_{1-\alpha}(p) \tag{7.15}$$

Nous pouvons aussi décider via la `p-value`. Si elle est plus petite que α, le modèle est globalement significatif.

```
# Test du rapport de vraisemblance
lr_test = pd.DataFrame({"statistic":model.llr,"pvalue":model.llr_pvalue},
                       index=["likelihood test"])
```

Table 7.5 – Test du rapport de vraisemblance

	statistic	pvalue
likelihood test	46.3897	0

Notre modèle est globalement significatif à 5%.

7.2.4.2 Test de Wald

Puisque $\hat{\beta}$ est un estimateur du maximum de vraisemblance, il est asymptotiquement sans biais et suit une loi normale. Le test de Wald revient à tester les hypothèses suivantes :

$$H_0 : R\beta \quad \text{contre} \quad H_1 : R\beta \neq 0$$

où R est une matrice d'ordre $q \times p$ avec $q \leq p$. La statistique de Wald s'écrit alors

$$W_{(q)} = \left(R\hat{\beta}\right)' \widehat{\Sigma}_{\widehat{\beta}_{(q)}} \left(R\hat{\beta}\right) \tag{7.16}$$

Sous l'hypothèse nulle, cette statistique suit une loi de χ^2 à q degrés de liberté.

```
# Statistique de Wald
R = np.identity(len(model.params))
statW = model.wald_test(R)
wtest = pd.DataFrame({"statistic" : statW.statistic[0],"ddl" : statW.df_denom,
                      "pvalue" : statW.pvalue},index=["Wald test"])
```

Table 7.6 – Test de Wald

	statistic	ddl	pvalue
Wald test	194.4728	4	0

Notre modèle est globalement significatif à 5%.

Remarque 7.1 *Le test de Wald est utilisé notamment pour tester la significativité individuelle des coefficients. Dans ce cas, le test de Wald revient à tester les hypothèses :*

$$H_0 : \beta_j = 0 \quad \textit{contre} \quad H_1 : \beta_j \neq 0$$

La statistique de test s'écrit :

$$z_{\widehat{\beta}_j} = \frac{\hat{\beta}_j}{\hat{\sigma}_{\widehat{\beta}_j}}$$

Cette statistique suit approximativement une loi normale centrée réduite.

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	1.8404	0.7366	2.4984	0.0125	0.3966	3.2841
C(Gender)[T.1]	0.7508	0.2676	2.8059	0.0050	0.2264	1.2753
YearsMarried	0.0763	0.0342	2.2330	0.0256	0.0093	0.1433
RatingMarriage	-0.5951	0.1444	-4.1224	0.0000	-0.8781	-0.3122

A la lecture du table ci - dessus, tous les coefficients semblent significatifs à 5%.

7.2.4.3 Intervalle de confiance des coefficients

Puisque

$$\frac{\hat{\beta}_j - \beta_j}{\hat{\sigma}_{\hat{\beta}_j}}$$

suit approximativement une loi normale centrée réduite ; alors on peut en déuite des intervalles de confiance des coeffcients au niveau $1-\alpha$:

$$IC_{1-\alpha}\left(\beta_j\right) = \left[\hat{\beta}_j - z_{1-\frac{\alpha}{2}}, \hat{\beta}_j + z_{1-\frac{\alpha}{2}}\right] \tag{7.17}$$

Pour le niveau de confiance $1-\alpha = 90\%$, on utilise le quantile à $z_{1-\alpha/2} = z_{0.95}$.

```
# Intervalle de confiance
confint = model.conf_int(alpha=0.1)
confint.columns = ["5%","95%"]
```

Table 7.7 – Intervalle de confiance des coefficients

	5%	95%
Intercept	0.6287	3.0520
C(Gender)[T.1]	0.3107	1.1910
YearsMarried	0.0201	0.1326
RatingMarriage	-0.8326	-0.3577

7.2.5 Interprétation des coefficients

7.2.5.1 Ecart en logarithme - Variable binaire

Pour une variable explicative binaire X_j, le coefficient $\hat{\beta}_j$ représente l'écart des logarithmes des nombres espérés (λ) selon l'apparition $\left(X_j = 1\right)$ ou pas $\left(X_j = 0\right)$ de la caractéristique.

$$\begin{aligned}\hat{\beta}_j &= \ln\left[\hat{\lambda}_{(X_j=1)}\right] - \ln\left[\hat{\lambda}_{(X_j=0)}\right] \\ &= \ln\left(\hat{\lambda}_1\right) - \ln\left(\hat{\lambda}_0\right)\end{aligned}$$

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	1.8404	0.7366	2.4984	0.0125	0.3966	3.2841
C(Gender)[T.1]	0.7508	0.2676	2.8059	0.0050	0.2264	1.2753
YearsMarried	0.0763	0.0342	2.2330	0.0256	0.0093	0.1433
RatingMarriage	-0.5951	0.1444	-4.1224	0.0000	-0.8781	-0.3122

Considérons la variable `Gender` (Gender = 1, homme ; Gender=0, femme). A années de mariage et satisfaction égales, le logarithme du nombre moyen d'infidélité chez les hommes est supérieur de 0.75 à celui des femmes.

Puisque

$$\hat{\beta}_j = \ln\left(\hat{\lambda}_1\right) - \ln\left(\hat{\lambda}_0\right) = \ln\left(\frac{\hat{\lambda}_1}{\hat{\lambda}_0}\right) \quad \Longrightarrow \quad e^{\hat{\beta}_j} = e^{\hat{\lambda}_1/\hat{\lambda}_0} = e^{0.75084} = 2.12$$

A années de mariage et satisfaction égales, les hommes trompent leur conjoint 2.12 fois plus souvent que les femmes.

Pour une variable X_j indicatrice d'une variable V à K modalités, le coefficient $\hat{\beta}_j$ correspond à une différence en logarithme de cette modalité avec la modalité de référence (celle qui a été omise).

$$\hat{\beta}_j = \ln\left(\hat{\lambda}_{(V=X_j)}\right) - \ln\left(\hat{\lambda}_{(V=\text{ref})}\right) \tag{7.18}$$

7.2.5.2 Ecart en logarithme - Autre type de variable

Pour une variable explicative X_j quantitative, le coefficient $\hat{\beta}_j$ correspond à uen différence en logarithme cosécutive à l'augmentation d'une unité de X_j :

$$\hat{\beta}_j = \ln\left(\hat{\lambda}_{(X_j+1)}\right) - \ln\left(\hat{\lambda}_{(X_j)}\right) \tag{7.19}$$

En prenant l'exponentiel, on a l'odds - ratio.

7.2.5.3 Coefficients standardisés

Pour apprécier les contributions relatives des variables quantitatives dans un modèle, une stratégie simple consiste à centrer et (surtout) réduire les variables (qui sont sans unités par conséquent). On a alors les coefficients α_j quit sont interprétables sous la forme de différence en écart - type de X_j.

```
# Régression avec les variables initiales
model2 = smf.poisson("Affairs~YearsMarried+RatingMarriage",data=D).fit(disp=0)
coef_model2 = model2.summary2().tables[1]
```

Table 7.8 – Régression avec les variables initiales

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	2.5690	0.6836	3.7582	0.0002	1.2292	3.9088
YearsMarried	0.0570	0.0320	1.7788	0.0753	-0.0058	0.1198
RatingMarriage	-0.6207	0.1472	-4.2179	0.0000	-0.9091	-0.3323

```
# Centrage et réduction des données
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.set_output(transform='pandas');
Z = sc.fit_transform(D)

# Régression sur variables centrées et réduites (pas la cible)
Z["Affairs"] = D["Affairs"]
model3 = smf.poisson("Affairs~YearsMarried+RatingMarriage",data=Z).fit(disp=0)
coef_model3 = model3.summary2().tables[1]
```

Table 7.9 – Régression sur variables centrées et réduites (pas la cible)

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	0.8774	0.1595	5.4994	0.0000	0.5647	1.1901
YearsMarried	0.2919	0.1641	1.7788	0.0753	-0.0297	0.6135
RatingMarriage	-0.6042	0.1432	-4.2179	0.0000	-0.8850	-0.3234

Nous pouvons retrouver les coefficients $\hat{\alpha}_j$ en multipliant les $\hat{\beta}_j$ par les écarts - type des variables σ_j.

```
# Ecarts - type des variables concernées
sd_vars = np.array(list(map(lambda x : np.std(D[x],ddof=0),
                            ["YearsMarried","RatingMarriage"])))
sd_vars
```

```
## array([5.121077  , 0.97339612])
```

```
# Coefficients standardisés
std_coef = (model2.params[1:]*sd_vars).to_frame("Coef.")
```

Table 7.10 – Coefficients standardisés

	Coef.
YearsMarried	0.2919
RatingMarriage	-0.6042

La situation dans le mariage influe comparativement plus que les années de mariage sur le nombre des infidélités.

7.2.6 Sélection de variables

7.2.6.1 Intérêt de la sélection de variables

La sélection de variables - ne conserver que les variables explicatives pertinentes - est importante pour (1) mieux situer les phénomènes de causalité (interprétation du modèle) ; (2) assurer la robutesse du modèle (principe de parcimonie).

L'automiser devient indispensable dès lors que le nombre de variables candidates augmente, le nombre de variables candidates augmente, le nombre de combinaisons devient important. On adopte souvent des démarches pas à pas, ascendantes (*forward*), descendante (*backward*) ou bidirectionnelles.

Les approches basées sur des tests statistiques (test de Wald) constituent une réponse possible, mais avec le danger des faux positifs induits par les comparaisons multiples (en multipliant les tests, on augmente les chances d'intégrer à tort une variable, il faut restreindre le risque α).

Une alternative est de s'appuyer sur un critère qui effectue un arbitrage entre la qualité de l'ajustement (la log - vraisemblance) et la complexité du modèle (nombre de paramètres), les critères AIC (*Akaike Information Criterion*) ou BIC (*Bayesian Information Criterion*) sont souvent utilisés dans ce contexte.

1. Critère Akaike

$$AIC(\beta) = -2 \times \log\left(\mathcal{L}\left(\beta\right)\right) + 2 \times (p+1)$$

2. Critère Schwarz

$$BIC(\beta) = -2 \times \log\left(\mathcal{L}\left(\beta\right)\right) + \log(n) \times (p+1)$$

Le critère BIC est plus restrictif dès que $\ln(n) > 2$, c'est - à - dire conduit à sélectionner moins de variables.

```
# Régression avce l'ensemble des variables
formula = create_formula("Affairs",D.drop("Affairs",axis=1).columns)
model_all = smf.poisson(formula,data=D).fit(disp=0)
coef_model_all = model_all.summary2().tables[1]
```

Table 7.11 – Coefficients du modèle avec toutes les variables

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	-5.3437	2.7476	-1.9449	0.0518	-10.7289	0.0414
Gender	0.3279	0.6469	0.5068	0.6123	-0.9401	1.5958
Age	-0.0505	0.0323	-1.5647	0.1177	-0.1137	0.0128
YearsMarried	0.1939	0.0814	2.3833	0.0172	0.0344	0.3534
Children	0.6817	0.6643	1.0261	0.3048	-0.6203	1.9837
Religiousness	-0.0595	0.1343	-0.4429	0.6579	-0.3227	0.2037
Education	0.6622	0.1582	4.1860	0.0000	0.3521	0.9722
Occupation	-0.8002	0.2672	-2.9955	0.0027	-1.3239	-0.2766
RatingMarriage	-0.6880	0.1514	-4.5437	0.0000	-0.9847	-0.3912

7.3 Diagnostics des résidus

7.3.1 Résidus

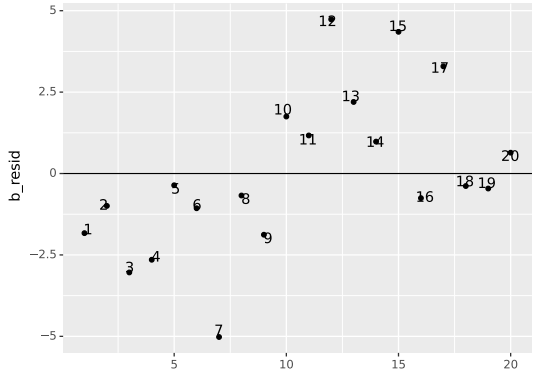
L'étude des résidus permet de diagnostiquer la régression, pour détecter les régularités (problème de spécification), ou identifier les points isolés (atypiques ou mal modélisés).

7.3.1.1 Résidus bruts

Les résidus bruts correspondent à l'écart entre les valeurs observées de la cible et la prédiction de $\mathbb{E}\left(Y\right) = \lambda$ du modèle.

$$r_i = y_i - \hat{\lambda}_i \quad \text{où} \quad \hat{\lambda}_i = \exp\left(X_i\hat{\beta}\right) \tag{7.20}$$

```
# Résidus bruts
bresid = pd.DataFrame(model.resid,columns=["b_resid"])
print((ggplot(bresid,aes(x=range(1,21),y="b_resid",label=range(1,21)))+
        geom_point()+geom_text(position=position_jitter(width=0.2,height=0.2))+
        geom_hline(yintercept=0)))
```

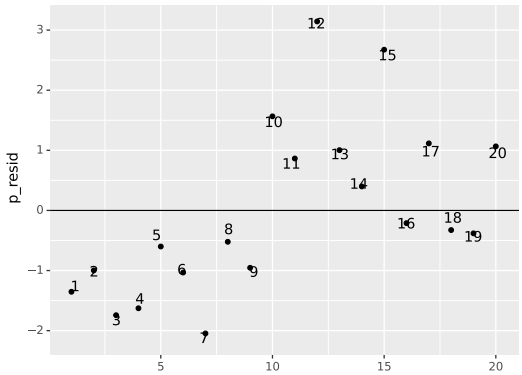


7.3.1.2 Résidus de Pearson

Les résidus de Pearson mesurent l'écart entre les valeurs observées de la cible et la prédiction, écart normalisé par l'écart type. En effet, $\mathbb{E}(Y) = \mathrm{V}(Y) = \lambda$ pour la loi de Poisson.

$$rp_i = \frac{r_i}{\sqrt{\widehat{\lambda}_i}} \tag{7.21}$$

```
# Résidus de Pearson
presid = pd.DataFrame(model.resid_pearson,columns=["p_resid"])
print((ggplot(presid,aes(x=range(1,21),y="p_resid",label=range(1,21)))+
        geom_point()+geom_text(position=position_jitter(width=0.2,height=0.2))+
        geom_hline(yintercept=0)))
```

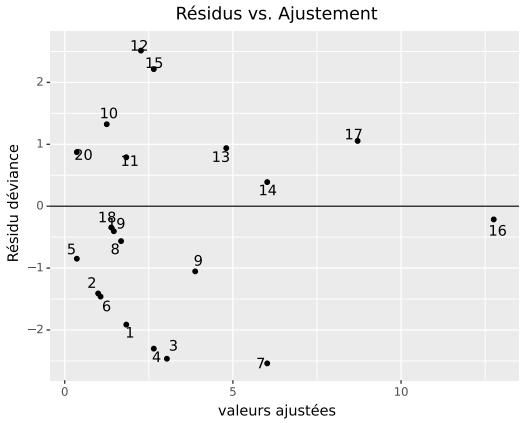


7.3.1.3 Résidus déviance

Le résidu de déviance est une composante (pour l'individu i) de la statistique de déviance utilisée pour évaluer la modélisation. Il est définit comme suit :

$$rd_i = \mathrm{sgn}(r_i) \times \sqrt{2 \times \left[y_i \log\left(\frac{y_i}{\widehat{\lambda}_i}\right) - r_i \right]} \tag{7.22}$$

```
# Résidus déviance
rd = np.sign(model.resid)*np.sqrt(2*(np.where(D.Affairs==0,0,
                                D.Affairs*np.log(D.Affairs/fitval))-model.resid))
data = pd.DataFrame(np.c_[fitval,rd],columns=["adj_val","dev_resid"])
print((ggplot(data,aes(x="adj_val",y="dev_resid",label=range(1,21)))+
         geom_point()+geom_text(position=position_jitter(width=0.2,height=0.2))+
         geom_hline(yintercept=0)+labs(x="valeurs ajustées",y="Résidu déviance",
         title="Résidus vs. Ajustement")))
```



Comme la statistique déviance :

$$D = \sum_{i=1}^{i=n} rd_i^2$$

On peut identifier les observations **à contre - courant** dans la modélisation, qui pèsent négativement dans l'ajustement (rd_i^2 élevés par rapport aux autres). Par exemple, le l'individu numéro 7, une femme avec beaucoup d'années de mariage (`years = 15`), pas très heureuse (`rating = 2`), et pourtant relativement fidèle (`affairs = 1`). Mais qu'est - ce qu'elle attend ? $\hat{\lambda}_7 = 6.02$. De même, l'individu numéro 15, un homme avec quelques années de mariage (`years=10`), assez heureux (`rating=7`), et pourtant chaud lapin (`affairs=7`). On s'y attendait, mais pas à ce point - là : $\hat{\lambda}_{15} = 2.64$.

7.3.2 Levier

Le levier d'un individu, h_{ii}, indique son éloignement dans l'espace de représentation, mais aussi son influence globale dans la régression. Elle est lue dans la diagonale de la matrice H :

$$H = W^{1/2} X' \left(X' W X\right)^{-1} X W^{1/2}$$

On peut aussi la calculer directement pour l'individu i :

$$h_{ii} = \hat{\lambda}_i X_i \left(X' W X\right)^{-1} X_i' \tag{7.23}$$

où $X_i = \left(1, X_{i,1}, \ldots, X_{i,p}\right)$.

Puisque $\sum_{i=1}^{i=n} h_i = p + 1$, on prend empiriquement comme seuil de suspicion $2 \times \frac{p+1}{n}$ (2 fois la moyenne).

```
# Points leviers
infl = model.get_influence()
hi = infl.hat_matrix_diag
data = pd.DataFrame({"index" : range(1,len(hi)+1),"h":hi})
print((ggplot(data,aes(x="index",y="h"))+
               geom_bar(stat="identity",color="black",fill="gray")+
               geom_hline(yintercept=2*(3+1)/D.shape[0],color="blue")+
               labs(x="index",y="Levier")))
```

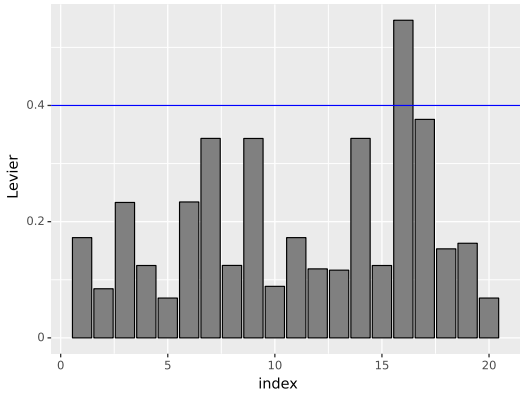


Figure 7.2 – Points leviers

L'individu numéro 16 est un homme qui endure 15 années de mariage malheureux (`rating`=2). **Très différent des autres individus de la base**. Pourtant il est bien modélisé (résidu faible), parce qu'il décharge à tire larigot (`affairs`=12).

7.3.2.1 Résidus de Studentisés

Il existe des versions studentisés des résidus de Person et de déviance. Ainsi, les résidus de Pearson studentisés s'écrivent :

$$srp_i = \frac{rp_i}{\sqrt{1 - h_{ii}}} \tag{7.24}$$

et les résidus de déviance studentisés s'écrivent :

$$srd_i = \frac{rd_i}{\sqrt{1 - h_{ii}}} \tag{7.25}$$

7.3.3 Surdispersion

Dans un modèle Poissonien, le paramètre $\lambda(x)$ est à la fois l'espérance et la variance conditionnelle. Il arrive souvent que les données présentent une sur-dispersion (la variance de la variable cible est largement supérieure à sa moyenne), invalidant l'hypothèse de la loi de Poisson.

La non-indépendance des observations individuelles peut causer une sur-dispersion des données par rapport aux suppositions de la distribution de Poisson. Cette sur-dispersion est représentée par un paramètre ϕ qui multiplie la variance attendue : pour une moyenne λ, la variance devient donc $\phi\lambda$.

De façon moins fréquente, il arrive que $\phi < 1$, correspondant à une **sous-dispersion** des observations. Contrairement à la sur-dispersion, où les observations tendent à être regroupées (ex. : quelques quadrats avec de nombreux individus, et plusieurs avec peu ou pas d'individus), la sous-dispersion signifie que les observations sont réparties de façon plus régulière que prévue.

7.3.3.1 Estimation du paramètre de dispersion

Pour estimer le paramètre de dispersion ϕ, nous utiliserons l'estimateur :

$$\hat{\phi} = \frac{\chi^2}{n - p - 1} \tag{7.26}$$

```
# Ratio = Statistique/ degré de liberté
ratio_surdispersion = khi2/model.df_resid
print("Ratio de surdispersion : %.4f"%(ratio_surdispersion))
```

```
## Ratio de surdispersion : 2.4612
```

L'estimé du paramètre de dispersion est égal à 2.4612. Lorsque $\hat{\phi}$ n'est pas trop élevé (typiquement, on suggère $\hat{\phi} < 4$), les estimés des coefficients de la régression de Poisson demeurent valides, mais il faut multiplier leurs erreurs-types par $\sqrt{\hat{\phi}}$. Autrement dit, la sur-dispersion n'introduit pas de biais, mais augmente l'incertitude sur les valeurs des coefficients. Si la sur-dispersion est très grande, il est préférable d'utiliser un autre modèle.

7.3.3.2 Correction de la surdispersion

Plusieurs solutions sont envisageables pour corriger la surdispersion : la première solution consiste à corriger l'estimation de l'écart - type des coeffcients (les coefficients ne sont pas modifiés) avec une procédure très simple issu du ratio calculé :

$$\tilde{\sigma}_{\widehat{\beta}_j} = \hat{\sigma}_{\widehat{\beta}_j} \times \sqrt{\hat{\phi}} \tag{7.27}$$

```
# Tableau des coefficients
coef_table = model.params.to_frame("Coef.")
# Correction à introduire
se_correction = np.sqrt(ratio_surdispersion)
# Modifier les écarts - types
coef_table.insert(1,"Std.Err.",model.bse*se_correction)
# Les autres colonnes
coef_table.insert(2,"z",coef_table.iloc[:,0]/coef_table.iloc[:,1]) # z - value
coef_table.insert(3,"P>|z|",2*st.norm.sf(np.abs(coef_table.iloc[:,2]))) # p - value
```

Table 7.12 – Tableau des coefficients après correction

	Coef.	Std.Err.	z	P>|z|
Intercept	1.8404	1.1556	1.5925	0.1113
C(Gender)[T.1]	0.7508	0.4198	1.7885	0.0737
YearsMarried	0.0763	0.0536	1.4233	0.1546
RatingMarriage	-0.5951	0.2265	-2.6277	0.0086

Una autre technique de correction est d'utiliser la famille `quasipoisson` qui effectue automatiquement l'estimation du paramètre de dispersion. Dans le modèle quasi - poisson, on modélise :

$$\mathbb{E}(Y|X=x) = \lambda(x) \quad \text{et} \quad \mathrm{V}(Y|X=x) = \theta \times \lambda(x)$$

où θ est un paramètre supplémentaire, facteur de surdispersion, estimé à partir des données.

```
# Modèle de famille quasipoisson
import statsmodels.api as sm
model_quasi = smf.glm('Affairs~C(Gender)+YearsMarried+RatingMarriage',data=D,
                      family=sm.families.Tweedie()).fit()
coef_quasi = model.summary2().tables[1]
```

Table 7.13 – Coefficients du modèle de quasi - poisson

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	1.8404	0.7366	2.4984	0.0125	0.3966	3.2841
C(Gender)[T.1]	0.7508	0.2676	2.8059	0.0050	0.2264	1.2753
YearsMarried	0.0763	0.0342	2.2330	0.0256	0.0093	0.1433
RatingMarriage	-0.5951	0.1444	-4.1224	0.0000	-0.8781	-0.3122

La valeur du paramètre de surdispersion est 2.4612.

Bibliographie

Agresti, Alan. 2012. *Categorical Data Analysis*. Vol. 792. John Wiley & Sons.

Clogg, Clifford C, and Edward S Shihadeh. 1994. “Statistical Models for Ordinal Variables.” *(No Title)*.

Cornillon, Pierre-André, Nicolas Hengartner, Eric Matzner-Løber, and Laurent Rouvière. 2019. *Régression Avec r-2e édition*. EDP sciences.

———. 2023. “Régression Avec r : 3ème édition.” In *Régression Avec r*. EDP Sciences.

Fu, Vincent Kang. 1999. “Estimating Generalized Ordered Logit Models.” *Stata Technical Bulletin* 8 (44).

Hastie, Trevor, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. 2009. *The Elements of Statistical Learning : Data Mining, Inference, and Prediction*. Vol. 2. Springer.

Hilbe, Joseph M. 2016. *Practical Guide to Logistic Regression*. crc Press.

Lemeshow, Stanley, Rodney X Sturdivant, and David W Hosmer Jr. 2013. *Applied Logistic Regression*. John Wiley & Sons.

Long, J Scott, and Jeremy Freese. 2006. *Regression Models for Categorical Dependent Variables Using Stata*. Vol. 7. Stata press.

O’Connell, Ann A. 2006. *Logistic Regression Models for Ordinal Response Variables*. Vol. 146. sage.

Rakotomalala, Ricco. 2011. “Pratique de La régression Logistique.” *Régression Logistique Binaire Et Polytomique, Université Lumière Lyon* 2 : 258.

Rouvière, L. 2017. “Régression Logistique Avec r, Université Rennes 2, En Ligne.”

Rouvière, Laurent. 2008. “Régression Sur Variables Catégorielles.” *Université de Rennes* 2.

———. 2015. “Régression Logistique Avec r.” *Universités Rennes* 2.

Saporta, Gilbert. 2006. *Probabilités, Analyse Des Données Et Statistique*. Editions technip.

Scherrer, B. 2007. “Biostatistique, Vol. 1.” *Gaëtan Morin éditeur (816 Pp.)*.

Tallarida, Ronald J, and Rodney B Murray. 1987. “Mann-Whitney Test.” In *Manual of Pharmacologic Calculations*, 149–53. Springer.

Tenenhaus, Michel. 2007. *Statistique : Méthodes Pour décrire, Expliquer Et Prévoir*. Vol. 680. Dunod Paris, France :

Williams, Richard. 2006. “Generalized Ordered Logit/Partial Proportional Odds Models for Ordinal Dependent Variables.” *The Stata Journal* 6 (1) : 58–82.

Econométrie des variables catégorielles :

Théorie et application sous Python

Cet ouvrage présente l'économétrie des variables catégorielles plus connue sous le nom d'économétrie des variables qualitatives. Il met l'accent sur les méthodes de régression logistique et plus généralement les méthodes linéaires généralisées. Réputés complexes, les modèles à variables catégorielles sont en réalité de plus en plus utilisés parmi les outils d'inférence statistique et leurs applications se révèlent être très diverses.

Ce manuel s'adresse aux étudiants de Licence et Master de mathématiques appliquées, d'économie et d'économétrie, ainsi qu'aux élèves des écoles d'ingénieurs. Il s'adresse également aux professionnels, praticiens de l'économétrie des variables qualitatives.

Duvérier DJIFACK ZEBAZE est ingénieur en Data Science et gestion des risques de l'école nationale de la statistique et de l'analyse de l'information (ENSAI) de Rennes en France et Ingénieur Statisticien Economiste (ISE) de l'école nationale de la statistique et de l'analyse économique (ENSAE) de Dakar au Sénégal. Par ailleurs, il est titulaire d'un master recherche en économie quantitative obtenu à la faculté des sciences économiques et de gestion appliquée de l'Université de Douala au Cameroun. Sa carrière débute en tant qu'ingénieur quantitatif au sein de l'équipe de modélisation de la Business Unit GLBA (Global Banking and Advisory) du groupe Société Générale.