1 Introduction

Graphique de dépendance partielle (partial dependence plot - PDP). Elaboration et interprétation. Etude sous R (package ''iml'') et Python (package ''scikit-learn'').

Récemment, j'avais étudié les outils agnostiques (applicables à tous types de classifieurs) pour mesurer l'importance des variables dans les modèles prédictifs (Février 2019). Toujours inspiré par l'excellent ouvrage de Christopher Molnar, ''Interpretable Machine Learning'', je m'essaie à l'étude de l'influence des variables cette fois-ci. L'objectif est de répondre à la question : de quelle manière la variable pèse sur la prédiction du modèle ? Pour schématiser, je dirais que l'importance traduit l'intensité de l'impact global de la variable. L'influence, elle, s'intéresse au sens et à la forme de la relation avec la cible, mais toujours à travers le modèle.

Dans ce tutoriel nous étudierons le graphique de dépendance partielle (''partial dependence plot'' en anglais, PDP) qui permet de caractériser, d'une certaine manière qu'on essaiera de délimiter, l'influence d'une variable dans un modèle. Nous travaillerons sous R dans un premier temps, je proposerai un programme pour le calculer, puis nous verrons si nos résultats concordent avec ceux du package ''iml'' pour R. Dans un deuxième temps, nous travaillerons sous Python, et nous explorerons cette fois-ci la procédure dédiée proposée par le package ''scikit-learn''.

2 Principe du graphique de dépendance partielle

Le graphique de dépendance partielle (PDP) montre l'effet marginal d'une variable sur la prédiction d'un classifieur. Il permet d'identifier à la fois le sens (positif ou négatif) et la forme (linéaire, non linéaire, par paliers) de la relation (''Interpretable Machine Learning'', section 5.1). En d'autres termes, il traduit l'impact d'une variable à différents stades des valeurs qu'elle peut prendre en moyennant l'influence de tous les autres descripteurs.

Le modèle ayant déjà été construit, on le calcule de la manière suivante pour une variable X_j d'un ensemble de données de taille ''n'' (qui peut être l'échantillon d'apprentissage) :

a. Définir une grille de M valeurs (v_m) également répartis entre min(X_j) et max(X_j)
b. Pour chaque valeur v_m
    1. Remplacer, dans la matrice des descripteurs X, les valeurs de X_j par v_m
    2. Appliquer le modèle sur cette matrice pour obtenir les probabilités d'affectation (π) à la classe cible
    3. Calculer les moyennes de ces probabilités ($\bar{\pi}_m$)
c. Les couples (v_m, $\bar{\pi}_m$) constituent les points du graphique de dépendance partielle.

Plutôt que la probabilité moyenne, une variante possible serait de moyenner les LOGIT

$$LOGIT = \ln \frac{\pi}{1-\pi}$$

Nous verrons pourquoi plus bas (section 4.1).

3 Importation et étude préalable des données

Nous utilisons les données ''Automobile Data Set'' du dépôt UCI – Machine Learning Repository. La variable cible ''risky'' est déduite de ''symboling'', cette dernière indique le degré de risque d'un véhicule relativement à son prix. Notre variable ''risky'' prend deux modalités {'no', 'yes'} (plus risquée par rapport à son prix ou non). Nous avons par la suite supprimé un certain nombre de variables de la base originelle (ex. make, symboling, fuel-type, etc.).

Pour corser l'affaire, nous avons ajouté la variable ''wb_bis'', fortement corrélée avec ''wheel_base'' qui s'avère être une des plus importantes dans les modèles prédictifs. Nous verrons si les dispositifs étudiés seront capables de distinguer leurs rôles.

Nous effectuons une étude préalable des données dans cette section, nous travaillons sous R.

3.1 Importation des données

Nous chargeons le fichier ''**autos_pdp.txt**'' et nous affichons la liste des variables.

```
#charger les données
setwd("...votre dossier ...")
autos <- read.table("Autos_pdp.txt",header=TRUE,sep="\t",dec=".")
str(autos)
```

```
'data.frame':205 obs. of  17 variables:
 $ risky            : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 1 ...
 $ wheel_base       : num  88.6 88.6 94.5 99.8 99.4 ...
 $ wb_bis           : num  89.3 90.3 96.7 100.8 100.2 ...
 $ normalized_losses: int  122 122 122 164 164 122 158 122 158 122 ...
 $ length           : num  169 169 171 177 177 ...
 $ width            : num  64.1 64.1 65.5 66.2 66.4 66.3 71.4 71.4 71.4 67.9 ...
 $ height           : num  48.8 48.8 52.4 54.3 54.3 53.1 55.7 55.7 55.9 52 ...
 $ curb_weight      : int  2548 2548 2823 2337 2824 2507 2844 2954 3086 3053 ...
 $ engine_size      : int  130 130 152 109 136 136 136 136 131 131 ...
 $ bore             : num  3.47 3.47 2.68 3.19 3.19 3.19 3.19 3.19 3.13 3.13 ...
 $ stroke           : num  2.68 2.68 3.47 3.4 3.4 3.4 3.4 3.4 3.4 3.4 ...
 $ compression_ratio: num  9 9 9 10 8 8.5 8.5 8.5 8.3 7 ...
 $ horsepower       : num  111 111 154 102 115 110 110 110 140 160 ...
 $ peak_rpm         : num  5000 5000 5000 5500 5500 5500 5500 5500 5500 5500 ...
 $ city_mpg         : int  21 21 19 24 18 19 19 19 17 16 ...
 $ highway_mpg      : int  27 27 26 30 22 25 25 25 20 22 ...
 $ price            : num  13495 16500 16500 13950 17450 ...
```

A l'exception de la cible ''risky'', toutes les variables sont quantitatives.

3.2 Etude des corrélations

Nous avons créé artificiellement la liaison entre ''wheel_base'' (une vraie variable de la base initiale) et ''wb_bis'' (une variable artificielle créée de toutes pièces), mais il y a peut-être d'autres corrélations dans l'ensemble de données. Pour nous en rendre compte, nous souhaitons calculer les corrélations croisées entre les variables. Pour mieux organiser les résultats, nous les trions d'abord selon leur position sur le premier axe factoriel d'une ACP (analyse en composantes principales).

Nous réalisons donc tout d'abord l'ACP avec l'outil princomp() de R…

```
#ACP
acp <- princomp(autos[-1],cor=TRUE,scores=TRUE)
print(acp)
```

```
Call:
princomp(x = autos[-1], cor = TRUE, scores = TRUE)

Standard deviations:
    Comp.1     Comp.2     Comp.3     Comp.4     Comp.5     Comp.6     Comp.7
2.83419327 1.63711073 1.13129598 0.97523053 0.90787409 0.76767858 0.66773872
    Comp.8     Comp.9    Comp.10    Comp.11    Comp.12    Comp.13    Comp.14
0.56989530 0.55907909 0.42888003 0.35815784 0.31703911 0.26522870 0.22679357
   Comp.15    Comp.16
0.13853208 0.07628436

 16  variables and  205 observations.
```

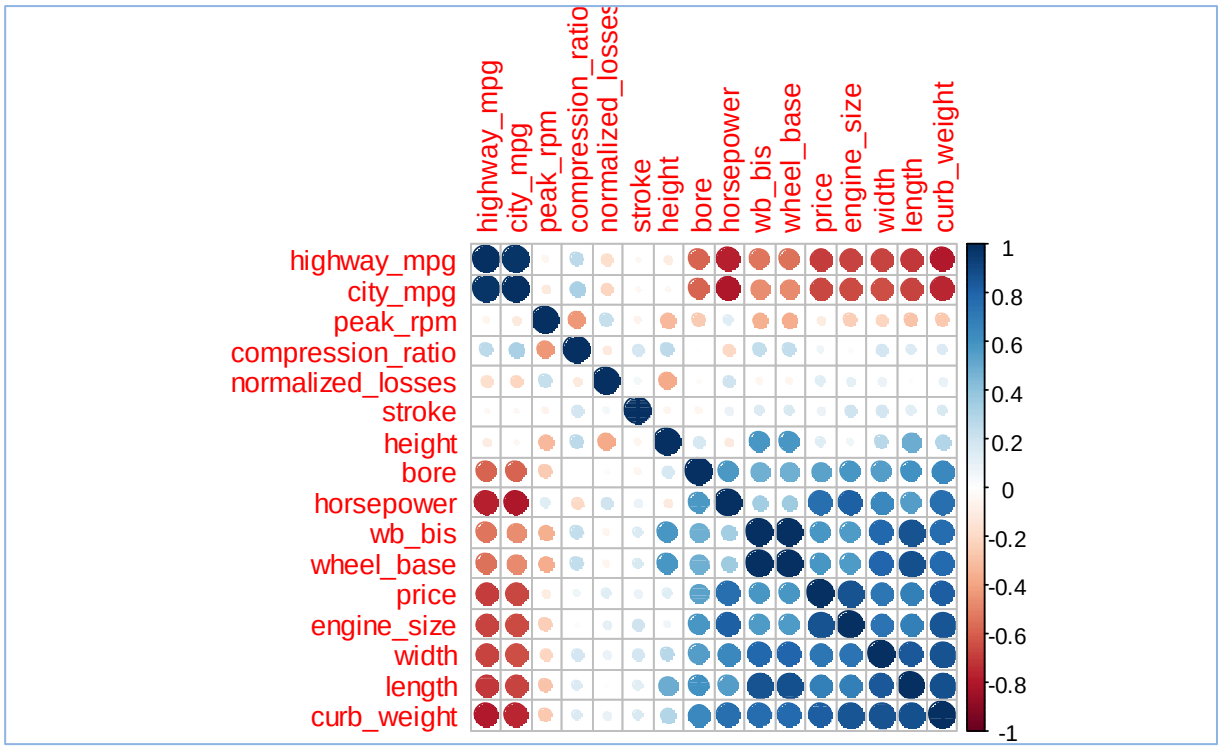
… et nous créons un index traduisant leurs coordonnées sur la première composante ($loadings est proportionnelle aux corrélations des variables avec les facteurs).

```
#index des variables selon le premier axe
idx1 <- order(acp$loadings[,1])
print(colnames(autos[-1])[idx1])
```

```
[1] "highway_mpg"       "city_mpg"          "peak_rpm"
 [4] "compression_ratio" "normalized_losses" "stroke"
 [7] "height"            "bore"              "horsepower"
[10] "wb_bis"            "wheel_base"        "price"
[13] "engine_size"       "width"             "length"
[16] "curb_weight"
```

Nous calculons les corrélations croisées cor(), représentées dans un graphique ''corrplot''.

```
#corrélation entre les variables
library(corrplot)
corrplot(cor(autos[-1][idx1]))
```



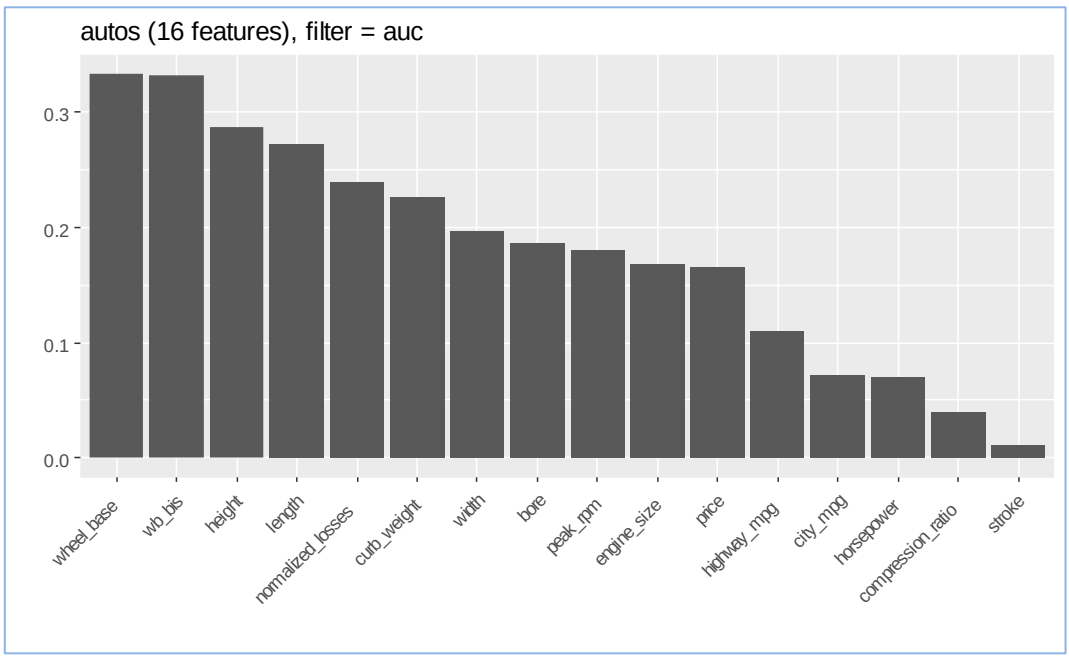
Nous observons les corrélation positives en bleu (concomitances entre les mesures de consommation [mpg], entre les mesures de dimensions [width, length, etc.], etc.) et les négatives en rouge (opposition entre consommations et dimensions par exemple). Avoir réorganisé les colonnes selon leur position sur la première composante de l'ACP permet de mieux discerner les ''blocs'' de variables.

3.3 Etude des liens entre la cible et les descripteurs

L'étape suivante consiste à mesurer le lien direct entre la cible et les descripteurs. Cette approche est souvent utilisée pour les filtrer avant la phase de modélisation, avec pour hypothèse que les variables ainsi mises en évidence seront les plus efficaces quel que soit l'algorithme de machine learning exploité par la suite. Le présupposé est hardi mais, quoiqu'il en soit, la rapidité de la procédure permet souvent de réduire drastiquement la dimensionalité à moindre frais. Nous utilisons le package ''mlr'' présenté dans un récent tutoriel (''R – Machine Learning avec mlr'', avril 2019). Le critère ''AUC'' (''aire sous la courbe'' de la courbe ROC) servira de mesure de pertinence des variables.

Avec ''mlr'', nous définissons tout d'abord la tâche à réaliser (makeClassifTask) en indiquant les données à traiter (data = autos), la variable cible (target = ''risky'') et la modalité cible (positive = ''yes''). Nous lançons ensuite les calculs (generateFilterValuesData) en précisant le critère de filtrage (method = ''auc''). Nous demandons un affichage graphique des résultats (plotFilterValues).

```
#librairie mlr
library(mlr)

#tâche pour le problème autos
autos.task <- mlr::makeClassifTask(data=autos,target="risky",positive="yes")

#filtrage selon l'AUC
filter.auc <- mlr::generateFilterValuesData(task=autos.task,method="auc")

#affichage
mlr::plotFilterValues(filter.auc)
```



La variable explicative la plus pertinente est ''wheel_base'', puis vient ''wb_bis'' qui, nous le savons, est corrélée avec la précédente, mais faussement liée avec la cible. La procédure tombe à pieds joints dans le piège. Un des enjeux de ce travail est justement de voir si nous serons capables de le déjouer par la suite. Les suivantes sont ''height'' et ''length'' qui sont des variables réelles, mais néanmoins également corrélées avec ''wheel_base'' (section 3.2).

4 PDP sous R

4.1 Modélisation avec la régression logistique (classifieur linéaire)

Nous utilisons la régression logistique, un classifieur linéaire, dans une première phase parce que la solution est évidente. La PDP doit correspondre une droite si nous utilisons le LOGIT pour apprécier l'influence de X_j dans le modèle. En effet, si l'on pose π le score [P(Y = + / X)] fourni par le modèle (''Y'' représente la variable cible, ''+'' la modalité cible), l'équation LOGIT s'écrit :

$$\text{LOGIT} = \ln\frac{\pi}{1-\pi} = a_0 + a_1x_1 + \cdots + a_jx_j + \cdots + a_px_p$$

Une augmentation d'une unité de X_j entraîne une augmentation a_j du LOGIT. La pente sera d'autant plus forte que $|a_j|$ est élevé, elle sera croissante si ($a_j > 0$), décroissante sinon.

Si l'on s'intéresse au score (π) lui-même cette fois-ci, nous devrions obtenir une courbe sigmoïde (croissante ou décroissante en fonction du signe de a_j) puisque la fonction de transfert reliant π au LOGIT s'écrit :

$$\pi = \frac{1}{1+e^{-LOGIT}}$$

Tout d'abord, nous construisons le modèle (method=''glm''). Nous passons par la librairie ''caret'' parce que l'outil ''iml'' que nous utiliserons plus loin (section 4.4) reconnaît uniquement ses objets et ceux du package ''mlr''. Nous estimons les performances en validation croisée [trControl = trainControl(method='cv', number=10)]

```
#librairie mlr
library(caret)

#modélisation et mesure des performances en validation croisée
m_reg <- caret::train(risky ~ ., data = autos, method="glm",trControl=trainControl(method='cv',number=10))
print(m_reg)
```

```
Generalized Linear Model

205 samples
 16 predictor
  2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 185, 185, 184, 185, 184, 185, ...
Resampling results:

  Accuracy   Kappa
  0.7857143  0.567714
```

Le taux de reconnaissance est de **78.57%**.

Avec le détail des résultats …

```
#coefficients du modèle élaboré sur la totalité des données
print(summary(m_reg$finalModel))
```

```
Call:
NULL

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.43480  -0.30241   0.02523   0.45274   2.13493

Coefficients:
```

```
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)       -1.208e+01  1.944e+01  -0.621 0.534327
wheel_base        -9.952e-01  4.093e-01  -2.432 0.015035 *
wb_bis             1.913e-01  3.358e-01   0.570 0.568997
normalized_losses  5.196e-02  1.145e-02   4.537  5.7e-06 ***
length            -1.178e-01  6.589e-02  -1.788 0.073757 .
width              1.515e+00  3.908e-01   3.877 0.000106 ***
height             2.791e-01  1.790e-01   1.559 0.118915
curb_weight       -3.497e-03  2.280e-03  -1.534 0.125109
engine_size       -8.889e-03  1.785e-02  -0.498 0.618461
bore               2.569e-01  1.249e+00   0.206 0.836962
stroke             9.147e-01  7.763e-01   1.178 0.238671
compression_ratio  7.156e-03  9.802e-02   0.073 0.941800
horsepower         1.603e-02  2.052e-02   0.781 0.434674
peak_rpm          -1.043e-03  7.757e-04  -1.345 0.178742
city_mpg          -1.797e-01  2.009e-01  -0.894 0.371178
highway_mpg        1.049e-01  1.711e-01   0.613 0.539980
price              1.415e-04  8.253e-05   1.715 0.086422 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 282.04  on 204  degrees of freedom
Residual deviance: 132.69  on 188  degrees of freedom
AIC: 166.69

Number of Fisher Scoring iterations: 7
```

… nous constatons que le coefficient de ‘’wheel_base’’ (-0.9952) est significatif à 5% (significativement différent de 0) et qu’il est de signe négatif. Celui de ‘’wb_bis’’ (0.1913) n’est pas significatif en revanche, et il est positif.

4.2 Un programme ad-hoc pour calculer le PDP

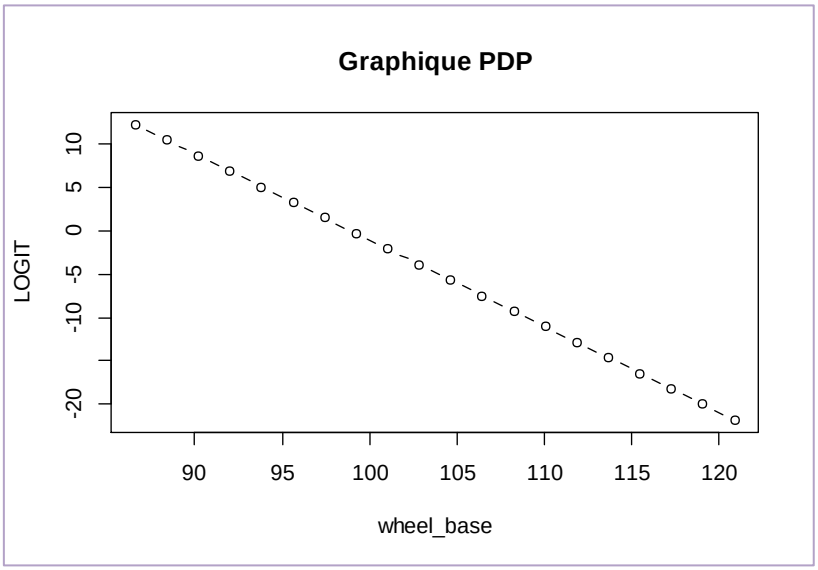
L’algorithme de PDP est relativement simple finalement (section 2). J’ai décidé d’écrire une petite fonction my.pdp() qui prend en entrée un modèle prédictif déjà entraîné, la matrice des descripteurs (X), le nom de la variable à traiter, le pas pour la génération de la séquence de valeurs à afficher en abscisse, et enfin l’utilisateur a la possibilité de présenter en ordonnée du graphique soit le LOGIT (use.logit = TRUE), soit le se score π (pour la probabilité P(risky = ‘yes’ / X) fourni par le modèle.

```
#fonction pour le graphique PDP
my.pdp <- function(modele,X,variable,pas=20,use.logit=TRUE){
  #extrêmes
  x_min <- min(X[,variable])
  x_max <- max(X[,variable])
  etendue <- (x_max - x_min)
  #séquence de valeurs
  valeurs <- seq(from=x_min,to=x_max,length.out=pas)
  #vecteur de resultat
  res <- c()
  #pour chaque valeur
  for (v in valeurs){
```

```
    #copier les données
    XPrim <- X
    #remplacer
    XPrim[,variable] <- rep(v,nrow(X))
    #prediction
    p <- predict(modele,newdata=XPrim,type='prob')[,'yes']
    #si logit demandé, transformer p en LOGIT
    if (use.logit == TRUE){
      p <- log(p/(1.0-p))
    }
    #récupération de la moyenne des probas ou des logit
    mp <- mean(p)
    #stocker le résultat
    res <- c(res,mp)
  }
  #titre du graphique
  titre.ord <- "P(yes/X)"
  #modifier le titre du graphique si logit demandé
  if (use.logit == TRUE){
    titre.ord <- "LOGIT"
  }
  #graphique
  plot(valeurs,res,type='b',xlab=variable,ylab=titre.ord,main="Graphique PDP")
  #liste des valeurs et moyenne des probas
  return(list(values=valeurs,probas=res))
}
```

Appliquons la fonction à ''wheel_base'' en demandant le LOGIT dans un premier temps.

```
#calcul sur la variable 'wheel_base' -- LOGIT
my.pdp(m_reg,autos[-1],'wheel_base',use.logit=TRUE)
```



La courbe est bien décroissante, et nous avons une droite ! Ah, c'est beau la théorie quand elle est confirmée par les données.

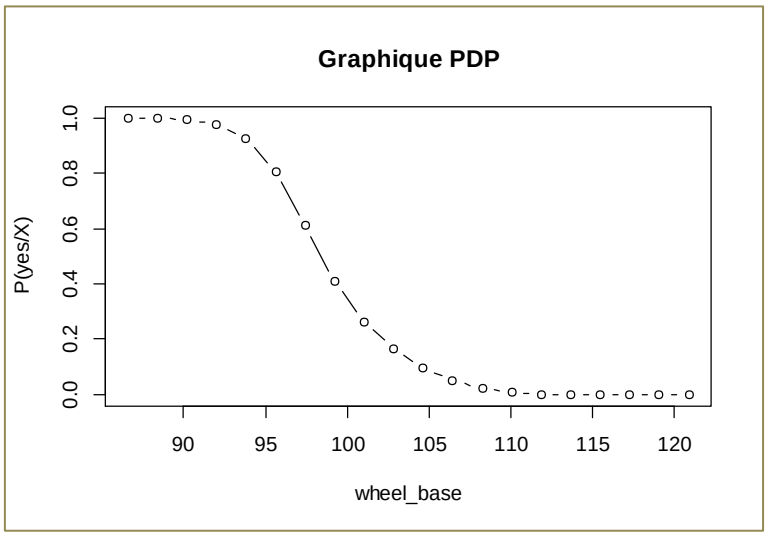
Dans la console apparaissent la série de valeurs qui ont permis de construire le graphique.

```
$values
 [1]  86.60000  88.40526  90.21053  92.01579  93.82105  95.62632
 [7]  97.43158  99.23684 101.04211 102.84737 104.65263 106.45789
[13] 108.26316 110.06842 111.87368 113.67895 115.48421 117.28947
[19] 119.09474 120.90000

$probas
 [1]  12.2789517  10.4823185   8.6856854   6.8890523   5.0924191
 [6]   3.2957860   1.4991529  -0.2974803  -2.0941134  -3.8907465
[11]  -5.6873797  -7.4840128  -9.2806459 -11.0772791 -12.8739122
[16] -14.6705453 -16.4671785 -18.2638116 -20.0604447 -21.8570779
```

Lorsque nous demandons les probabilités d'affectation en ordonnée du graphique PDP…

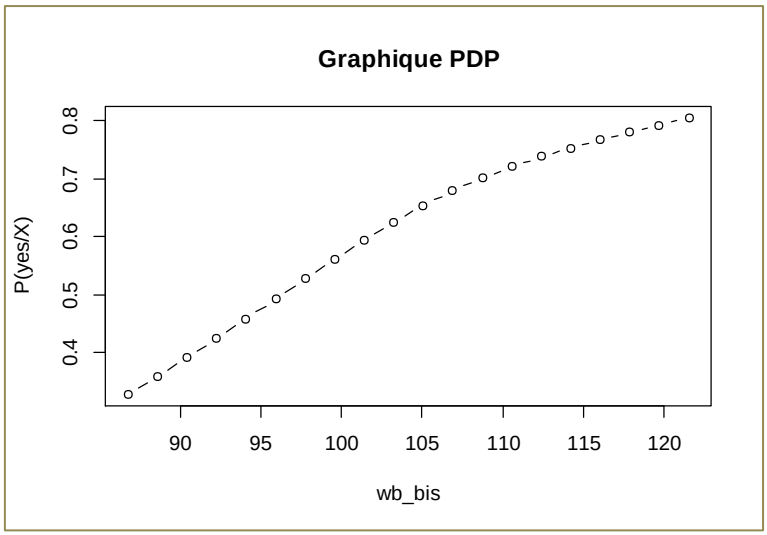
```
#calcul sur la variable 'wheel_base' -- probas
my.pdp(m_reg,autos[-1],'wheel_base',use.logit=FALSE)
```

… la forme sigmoïde apparait clairement, avec des valeurs en ordonnée allant de 1 à 0.



Si l'on s'intéresse à la variable 'wb_bis', toujours pour les probabilités d'affectation :

```
#pour 'wb_bis'
my.pdp(m_reg,autos[-1],'wb_bis',use.logit=FALSE)
```



La courbe est croissante ($a_{wb\_bis} > 0$). Mais comme le coefficient est relativement faible (en valeur absolue), la forme sigmoïde est moins marquée, et l'étendue en ordonnée est réduite [0.328, 0.805].

Conclusion : Le graphique de dépendance partielle traduit fidèlement le rôle joué par ces variables dans le modèle prédictif issu de la régression logistique. L'approche tient la route pour apprécier l'influence des variables, *au moins dans ce cas*.

4.3 Modélisation avec un GBM (gradient boosting machine) (non-linéaire)

Le graphique PDP sait reporter des évidences pour la régression logistique qui est un classifieur linéaire. Le contraire eût été très dommage. Voyons maintenant ce qu'il en est pour un modèle non-linéaire, pour lequel nous ne disposons pas de la solution pour caler notre lecture des résultats.

Nous avons choisi un gradient boosting (GBM – gradient boosting machine). Nous l'instancions (method = ''gbm'') et nous mesurons ses performances en validation croisée.

```
#gradient boosting machine
m_gbm <- caret::train(risky ~ ., data = autos, method="gbm",trControl=trainControl(method='cv',number=10))
print(m_gbm)
```

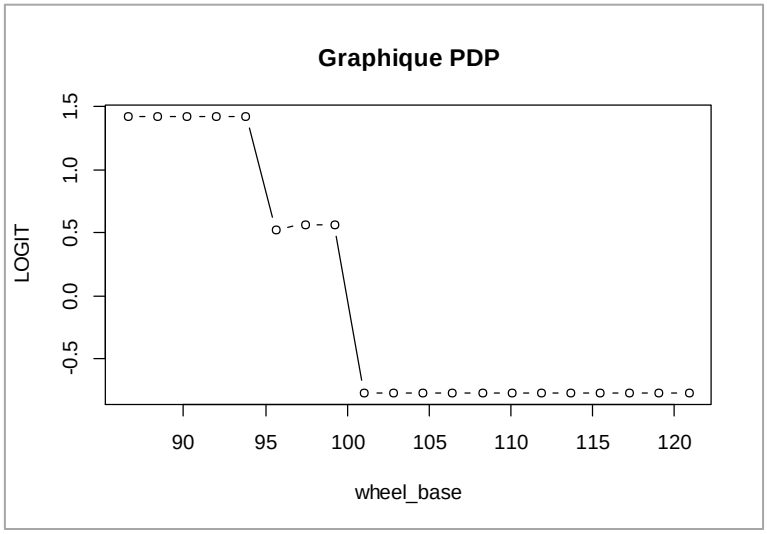
```
Stochastic Gradient Boosting

205 samples
 16 predictor
  2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 185, 185, 185, 184, 185, 183, ...
Resampling results across tuning parameters:

  interaction.depth  n.trees  Accuracy   Kappa
  1                   50      0.8387229  0.6754987
  1                  100      0.8577922  0.7164837
  1                  150      0.8975541  0.7935284
```

```
  2                    50      0.8875541  0.7737206
  2                   100      0.9173160  0.8327837
  2                   150      0.9127922  0.8236561
  3                    50      0.8930303  0.7840741
  3                   100      0.9127922  0.8236205
  3                   150      0.9175541  0.8334797

Tuning parameter 'shrinkage' was held constant at a value
 of 0.1
Tuning parameter 'n.minobsinnode' was held
 constant at a value of 10
Accuracy was used to select the optimal model using
 the largest value.
The final values used for the model were n.trees =
 150, interaction.depth = 3, shrinkage = 0.1
 and n.minobsinnode = 10.
```

L'outil optimise automatiquement les paramètres d'apprentissage, le meilleur modèle (interaction.depth = 3, n.trees = 150) propose un taux de reconnaissance égal à **91.75%**, bien supérieur à la régression logistique.

Nous produisons le graphique de dépendance partielle de ''wheel_base'', sur le LOGIT tout d'abord :

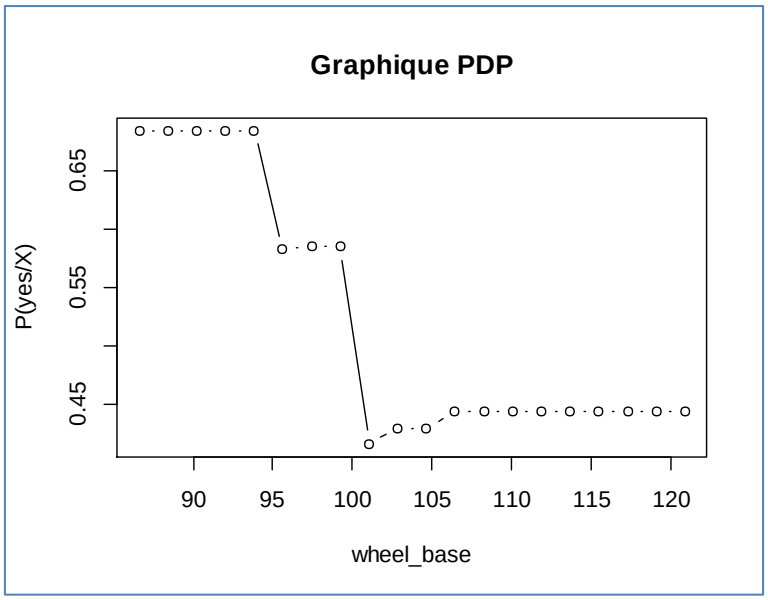
```
#PDP pour wheel_base -- LOGIT
my.pdp(m_gbm,autos[-1],'wheel_base',pas=20,use.logit=TRUE)
```



Dans le modèle, le LOGIT est toujours décroissant en fonction de ''wheel_base''. Mais la relation est clairement non-linéaire cette fois-ci. Il y a deux gaps clairement de l'influence, vers (wheel_base = 95), puis (wheel_base = 100).

Lorsqu'on passe sur le graphique des probabilités d'affectation, la différenciation par rapport à celui de la régression logistique (forme sigmoïde) est moins frappante.

```
#PDP pour wheel_base -- PROBAS
my.pdp(m_gbm,autos[-1],'wheel_base',pas=20,use.logit=FALSE)
```



On remarque néanmoins, avec les plages de valeurs prises en ordonnée, que l'influence de 'wheel_base' est moindre dans le modèle GBM.

4.4 Utilisation du package ''iml''

Le package ''iml'' propose plusieurs outils pour l'interprétation en machine learning. J'avais étudié précédemment la mesure agnostique d'importance des variables dans les modèles (''Importance des variables dans les modèles'', février 2019). L'outil FeatureEffect permet de produire, relativement simplement, le graphique de dépendance partielle.

Après avoir importé le package, nous créons un objet Predictor qui permet de spécifier les données à traiter (data = autos), la variable cible (y = risky) et le modèle à étudier (m_gbm).

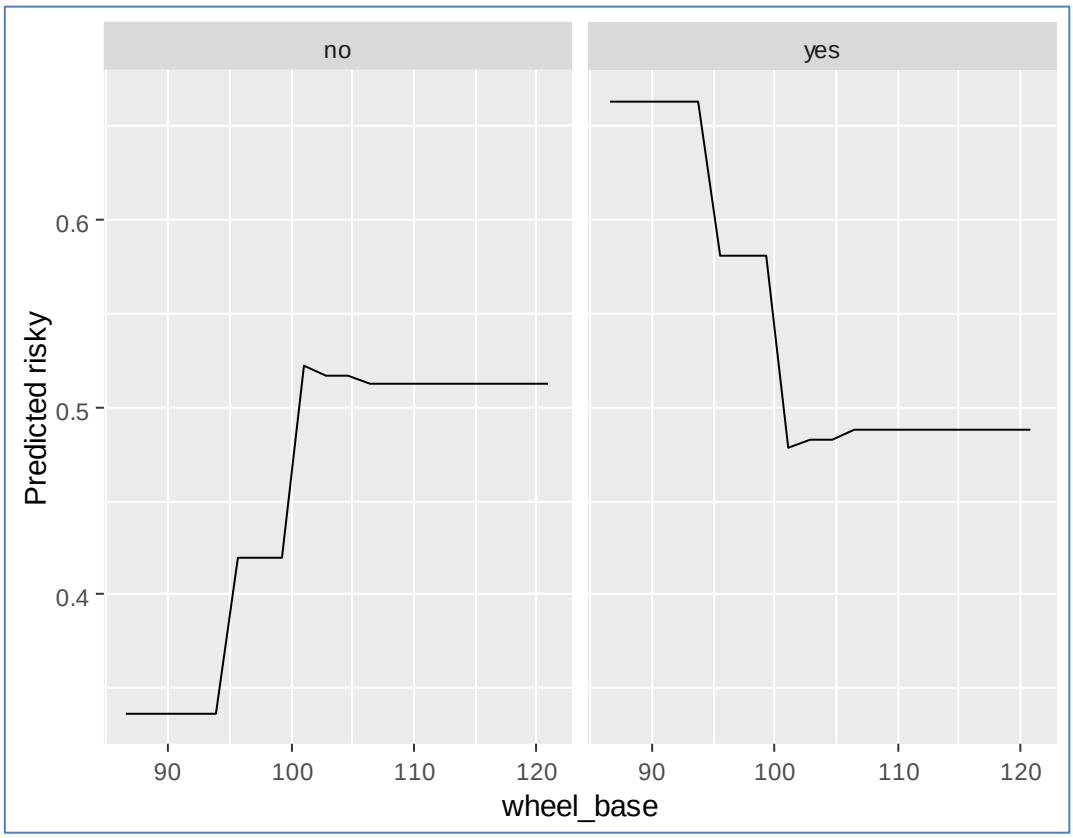
```
#package iml
library(iml)

#objet predictor pour le modèle gbm
obj_gbm <- iml::Predictor$new(m_gbm,data=autos,y="risky")
print(obj_gbm)
```

```
Prediction task: classification
Classes:
```

Nous pouvons calculer les effets de la variable ''wheel_base'' (feature = 'wheel_base'), nous optons pour le graphique de dépendance partielle (method = ''pdp''). D'autres graphiques d'interprétation sont possibles (''ale'', accumulated local effect ; ''ice'', individual conditionnal expectations -- https://rdrr.io/cran/iml/man/FeatureEffect.html).

```
#calculer l'effet – partial dependence plot
pdp <- iml::FeatureEffect$new(obj_gbm,method="pdp",feature='wheel_base')
```

```
plot(pdp)
```



C'est la modalité ''yes'' (à droite) qui nous intéresse. Le graphique correspond à celui que nous avions produit avec notre fonction ad hoc. Il n'est pas possible en revanche de produire le graphique des LOGIT avec cet outil.

Le champ ($result) fournit le détail des résultats.

```
#détail des valeurs
print(pdp$results)
```

```
  wheel_base .class    .y.hat .type
1   86.60000     no 0.3365854   pdp
2   88.40526     no 0.3365854   pdp
3   90.21053     no 0.3365854   pdp
4   92.01579     no 0.3365854   pdp
5   93.82105     no 0.3365854   pdp
6   86.60000    yes 0.6634146   pdp
7   88.40526    yes 0.6634146   pdp
8   90.21053    yes 0.6634146   pdp
9   92.01579    yes 0.6634146   pdp
10  93.82105    yes 0.6634146   pdp
11  95.62632     no 0.4195122   pdp
12  97.43158     no 0.4195122   pdp
13  99.23684     no 0.4195122   pdp
14 101.04211     no 0.5219512   pdp
15 102.84737     no 0.5170732   pdp
16  95.62632    yes 0.5804878   pdp
17  97.43158    yes 0.5804878   pdp
18  99.23684    yes 0.5804878   pdp
```

```
19  101.04211    yes 0.4780488   pdp
20  102.84737    yes 0.4829268   pdp
21  104.65263     no 0.5170732   pdp
22  106.45789     no 0.5121951   pdp
23  108.26316     no 0.5121951   pdp
24  110.06842     no 0.5121951   pdp
25  111.87368     no 0.5121951   pdp
26  104.65263    yes 0.4829268   pdp
27  106.45789    yes 0.4878049   pdp
28  108.26316    yes 0.4878049   pdp
29  110.06842    yes 0.4878049   pdp
30  111.87368    yes 0.4878049   pdp
31  113.67895     no 0.5121951   pdp
32  115.48421     no 0.5121951   pdp
33  117.28947     no 0.5121951   pdp
34  119.09474     no 0.5121951   pdp
35  120.90000     no 0.5121951   pdp
36  113.67895    yes 0.4878049   pdp
37  115.48421    yes 0.4878049   pdp
38  117.28947    yes 0.4878049   pdp
39  119.09474    yes 0.4878049   pdp
40  120.90000    yes 0.4878049   pdp
```

5 PDP sous Python avec ''scikit-learn''

Dans cette section, nous réitérons sous Python la construction du graphique PDP via les outils du package ''sckit-learn''. L'idée est de mettre les résultats en parallèle avec ceux de R.

5.1 Importation des données

L'importation des données est réalisée avec la librairie ''pandas''.

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

#charger les données
import pandas
autos = pandas.read_csv("autos_pdp.txt",sep="\t",header=0)
print(autos.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 17 columns):
risky                205 non-null object
wheel_base           205 non-null float64
wb_bis               205 non-null float64
normalized_losses    205 non-null int64
length               205 non-null float64
width                205 non-null float64
height               205 non-null float64
curb_weight          205 non-null int64
engine_size          205 non-null int64
bore                 205 non-null float64
stroke               205 non-null float64
compression_ratio    205 non-null float64
horsepower           205 non-null float64
peak_rpm             205 non-null float64
```

```
city_mpg             205 non-null int64
highway_mpg          205 non-null int64
price                205 non-null float64
dtypes: float64(11), int64(5), object(1)
```

5.2 Gradient boosting avec ''scikit-learn''

Nous instancions un objet gradient boosting avec les paramètres par défaut, puis nous lançons l'apprentissage sur notre jeu de données.

```
#gradient boosting
from sklearn.ensemble import GradientBoostingClassifier
m_gbm = GradientBoostingClassifier()

#modélisation
m_gbm.fit(autos.iloc[:,1:],autos.iloc[:,0])
```

5.3 Importance des variables

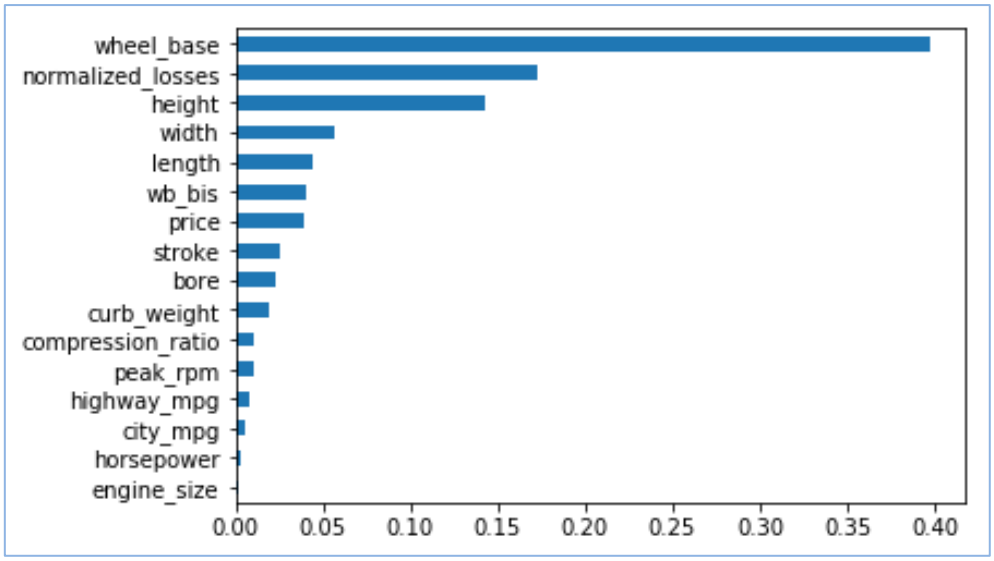
L'importance des variables mesure l'impact global de chaque descripteur dans le modèle.

```
#importance des variables
print(pandas.DataFrame({'variable':autos.columns[1:],'coef':m_gbm.feature_importances_}))
```

```
             variable      coef
0          wheel_base  0.397768
1              wb_bis  0.040424
2   normalized_losses  0.172825
3              length  0.044172
4               width  0.056606
5              height  0.142380
6         curb_weight  0.019245
7         engine_size  0.002321
8                bore  0.023309
9              stroke  0.024926
10  compression_ratio  0.010871
11         horsepower  0.002407
12           peak_rpm  0.010078
13           city_mpg  0.005384
14        highway_mpg  0.008406
15              price  0.038878
```

Réorganiser les variables selon leur importance dans un graphique permet de mieux situer leur rôle.

```
#position des variables selon l'importance
import numpy
pos = numpy.argsort(m_gbm.feature_importances_)

#affichage graphique
pandas.Series(m_gbm.feature_importances_[pos],index=autos.columns[1:][pos]).plot(kind='barh')
```

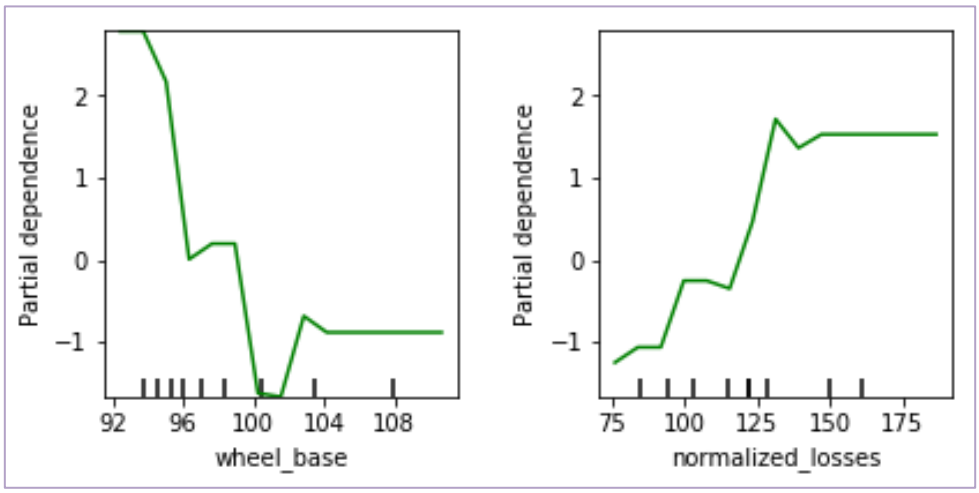


''wheel_base'' se démarque dans le modèle, ''normalized_losses'' est en seconde position. ''wb_bis'' est, à juste titre, relégué plus loin.

5.4 Graphique de dépendance partielle

La fonction plot_partial_dependence() produit le graphique de dépendance partielle pour un ensemble de variables, *pris individuellement*. Nous présentons les numéros de variables dans une liste (features = [0, 2]). Attention, la procédure fonctionne uniquement pour les instances de GradientBoosting (version 0.20.3 de scikit-learn). Elle n'est pas opérationnelle pour les autres. Je me demande pourquoi parce que, à la lecture de la description de l'algorithme (section 2), l'approche n'est absolument pas dépendante de la méthode de modélisation.

```
#classe de calcul pour partial dependence plot
from sklearn.ensemble.partial_dependence import plot_partial_dependence
#graphique pour wheel_base (n°0) et normalized_loss (n°2)
plot_partial_dependence(m_gbm,autos.iloc[:,1:],feature_names=autos.columns[1:],features=[0,2],grid_resolution=15)
```

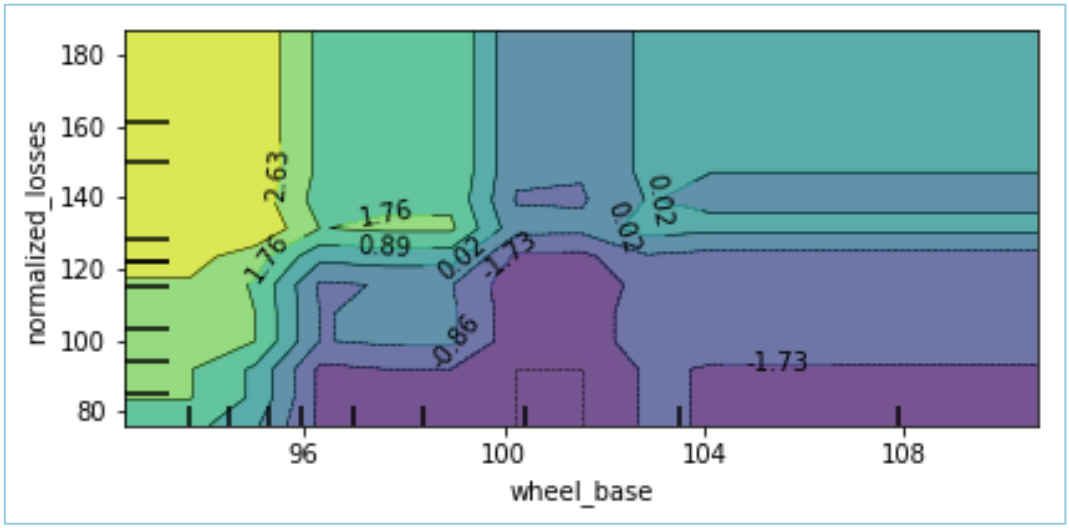


En ordonnée, nous observons le LOGIT, mais avec une certaine normalisation pour que leur moyenne soit nulle (d'après ce que j'ai pu trouver en furetant sur le net, la documentation n'est pas très loquace à ce sujet). Le résultat pour ''wheel_base'' est conforme à ce que nous avions pu observer sous R à l'aide de notre fonction ad hoc (section 4.3). En ce qui concerne ''normalized_losses'', elle pèse positivement dans le modèle, la relation n'est pas linéaire non-plus.

5.5 Graphique pour l'effet conjoint de deux variables

Fonctionnalité intéressante, plot_partial_dependence() peut caractériser le rôle conjoint des variables (2 variables maximum), notamment pour identifier les possibles interactions. Il faut les présenter sous forme de tuple dans ce cas (features = [(0,2)]).

```
#graphique pour l'effet conjoint de wheel_base et normalized_losses
plot_partial_dependence(m_gbm,autos.iloc[:,1:],feature_names=autos.columns[1:],features=[(0,2)],grid_resolution=15)
```



Nous avons un ''heatmap''. Le surcroît de chances (LOGIT) d'être (risky = 'yes') apparaît dans la zone coloriée en vert-jaune, dans la partie nord-ouest du graphique c.-à-d. la conjonction de valeurs élevées de ''normalized_losses'' et faibles de ''wheel_base''.

Au-delà de 2 variables, la représentation graphique n'est plus possible.

6 Conclusion

Comprendre ce qu'il se passe dans nos modèles est toujours important, même lorsque la performance est la visée première de l'étude. En effet, savoir appréhender le mécanisme prédictif permet d'identifier les raisons pour lesquels ils seraient faillibles dans certaines situations, et nous donne des pistes pour en améliorer le comportement.

Je ne l'ai pas abordé dans ce tutoriel, mais le principe de la dépendance partielle peut s'appliquer aux prédicteurs qualitatifs. Nous le calculons toujours avec le même principe : nous affectons tour à tour à la variable X_j la valeur d'une des modalités, puis nous calculons la moyenne des scores π. Le graphique se présente alors comme un ''barplot'' où, pour chaque modalité de X_j, nous observons la probabilité moyenne $\bar{\pi}$ d'affecter à la classe cible (ou la moyenne du LOGIT).

Enfin, le PDP n'est pas la panacée bien sûr. Il ne sait pas appréhender les corrélations entre les variables en définissant la grille des valeurs sur une variable indépendamment des autres. Pour reprendre un des exemples que l'on retrouve souvent sur le net, si l'on a dans la base le poids et la taille des personnes, avec ce dispositif, nous pouvons très bien créer une situation artificielle d'une personne pesant 50 kg et mesurant 2m10. La probabilité d'affectation calculée dans cette configuration irréaliste n'a pas vraiment de sens.

7 Références

C. Molnar, "Interpretable Machine Learning - A Guide for Making Black Box Models Explainable", 2019, Chapter 5 : Model-Agnostics Methods, Section 5.1 : Partial Dependence Plot (PDP). https://christophm.github.io/interpretable-ml-book/pdp.html (version 03 avril 2019).

R : ''iml : Interpretable Machine Learning'' - https://cran.r-project.org/package=iml

Python : ''scikit-learn'' - https://scikit-learn.org/stable/