1 Introduction

Explication du processus de classement. Utilisation du package ''lime''.

Mon attention a été attirée récemment sur un post (mars 2019) du blog ''Lovely Analytics'' concernant l'interprétation du processus de classement à l'aide du package ''lime'' pour Python. Très peu souvent abordée dans les articles scientifiques, l'affaire est pourtant d'importance en pratique. En effet, quoi de plus naturel que d'essayer d'identifier les caractéristiques qui ont prédominé lors de l'attribution d'une classe à un individu en prédiction. Dans de nombreux domaines, cette justification est primordiale pour asseoir la crédibilité de la décision. On sait que le modèle refuse l'attribution d'un crédit à une personne parce qu'elle est au chômage, ou parce qu'elle a bouton sur le nez, ou, pire, parce qu'elle a un revers à deux mains, ou que sais-je encore… en tous les cas, on ne peut certainement pas se retrancher derrière la décision de l'ordinateur en invoquant l'infaillibilité de la data science et des fameux algorithmes.

La question est essentielle mais la réponse n'est pas toujours évidente. Elle est simple avec un arbre de décision, elle l'est un peu moins avec un modèle linéaire, elle devient ardue avec les modèles non-linéaires. Le package ''lime'' propose une approche agnostique – applicable à tout type de modèle prédictif – en procédant par augmentation des données et approximation à l'aide d'un modèle linéaire dans la zone de prise de décision. Son avantage est de pouvoir proposer une lecture des résultats lorsque l'on applique un modèle ''boîte noire''. Mais, dans ce cas, nous ne pouvons que prendre pour argent comptant l'interprétation fournie puisque nous n'avons aucun recul sur les résultats.

Dans ce tutoriel, nous essayons d'expertiser les solutions avancées par ''lime'' en vérifiant leur adéquation avec l'interprétation usuelle que l'on peut faire du classement dans les situations où on sait le faire, à savoir lorsqu'on utilise les arbres de décision et les classifieurs linéaires.

2 Importation et préparation des données

Nous utilisons les données ''spam'' accessible sur le serveur UCI Machine Learning Repository. L'objectif est d'identifier les courriels frauduleux (spam = yes) à partir de leurs caractéristiques (fréquences des mots, de certains caractères, nombre de caractères en majuscules). Nous avons au préalable scindé les données en 4599 observations pour la modélisation (TRAIN) et 2 observations (TEST) pour le déploiement. L'objectif est de décortiquer dans le détail le mécanisme d'affectation sur ces 2 individus.

Nous utilisons la version 3.6.8 de l'interpréteur Python. Je le précise parce que je n'ai pas pu installer ''lime'' sous Python 3.7.1. J'ai donc dû downgrader ma configuration.

```
#version de Python
import sys
print(sys.version)
```

```
3.6.8 |Anaconda, Inc.| (default, Feb 21 2019, 18:30:04) [MSC v.1916 64 bit (AMD64)]
```

Nous chargeons les données d'apprentissage (sheet_name=0) à l'aide de la librairie ''pandas''. Nous disposons de 4599 observations et 56 variables, dont la cible ''spam''.

```
#changement de dossier
import os
os.chdir("... votre dossier ...")

#train
import pandas
D = pandas.read_excel("spam_lime.xlsx",sheet_name=0)
print(D.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4599 entries, 0 to 4598
Data columns (total 56 columns):
spam                          4599 non-null object
wf_make                       4599 non-null float64
wf_address                    4599 non-null float64
wf_all                        4599 non-null float64
wf_3d                         4599 non-null float64
wf_our                        4599 non-null float64
wf_over                       4599 non-null float64
wf_remove                     4599 non-null float64
wf_internet                   4599 non-null float64
wf_order                      4599 non-null float64
wf_mail                       4599 non-null float64
wf_receive                    4599 non-null float64
wf_will                       4599 non-null float64
wf_people                     4599 non-null float64
wf_report                     4599 non-null float64
wf_addresses                  4599 non-null float64
wf_free                       4599 non-null float64
wf_business                   4599 non-null float64
wf_email                      4599 non-null float64
wf_you                        4599 non-null float64
wf_credit                     4599 non-null float64
wf_your                       4599 non-null float64
wf_font                       4599 non-null float64
wf_000                        4599 non-null float64
wf_money                      4599 non-null float64
wf_hp                         4599 non-null float64
wf_hpl                        4599 non-null float64
wf_lab                        4599 non-null float64
```

```
wf_labs                         4599 non-null float64
wf_telnet                       4599 non-null float64
wf_857                          4599 non-null float64
wf_data                         4599 non-null float64
wf_415                          4599 non-null float64
wf_85                           4599 non-null float64
wf_technology                   4599 non-null float64
wf_1999                         4599 non-null float64
wf_parts                        4599 non-null float64
wf_pm                           4599 non-null float64
wf_direct                       4599 non-null float64
wf_cs                           4599 non-null float64
wf_meeting                      4599 non-null float64
wf_original                     4599 non-null float64
wf_project                      4599 non-null float64
wf_re                           4599 non-null float64
wf_edu                          4599 non-null float64
wf_table                        4599 non-null float64
wf_conference                   4599 non-null float64
cf_comma                        4599 non-null float64
cf_bracket                      4599 non-null float64
cf_sqbracket                    4599 non-null float64
cf_exclam                       4599 non-null float64
cf_dollar                       4599 non-null float64
cf_hash                         4599 non-null float64
capital_run_length_average      4599 non-null float64
capital_run_length_longest      4599 non-null int64
capital_run_length_total        4599 non-null int64
dtypes: float64(53), int64(2), object(1)
```

Nous centrons et réduisons les variables pour la modélisation. L'opération ne s'impose pas pour la modélisation à l'aide des arbres de décision ; elle est en revanche nécessaire pour la régression logistique car, en ramenant les variables sur la même échelle, elle permet de comparer directement les coefficients de la régression pour apprécier l'impact des variables dans le modèle.

```
#transformation échantillon
from sklearn.preprocessing import StandardScaler
stds = StandardScaler()
Z = stds.fit_transform(X=D.iloc[:,1:])
print(Z.mean(axis=0))
```

```
[-6.95247295e-18  0.00000000e+00 -9.26996394e-18  1.66086854e-17
  5.09848017e-17  1.39049459e-17 -1.73811824e-17  1.23599519e-17
  7.64772025e-17 -4.01698437e-17 -1.69949339e-17  1.54499399e-18
  4.94398077e-17 -2.16299159e-17  2.70373948e-17 -1.04287094e-17
  3.08998798e-17  3.88179740e-17  4.63498197e-17  1.54499399e-18
 -2.93548858e-17  2.16299159e-17  3.43761163e-17 -1.39049459e-17
  4.17148377e-17  1.62224369e-17  3.24448738e-17  6.17997596e-18
```

```
  4.21010862e-17  1.31324489e-17 -4.32598317e-17 -2.16299159e-17
  1.42911944e-17 -3.12861283e-17  7.60909540e-17 -3.08998798e-18
 -1.04287094e-17  8.49746694e-18 -3.55348618e-17  0.00000000e+00
 -4.94398077e-17  3.32173708e-17  2.70373948e-17 -1.08149579e-17
  1.23599519e-17 -1.23599519e-17 -2.62648978e-17 -1.05059591e-16
  1.31324489e-17 -1.54499399e-18  9.26996394e-18  2.08574189e-17
  3.08998798e-18 -1.23599519e-17 -1.08149579e-17]
```

Forcément, les moyennes sont nulles après cette transformation.

Nous isolons le vecteur cible dans une structure spécifique par commodité.

```
#isoler la cible par commodité
Y = D.iloc[:,0]
```

Nous chargeons ensuite l'échantillon de déploiement (sheet_name = 1).

```
#données à classer
DC = pandas.read_excel("spam_lime.xlsx",sheet_name=1)
print(DC)
```

```
  spam  wf_make  ...  capital_run_length_longest  capital_run_length_total
0  yes      0.4  ...                          53                       608
1   no      0.0  ...                          18                       214
```

L'individu n°0 appartient à la classe 'spam = yes', le n°1 à 'spam = no'. Il faudra s'en rappeler lorsque nous calculerons les probabilités d'affectation attribuées par les modèles en déploiement.

Nous centrons et réduisons également ces données, mais en utilisant les paramètres (moyennes, écarts-type) calculés sur l'échantillon d'apprentissage.

```
#centrage réduction avec paramètre de l'apprentissage
ZC = stds.transform(DC.iloc[:,1:])
print(ZC)
```

```
[[ 9.68811998e-01  9.83125679e-02 -2.11714587e-02 -4.69098029e-02
  -2.71144196e-01  1.11227266e+00 -1.38596889e-01  2.35994951e-01
  -3.23070556e-01  1.73770474e+00  1.04360475e+00  1.61328523e-01
   2.84336492e+00 -1.74966363e-01  5.85480453e-01 -3.01322293e-01
  -3.20993964e-01 -3.47633646e-01  1.13622353e+00 -1.67930645e-01
   7.96691232e-05 -1.18197560e-01  8.08137050e-02  3.97146180e-01
  -3.28677838e-01 -2.98966372e-01 -1.66768708e-01 -2.25290974e-01
  -1.59929969e-01 -1.43243796e-01 -1.74780149e-01 -1.45247386e-01
  -1.98112143e-01 -2.42185953e-01 -3.23533297e-01 -5.98492928e-02
  -1.80951965e-01 -1.85345521e-01 -1.20931355e-01 -1.72638606e-01
  -2.06039801e-01 -1.26557468e-01 -2.97846692e-01 -1.97319020e-01
  -7.14035045e-02 -1.11570784e-01 -1.05099812e-01 -3.21671901e-01
  -1.55232230e-01  2.51118991e-01  4.93241984e-01  4.85234894e-02
  -6.00391289e-02  4.43272166e-03  5.35570641e-01]
 [-3.42121023e-01 -1.65108780e-01  6.33366894e-01 -4.69098029e-02
  -4.64437124e-01 -3.49963844e-01 -2.91862730e-01 -2.62622580e-01
  -3.23070556e-01 -3.71456265e-01 -2.96549266e-01 -1.29167420e-01
```

```
 -4.64038010e-02 -1.74966363e-01 -1.89542819e-01 -3.01322293e-01
 -3.20993964e-01 -3.47633646e-01  7.64462754e-01 -1.67930645e-01
  1.83278818e-01 -1.18197560e-01 -2.90277615e-01 -2.12823548e-01
 -3.28677838e-01 -2.98966372e-01 -1.66768708e-01 -2.25290974e-01
 -1.59929969e-01 -1.43243796e-01 -1.74780149e-01 -1.45247386e-01
 -1.98112143e-01 -2.42185953e-01 -3.23533297e-01 -5.98492928e-02
  3.07613838e-03 -1.85345521e-01 -1.20931355e-01 -1.72638606e-01
 -2.06039801e-01 -1.26557468e-01  2.06216600e-01 -1.97319020e-01
 -7.14035045e-02 -1.11570784e-01 -1.58488677e-01 -2.06938477e-01
 -1.55232230e-01 -2.08580968e-01 -3.08097023e-01 -1.02861107e-01
 -1.21710328e-01 -1.75157775e-01 -1.14197437e-01]]
```

3 Arbre de décision

3.1 Modélisation

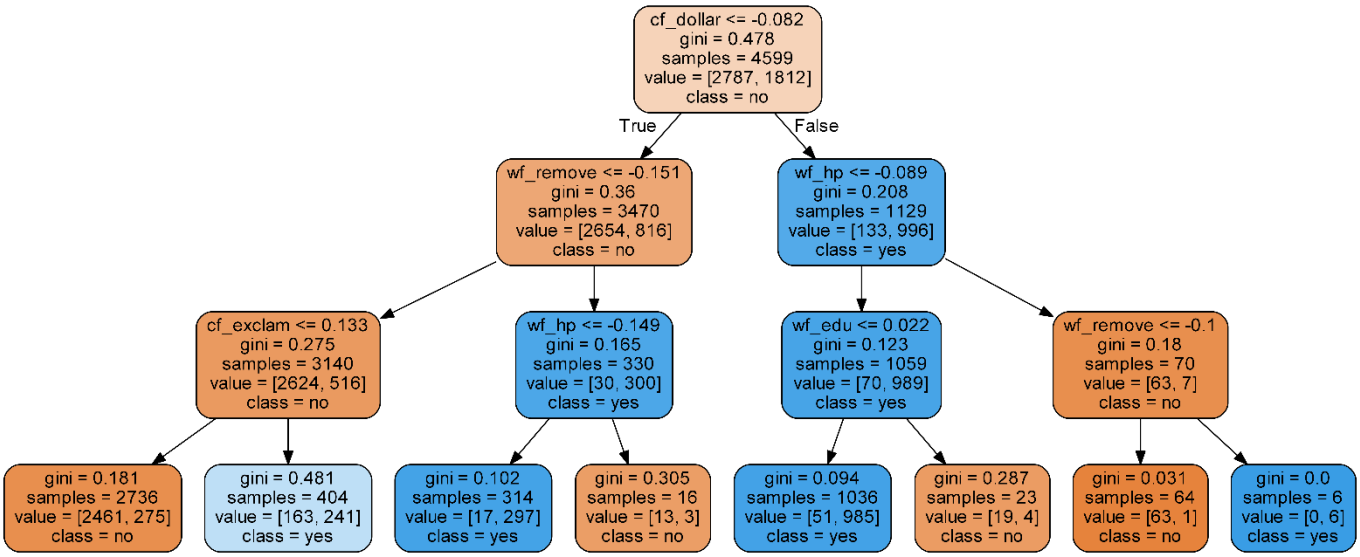
Nous instancions et nous construisons un arbre de décision avec la classe DecisionTreeClassifier de la librairie ''scikit-learn''. Nous avons limité la profondeur de l'arbre à 3 niveaux (max_depth = 3) pour faciliter la lecture.

```
#arbre de décision
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=3)

#apprentissage
dt.fit(Z,Y)
```

L'affichage d'un arbre n'est pas aisé avec cet outil. Il faut travailler en deux temps : (1) créer un fichier '.dot' à partir de l'arbre généré ; (2) que l'on fait traduire en image PNG avec le logiciel GraphViz (qu'il faut installer au préalable) (cf. Russel, ''Creating and Visualizing Decision Trees with Python'', août 2017).

```
#exportation de l'arbre au format dot
from sklearn import tree
import numpy
tree.export_graphviz(dt,out_file='tree.dot',feature_names=D.columns[1:],class_
names=numpy.unique(Y),filled=True,rounded=True)

#transformation du fichier .dot au format .png
#il faut installer le logiciel GraphViz au préalable -- https://www.graphviz.org/
from subprocess import call
call(['C:\\Graphviz2.38\\bin\\dot','-Tpng','tree.dot','-o','tree.png','-Gdpi=600'])
```

Voici le fichier 'tree.png' obtenu.



La racine a été segmentée avec la variable ''cf_dollar'' (fréquence du caractère $), avec pour valeur seuil ''-0.082''. La branche à gauche (True) correspond à la condition ''cf_dollar ≤ -0.082'', la branche à droite (False) à ''cf_dollar > -0.082''. Etc. Chaque chemin aboutissant à une feuille correspond à une règle. Pour comprendre le mécanisme d'affectation d'une classe à un individu, il suffit de suivre la voie qu'il emprunte de la racine vers la feuille.

3.2 Importance des variables

L'arbre ''scikit-learn'' fournit l'importance des variables dans l'arbre. Nous créons une petite fonction pour afficher les résultats de manière plus sympathique : ''refs'' correspond au vecteur permettant de trier les variables, ''values'' indique les valeurs à afficher.

```
#fonction pour affichage des indicateurs des variables
#décroissantes en valeur absolue
#var_names : nom des variables
#refs: sert à trier les valeurs
#values : indicateurs à afficher
#nb nombre de valeurs à afficher
def affichage_influence(var_names,refs,values=None,nb=-1):
    #vérification du nombre d'éléments à afficher
    if (nb == -1) or (nb > refs.shape[0]):
        nb = refs.shape[0]
    #index des valeurs triés en valeur absolue - décroissantes
    index = numpy.argsort(numpy.abs(refs))[::-1]
    #valeurs à afficher
    if (values is None):
        display = refs
    else:
        display = values
```

```
    #table
    temp = pandas.DataFrame({'variable':var_names[index],'indicateur':display[index]})
    #affichage des nb premiers
    print(temp.head(nb))
```

Ainsi, si nous souhaitons identifier les 15 variables les plus importantes :

```
#affichage des 15 variables les plus pertinentes dans l'arbre
affichage_influence(D.columns[1:],refs=dt.feature_importances_,nb=15)
```

```
       variable  indicateur
0     cf_dollar    0.522779
1     wf_remove    0.250590
2     cf_exclam    0.126961
3         wf_hp    0.079765
4        wf_edu    0.019906
5     wf_report    0.000000
6      wf_money    0.000000
7        wf_000    0.000000
8       wf_font    0.000000
9       wf_your    0.000000
10    wf_credit    0.000000
11       wf_you    0.000000
12     wf_email    0.000000
13  wf_business    0.000000
14      wf_free    0.000000
```

Contrairement à d'autres outils (rpart de R par exemple), l'arbre de ''scikit-learn'' n'accorde de l'importance qu'aux variables qui apparaissent explicitement dans l'arbre.

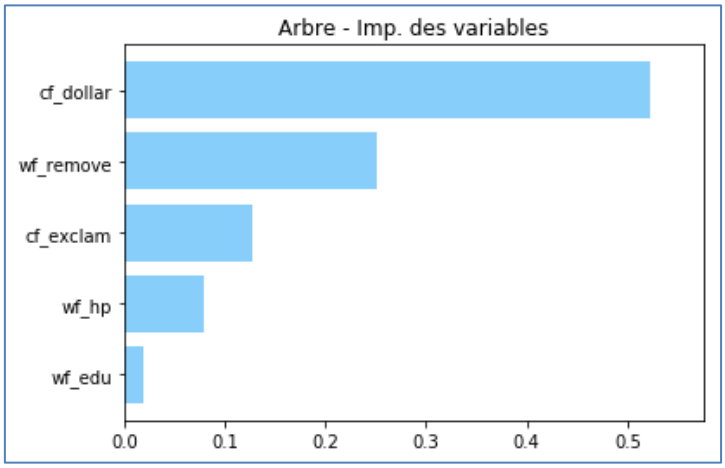
Nous pouvons aussi opter pour un affichage graphique avec des couleurs tant qu'à faire. Cette nouvelle fonction nous servira surtout plus loin lorsqu'il faudra évaluer l'importance et le sens de l'influence des variables en classement.

```
#package graphique
import matplotlib.pyplot as plt

#fonction pour affichage graphique
def affichage_influence_graphique(titre,var_names,refs,values=None,nb=-1):
    #vérification du nombre d'éléments à afficher
    if (nb == -1) or (nb > refs.shape[0]):
        nb = refs.shape[0]+1
    #index
    index = numpy.argsort(numpy.abs(refs))[::-1]
    #valeurs à afficher
    if (values is None):
        display = refs
    else:
```

```
        display = values
    #min/max
    lv = numpy.min(display)
    hv = numpy.max(display)
    hh = max(abs(lv),abs(hv))
    if (lv >= 0):
        lv = 0
    else:
        lv = -hh
    #code couleur
    code_couleur = numpy.array(['lightgreen','lightskyblue'])[numpy.array((numpy.sign(refs[index][:nb][::-1])+1)//2,dtype='int')]
    #affichage du barplot
    plt.barh(var_names[index][:nb][::-1],display[index][:nb][::-1],color=code_couleur)
    plt.xlim(lv*1.1,+hh*1.1)
    plt.title(titre)
    plt.show()
```

Ainsi, pour les 5 variables les plus importantes.

```
#importances des 5 variables (les plus importantes)
affichage_influence_graphique('Arbre - Imp. des variables',D.columns[1:],refs=dt.feature_importances_,nb=5)
```



Ces seules variables interviendront lors de l'attribution des classes aux individus supplémentaires en classement. Mais comment ? C'est ce que nous allons essayer de détailler pour les individus n°0 et n°1 des données TEST.

3.3 Mécanisme d'attribution des classes de l'arbre – Individu n°0

Avec la commande predict_proba(), nous calculons les probabilités d'affectation aux classes de l'individu à classer.

```
#proba. pour l'individu n°0
pdt0 = dt.predict_proba(numpy.reshape(ZC[0,:],(1,-1)))
print(pdt0)
```

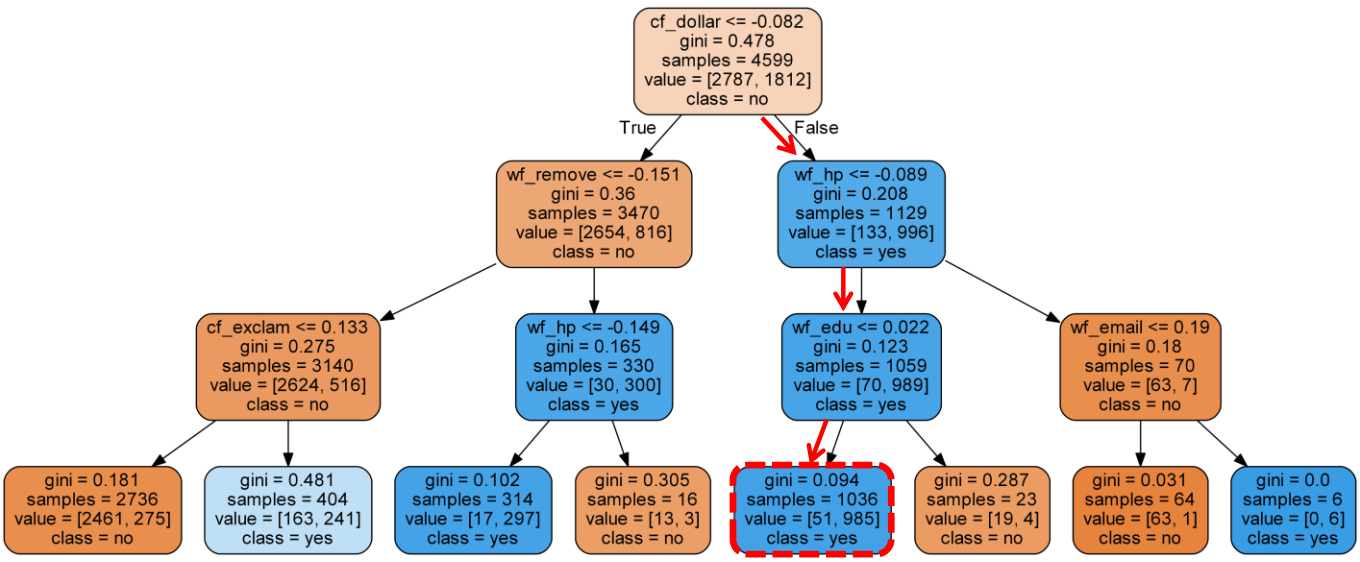
```
[[0.0492278 0.9507722]]
```

D'où viennent ces valeurs ? Pour répondre, nous devons retracer le chemin suivi par l'individu. Voici sa description en valeurs centrées et réduites. Pour faciliter la lecture, je n'ai conservé que les variables qui apparaissent dans l'arbre.

```
#affichage de ses valeurs
print(pandas.DataFrame({'variable':D.columns[1:],'valeurs':ZC[0,:]}))
```

```
                    variable   valeurs
...
6                  wf_remove -0.138597
...
24                     wf_hp -0.328678
...
43                    wf_edu -0.197319
...
49                 cf_exclam  0.251119
50                 cf_dollar  0.493242
...
```

Le chemin emprunté par l'individu de la racine à la feuille concernée (la règle déclenchée) est matérialisé dans l'arbre (flèches rouges).



La feuille aboutit à la conclusion ''spam = yes'' avec la probabilité $985/(985+51) = 0.9507722$. Exactement le résultat obtenu avec predict_proba(). De fait, comme les variables sont centrées, on peut en déduire que l'on affecte la classe 'yes' à cet individu parce qu'il présente :

- Une valeur élevée (supérieure au seuil -0.082) de 'cf_dollar' c.-à-d. le terme 'dollar' apparaît plus souvent que la moyenne (à peu près) dans le document à classer ;
- Une valeur faible de 'wf_hp' ;
- Une valeur faible de 'wf_edu'.

Disposer d'un modèle explicite avec des règles d'affectation aisément interprétables fait partie des énormes atouts des arbres de décision. Un expert du domaine, non statisticien, peut comprendre la règle et vérifier si elle est raccord avec les connaissances du domaine. Il peut ainsi la valider ou l'invalider en s'appuyant sur des informations qui ne sont pas inscrites dans les données. Cette synergie fait toute la puissance de la data science.

3.4 Interprétation avec ''lime''

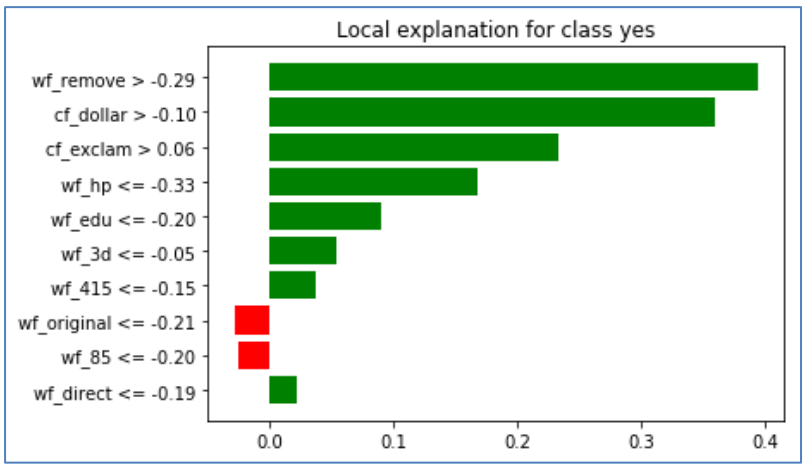
Voyons ce que donne ''lime'' dans ce contexte où nous connaissons la solution. Il faut avoir installer la librairie au préalable. Attention encore une fois, je n'en suis pas arrivé à bout sous Python 3.7. Je me suis rabattu sur Python 3.6.8.

Après avoir importé la classe chargée des calculs, nous l'instancions en lui passant les données d'apprentissage. Cette opération permet à l'outil d'établir le voisinage de l'individu à traiter en prédiction.

```
#librairie lime
from lime.lime_tabular import LimeTabularExplainer

#passer les données d'apprentissage
explainer = LimeTabularExplainer(Z,feature_names=D.columns[1:],class_names=numpy.unique(Y))
```

Nous lui transmettons ensuite les coordonnées de l'individu n°0 à traiter (ZC[0, :]), puis le modèle à utiliser (dt.predict_proba) pour produire les probabilités d'affectation. Nous limitons le nombre de variable à afficher à (num_features=10). Nous affichons les résultats sous forme graphique.

```
#explication pour l'individu test n°0
expdt0 = explainer.explain_instance(ZC[0,:],dt.predict_proba,num_features=10)

#affichage sous forme graphique
expdt0.as_pyplot_figure()
```



Du côté des résultats attendus. Les variables 'cf_dollar', 'wf_hp' et 'wf_edu' contribuent bien positivement à l'attribution de la classe 'yes', dans le sens identifié dans l'arbre : 'cf_dollar' élevé, 'wf_hp' faible, 'wf_edu'. C'est un résultat très encourageant.

Du côté des résultats inattendus. Les variables 'wf_remove' et 'cf_exclam' semblent jouer un rôle important également alors qu'elles n'apparaissent pas dans la règle d'affectation. En réalité, ce n'est pas une erreur en soi. Dans les arbres en particulier, les variables intégrées dans la structure masquent le rôle qu'arait pu jouer les autres prédicteurs. C'est pour cette raison que des outils tels que rpart de R affichent l'importance de certaines variables même si elles n'apparaissent jamais explicitement dans la structure. De fait, les résultats proposés par ''lime'' sont certes déroutants si l'on ne connait pas cette singularité des arbres de décision, mais ils ne sont pas foncièrement erronés. Finalement, avec un peu recul, on pourrait même dire qu'ils enrichissent les résultats de l'arbre car ce dernier élude une partie de l'information.

3.5 Classement de l'individu n°1

Pour l'individu n°1 à classer, predict_proba() propose les probabilités suivantes.

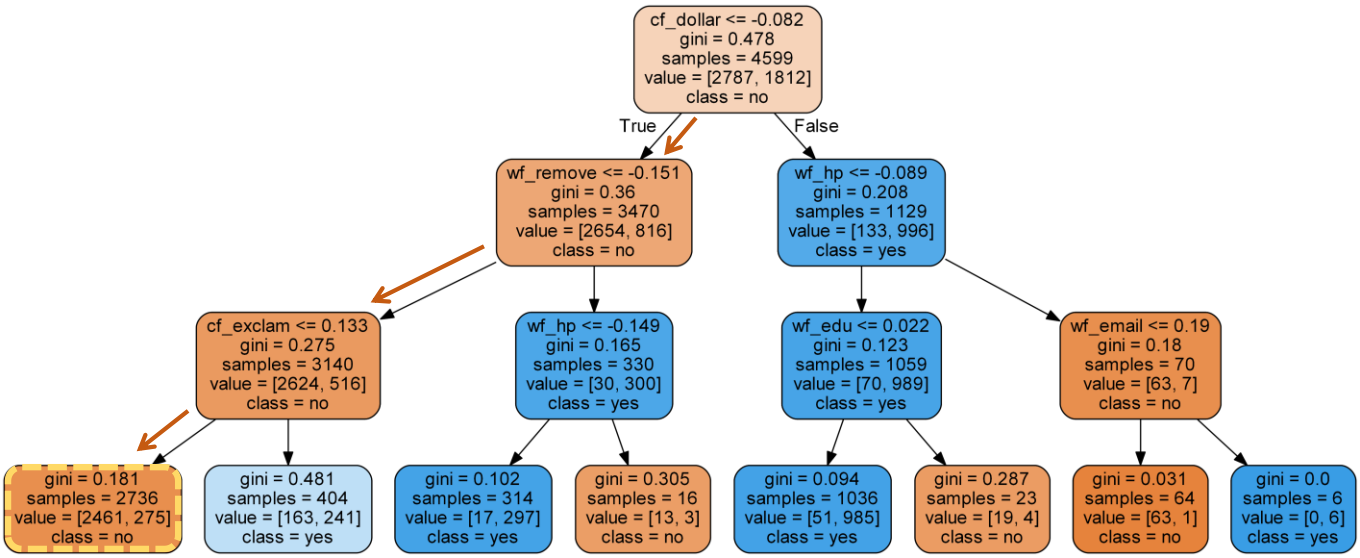
```
#proba pour l'individu n°1
pdt1 = dt.predict_proba(numpy.reshape(ZC[1,:],(1,-1)))
print(pdt1)
```

```
[[0.8994883 0.1005117]]
```

Voici sa description pour que nous puissions identifier la règle déclenchée en prédiction.

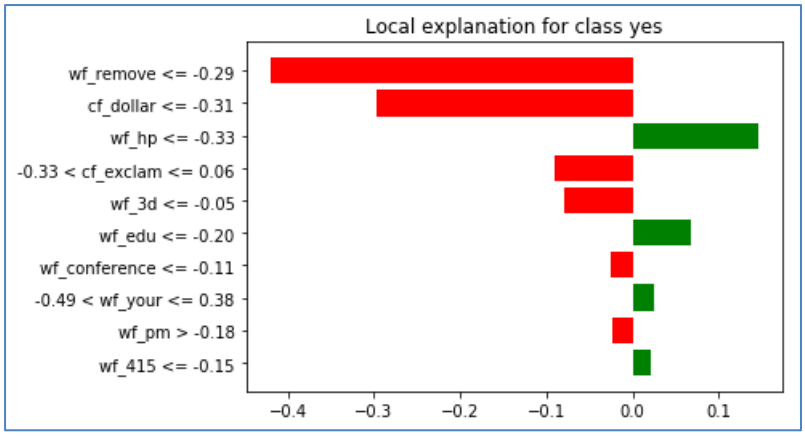
```
                    variable   valeurs
...
6                  wf_remove -0.291863
...
24                     wf_hp -0.328678
...
43                    wf_edu -0.197319
...
49                 cf_exclam -0.208581
50                 cf_dollar -0.308097
...
```

La distribution des classes dans la feuille correspond bien à la probabilité d'affectation fournie par le modèle pour l'individu à traiter.



Et que nous dit ‘’lime’’ ?

```
#explication pour l'individu test n°0
expdt1 = explainer.explain_instance(ZC[1,:],dt.predict_proba,num_features=10)

#affichage sous forme graphique
expdt1.as_pyplot_figure()
```



Le rôle déterminant de ‘wf_remove’ et ‘cf_dollar’, parce qu’elles prennent des valeurs faibles, est bien identifié dans l’attribution de la classe ‘no’. Il en est de même pour ‘cf_exclam’. Pour cet individu également, ‘’lime’’ se comporte plutôt bien.

4 Régression logistique

L’arbre de décision propose une lecture simple et claire du processus d’affectation. Mais en éclipsant le rôle des variables qui n’apparaissent pas dans le modèle, il ne permet pas de situer réellement l’intérêt de ‘’lime’’. Pour aller plus loin dans notre étude, voyons ce qu’il en est cette fois-ci avec un classifieur linéaire, en l’occurrence la régression logistique. Etant linéaire, les

interprétations locales et globales du processus d'affectation se confondent. Les résultats devraient directement concorder. On verra ce qu'il en est.

4.1 Modélisation

Nous implémentons la régression logistique de ''scikit-learn'' et nous récupérons les coefficients du modèle dans un vecteur dédié.

```
#regression logistique
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='lbfgs')

#apprentissage
lr.fit(Z,Y)

#coefficients
lr_coef = lr.coef_[0,:]
print(lr_coef)

#affichage avec les noms de variabes
print(pandas.DataFrame({'variable':D.columns[1:],'coef':lr_coef}))
```

```
                      variable      coef
0                      wf_make -0.076466
1                   wf_address -0.199071
2                       wf_all  0.086670
3                        wf_3d  1.091065
...
49                   cf_exclam  0.441162
50                   cf_dollar  1.408155
51                     cf_hash  0.763532
52  capital_run_length_average -0.370344
53  capital_run_length_longest  1.611127
54    capital_run_length_total  0.485118
```

Le LOGIT du modèle s'écrit donc :

$$\text{LOGIT} = -0.076466 \times \text{wf\_make} -0.199071 \times \text{wf\_address} + 0.086670 \times \text{wf\_all} + \ldots$$

4.2 Influence des variables

Distinguer ce qu'il y a d'important là-dedans n'est pas facile. Nous trions les variables selon la valeur absolue du coefficient. L'influence d'une variable exprime à la fois son importance relative dans le modèle et le sens de la relation avec la variable cible. Les coefficients de la régression logistique calculés sur des données centrées et réduites traduit exactement cette idée.

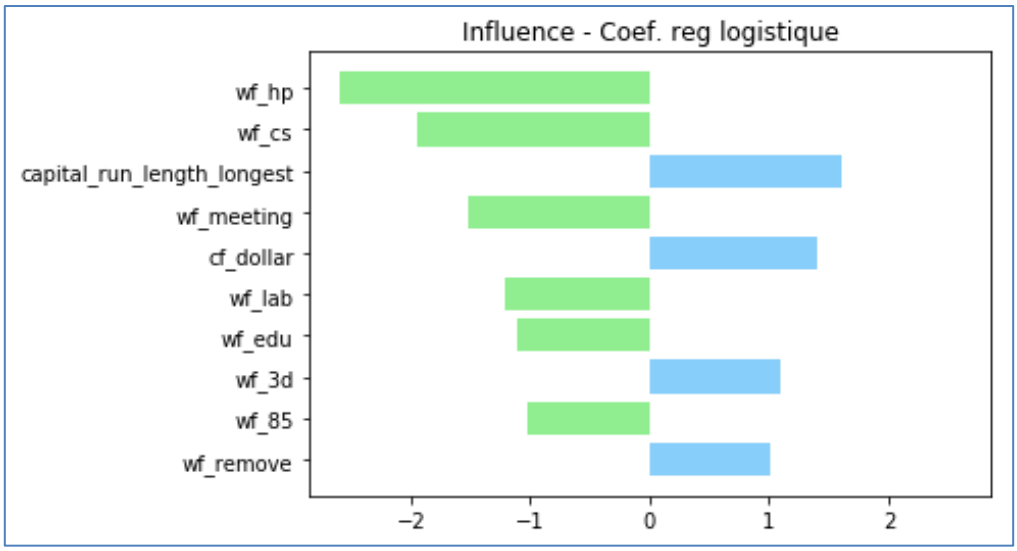
```
#affichage des coefficients de la régression - 10 premiers (en valeur absolue)
affichage_influence(var_names=D.columns[1:],refs=lr_coef,nb=10)
```

```
                     variable  indicateur
0                       wf_hp   -2.602000
1                       wf_cs   -1.940503
2  capital_run_length_longest    1.611127
3                  wf_meeting   -1.526795
4                   cf_dollar    1.408155
5                      wf_lab   -1.215523
6                      wf_edu   -1.104762
7                       wf_3d    1.091065
8                       wf_85   -1.017639
9                   wf_remove    1.001434
```

Le coefficient de 'wf_hp' est le plus élevé en valeur absolue (-2.602) c.-à-d. toutes choses égales par ailleurs, si 'wf_hp' augmente d'une unité (d'un écart-type dans l'espace initial puisque les variables ont été réduites), le LOGIT diminuera (puisque le signe est négatif) de 2.602. Autrement dit, le courriel a plus tendance à appartenir à la classe 'no' lorsque la fréquence relative du terme 'hp' prend des valeurs élevées. Etc.

Un affichage graphique permet de mieux distinguer visuellement l'influence des variables.

```
#affichage graphique pour les coefs de la régression
affichage_influence_graphique('Influence - Coef. reg logistique',var_names=D.columns[1:],refs=lr_coef,nb=10)
```



4.3 Interprétation du classement de l'individu n°0

Nous calculons les probabilités d'appartenance aux classes pour l'individu n°0.

```
#prédiction pour le n°0
plr0 = lr.predict_proba(numpy.reshape(ZC[0,:],(1,-1)))
print(plr0)
```

```
[[0.0967504 0.9032496]]
```

Pour comprendre le mécanisme d'attribution des probabilités, inspectons les valeurs des descripteurs pour les 10 variables les plus influentes.

```
#valeurs des 10 variables les plus influentes
affichage_influence(D.columns[1:],refs=lr_coef,values=ZC[0,:],nb=10)
```

	variable	indicateur
0	wf_hp	-0.328678
1	wf_cs	-0.120931
2	capital_run_length_longest	0.004433
3	wf_meeting	-0.172639
4	cf_dollar	0.493242
5	wf_lab	-0.166769
6	wf_edu	-0.197319
7	wf_3d	-0.046910
8	wf_85	-0.198112
9	wf_remove	-0.138597

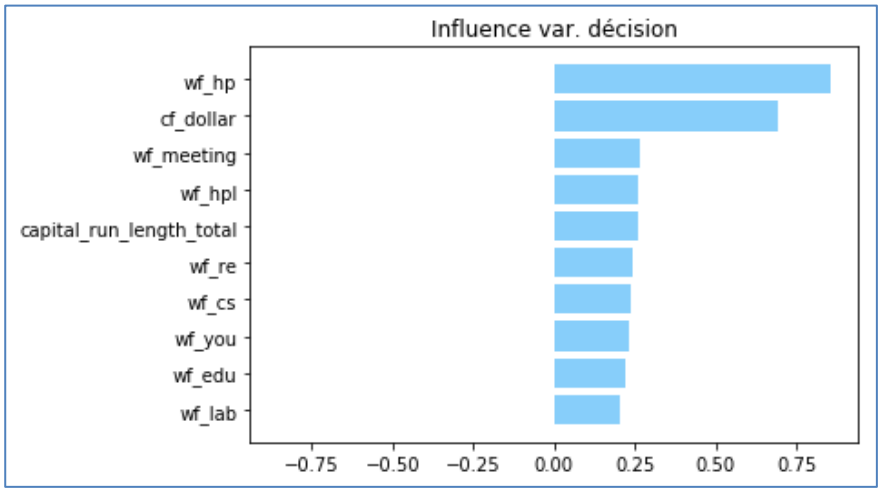
Le LOGIT calculé pour l'individu n°0 devient :

$$\text{LOGIT}(0) = -2.602 \times (-0.329)\ [\text{wf\_hp}] - 1.941 \times (-0.121)\ [\text{wf\_cs}] + 1.611 \times 0.004\ [\text{capit...}] + \ldots$$

Pour un individu à classer, la forte contribution d'un descripteur dans la prise de décision résulte de la conjonction d'un coefficient élevé (en valeur absolue) et d'une valeur élevée prise par la variable (en valeur absolue toujours). Et le sens de la contribution dépend du produit des signes de ces deux éléments.

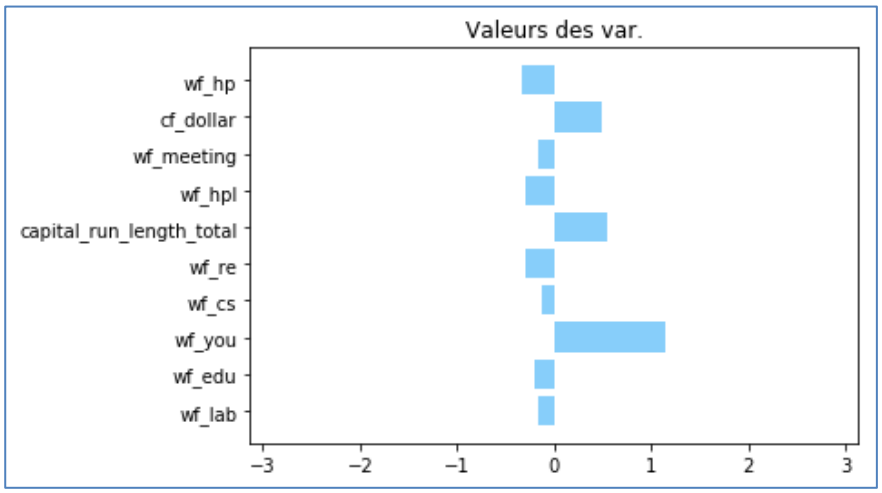
Formons ce produit et identifions les couples qui contribuent le plus dans le classement de l'individu n°0 :

```
#influence dans la prise de décision
inf_lr0 = lr_coef * ZC[0,:]
affichage_influence_graphique('Influence var. décision',var_names=D.columns[1:],refs=inf_lr0,nb=10)
```



Nous constatons que les 10 tandems les plus influents concourent à l'attribution de la classe 'yes' à l'individu n°0. Deux variables pèsent très fortement ('wf_hp' et 'cf_dollar'), mais pour des raisons différentes :

```
#influence avec valeurs des variables
affichage_influence_graphique('Valeurs des var.',var_names=D.columns[1:],refs=inf_lr0,values=ZC[0,:],nb=10)
```

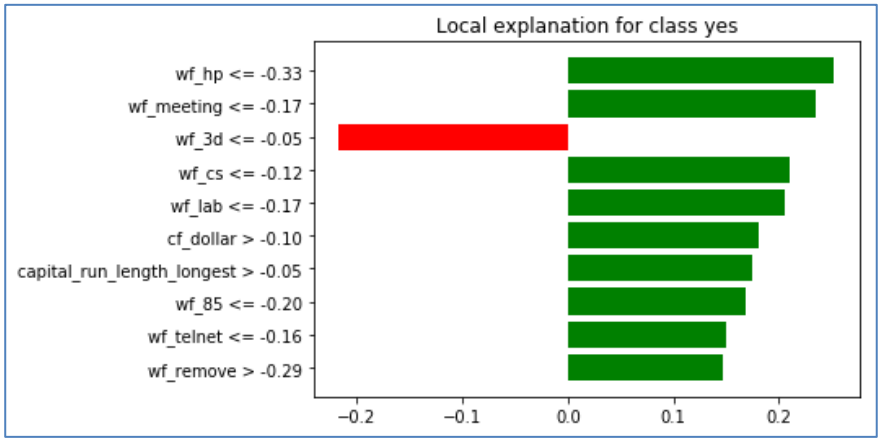


- Parce que 'wf_hp' prend une valeur faible, inférieure à la moyenne (-0.329) ;
- Parce que 'cf_dollar' présente une valeur élevée (0.493)

4.4 Comparaison avec ''lime''

Voyons ce que dit ''lime'' dans nous lui passons la régression logistique comme modèle sous-jacent.

```
#calcul influence avec lime
explr0 = explainer.explain_instance(ZC[0,:],lr.predict_proba,num_features=10)
explr0.as_pyplot_figure()
```



Le rôle prééminent de la variable 'wf_hp', parce qu'elle prend une valeur négative, est détecté. Celui de 'cf_dollar' est minimisé en revanche (même s'il apparaît quand-même en 6ème position).

A contrario, on s'interroge sur l'importance accordée à la variable 'wf_3d' dont le produit entre le coefficient (1.091065) et la valeur (-0.046910) ne pèse pas vraiment dans le LOGIT (-0.051182) par rapport à d'autres variables (32ème position pour le produit coefficient x valeur).

4.5 Classement de l'individu n°1

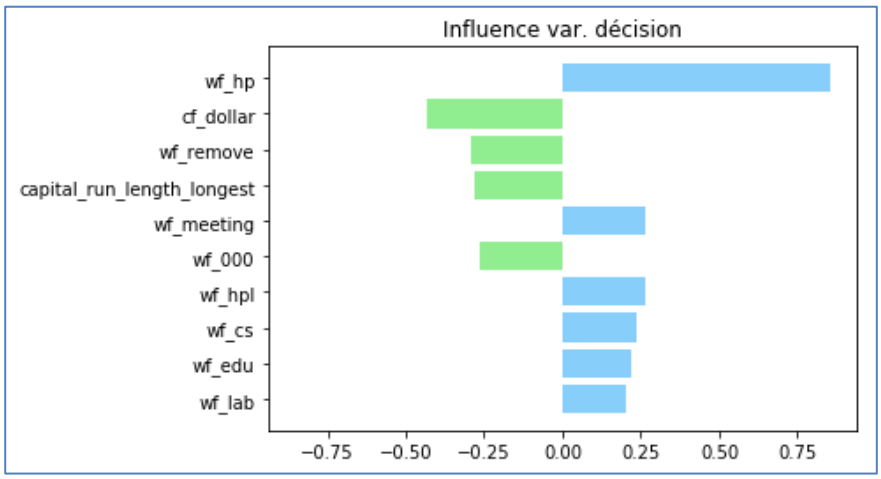
Pour l'individu n°1, les probabilités d'affectation tendent vers la prédiction de la classe 'no'.

```
#prédiction pour le n°1
plr1 = lr.predict_proba(numpy.reshape(ZC[1,:],(1,-1)))
print(plr1)
```

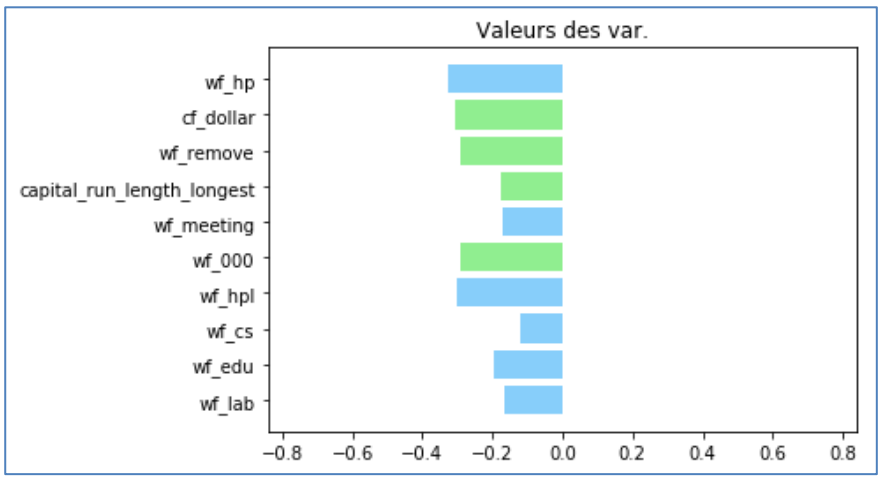
```
[[0.78842157 0.21157843]]
```

Essentiellement à cause des variables 'cf_dollar', 'cf_remove', 'capital_run_length_longest', 'wf_000', etc.

```
#influence dans la prise de décision (produit coef x valeur)
inf_lr1 = lr_coef * ZC[1,:]
affichage_influence_graphique('Influence var. décision',var_names=D.columns[1:],refs=inf_lr1,nb=10)
```

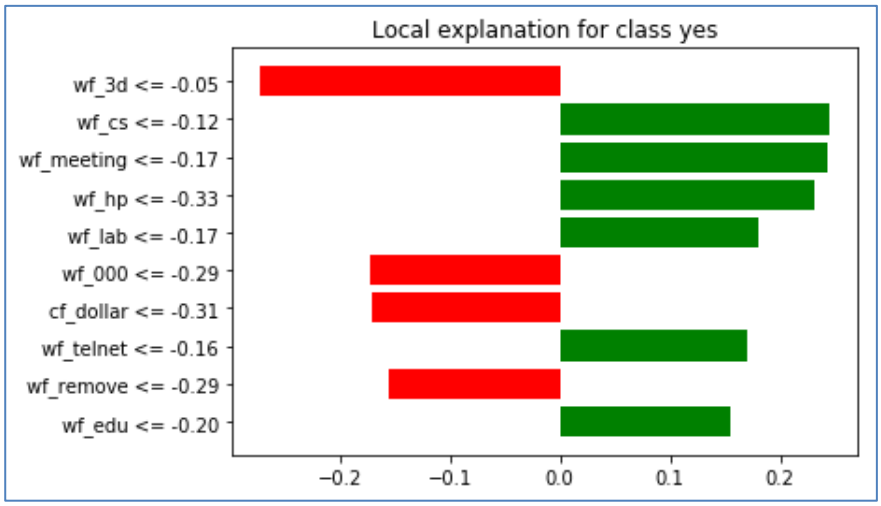


Parce que, pour cet individu n°1 à classer, elles ont toutes prises des valeurs négatives.



De son côté, ''lime'' propose l'interprétation suivante :

```
#calcul influence avec lime
explr1 = explainer.explain_instance(ZC[1,:],lr.predict_proba,num_features=10)
explr1.as_pyplot_figure()
```



De nouveau, 'wf_3d' est mis en avant de manière assez étrange. Sinon, pour ce qui est des autres variables, les résultats sont assez cohérents avec les contributions au LOGIT.

5 Conclusion

Le package ''lime'' répond à une question cruciale en pratique : sur la base de quelles informations (valeurs prises par les variables) le modèle a-t-il attribué telle classe à tel individu ? Ses auteurs proposent une approche agnostique, applicable à tout type de classifieur, y compris les modèles boîtes noires, souvent puissants mais inutilisables dans certains domaines où l'interprétation est primordiale.

Dans ce tutoriel, nous nous sommes placés dans une situation relativement facile où l'on sait réaliser cette interprétation, lorsque nous utilisons les arbres de décision et les modèles linéaires. En confrontant les résultats, nous avons pu constater que ''lime'' produisait des résultats assez pertinents avec, quand-même, des incongruités dans certaines situations.

6 Références

M.T. Ribeiro, S. Singh, C. Guestrin, ''Local Interpretable Model-Agnostic Explanations (LIME) : An Introduction'', https://www.oreilly.com/learning/introduction-to-local-interpretable-model-agnostic-explanations-lime

''lime'' -- https://github.com/marcotcr/lime