Hybrid machine learning forecasts UEFA EURO 2020

Moudila Marcel

▶ **To cite this version:**

Moudila Marcel. Hybrid machine learning forecasts UEFA EURO 2020. Mathématiques [math]. 2022. hal-03782077

HAL Id: hal-03782077
https://hal.univ-lorraine.fr/hal-03782077

Submitted on 21 Sep 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Institut ÉLIE CARTAN

Travaux encadrés de recherche

Master 1 Mathématiques

2021-2022

Hybrid Machine Learning Forecasts for UEFA EURO 2020

Réalisé par :
Marcel MOUDILA

Référent :
Efoevi KOUDOU

1er juin 2022

Avant-propos

Le machine learning est utilisé , entre autres, dans le milieu sportif, par exemple dans le football pour prédire les scores des différentes rencontres. Le modèle de régression de Poisson est beaucoup utilisé pour ce genre de prédiction car le score ou la variable cible suit une distribution probabiliste : la loi de Poisson.

Il est courant de combiner des techniques distinctes pour faire des prédictions si cette combinaison fournit le meilleur modèle car on est relativement sûr que notre modèle est le bon qu'en fonction de ce qu'il nous renvoie lorsqu'il est entraîné, testé sur les données qui lui sont soumis. [GHL+21] mentionne que les arbres de forêts décisionnels, et l'algorithme de gradient boosting " Xgboost" sont des techniques qui ont été utilisé à côté du modèle de régression de Poisson sur les données des rencontres des coupes d'EURO 2004 à 2016 (une coupe d'EURO a lieu tous les 4 ans) afin de prédire les scores des rencontres de l'EURO 2020.

D'après le site wikipedia.fr (en Avril 2022) , le machine learning (ou apprentissage automatique en Anglais) est *un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d'« apprendre » à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune. Plus largement, il concerne la conception, l'analyse, l'optimisation, le développement et l'implémentation de telles méthodes.*

Dans ce mémoire, nous parlerons de deux méthodes : régression de Poisson, et forêts aléatoires.

Nous donnerons les notions théoriques de ces méthodes et nous utiliserons le language Python pour implémenter ces méthodes. Éventuellement, nous comparerons les résultats obtenus par nos propres programmes écrits sur Python avec les résultats des bibliothèques classiques de la régression de poisson ,et des forêts aléatoires :

sklearn.linear_model.PoissonRegressor,
sklearn.ensemble.RandomForestClassifier,
sklearn.ensemble.RandomForestRegressor,
Ces trois bibliothèques sont intégrés dans le module **sklearn (scikit-learn)**.Ce dernier s'est imposé comme étant le module de référence sur Python concernant le machine learning.

Nous nous donnons le plan suivant :
- 1ère partie : Apprentissage supervisé : cadre théorique ;
- 2ème partie : Apprentissage supervisé : programmation sur Python ;
- 3ème partie (en fonction du temps) : Apprentissage supervisé : Étude sur données réelles.

Dans la 1ère partie, il sera l'objet de l'étude de la régression de poisson, et des forêts aléatoires. Dans la seconde partie, les programmations de ces méthodes. Enfin, dans la dernière partie, une application sur un jeux des données réelles classique. Les jeux des données sur les matchs de l'EURO 2004 à 2016 seront eux l'objet de la suite de ce mémoire, en master 2 dans le cadre d'un projet long par exemple.

Table des matières

1 Régression de Poisson **3**
1.1 notions de modèles linéaires généralisés . . . 3
1.2 cas de la distribution de Poisson . . . 3
1.3 Estimation des paramètres . . . 4
1.4 Intervalles de confiance des paramètres . . . 4
1.5 Test des coefficients du modèle . . . 4
1.5.1 Test de Wald . . . 4
1.6 Pertinence du Modèle . . . 5
1.6.1 Résidus de Pearson . . . 5
1.6.2 Résidus de la déviance . . . 5
1.7 Détection des valeurs aberrantes . . . 5
1.7.1 Méthode des résidus normalisés de Pearson . . . 5

2 Forêts aléatoires **6**
2.1 notions d'arbres de décision . . . 6
2.2 les forêts aléatoires . . . 7
2.2.1 les forêts d'arbres de régression . . . 7
2.2.2 les forêts d'arbres de classification . . . 7

3 Création des algorithmes d'apprentissage pour l'exploitation des données **9**
3.1 Notions de dictionnaire sur Python . . . 9
3.2 Création de moudilaRegpoisson . . . 10

Bibliographie **18**

Chapitre 1

Régression de Poisson

1.1 notions de modèles linéaires généralisés

Le modèle linéaire généralisé est dû à John Nelder et R. W. M. Wedderburn en 1972. Ce modèle forme une famille des méthodes statistiques pour la prédiction. Il s'agit de prédire une variable Y appelée **variable réponse**, ou variable cible, à partir de p variables déterministes $X_1, \ldots, X_p$ appelées **variables explicatives** ou **covariables**. Le modèle linéaire généralisé s'identifie par l'équation :

$$g(\mathbb{E}(Y \mid X = x) = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p$$

où g est une fonction nommée **fonction de lien**. g est monotone et différentiable. $\beta = (\beta_0, \ldots, \beta_p)$ est le paramètre du modèle. L'estimation de β se fait par la méthode de maximum de vraisemblance. la loi de distribution de Y fait toujours partie de la famille exponentielle des lois.

loi de Y	**méthode**	**fonction de lien (exemples)**
Gaussienne	régression linéaire	fonction identité
Bernoulli	régression logistique	fonction logit
Poisson	régression de Poisson	fonction logarithme

Étudions plus en détail la régression de Poisson car cette méthode est citée dans [GHL+21] qui est l'article de recherche sur laquelle nous travaillons.

1.2 cas de la distribution de Poisson

Définition 1.2.1. *La loi de Poisson a été introduite en 1838 par Denis Poisson (1781–1840). C'est une loi de probabilité discrète qui décrit le nombre d'événements se produisant dans un intervalle de temps fixé. On dit qu'une variable Y suit une loi de Poisson de paramètre $\lambda > 0$ si elle prend des valeurs entières, avec la probabilité*

$$\mathbb{P}(Y = y) = \frac{\lambda^y e^{-\lambda}}{y!}$$

Propriété 1.2.2. *C'est la seule distribution de probabilité classique dont l'espérance et la variance sont égales*

$$\mathbb{E}(Y) = Var(Y) = \lambda$$

Définition 1.2.3. *L'équation du modèle de régression linéaire de poisson entre une variable réponse Y, qui suit la loi de Poisson de paramètre $\lambda > 0$, et p variables explicatives déterministes $X_1, \ldots, X_p$ est :*

$$\log(\mathbb{E}(Y|X = x)) = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p = \log(\lambda)$$

1.3 Estimation des paramètres

On souhaite maintenant estimer $\beta = (\beta_0, \ldots, \beta_p)$ le vecteur des paramètres du modèle de régression de poisson.

Définition 1.3.1. *Soit $(y_i, x_i)_{1 \leq i \leq n}$ la réalisation d'un échantillon de (Y, X) où $x_i = (x_{i1}, \ldots, x_{ip})$ et p le nombre de variables explicatives. On appelle la vraisemblance du modèle de régression de poisson, la fonction*

$$\mathcal{L}(\beta, (y_i, x_i)) = \prod_{i=1}^{n} \frac{\lambda(x_i)^{y_i}}{y_i!} \exp(-\lambda(x_i))$$

En prenant le logarithme de la vraisemblance, on simplifie l'écriture, mais seul un algorithme nous permet de maximiser $\log(\mathcal{L})$ et ainsi obtenir $\hat{\beta}$ l'estimateur du maximum de vraisemblance. Au chapitre 3, nous avons fait usage des estimateurs donnés par scikit-learn, qui lui-même fait usage de la librairie scipy.optimize pour résoudre ce problème d'optimisation.

Définition 1.3.2. *On définit l'estimation de la variance de $\hat{\beta}$ par :*

$$\hat{Var}(\hat{\beta}) = (X^t W X)^{-1} = \begin{pmatrix} \hat{Var}(\hat{\beta}_0) & \hat{Cov}(\hat{\beta}_0, \hat{\beta}_1) & \cdots & \hat{Cov}(\hat{\beta}_0, \hat{\beta}_p) \\ \hat{Cov}(\hat{\beta}_1, \hat{\beta}_0) & \hat{Var}(\hat{\beta}_1) & \cdots & \hat{Cov}(\hat{\beta}_1, \hat{\beta}_p) \\ \vdots & \vdots & \ddots & \vdots \\ \hat{Cov}(\hat{\beta}_p, \hat{\beta}_0) & \hat{Cov}(\hat{\beta}_p, \hat{\beta}_1) & \cdots & \hat{Var}(\hat{\beta}_p) \end{pmatrix}$$

où $W = diag(\hat{\lambda}(x_1), \ldots, \hat{\lambda}(x_n)$ *et* $X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}$

Définition 1.3.3. *on définit l'estimation de l'écart-type de $\hat{\beta}_j$,$0 \leq j \leq p$ par :*

$$\hat{\sigma}(\hat{\beta}_j) = \sqrt{\hat{Var}(\hat{\beta}_j)}$$

1.4 Intervalles de confiance des paramètres

Soit $j = 0, \ldots, p$, un intervalle de confiance pour le paramètre β_j au niveau $100(1-\alpha)\%$,$0 < \alpha < 1$ est :

$$\left[\hat{\beta}_j \pm qnorm(1 - \frac{\alpha}{2})\hat{\sigma}(\hat{\beta}_j)\right]$$

1.5 Test des coefficients du modèle

L'objectif est d'évaluer l'influence de la variable explicative X_j, $1 \leq j \leq p$, sur la variable cible Y.

1.5.1 Test de Wald

Hypothèse nulle	$H_o : \beta_j = 0$
Hypothèse alternative	$H_1 : \beta_j \neq 0$
statistique de test $\mathcal{Z}_{obs}$	$\frac{\hat{\beta}_j}{\hat{\sigma}(\hat{\beta}_j)}$
p-value	$\mathbb{P}(\lvert\mathcal{N}(0,1)\rvert \geq \lvert\mathcal{Z}_{obs}\rvert)$

si la p-value < 0.05 , la variable explicative X_j influence significativement la variable réponse Y

1.6 Pertinence du Modèle

1.6.1 Résidus de Pearson

Hypothèse nulle	$H_0 : \lambda(x) = \exp(\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p)$
Hypothèse alternative	$H_1 : \lambda(x) \neq \exp(\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p)$
résidu de Pearson	$\hat{\epsilon_i} = \frac{Y_i - \exp(\hat{\beta_0} + \hat{\beta_1} x_{1i} + \ldots + \hat{\beta_p} x_{pi})}{\sqrt{\exp(\hat{\beta_0} + \hat{\beta_1} x_{1i} + \ldots + \hat{\beta_p} x_{pi})}}$
χ^2_{obs}	$\sum_{i=1}^{n} \hat{\epsilon_i}^2$
p-value :	$\mathbb{P}(\text{chi-deux}(n-(p+1)) \geq \chi^2_{obs})$

Si la p-value > 0.05, le modèle est pertinent par rapport aux données.

1.6.2 Résidus de la déviance

il y'a beaucoup des similarités avec les résidus de Pearson, ici on parle des déviances résiduelles. Pour alléger les notations, on utilise

$$\hat{\lambda}(x_i) = \exp(\hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \ldots + \hat{\beta}_p x_{pi})$$

Hypothèse nulle	$H_0 : \lambda(x) = \exp(\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p)$
Hypothèse alternative	$H_1 : \lambda(x) \neq \exp(\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p)$
déviance résiduelle	$\hat{D}_i = \pm\sqrt{2\left(Y_i \log\left(\frac{Y_i}{\hat{\lambda}(x_i)}\right) - (Y_i - \hat{\lambda}(x_i))\right)}$
χ^2_{obs}	$\sum_{i=1}^{n} \hat{D}_i^2$
p-value :	$\mathbb{P}(\text{chi-deux}(n-(p+1)) \geq \chi^2_{obs})$

Si la p-value > 0.05, le modèle est pertinent par rapport aux données.

1.7 Détection des valeurs aberrantes

Étant donné $(y_i, x_i)_{1 \leq i \leq n}$ la réalisation d'un échantillon de (Y, X) où $x_i = (x_{i1}, \ldots, x_{ip})$, la détection des valeurs aberrantes dans les données est cruciale car ces valeurs peuvent avoir une influence négative dans les estimations des paramètres de notre modèle de régression. On dispose de deux méthodes pour détecter les valeurs aberrantes : la méthode des résidus normalisés de Pearson, et le critère des distances de Cook.

1.7.1 Méthode des résidus normalisés de Pearson

Pour tout $i = 1, \ldots, n$, on rappelle que le résidu de Pearson de l'observation i est estimé par

$$\hat{\epsilon_i} = \frac{Y_i - \exp(\hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \ldots + \hat{\beta}_p x_{pi})}{\sqrt{\exp(\hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \ldots + \hat{\beta}_p x_{pi})}} = \frac{Y_i - \hat{\lambda}(x_i)}{\sqrt{\hat{\lambda}(x_i)}}$$

On note $W = \text{diag}(\hat{\lambda}(x_1), \ldots, \hat{\lambda}(x_n)$

Définition 1.7.1. *On appelle le résidu standardisé de Pearson de l'observation i la réalisation e_i^* de*

$$\hat{\epsilon_i^*} = \frac{\hat{\epsilon_i}}{\sqrt{1 - [W^{\frac{1}{2}} X (X^t W X)^{-1} X^t W^{\frac{1}{2}}]_{i,i}}}$$

Si $|e_i^*| > 2$, alors on envisage que l'observation i a des valeurs aberrantes.

Le critère des distances de Cook utilise les résidus standardisés de Paerson, d'où on laisse de côté cette deuxième méthode.

Chapitre 2

Forêts aléatoires

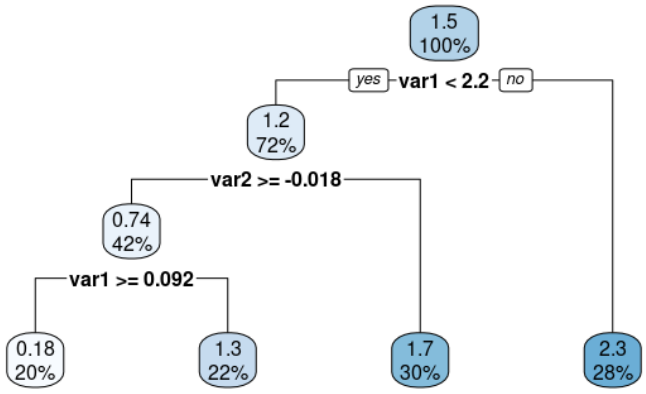
2.1 notions d'arbres de décision

Un arbre de décision est un algorithme d'apprentissage supervisé non paramétrique qui consiste à partir de p variables explicatives $X_1, \ldots, X_p$ de faire de la prédiction d'une variable réponse Y quantitative ou la classification si Y est qualitative.

Exemple d'algorithme de construction d'arbre de décision :
- CART, ID3, C4.5

Les bibliothèques classiques de machine learning sous Python ou R implémentent au moins un de ces algorithmes.

```
######### Création d'un data frame et d'un arbre de décision type régression
x1 <- runif(50,-2,5)
x2 <- runif(50,-2,5)
y <- runif(50,-2,5)
mydata <- data.frame(var1 = x1,var2 = x2, cible = y)
attach(mydata)
library(rpart)
arbre = rpart(cible ~ var1 + var2)
library(rpart.plot)
rpart.plot(arbre)
```



Cet arbre de décision comprend 2 nœuds internes, 6 branches, et 4 feuilles. On a le vocabulaire suivant
- racine (ou nœud principal) : c'est le sommet de l'arbre.
- nœuds internes : des noeuds qui ont des descendants.
- feuilles (ou nœuds terminaux) : noeuds qui n'ont pas des descendants.
- branches : ce sont les traits qui partent d'un nœud principal ou interne. De chaque nœud partent, 0 ou 2 branches.

Sur Python , la bibliothèque **sklearn.tree.DecisionTreeClassifier** pourraît être utilisé pour obtenir un arbre de classification, et **sklearn.tree.DecisionTreeRegressor** pour obtenir un arbre de régression.

2.2 les forêts aléatoires

une forêt aléatoire encore appelé forêt d'arbres décisionnels est une technique d'apprentissage supervisé proposé par Leo Breimann en 2001. Sa popularité et son efficacité justifient son usage parmi les méthodes utilisées dans [GHL+21]. On distingue les forêts d'arbres de régression et les forêts d'arbres de classification.

2.2.1 les forêts d'arbres de régression

Soient n le nombre des observations et p le nombre de covariables. Soient N , m et q des entiers tels que $1 \leq m \leq n$ et $1 \leq q \leq p$. L'entier N désigne le nombre d'arbres de régression dans la forêt. pour k allant de 1 à N, le k-ème arbre de décision est construit de la façon suivante :

- On sélectionne au hasard et avec remise m couples parmi $(y_1, x_1), \ldots, (y_n, x_n)$ où $x_i = (x_{i1}, \ldots, x_{ip})$. L'arbre sera alors construit exclusivement avec les couples sélectionnés.
- Pour chaque noeud de l'arbre, on sélectionne sans remise q variables parmi $X_1, \ldots, X_p$ avec q généralement petit pour que l'arbre ne soit pas grand. Dès lors, une variable de coupure sera déterminée parmi les variables sélectionnées, et une valeur seuil sera calculée en fonction.

On obtient alors par cet algorithme récursif les N arbres partiellement indépendants. Ainsi se construit une forêt aléatoire. Pour donner un ordre de grandeur, des choix raisonnables sont : $N = 500$, $m \simeq \frac{2n}{3}$, et $q \simeq \sqrt{p}$

Utilisation : Étant donné une forêt d'arbres de régression, une valeur prédite de y_{n+1} pour un individu dont on connaît les réalisations de $X_1, \ldots, X_p$ est obtenue en faisant la moyenne des valeurs prédites de y_{n+1} par chaque arbre de la forêt.

Remarque : On ne peut pas visualiser sur ordinateur une forêt aléatoire comme on le ferait pour un arbre unique.

2.2.2 les forêts d'arbres de classification

Dans ce contexte, la variable cible Y est une variable qualitative. Une forêt d'arbres de classification est construit par le même algorithme récursif que pour les forêts d'arbres de régression.

Utilisation : Étant donné une forêt d'arbres de classification, une modalité plausible de y_{n+1} est donnée par un vote majoritaire parmi les modalités plausibles de y_{n+1} obtenues par chaque arbre de la forêt.

Exemple :

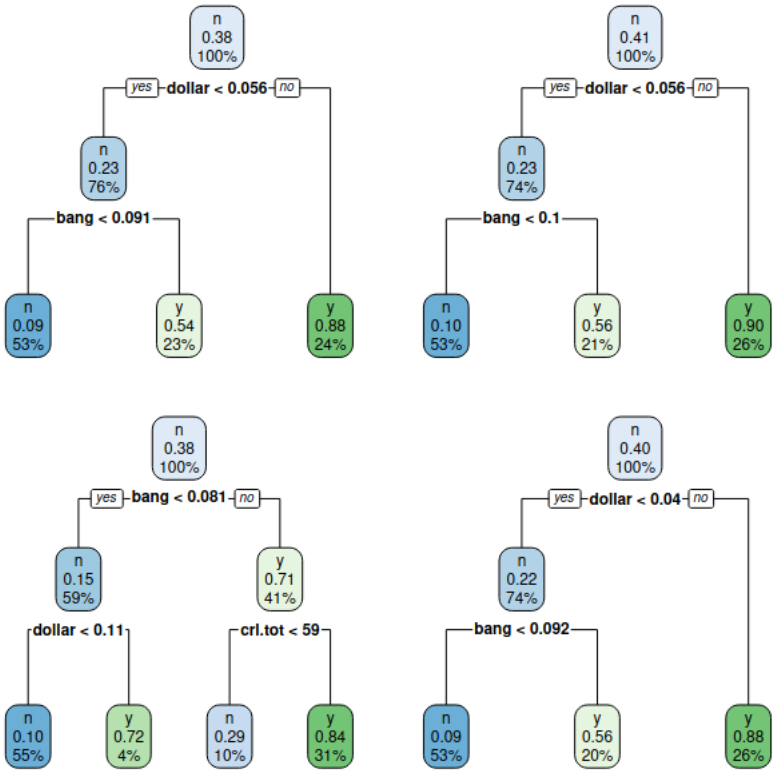
spam7 ×

Filter

	crl.tot	dollar	bang	money	n000	make	yesno
1	278	0.000	0.778	0.00	0.00	0.00	y
2	1028	0.180	0.372	0.43	0.43	0.21	y
3	2259	0.184	0.276	0.06	1.16	0.06	y
4	191	0.000	0.137	0.00	0.00	0.00	y
5	191	0.000	0.135	0.00	0.00	0.00	y
6	54	0.000	0.000	0.00	0.00	0.00	y

Showing 1 to 6 of 4,601 entries, 7 total columns

On considère le jeu de données `spam7`, avec lequel on connait la nature spam ou non d'un email en fonction de certaines de ses caractéristiques, notamment des caractères `bang`, `dollar` et `crl.tot`. Le caractère que l'on veut expliquer est `yesno`, lequel est qualitatif de modalités y et n. Pour ce faire, on utilise une forêt aléatoire d'arbres de classification avec 4 arbres (c'est un cas d'école, car en pratique, on la construit plutôt avec 500 arbres ...). On obtient les 4 arbres de classification suivants :



À partir de cette forêt aléatoire, donnons une modalité plausible de yesno pour un individu vérifiant bang = 0.096, dollar = 0.045 et crl.tot = 55.

Réponse : Il n'y a pas de modalités plausibles de yesno en supériorité numérique, on est dans un cas très particulier : l'arbre en haut à gauche prédit *yes*, en haut à droite prédit *no*, en bas à gauche prédit *no* et le dernier prédit *yes*.

Chapitre 3

Création des algorithmes d'apprentissage pour l'exploitation des données

Nous avons développé des algorithmes pour pratiquer la régression de poisson en se basant sur l'exposé théorique du chapitre 1.

3.1 Notions de dictionnaire sur Python

Un dictionnaire en Python est une structure qui est composée d'un ensemble de clé. A chaque clé, on lui associe une valeur. L'exemple ci-dessous est le dictionnaire que nous avons crée pour la régression de poisson .

Exemple : création d'un dictionnaire appelé `moudilaRegpoisson`

```
def moudilaRegpoisson(y,x,alpha):

    beta = param(y,x)                                  # coeff du modele
    X = matrice_X(x)                                   # matrice X
    ld = lambda_x(beta,X)                              # lambda
    V = var_b(ld,X)                                    # variance de beta
    sd = ecarttype(V)                                  # ecart-type de beta_j
    IC = ICbeta(beta,alpha,sd)                         # intervalle de confiance de beta_j
    n = len(ld)
    tab = testWald(beta,sd,n)                          # test de nullité de beta_j
    resp = residu_pearson(y,ld,beta)                   # residu de pearson
    resd = residu_deviance(y,ld,beta)                  # residu de deviance
    tab2 = pertinence(y,resd,resp,beta)                # test sur la pertinence du modèle
    res = residu_pearson_standardise(resp,ld,X,V)      # résidus standardisés de pearson
    tab3 = aberrantes(res)                             # valeurs anormales

    dictionnaire = {"coeff":beta,"matrice": X,"lambda" :ld,"variance" :V,"ecarttype":sd,
                    "intervalle.confiance":IC,"test.wald":tab,"residu.pearson":resp,
                    "residu.deviance":resd,"pertinence":tab2,
                    "residu.pearson.standard":res,"anormale":tab3}

    return dictionnaire
```

Nous l'avons définit comme une fonction python, mais la sortie est un dictionnaire. Il recense l'implémentation que nous avons fait concernant le modèle de régression de poisson.

L'intérêt de telle structure réside dans le fait qu'elle va servir à ce que l'application d'un modèle de régression sur un jeux des données renvoie tout un ensemble des résultats bien structurés. En fait un modèle de régression n'est en fait qu'un dictionnaire dont les valeurs des clés sont les résultats des fonctions préalablement crées. On peut ainsi vouloir voir un résultat précis en écrivant **nom_du_dictionnaire["nom_de_la_clé"]**.

Pour voir joliment l'ensemble des valeurs associées aux clés d'un dictionnaire, ou pour extraire un résumé du dictionnaire, il plus que recommandé de créer un nouveau code comme ci-dessous

```
def affichage(dictionnaire) :

    for cle,valeur in dictionnaire.items():
        print (cle)
        print()
        print(valeur)
        print()
        print("==========================================================")
    return "implemente par Marcel MOUDILA---M1 ingenierie mathematiques----Nancy"
```

3.2 Création de moudilaRegpoisson

Ici, nous rendons l'intégralité des implémentations que nous avons effectuées pour la régression de poisson. l'implémentation sur R (commande `glm`) est proposée pour servir de comparaison avec les résultats sur Python (via notre commande `moudilaRegpoisson`). L'exemple choisi est juste à titre illustratif. Lors de la soutenance, nous ferons usage d'un exemple sur données réelles + commentaires sur l'interprétation. Nous verrons aussi comment notre algorithme fait de la prédiction.

Pour l'implémentation brute (comme nous l'avons fait dans le cas régression de poisson), des arbres de décision, les algorithmes implémentés pour exécuter les méthodes CARD , ID3, CHAID, etc font appel à une autre notion en programmation qu'on appelle la notion de "diviser pour régner" ou encore "algorithme glouton", et pourrait être la suite de nos travaux dans un avenir proche. C'est pourquoi, nous ne sommes pas rentrer trop en détails sur les concepts mathématiques des arbres de décisions : entropie, gain de l'information, impureté de Gini, etc parce que nous savions à l'avance que nous n'implémenterons pas de façon brute les arbres de décision dans ce mémoire. Nous donnerons à la soutenance des exemples d'applications théoriques sur ces notions qui ont pour but d'expliquer la construction des arbres de décision. Pour les forêts aléatoires, nous avons donné les concepts mathématiques de leur construction dans ce mémoire.

Comparaison résultats de mon propre module de regression de poisson avec celui de R

Marcel

2022-06-01

A) Présentation de la régression de poisson sur R.

```
y = c(2,4,1,2,1)
x1 = c(7,3,-2,1,2)
x2 = c(12,2,5,4,3)
regpois = glm(y~x1+x2,family = poisson(link = "log"))
summary(regpois)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = poisson(link = "log"))
##
## Deviance Residuals:
##        1         2         3         4         5
##  0.01598   0.44603   0.16675   0.23099  -0.97603
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.8290     0.5289   1.567    0.117
## x1            0.1910     0.1762   1.084    0.278
## x2           -0.1237     0.1184  -1.045    0.296
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2.7726  on 4  degrees of freedom
## Residual deviance: 1.2330  on 2  degrees of freedom
## AIC: 19.726
##
## Number of Fisher Scoring iterations: 4
```

B) Présentation de la régression de poisson sur python du même exemple mais avec mon algorithme développé.

```
y = np.array([2,4,1,2,1])
x = np.array([[7,12],[3,2],[-2,5],[1,4],[2,3]])
alpha = 0.05
dico = moudilaRegpoisson(y,x,alpha)
print(affichage(dico))
```

```
## coeff
```

```
##
## [0.8289915381214595, 0.19102378556940222, -0.12369409396633564]
##
## ========================================================
## matrice
##
## [[ 1.  7. 12.]
##  [ 1.  3.  2.]
##  [ 1. -2.  5.]
##  [ 1.  1.  4.]
##  [ 1.  2.  3.]]
##
## ========================================================
## lambda
##
## [1.97749108 3.1729911  0.84237794 1.69086283 2.31626698]
##
## ========================================================
## variance
##
## [[ 0.27974267 -0.01791875 -0.0269937 ]
##  [-0.01791875  0.03104419 -0.01437607]
##  [-0.0269937  -0.01437607  0.01400973]]
##
## ========================================================
## ecarttype
##
## [0.52890705 0.17619361 0.11836272]
##
## ========================================================
## intervalle.confiance
##
##         borne inf  borne sup
## beta_0  -0.207647   1.865630
## beta_1  -0.154309   0.536357
## beta_2  -0.355681   0.108293
##
## ========================================================
## test.wald
##
##         coefficients  ecart-type  statistique de test   p-value
## beta_0      0.828992    0.528907             1.567367  0.117029
## beta_1      0.191024    0.176194             1.084170  0.278290
## beta_2     -0.123694    0.118363            -1.045043  0.296003
##
## ========================================================
## residu.pearson
##
## [ 0.01600654  0.46427549  0.17173685  0.23773714 -0.8648677 ]
##
## ========================================================
## residu.deviance
##
## [ 0.01597632  0.44602626  0.16675998  0.23099478 -0.97602269]
```

```
##
## ======================================================
## pertinence
##
##           statistique de test   p-value
## deviance             1.232982  0.266828
## pearson              1.049817  0.305549
##
## ======================================================
## residu.pearson.standard
##
## [ 0.31905096  0.87871335  0.31917167  0.28101894 -1.02416554]
##
## ======================================================
## anormale
##
## []
##
## ======================================================
## implemente par Marcel MOUDILA---M1 ingenierie mathematiques----Nancy
```

C) Mon code en détails : Auteur : moudilamarcel@gmail.com : licence : libre

```
import numpy as np
import pandas as pd
from math import exp,sqrt, log
from scipy.stats import chi2,norm
from numpy.linalg import inv, multi_dot


from sklearn.linear_model import PoissonRegressor
def param(y,x):
    reg = PoissonRegressor(alpha=0)
    reg.fit(x, y)
    beta =[reg.intercept_]+[reg.coef_[i] for i in range(len(reg.coef_))]
    return beta

def matrice_X(x):
    # x : vecteur des covariables
    # X : matrice des donnees des covariables
    n = len(x)
    p = len(x[0])
    X = np.zeros((n,p+1))
    col = np.ones((n,1))
    for i in range(n):
        X[i] = np.concatenate((col[i],x[i]))
    return X

def lambda_x(beta, X):
    # beta : beta(beta_0,...,beta_p)
    # X : matrice des donnees des covariables
    # sortie : lambda_x(lambda (x_1),...lambda(x_n))
    n = len(X)
    lambda_x = np.zeros(n)
```

```
    for i in range(n):
        lambda_x[i] = np.exp(beta@X[i])
    return lambda_x

def var_b(ld,X):
    # ld : ld(lambda (x_1),...lambda(x_n))
    # X : matrice des données des covariables
    # sortie : matrice des covariances/variances de beta
    n = len(X)
    W = np.diag(ld)
    var = inv(multi_dot([np.transpose(X),W,X]))
    return var

def ecarttype(V):
    # V:  variance de beta
    # sortie : ecart type sd(sd(beta_0),..sd(beta_p))

    n = len(V)
    sd = np.zeros(n)
    for i in range(n):
        sd[i] = sqrt(V[i,i])
    return sd

def ICbeta(beta, alpha, sd):
    # beta : coefficients du modele
    # alpha : seuil de confiance
    # sd : vecteur des ecart-type de beta
    # sortie : tableau des intervalles de confiance de beta_j

    z_alpha = norm.ppf(1-(alpha/2),0,1)
    p = len(beta)-1
    intervalle = np.zeros((p+1,2))
    for i in range(p+1):
        intervalle[i]= np.array([beta[i]-z_alpha*sd[i],
                                 beta[i]+z_alpha*sd[i]])
    resultat = pd.DataFrame(intervalle,
               columns=['borne inf','borne sup'],
               index = ["beta_"+str(i) for i in range(p+1)])
    return resultat

def testWald(beta,sd,n):
    # beta : coefficients du modele
    # sd : vecteur des ecart-type de beta
    # n : nombre d'individus
    # sortie : un tableau des resultats du test de Wald
    p = len(beta)-1
    tab = np.zeros((p+1,4))
    Zobs = np.zeros(p+1)
    pvalue = np.zeros(p+1)
    for i in range(p+1):
        Zobs[i]= beta[i]/sd[i]
        pvalue[i]= (1-norm.cdf(abs(Zobs[i]),0,1))*2
        tab[i] = np.array([beta[i],sd[i],Zobs[i],pvalue[i]])
```

```
    resultat = pd.DataFrame(tab,
               columns=['coefficients','ecart-type','statistique de test','p-value'],
               index = ["beta_"+str(i) for i in range(p+1)])
    return resultat

def residu_pearson(y,ld,beta):
    # y : donnees de la avariable cible
    # ld : ld(lambda (x_1),...lambda(x_n))
    # beta : coefficients du modele
    # sortie : les residus de pearson

    n = len(y)
    p = len(beta)
    rsp = np.zeros(n)
    lambda_x = ld
    for i in range(n) :
        rsp[i] = np.array((y[i] - lambda_x[i])/sqrt(lambda_x[i]))
    return rsp

def residu_deviance(y,ld,beta):
    # y : donnees de la avariable cible
    # ld : ld(lambda (x_1),...lambda(x_n))
    # beta : coefficients du modele
    # sortie : les residus de deviance

    n = len(y)
    rsd = np.zeros(n)
    lambda_x = ld
    for i in range(n) :
        rsd[i] = np.sign(y[i]-lambda_x[i])*sqrt(2*(y[i]*log(y[i]/
                 lambda_x[i])-(y[i]-lambda_x[i])))
    return rsd

def pertinence(y,resd,resp,beta):
    # y : donnees de la avariable cible
    # resd : les residus de deviance
    # resp : les residus de pearson
    # beta : coefficients du modele
    # sortie : tableau de pertinence
    n = len(y)
    p = len(beta)
    #deviance
    Zobs_d = np.sum(resd**2)
    ddl = n-(p+1)
    pvalue_d = 1 - chi2.cdf(Zobs_d,ddl)
    #pearson
    Zobs_p = np.sum(resp**2)
    ddl = n-(p+1)
    pvalue_p = 1 - chi2.cdf(Zobs_p,ddl)
    tab = pd.DataFrame(np.array([[Zobs_d,pvalue_d ],[Zobs_p,pvalue_p]]),
               columns=['statistique de test','p-value'],
               index = ["deviance","pearson"])
    return tab
```

```
def residu_pearson_standardise(resp,ld,X,V):
    # resp : les residus de pearson
    # ld : ld(lambda (x_1),...lambda(x_n))
    # X : matrice des données des covariables
    # V : matrice des covariances/variances de beta

    # sortie : un tableau des observations aberrantes
    n = len(resp)
    W = np.diag(ld)**(1/2)
    mat = multi_dot([W,X,V,np.transpose(X),W])
    resp_stand = np.zeros(n)
    for i in range(n):
        resp_stand[i] = resp[i]/sqrt(1-mat[i][i])
    return resp_stand

def aberrantes(resp_stand):
    # resp_stand : les résidus standardises de pearson
    # sortie : la liste des observations aberrantes

    n = len(resp_stand)
    liste = []
    for i in range(n):
        if abs(resp_stand[i]) > 2:
            liste = liste +[i]
    return liste

def moudilaRegpoisson(y,x,alpha):

    beta = param(y,x)                                  # coeff du modele
    X = matrice_X(x)                                   # matrice X
    ld = lambda_x(beta,X)                              # lambda
    V = var_b(ld,X)                                    # variance de beta
    sd = ecarttype(V)                                  # ecart-type de beta_j
    IC = ICbeta(beta,alpha,sd)                         # intervalle de confiance de beta_j
    n = len(ld)
    tab = testWald(beta,sd,n)                          # test de nullité de beta_j
    resp = residu_pearson(y,ld,beta)                   # residu de pearson
    resd = residu_deviance(y,ld,beta)                  # residu de deviance
    tab2 = pertinence(y,resd,resp,beta)                # test sur la pertinence du modèle
    res = residu_pearson_standardise(resp,ld,X,V)      # résidus standardisés de pearson
    tab3 = aberrantes(res)                             # valeurs anormales

    dictionnaire = {"coeff":beta,"matrice": X,"lambda" :ld,"variance" :V,"ecarttype":sd,
                    "intervalle.confiance":IC,"test.wald":tab,"residu.pearson":resp,
                    "residu.deviance":resd,"pertinence":tab2,
                    "residu.pearson.standard":res,"anormale":tab3}

    return dictionnaire

def pred_Moud_reg_pois(dictionnaire,x):
    beta = np.array(dictionnaire["coeff"])
    xnew = np.array([1]+[i for i in x])
    if len(beta) != len(x)+1 :
```

```
        pass
    else :
        return np.exp(beta@xnew)

def affichage(dictionnaire) :

    for cle,valeur in dictionnaire.items():
        print (cle)
        print()
        print(valeur)
        print()
        print("============================================================")
    return "implemente par Marcel MOUDILA---M1 ingenierie mathematiques----Nancy"
```

Bibliographie

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1) :5–32, 2001.

[GHL+21] Andreas Groll, Lars Magnus Hvattum, Christophe Ley, Franziska Popp, Gunther Schauberger, Hans Van Eetvelde, and Achim Zeileis. Hybrid machine learning forecasts for the uefa euro 2020. *arXiv preprint arXiv :2106.05799*, 2021.

[SSL+16] Erlandson F Saraiva, Adriano K Suzuki, Francisco Louzada, et al. Predicting football scores via poisson regression model : applications to the national football league. *Communications for Statistical Applications and Methods*, 23(4) :297–319, 2016.