1 Introduction

Instanciation et interprétation d'un auto-encodeur, utilisation du tandem de librairies Tensorflow / Keras sous Python.

Ce tutoriel fait suite au support de cours consacré aux auto-encodeurs (''Deep learning : les Auto-encodeurs'', novembre 2019). Nous mettons en œuvre la technique sur un jeu de données jouet (des automobiles pour ne pas changer).

Il y a différentes manières de considérer les auto-encodeurs. Dans notre cas, nous adoptons le point de vue de la description des données dans un espace de dimension réduite. Comme une alternative à l'ACP (analyse en composantes principales) en somme. L'objectif est de cerner au mieux les attentes que l'on pourrait avoir par rapport aux résultats qu'elle fournit dans ce contexte, notamment en matière de qualité de reconstitution des données.

Un repère simple pour suivre les étapes décrites dans ce document serait de se référer à un précédent tutoriel consacré à l'ACP sous Python (''ACP sous Python'', juin 2018).

2 Données

Le fichier Excel ''**cars_autoencoder.xlsx**'' est formé de 3 feuilles :

- VAR.IND.ACTIFS contient les 28 individus actifs décrits par 6 variables (puissance, cylindrée, vitesse, longueur, hauteur, poids).
- IND.ILLUS recense les individus illustratifs. Nous cherchons à positionner quelques modèles Peugeot (407, 307 CC, 1007 et 607) par rapport à la cartographie définie par les autres véhicules.
- VAR.ILLUS contient les variables illustratives (co2, prix, carburant) qui permettront de mieux situer la portée des résultats.

3 Librairies Tensorflow / Keras

Nous avons étudié à plusieurs reprises ces librairies dans des anciens tutoriels (voir http://tutoriels-data-mining.blogspot.com/search?q=keras). Pour rappel, en simplifiant un peu, Keras permet d'accéder relativement facilement aux fonctionnalités de Tensorflow. J'utilise la dernière (au 30.11.2019) version d'Anaconda avec **Python 3.7.4** durant cette expérimentation. Je n'ai rencontré aucune difficulté pour installer Tensorflow tout d'abord (https://www.tensorflow.org/install), puis Keras par la suite (https://anaconda.org/conda-forge/keras).

4 Importation et préparation des données

4.1 Importation des données actives

Nous chargeons le tableau des d'individus actifs. Il est situé dans la première feuille (sheet_name = 0) de notre fichier Excel.

```
#modif. du dossier de travail
import os
os.chdir("… votre dossier de travail …")

#librairie pandas
import pandas

#chargement de la première feuille de données - n°0
X = pandas.read_excel("cars_autoencoder.xlsx",header=0,index_col=0,sheet_name=0)

#liste des variables
print(X.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 28 entries, CITRONC5     to AUDIA8
Data columns (total 6 columns):
puissance    28 non-null int64
cylindree    28 non-null int64
vitesse      28 non-null int64
longueur     28 non-null int64
hauteur      28 non-null int64
poids        28 non-null int64
dtypes: int64(6)
memory usage: 1.5+ KB
```

La première colonne (index_col = 0) correspond à des étiquettes, les modèles des véhicules en l'occurrence. La première ligne décrit les noms des variables (header = 0). Affichons les données pour en avoir le cœur net.

```
#affichage
print(X)
```

```
             puissance  cylindree  vitesse  longueur  hauteur  poids
Modele
CITRONC5           210       2946      230       475      148   1589
LAGUNA             165       1998      218       458      143   1320
CITRONC4           138       1997      207       426      146   1381
CLIO               100       1461      185       382      142    980
CITRONC2            61       1124      158       367      147    932
MODUS              113       1598      188       380      159   1170
MEGANECC           165       1998      225       436      141   1415
TWINGO              60       1149      151       344      143    840
MONDEO             145       1999      215       474      143   1378
VECTRA             150       1910      217       460      146   1428
PASSAT             150       1781      221       471      147   1360
MERC_A             140       1991      201       384      160   1340
```

```
ALFA 156      250   3179   250   443   141   1410
AUDIA3        102   1595   185   421   143   1205
GOLF           75   1968   163   421   149   1217
MUSA          100   1910   179   399   169   1275
FIESTA         68   1399   164   392   144   1138
CORSA          70   1248   165   384   144   1035
PANDA          54   1108   150   354   154    860
AVENSIS       115   1995   195   463   148   1400
CHRYS300      340   5654   250   502   148   1835
MAZDARX8      231   1308   235   443   134   1390
PTCRUISER     223   2429   200   429   154   1595
YARIS          65    998   155   364   150    880
VELSATIS      150   2188   200   486   158   1735
BMW530        231   2979   250   485   147   1495
MERC_E        204   3222   243   482   146   1735
AUDIA8        280   3697   250   506   145   1770
```

Nous récupérons à toutes fins utiles le nombre d'observations (n) et de variables (p).

```
#dimension
print(X.shape) #(28, 6)

#nombre d'observations
n = X.shape[0]

#nombre de variables
p = X.shape[1]
```

4.2 Standardisation des variables

Nous souhaitons centrer et surtout réduire les variables pour éviter les disparités d'échelle. Nous utilisons la librairie ''scikit-learn''. ''**Z**'' correspond aux données transformées. L'objet ''**std**'' représente l'opérateur qui permettra d'appliquer le même processus sur d'autres observations (individus illustratifs), ou la transformation inverse lorsque nous aurons à revenir dans l'espace initial (données restituées par le réseau).

```
#outil centrage réduction
from sklearn.preprocessing import StandardScaler

#instanciation
std = StandardScaler()

#transformation
Z = std.fit_transform(X)
print(Z)
```

```
[[ 0.84249298  0.87217654  0.86954849  0.96525293  0.02554647  0.9621173 ]
 [ 0.2271068  -0.106354    0.49971521  0.603188   -0.68975475 -0.01928404]
...
 [ 1.12967319  0.90623931  1.4859373   1.17823229 -0.11751377  0.61917408]
 [ 0.76044149  1.15706517  1.27020122  1.11433848 -0.26057402  1.49477379]
 [ 1.79976036  1.64736265  1.4859373   1.62548897 -0.40363426  1.62246541]]
```

Nous calculons l'étendue des valeurs pour chaque variable après transformation.

```
#min pour chaque variable
import numpy
numpy.min(Z,axis=0)
```

```
array([-1.29084576, -1.13855921, -1.59600673, -1.82477681, -1.97729694, -1.77048346])
```

```
#max. pour chaque variable
numpy.max(Z,axis=0)
```

```
array([2.62027526, 3.66738824, 1.4859373 , 1.62548897, 3.02981159, 1.859607  ])
```

5 Auto-encodeur avec Keras

5.1 Construction du réseau

Pour définir l'architecture de l'auto-encodeur, nous avons besoin de plusieurs outils de Keras : ceux qui permettent de définir les différentes couches ; ceux qui permettent de définir le réseau dans son ensemble, en tant que modèle.

```
#outil couches
from keras.layers import Input, Dense

#outil modélisation
from keras.models import Model
```

```
Using TensorFlow backend.
```

Keras nous indique qu'il s'appuie sur la librairie Tensorflow en sous-main.

Nous disposons de p = 6 variables, nous désirons une représentation des individus dans le plan. Nous construisons donc le réseau suivant :

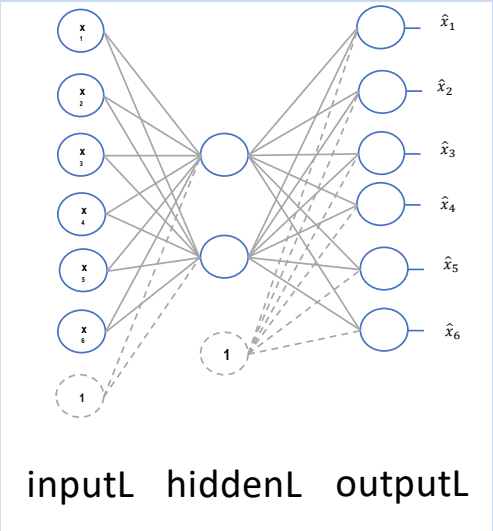

Figure 1 - Architecture de l'auto-encodeur

A l'instar d'un perceptron, mis à part la sortie, chaque couche est munie d'un biais (un neurone qui prend systématiquement la valeur 1).

Nous définissons les 3 couches successives du réseau et le modèle dans son ensemble :

```
#définition de la structure - couche d'entrée
inputL = Input(shape=(p,))

#couche intermédiaire
hiddenL = Dense(2,activation='sigmoid')(inputL)

#couche de sortie
outputL = Dense(p,activation='linear')(hiddenL)

#modele
autoencoder = Model(inputL,outputL)
```

La couche ''inputL'' est créée ex-nihilo, nous spécifions le nombre de neurones ''p'', le biais est rajouté automatiquement (c'est une option modifiable). ''hiddenL'' est la couche cachée à 2 neurones avec une fonction d'activation sigmoïde, elle vient après ''inputL''. ''outputL'' est la couche de sortie avec autant de neurones qu'en entrée, elle vient après ''hiddenL''. J'ai fait le choix d'une fonction d'activation linéaire en sortie parce que le réseau doit pouvoir produire des valeurs s'étalant sur la même étendue que les variables de Z.

La variable ''autoencoder'' représente le réseau dans son ensemble (le modèle), commençant par ''inputL'' et finissant par ''outputL''. Nous en définissons les caractéristiques d'apprentissage en spécifiant l'algorithme d'optimisation (optimizer = 'adam') et le critère à optimiser (loss = 'mse' ; mean squared error).

```
#compilation
autoencoder.compile(optimizer='adam',loss='mse')

#affichage des carac. du modèle
print(autoencoder.summary())
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 6)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 14
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 18
=================================================================
Total params: 32
Trainable params: 32
Non-trainable params: 0
_________________________________________________________________
```

Le nombre de paramètres (Param #) est important, il nous dit :

- Qu'il y a 14 coefficients à estimer pour la couche intermédiaire c.-à-d. 2 x (6 +1) = 14. En effet, il est composé de 2 neurones, et la couche précédente (d'entrée) est porté par 6 variables + le biais.
- Pour la couche de sortie, nous avons 18 coefficients, soit 6 x (2 + 1).

C'est bien l'architecture que nous voyons dans la Figure 1.

5.2 Entraînement du réseau

Nous lançons l'apprentissage des poids synaptiques (les coefficients) à l'aide de la fonction **fit()**.

```
#apprentissage à partir des données
n_epochs = 10000
historique = autoencoder.fit(x=Z,y=Z,epochs=n_epochs)
```

Nous indiquons les variables à prendre en entrée (x) ; les variables de référence en sortie pour le calcul de la fonction de perte (y). Nous mettons la même matrice Z dans notre cas, nous sommes dans le cadre de l'apprentissage non-supervisé. Mais je fais remarquer que l'outil peut dépasser ce périmètre restreint. Nous pouvons placer en (x) et (y) des données différentes, par exemple une version corrompue des données en (x), propre en (y) pour bénéficier des propriétés de ''débruiteur'' de l'auto-encodeur (voir le cours). Ou même pourquoi pas faire de l'apprentissage supervisé multi-cible en plaçant de matrices de nature différente en (x) et (y) à l'instar de ce que l'on ferait par exemple dans une régression PLS (voir ''Régression PLS – Comparaison de logiciels'', mai 2008). Je trouve l'analogie très séduisante j'avoue. Il faudra creuser cette idée dans un prochain tutoriel.

Enfin, j'ai fixé un nombre d'itération (epochs = 10000) très élevé parce que j'ai remarqué que le dispositif avait du mal à converger sur les petits ensembles de données. Le processus d'apprentissage est de ce fait relativement long (pour un fichier de n = 28 observations et p = 6 variables). Nous pouvons représenter la décroissance de la fonction de perte en utilisant l'objet ''**historique**'' renvoyé par **fit()** (Figure 2).

```
#importation librairie graphique
import matplotlib.pyplot as plt

#affichage de l'évolution de l'apprentissage
plt.plot(numpy.arange(1,n_epochs+1),historique.history['loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Décroissance fnct de perte')
plt.show()
```

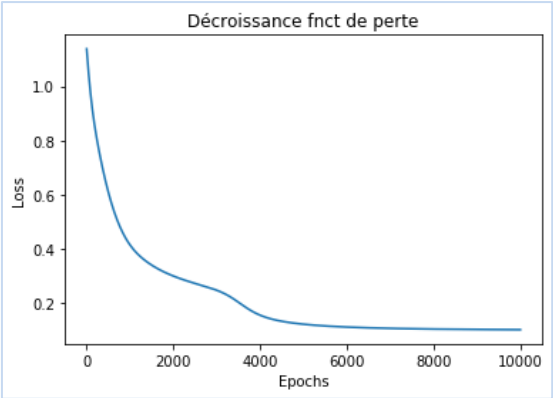

Figure 2 - Décroissance de la fonction de perte (MSE) au fil des epochs

Nous constatons que la convergence n'intervient qu'à partir de la 6000ème itération (et encore…).

5.3 Evaluation des performances

Nous pouvons calculer la perte sur un jeu de données avec la fonction **evaluate()**. Elle prend la matrice (x) que l'on présente en entrée, et (y) qui représente la matrice de référence que l'on cherche à reproduire. Dans notre cas, les deux matrices sont identiques (x = Z) et (y = Z), et nous effectuons les calculs sur les données d'apprentissage.

```
#qualité de l'approximation
print(autoencoder.evaluate(x=Z,y=Z))
```

```
0.1006258875131607
```

Pour vérifier la formule de Keras, j'ai calculé la projection ($\hat{Z}$) en sortie du réseau avec **predict()**.

```
#projection des individus dans l'espace initial
ZP = autoencoder.predict(Z)
print(numpy.round(ZP,3))
```

```
[[ 9.880e-01  9.650e-01  9.560e-01  9.500e-01  1.900e-02  1.026e+00]
 [ 3.030e-01  9.800e-02  3.960e-01  3.010e-01 -8.450e-01  1.070e-01]
 ...
 [ 1.245e+00  1.170e+00  1.229e+00  1.200e+00 -1.770e-01  1.244e+00]
 [ 1.665e+00  1.556e+00  1.647e+00  1.605e+00 -2.720e-01  1.655e+00]]
```

Puis j'ai essayé de reproduire la formule du MSE en opposant la projection ($\hat{Z}$) à (Z) observé.

```
#MSE que l'on peut calculer manuellement
MSE = 0.0
for i in range(n):
    MSE = MSE + numpy.mean((Z[i,:]-ZP[i,:])**2)
MSE = MSE/n
print(MSE)
```

```
0.10062589093778788
```

La formule réellement utilisée par Keras s'écrit :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{j=1}^{p}(z_{ij} - \hat{z}_{ij})^2}{p}$$

5.4 Les poids synaptiques et structure du réseau

Le réseau comporte 42 (14 + 18) coefficients estimés à partir des données. Nous les obtenons avec la commande **get_weights()**.

```
#affichage des poids - reconstitution du réseau
print(autoencoder.get_weights())
```

```
[array([[ 0.22956452,  0.07013915],
       [ 0.13161308,  0.27462834],
       [ 0.16323493,  0.02009064],
       [ 0.17092122,  0.1250671 ],
       [-0.4811064 ,  0.61330754],
       [ 0.13426608,  0.20653647]], dtype=float32),

array([ 0.03020703, -0.22764741], dtype=float32),

array([[ 3.6982296,  2.7750626,  4.015049 ,  3.597457 , -3.5301135, 2.9627585],
       [ 2.1801682,  3.0699747,  1.616154 ,  2.0517163,  4.0933795, 3.2495985]],
dtype=float32),

array([-2.8510175 , -2.782167  , -2.7587035 , -2.7424018 , -0.04747986, -2.9580164 ],
dtype=float32)]
```

Entre la couche intermédiaire et l'entrée, nous avons les 6 x 2 coefficients et les 2 constantes (intercept). Entre la sortie et la couche intermédiaire, nous avons les 2 x 6 coefficients et les 6 constantes. Avec un peu de courage et d'organisation, il serait possible de reproduire les projections du modèle à partir de ces paramètres estimés.

6 Exploitation des résultats

6.1 Représentation des individus dans l'espace réduit

La réduction de la dimensionalité est une des fonctionnalités majeures des auto-encodeurs. Dans notre structure de réseau, nous disposons d'une représentation dans le plan avec les deux neurones de la couche centrale. Pour obtenir les coordonnées des individus dans cet espace, nous formons un modèle, l'encodeur, avec l'entrée (**inputL**) et la couche centrale (**hiddenL**).

```
#modèle "encodeur"
encoder = Model(inputL,hiddenL)
```

Remarques : L'encodeur les prend en compte s'il y avait eu d'autres couches de neurones entre (inputL, hiddenL).

Nous présentons en entrée de ce nouveau modèle ''encoder'' la matrice des données Z pour obtenir les coordonnées ''factorielles'' des individus dans le plan.

```
#projection - coordonnées "factorielles"
coord = encoder.predict(Z)
print(coord)
```

```
[[0.6818345  0.60433006]
 [0.6415895  0.35836625]
 ...
 [0.7466697  0.61236197]
 [0.83098197 0.6618823 ]]
```

Puisque nous sommes à 2 dimensions, nous pouvons produire une représentation graphique.

```
#positionnement des individus dans le plan
fig, axes = plt.subplots(figsize=(15,15))
axes.set_xlim(0.1,0.9)
axes.set_ylim(0.1,0.9)
#étiquettes des points
for i in range(coord.shape[0]):
    plt.annotate(X.index[i],(coord[i,0],coord[i,1]))
#titre et axes
plt.title('Position des véhicules dans le plan')
plt.xlabel('Coord 1')
plt.ylabel('Coord 2')
plt.show()
```

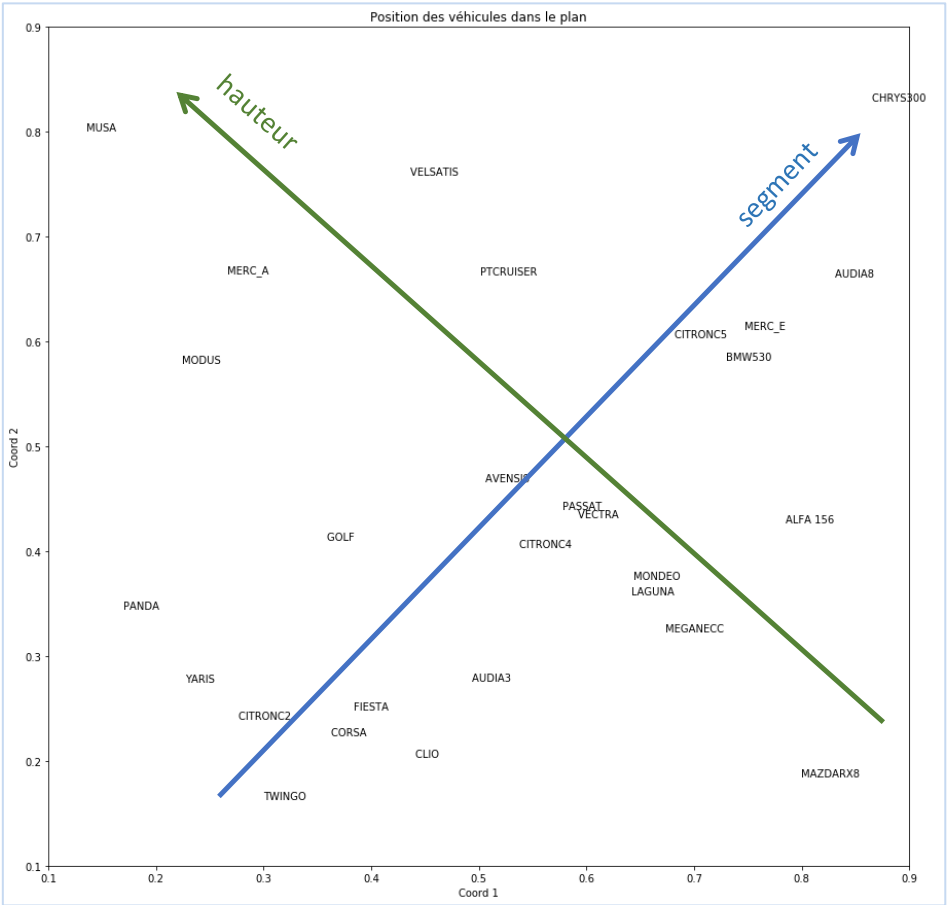

Figure 3 - Représentation des véhicules dans le plan défini par l'auto-encodeur

Il n'est pas nécessaire d'être grand clerc pour identifier les deux principales dimensions que recèlent les données : un axe ''segment'', principalement influencé par les caractéristiques de taille et de performances comme nous le verrons plus loin ; un axe ''hauteur'' opposant les véhicules ''hautes'', relativement à leurs caractéristiques (petits monospaces), aux voitures ''basses'', plutôt sportives si j'en juge à la Mazda RX8 située à l'extrémité sud-est du graphique.

6.2 Représentation des variables – ''Cercle des corrélations''

Une solution simple pour identifier le rôle des variables dans la définition de ces deux ''facteurs'' consiste à calculer les corrélations. Attention, si la représentation est hautement non-linéaire, cette piste n'est absolument pas adaptée. Dans notre cas, une seule couche cachée avec une fonction d'activation sigmoïde, une fonction de transfert linéaire en sortie, nous pouvons raisonnablement penser que les calculs revêtent une certaine signification.

Nous calculons les corrélations des variables actives avec les coordonnées factorielles.

```
#corrélations des variables avec les "composantes"
correlations = numpy.zeros(shape=(6,2))
for j in range(coord.shape[1]):
    for i in range(Z.shape[1]):
        correlations[i,j]=numpy.corrcoef(coord[:,j],Z[:,i])[0,1]

#data.frame pour correlations
dfCorr = pandas.DataFrame(correlations,columns=['Corr_1','Corr_2'],index=X.columns)
print(dfCorr)
```

```
             Corr_1    Corr_2
puissance  0.856896  0.558393
cylindree  0.692700  0.696586
vitesse    0.904411  0.461050
longueur   0.836163  0.530535
hauteur   -0.606759  0.664756
poids      0.737113  0.737782
```

Avec un ''cercle des corrélations'', on perçoit le positionnement relatif des variables (Figure 4).

```
#cercle des corrélations
fig, axes = plt.subplots(figsize=(10,10))
axes.set_xlim(-1.0,1.0)
axes.set_ylim(-1.0,1.0)
#position des variables
for i in range(dfCorr.shape[0]):
    plt.annotate(dfCorr.index[i],(correlations[i,0],correlations[i,1]))
#axes
plt.plot([0,0],[-1,+1],linestyle='--',c='gray',linewidth=1)
plt.plot([-1,+1],[0,0],linestyle='--',c='gray',linewidth=1)
#cosmétique
circle = plt.Circle((0,0),radius=1,fill=False,edgecolor='gray')
```

```
axes.add_artist(circle)
axes.set_aspect(1)
plt.show()
```

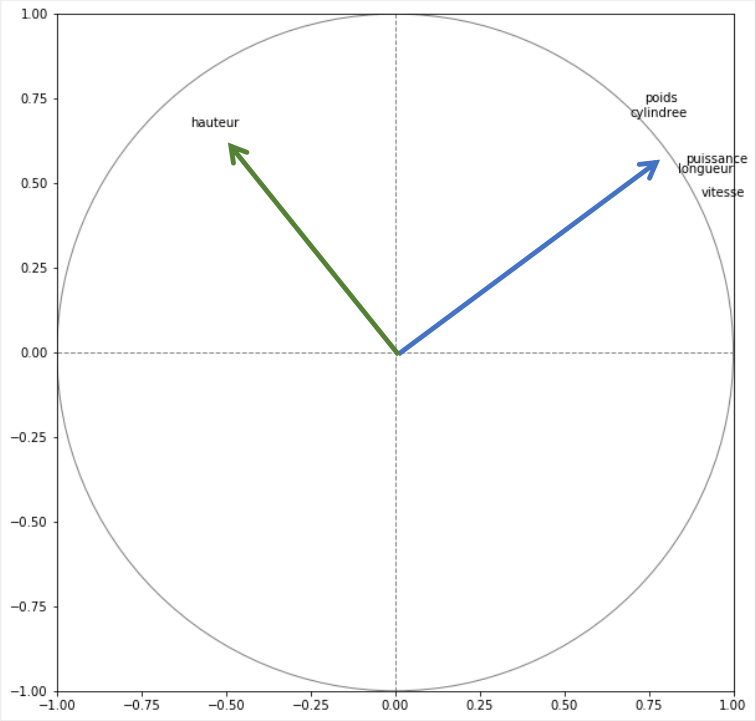

Figure 4 - "Cercle" des corrélations

L'intuition née de la visualisation du nuage de points des véhicules est confirmée ici. Il y a bien deux principales ''dimensions'' dans les données.

6.3 Restitution des données dans l'espace initial

L'autre fonctionnalité majeure de l'auto-encodeur est d'agir comme un nettoyeur de données. La partie centrale opère comme un goulot d'étranglement qui filtre le ''bruit'' qu'elles comportent. Nous pouvons reproduire les coordonnées des observations dans l'espace initial qui ne conserveraient que les informations importantes que les observations ont en commun.

Pour cela, il suffit de ''déstandardiser'' la projection ($\hat{Z}$) fournie par l'auto-encodeur. L'objet ''std'' utilisé pour la transformation des variables fait merveille ici (section 4.2). Nous n'affichons que les valeurs entières pour alléger la présentation.

```
#et déstandardisation de la projection dans l'espace initial
XP = std.inverse_transform(ZP)

#affichage - valeurs tronquées pour affichage plus clair
dp = pandas.DataFrame(XP.astype('int'),columns=X.columns,index=X.index)
print(dp)
```

Modele	puissance	cylindree	vitesse	longueur	hauteur	poids
CITRONC5	220	3036	232	474	147	1606
LAGUNA	170	2196	214	443	141	1354
CITRONC4	149	2049	203	430	145	1309
CLIO	91	1198	180	395	142	1054
CITRONC2	53	864	160	370	147	953
MODUS	92	1728	171	394	158	1211
MEGANECC	173	2175	216	445	140	1348
TWINGO	46	697	159	367	144	903
MONDEO	173	2247	215	445	142	1369
VECTRA	168	2282	212	442	145	1380
PASSAT	166	2267	210	440	145	1375
MERC_A	118	2103	181	410	159	1324
ALFA 156	220	2787	236	474	140	1532
AUDIA3	117	1553	191	410	143	1161
GOLF	102	1592	180	401	150	1171
MUSA	104	2153	171	400	167	1338
FIESTA	83	1178	175	389	145	1047
CORSA	73	1047	171	383	144	1008
PANDA	40	886	152	362	153	959
AVENSIS	151	2153	202	431	148	1340
CHRYS300	306	4199	268	526	149	1956
MAZDARX8	185	2104	226	453	133	1328
PTCRUISER	181	2726	212	449	154	1512
YARIS	45	834	156	365	149	944
VELSATIS	178	2836	208	447	158	1545
BMW530	230	3099	237	480	146	1625
MERC_E	239	3234	241	486	146	1666
AUDIA8	270	3608	255	505	145	1778

En ne conservant que les informations essentielles communes aux observations, voici donc ce que devrait être le tableau de données débarrassé de ses scories. Le confronter au tableau initial permet d'identifier les particularités de certains véhicules, ceux dont les caractéristiques pourraient dénoter par rapport au positionnement relatif qu'ils occupent relativement aux autres.

Pour ce faire, nous calculons tout d'abord l'écart-type des variables.

```
#calculer les écarts-type des variables initiales
etX = X.apply(lambda x: numpy.std(x))
print(etX)
```

```
puissance     73.124815
cylindree    968.799606
vitesse       32.447053
longueur      46.952905
hauteur        6.990062
poids        274.097852
```

Puis nous confrontons les données observées et reproduites, nous signalons les écarts s'ils sont supérieurs à **0.6** fois (paramètre à moduler en fonction de ce que l'on cherche à mettre en évidence) l'écart-type : ''+'' si la valeur observée est supérieure, ''-'' dans le cas contraire.

```
#signaler les écarts forts et leur sens
gap_et = 0.6
dstr = X.transform(lambda x: x.astype('str'))
```

```
for j in range(X.shape[1]):
    for i in range(X.shape[0]):
        if (X.values[i,j]-XP[i,j]) > gap_et * etX[j]:
            dstr.iloc[i,j] = '+%s' % (X.values[i,j])
        elif (XP[i,j] - X.values[i,j]) > gap_et * etX[j]:
            dstr.iloc[i,j] = '-%s' % (X.values[i,j])
        else:
            dstr.iloc[i,j] = '.'
print(dstr)
```

```
             puissance cylindree vitesse longueur hauteur  poids
Modele
CITRONC5             .         .       .        .       .      .
LAGUNA               .         .       .        .       .      .
CITRONC4             .         .       .        .       .      .
CLIO                 .         .       .        .       .      .
CITRONC2             .         .       .        .       .      .
MODUS                .         .       .        .       .      .
MEGANECC             .         .       .        .       .      .
TWINGO               .         .       .        .       .      .
MONDEO               .         .       .     +474       .      .
VECTRA               .         .       .        .       .      .
PASSAT               .         .       .     +471       .      .
MERC_A               .         .       .        .       .      .
ALFA 156             .         .       .     -443       .      .
AUDIA3               .         .       .        .       .      .
GOLF                 .         .       .        .       .      .
MUSA                 .         .       .        .       .      .
FIESTA               .         .       .        .       .      .
CORSA                .         .       .        .       .      .
PANDA                .         .       .        .       .      .
AVENSIS              .         .       .     +463       .      .
CHRYS300             .     +5654       .        .       .      .
MAZDARX8          +231     -1308       .        .       .      .
PTCRUISER            .         .       .        .       .      .
YARIS                .         .       .        .       .      .
VELSATIS             .     -2188       .     +486       .  +1735
BMW530               .         .       .        .       .      .
MERC_E               .         .       .        .       .      .
AUDIA8               .         .       .        .       .      .
```

On observe des valeurs saillantes forts instructives. Par exemple :

- Relativement à la situation qu'elle occupe sur l'échiquier des véhicules, la Mazda RX8 (qui n'est plus produite hélas) est à la fois plus puissante et dispose d'un petit moteur. Rien d'étonnant à cela, il s'agit d'une voiture sportive équipée d'un moteur rotatif. C'est évident si on connait bien les voitures, pour un non-initié l'information du tableau interpelle.
- Par rapport à sa situation de berline haut de gamme, la Vel Satis est munie d'un petit moteur, mais elle s'avère lourde et haute. Elle constituait une tentative de Renault de renouveler le concept de berline statutaire qui a abouti à un échec, malheureusement également. Il ne fait pas bon d'être trop novateur pour certaines catégories de véhicules.

- La Chrysler 300 C est irrémédiablement une berline très haut de gamme, sa position dans le plan factoriel le montre bien, mais sa cylindrée hors norme reste singulière.

Nous pouvons ainsi disserter longuement sur les caractéristiques des individus qui n'ont pas pu être captés par la représentation ''factorielle''. Je note surtout que la piste peut être intéressante pour identifier les points atypiques dans un jeu de données par exemple. Plutôt que la MSE, nous adoptons la MAE (mean absolute error, plus robuste aux points aberrants) comme fonction de perte pour l'auto-encodeur. Nous identifions alors comme atypiques les observations pour lesquels les écarts entre valeurs initiales et reconstituées sont anormalement élevées.

7 Individus et variables supplémentaires

7.1 Individus supplémentaires

Nous cherchons à placer des véhicules supplémentaires sur l'échiquier défini par l'auto-encodeur, des modèles Peugeot qui n'ont pas participé à l'étude.

Nous chargeons les données dans un premier temps.

```
#chargement des individus supplémentaires
ind_supp = pandas.read_excel("cars_autoencoder.xlsx",header=0,index_col=0,sheet_name=1)
print(ind_supp)
```

```
             puissance  cylindree  vitesse  longueur  hauteur  poids
Modele
P407               136       1997      212       468      145   1415
P307CC             180       1997      225       435      143   1490
P1007               75       1360      165       374      161   1181
P607               204       2721      230       491      145   1723
```

Puis nous transformons les variables avec l'objet ''std'' (section 4.2) dont les paramètres (moyenne, écart-type) ont été calculés sur l'échantillon d'apprentissage.

```
#transformation
z_ind_supp = std.transform(ind_supp)
print(z_ind_supp)
```

```
[[-0.1694754  -0.10738621  0.31479857  0.81616737 -0.40363426  0.32730751]
 [ 0.43223553 -0.10738621  0.71545129  0.11333545 -0.68975475  0.60093242]
 [-1.00366555 -0.76490092 -1.13371512 -1.1858387   1.88532964 -0.52640221]
 [ 0.76044149  0.63993037  0.86954849  1.30601992 -0.40363426  1.45099381]]
```

Nous projetons les individus dans le plan ''factoriel'' en utilisant le modèle ''encoder'' constitué uniquement de l'entrée et de la couche intermédiaire.

```
#projection
coord_ind = encoder.predict(z_ind_supp)
print(coord_ind)
```

```
[[0.60017437 0.4156695 ]
```

```
 [0.66015565 0.37820107]
 [0.15891293 0.5910655 ]
 [0.73946273 0.5583266 ]]
```

Représentés graphiquement, nous avons :

```
#positionnement des individus dans le plan
fig, axes = plt.subplots(figsize=(15,15))
axes.set_xlim(0.1,0.9)
axes.set_ylim(0.1,0.9)
#individus actifs
for i in range(coord.shape[0]):
    plt.annotate(X.index[i],(coord[i,0],coord[i,1]),c="gray")
#individus supplémentaires
for i in range(coord_ind.shape[0]):
    plt.annotate(ind_supp.index[i],(coord_ind[i,0],coord_ind[i,1]),c="blue",fontweight='bold')
#cosmétique
plt.title('Individus supplémentaires - Les modèles Peugeot',c='blue')
plt.xlabel('Coord 1')
plt.ylabel('Coord 2')
plt.show()
```

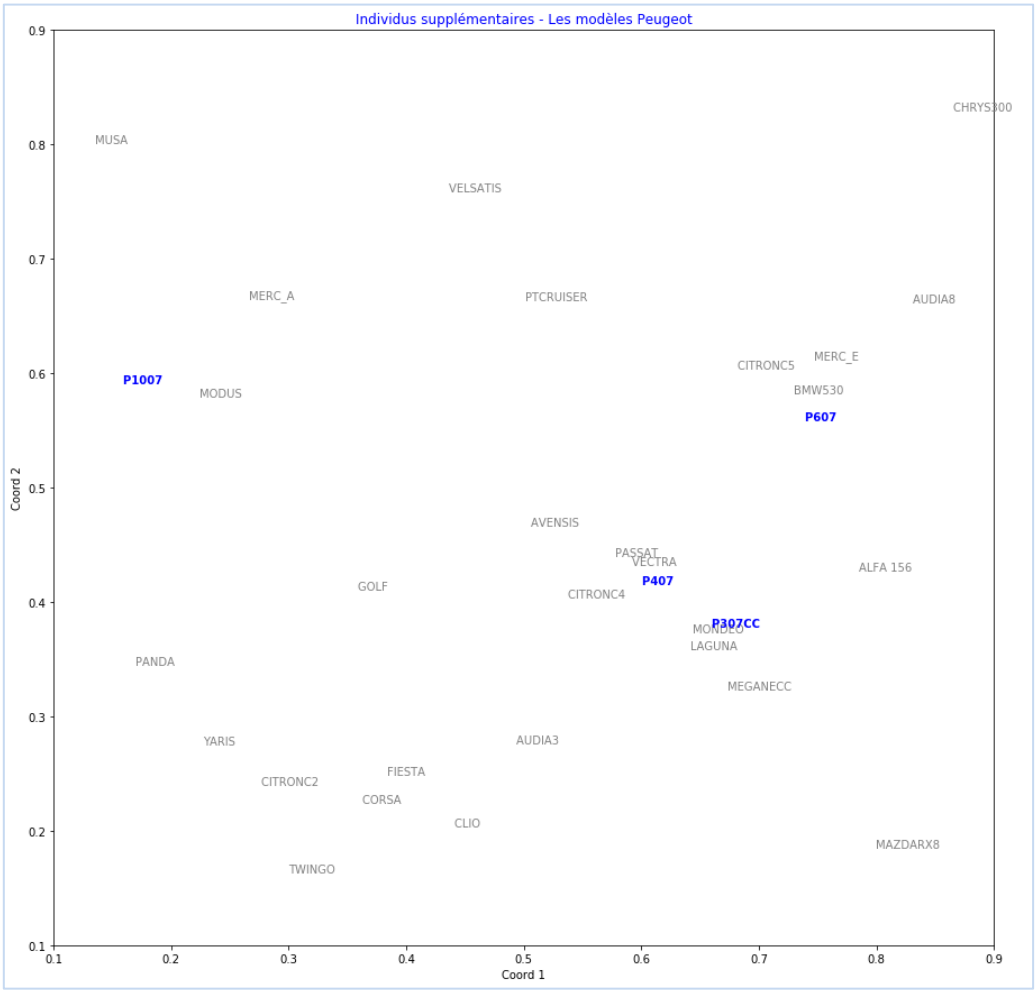

Figure 5 - Position des modèles supplémentaires "Peugeot" dans le plan "factoriel"

Connaissant un peu les voitures, les positions occupées par ces modèles sont totalement cohérentes.

7.2 Variables supplémentaires

Pour mieux interpréter les résultats fournis par l'auto-encodeur, nous introduisons les variables supplémentaires suivantes dans notre analyse :

```
#variables supplémentaires
var_supp = pandas.read_excel("cars_autoencoder.xlsx",header=0,index_col=0,sheet_name=2)
print(var_supp)
```

```
               co2   prix carburant
Modele
CITRONC5       238  33000   Essence
LAGUNA         196  25350   Essence
CITRONC4       142  23400    Diesel
CLIO           113  17600    Diesel
CITRONC2       141  10700   Essence
MODUS          163  16950   Essence
MEGANECC       191  27800   Essence
TWINGO         143   8950   Essence
MONDEO         189  23100   Essence
VECTRA         159  26550    Diesel
PASSAT         197  27740   Essence
MERC_A         141  24550    Diesel
ALFA 156       287  40800   Essence
AUDIA3         168  21630   Essence
GOLF           143  19140    Diesel
MUSA           146  17900    Diesel
FIESTA         117  14150    Diesel
CORSA          127  13590    Diesel
PANDA          135   8070   Essence
AVENSIS        155  26400    Diesel
CHRYS300       291  54900   Essence
MAZDARX8       284  34000   Essence
PTCRUISER      235  27400   Essence
YARIS          134  10450   Essence
VELSATIS       188  38250    Diesel
BMW530         231  46400   Essence
MERC_E         183  46450    Diesel
AUDIA8         281  78340   Essence
```

Première vérification très simple, assurons-nous que nous avons bien les mêmes individus dans les deux tables, celle des individus actifs, et celle que nous venons d'importer.

```
#vérifier que nous avons bien les mêmes individus
print(numpy.sum(X.index != var_supp.index)) # 0
```

Oui, c'est le cas. Ce type de vérification ne coûte rien et évite bien des désillusions par la suite.

Nous créons un data frame ''pandas'' où nous réunissons les coordonnées factorielles des individus et ces nouvelles variables. Les manipulations en seront facilitées par la suite.

```
#créer un data.frame intermédiaire
dfSupp = pandas.DataFrame(coord,columns=['coord1','coord2'],index=X.index)
dfSupp = pandas.concat([dfSupp,var_supp],axis=1,sort=False)
print(dfSupp)
```

```
                coord1    coord2  co2   prix carburant
Modele
CITRONC5      0.681835  0.604330  238  33000   Essence
LAGUNA        0.641590  0.358366  196  25350   Essence
CITRONC4      0.537167  0.403354  142  23400    Diesel
CLIO          0.441485  0.203617  113  17600    Diesel
CITRONC2      0.276590  0.240321  141  10700   Essence
MODUS         0.223585  0.579018  163  16950   Essence
MEGANECC      0.672957  0.322868  191  27800   Essence
TWINGO        0.299458  0.163738  143   8950   Essence
MONDEO        0.643636  0.373578  189  23100   Essence
VECTRA        0.591917  0.432296  159  26550    Diesel
PASSAT        0.577471  0.440095  197  27740   Essence
MERC_A        0.266523  0.666097  141  24550    Diesel
ALFA 156      0.785444  0.427036  287  40800   Essence
AUDIA3        0.493176  0.276323  168  21630   Essence
GOLF          0.359128  0.410699  143  19140    Diesel
MUSA          0.135138  0.801592  146  17900    Diesel
FIESTA        0.383783  0.249002  117  14150    Diesel
CORSA         0.363015  0.223892  127  13590    Diesel
PANDA         0.169353  0.344834  135   8070   Essence
AVENSIS       0.505610  0.466746  155  26400    Diesel
CHRYS300      0.865020  0.829988  291  54900   Essence
MAZDARX8      0.799053  0.185262  284  34000   Essence
PTCRUISER     0.501098  0.663585  235  27400   Essence
YARIS         0.227239  0.275157  134  10450   Essence
VELSATIS      0.436161  0.759375  188  38250    Diesel
BMW530        0.729567  0.582608  231  46400   Essence
MERC_E        0.746670  0.612362  183  46450    Diesel
AUDIA8        0.830982  0.661882  281  78340   Essence
```

7.2.1 Quantitatives

Une approche simple pour les variables illustratives quantitatives consiste à calculer leurs corrélations avec les axes et à les placer dans le cercle des corrélations.

Pour effectuons le calcul pour la variable ‘’prix’’ :

```
#corrélations avec prix
corPrix = dfSupp[['coord1','coord2']].corrwith(dfSupp.prix)
print(corPrix)
```

```
coord1    0.795658
coord2    0.577824
```

Puis pour ‘’co2’’ :

```
#corrélations avec co2
corCo2 = dfSupp[['coord1','coord2']].corrwith(dfSupp.co2)
print(corCo2)
```

```
coord1    0.808652
coord2    0.402165
```

Insérons ces informations dans le cercle des corrélations maintenant :

```
#rajouter dans le cercle des corrélations
fig, axes = plt.subplots(figsize=(10,10))
axes.set_xlim(-1.0,1.0)
axes.set_ylim(-1.0,1.0)
#var. actives
for i in range(dfCorr.shape[0]):
    plt.annotate(dfCorr.index[i],(correlations[i,0],correlations[i,1]))
#var illustratives
plt.annotate('Prix',(corPrix[0],corPrix[1]),c='blue',fontweight='bold')
plt.annotate('CO2',(corCo2[0],corCo2[1]),c='green',fontweight='bold')
#cosmétique
plt.plot([0,0],[-1,+1],linestyle='--',c='gray',linewidth=1)
plt.plot([-1,+1],[0,0],linestyle='--',c='gray',linewidth=1)
circle = plt.Circle((0,0),radius=1,fill=False,edgecolor='gray')
axes.add_artist(circle)
axes.set_aspect(1)
plt.show()
```

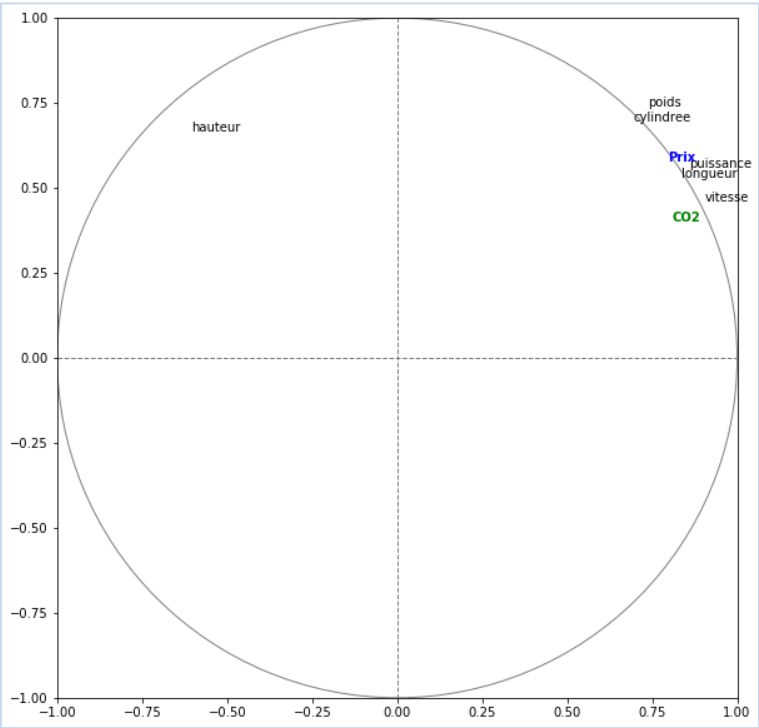

Figure 6 - Cercle des corrélations avec les variables illustratives (CO2, PRIX)

Manifestement, le prix et la pollution (en CO2) vont de pair avec le segment des véhicules. Rien d'étonnant à cela si on connaît un peu les voitures.

Une autre manière simple d'illustrer les véhicules du plan factoriel selon le prix serait de faire varier la teinte des points en fonction des valeurs prises par ''prix'' (Figure 7). Il y en une

visiblement qui est à un niveau de prix stratosphérique. Il s'agit de l'Audi A8 si l'on se réfère au tableau de données et au graphique factoriel étiqueté (Figure 3).

```
#graphique - prix
plt.scatter(data=dfSupp,x='coord1',y='coord2',c='prix',cmap='Oranges')
plt.title('Véhicules illustrés par les prix')
plt.show()
```

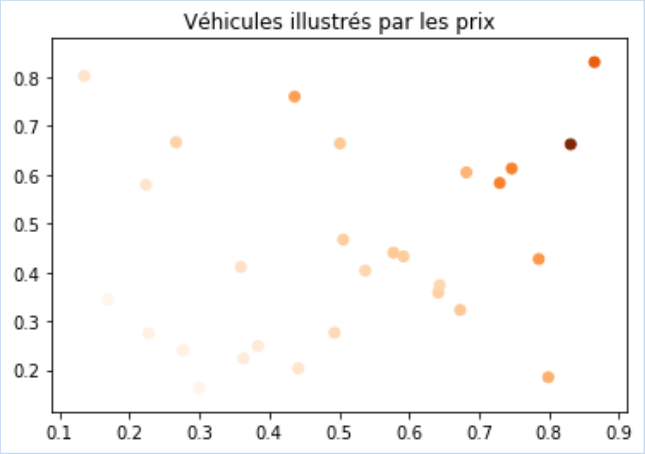

Figure 7 - Illustration des véhicules selon le prix

Nous pouvons faire de même pour la variable ''co2''.

```
#graphique - co2
plt.scatter(data=dfSupp,x='coord1',y='coord2',c='co2',cmap='Blues')
plt.annotate('MAZDA RX8',(dfSupp.coord1[21]-0.09,dfSupp.coord2[21]+0.025))
plt.title('Véhicules illustrés par le CO2')
plt.show()
```

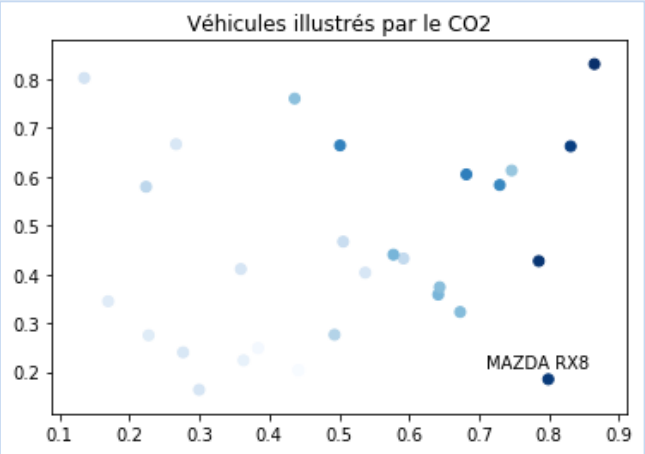

Figure 8 - Illustration des véhicules selon le CO2 - Mazda RX8 se distingue

C'est son niveau de pollution qui a condamné la Mazda RX8 et les moteurs rotatifs en général.

7.2.2 Qualitatives

Pour la variable qualitative ‘’carburant’’, une solution simple consiste à calculer les moyennes conditionnelles sur chaque facteur. Elles ne se démarquent pas vraiment :

```
#moyennes conditionnelles
dfSupp.pivot_table(index='carburant',values=['coord1','coord2'],aggfunc=pandas.Series.mean)
```

```
             coord1    coord2
carburant
Diesel     0.433327  0.475367
Essence    0.554003  0.431117
```

On pourrait placer ces barycentres conditionnels dans le graphique factoriel pour mieux les situer. Je préfère une autre solution, nous colorions les points en fonction du type de carburant.

Nous listons les différentes catégories de carburant tout d’abord :

```
#catégories de carburant
catCarburant = numpy.unique(dfSupp.carburant)
print(catCarburant)
```

```
['Diesel' 'Essence']
```

Puis nous créons le graphique en en tenant compte.

```
#graphique -- illustration selon le type de carburant
fig, ax = plt.subplots()
for cat,col in zip(catCarburant,['blue','green','red']):
    id = numpy.where(dfSupp.carburant==cat)[0]
    ax.scatter(dfSupp.coord1[id],dfSupp.coord2[id],c=col,label=cat)
ax.legend()
plt.title('Véhicules selon le type de carburant')
plt.show()
```

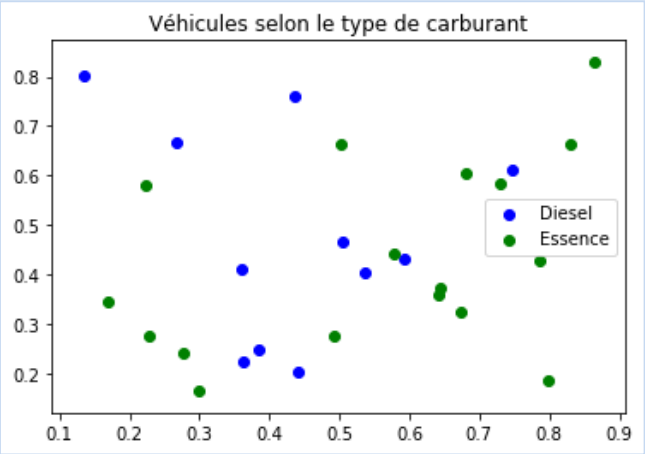

Figure 9- Illustration des véhicules selon le type de carburant

Distinguer la cartographie des véhicules avec le type de carburant n’amène rien de probant dans cette étude.

8 Conclusion

Nous nous en sommes tenus à un exemple basique dans ce tutoriel pour montrer le mode opératoire des auto-encodeurs à une couche cachée. Nous constatons que l'outil est simple et pratique, et qu'il produit des résultats similaires à ce qu'une analyse en composantes principales (ACP) pourrait nous fournir dans le même contexte.

Pour aller plus loin dans l'exploration de l'outil, une voie intéressante serait de travailler sur des auto-encodeurs avec plusieurs couches cachées pour essayer de capter des formes non-linéaires dans les données. Chose qu'une ACP, classique tout du moins, ne sait pas faire.

9 Références

Cholet F., ''Building Autoencoders in Keras'', in the Keras Blog, mai 2016.

Cohen O., ''PCA vs. Autoencoders'', in Towards Data Science, avril 2018.

Keras : The Python Deep Learning Library, https://keras.io/

Tutoriel Tanagra, ''Deep learning : les auto-encodeurs'', novembre 2019.

Tutoriel Tanagra, ''ACP avec Python'', juin 2018.