# TP 6

## 1. L'imputation dans Scikit-learn :

La description de l'implémentation de l'imputation de données manquantes de `scikit-learn` se trouve dans le guide utilisateur. Trois types d'imputeurs sont disponibles à l'heure actuelle : le SimpleImputer, qui implémente des stratégies classiques d'imputation univariée, le KNNImputer, qui implémente la stratégie par k-plus proches voisins, et l'"IterativeImputer qui implémente à titre expérimental l'imputation itérative.

## 2. Imputation par une valeur unique, pour des données générées

Dans un premier temps, intéressons à l'imputation univariée, c'est-à-dire imputer les valeurs manquantes par une valeur unique à chaque colonne.

Pour définir un tel *imputer*, on appelle `SimpleImputer()`. Les paramètres d'appel sont (pour des explications plus détaillées voir le lien ci-dessus) :

- `missing_values` : codage des valeurs manquantes dans le tableau de données. Ce codage peut être fait de différentes façons, par ex. par « NaN » (*Not a Number*), `np.nan`, parfois par `0` ou par des valeurs qui sont éloignées du domaine de variation de la variable concernée, par ex. `-999`. En général, le même code est employé pour les valeurs manquantes dans tout le tableau de données.
- `strategy` : méthode employée pour l'imputation : `'mean'` (par défaut) ou `'median'`, `'most_frequent'` ou `'constant'` ; en général, `'most_frequent'` n'a pas de sens pour des variables continues ; le calcul de la moyenne, respectivement de la médiane ou de la valeur la plus fréquente est réalisé sur les observations complètes ; si `'constant'`, alors c'est la valeur indiquée dans `'fill_value'` qui est systématiquement employée.
- `fill_value` : indique la valeur constante (chaîne de caractères ou valeur numérique, suivant le cas) utilisée pour compléter les valeurs manquantes (par défaut `None`).
- `verbose` : contrôle la verbosité (`0` ou `1`, par défaut `0` c'est à dire moins verbeux).
- `copy` : si `False`, la matrice de données est écrasée (si cela est possible) par les données transformées (par défaut `True`).

Le seul attribut accessible est :

- `statistics_` : array `[n_features]` dans lequel on trouve les valeurs calculées pour réaliser l'imputation.

Les méthodes habituelles peuvent être employées :

- `fit(X, y=None)` : calcul des valeurs à utiliser pour l'imputation.
- `transform(X)` : imputation des données manquantes en utilisant les valeurs obtenues lors de l'appel à `fit`.
- `fit_transform(X, y=None)` : calcul des valeurs à utiliser pour l'imputation et application de l'imputation.
- `get_params([deep])` : lire les valeurs des paramètres de l'estimateur employé.
- `set_params(**params)` : donner des valeurs aux paramètres de l'estimateur employé.

Nous examinerons d'abord le résultat de l'application de méthodes d'imputation proposées par Scikit-learn à des données simples bidimensionnelles. Après avoir généré les données complètes, nous supprimons les valeurs de la seconde variable pour certaines observations afin d'obtenir des données incomplètes.

Générons d'abord des données bidimensionnelles réparties dans trois gaussiennes.

```
# importations préalables
import numpy as np
from sklearn.impute import SimpleImputer

# Génération des données bidimensionnelles complètes
n_base = 100
data1 = np.random.randn(n_base,2) + [5,5]
data2 = np.random.randn(n_base,2) + [3,2]
data3 = np.random.randn(n_base,2) + [1,5]
data = np.concatenate((data1,data2,data3))
data.shape   # vérification
(300, 2)

# On mélange les données
np.random.shuffle(data)

# visualisation (optionnelle) des données générées
import matplotlib.pyplot as plt
plt.plot(data[:,0],data[:,1],'r+')
plt.show()
```

Nous avons donc une matrice d'observations (300, 2). Nous allons supprimer certaines valeurs de la seconde volonne. La variable `missing_rate` nous permettra de contrôler le nombre de valeurs manquantes. Pour ce faire, on génère un tableau de booléens `missing_samples` (un booléen par ligne de la matrice d'observations) qui indiquera `True` si la valeur est manquante et `False` sinon.

```
n_missing_samples = int(np.floor(n_samples * missing_rate))
print("Nous allons supprimer {} valeurs".format(n_missing_samples))

# choix des lignes à valeurs manquantes
present = np.zeros(n_samples - n_missing_samples, dtype=np.bool)
missing = np.ones(n_missing_samples, dtype=np.bool)

missing_samples = np.concatenate((present, missing))
# On mélange le tableau des valeurs absentes
np.random.shuffle(missing_samples)
print(missing_samples)
```

Nous pouvons maintenant créer la matrice des observations avec des valeurs manquantes. Nous choisissons ici de représenter une valeur manquante par `NaN` (*Not A Number*).

```
# obtenir la matrice avec données manquantes : manque indiqué par
#  valeurs NaN  dans la seconde colonne pour les lignes True dans
#   missing_samples
data_missing = data.copy()
data_missing[np.where(missing_samples), 1] = np.nan
print(data_missing)
```

`data_missing` est désormais notre matrice d'observations pour lesquelles une fraction des valeurs de la seconde colonne sont manquantes.

**Question 1:**

*Comment caractériser ici le mécanisme par lequel des données manquent : MCAR, MAR, MNAR ?*

Nous pouvons commencer par réaliser une imputation par la moyenne, c'est-à-dire que lorsque la valeur de la seconde colonne est absente, nous utilisons à la place la valeur moyenne de cette colonne.

```
# imputation par la moyenne: les NaN sont remplacés par la moyenne
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
data_imputed = imp.fit_transform(data_missing)
print(data_imputed)
```

Nous pouvons visualiser dans le plan l'effet de cette imputation :

```
plt.scatter(data_imputed[:,0],data_imputed[:,1], marker='+', c=missing_samples)
plt.show()
```

Comme dans ce cas illustratif nous avons accès aux données réelles, nous pouvons estimer l'erreur que nous faisons en réalisant cette imputation :

```
# calculer l'"erreur" d'imputation
from sklearn.metrics import mean_squared_error
mean_squared_error(data[missing_samples,1],data_imputed[missing_samples,1])
```

**Question 2:**

*Affichez les valeurs calculées (les moyennes) employées pour l'imputation.*

**Question 3:**

*Appliquez une imputation par la médiane et examinez l'erreur résultante.*

**Question 4:**

*Appliquez une imputation par `0`, visualisez les résultats et calculez l'erreur d'imputation. Pourquoi est-elle nettement supérieure à celle obtenue lors de l'imputation par la moyenne ou par la médiane?*

### 3. Imputation par le centre du groupe, pour des données générées :

Cette méthode n'est pas directement disponible dans Scikit-learn mais nous pouvons néanmoins l'appliquer de façon assez simple. Pour cela, nous ferons d'abord une classification automatique avec *K-means* des observations sans données manquantes. Nous trouverons ensuite, pour chaque observation incomplète (à données manquantes), quel est le centre de groupe le plus proche de cette observation (en utilisant une distance partielle, calculée seulement sur les coordonnées renseignées pour cette observation) ; nous nous servirons pour cela de la méthode de classement *Nearest centroid*, voir http://scikit-learn.org/stable/modules/neighbors.html#nearest-centroid-classifier. Enfin, pour chaque observation incomplète nous complèterons la valeur manquante par la coordonnée correspondante du centre de groupe le plus proche.

```
# obtenir le tableau composé des seules observations complètes
# ~ permet d'inverser un tableau de booléens
data_filtered = data[~missing_samples, :]
print(data_filtered.shape)    # vérification
# (210, 2)
```

On commence par réaliser un *K-means* sur les observations qui sont complètes (pour lesquelles aucune colonne n'est manquante).

```
# application de la classification automatique aux observations complètes
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3).fit(data_filtered)
# affichage des centres obtenus pour les groupes
centers = kmeans.cluster_centers_
print("Centres : {}".format(centers))

plt.scatter(data_filtered[:,0], data_filtered[:,1], marker='+', c=kmeans.labels_)
plt.scatter(centers[:,0], centers[:,1], edgecolors='k', s=300, marker='*', label="Centres", c=range(len(centers)))
plt.legend()
plt.show()
```

Pour chaque observation incomplète, nous allons déterminer le centre le plus proche de cette observation (à l'aide de l'outil `NearestCentroid` de scikit-learn). L'indice du centre le plus proche sera déterminé à partir des valeurs non manquantes (en l'occurrence, à partir de la valeur de la première colonne).

```
# les centres calculés par k-means sont associés aux 3 étiquettes des groupes
#  (les 'observations' pour NearestCentroid sont les centres issus de k-means)
#   seules les coordonnées des centres sur l'axe 1 sont employées
ncPredictor.fit(centers[:,0].reshape(-1, 1), y)

# l'index du centre le plus proche est déterminé pour chaque observation à
#  donnée manquante (à partir de la valeur de la variable non manquante, axe 1
nearest = ncPredictor.predict(data_missing[missing_samples, 0].reshape(-1,1))
# détermination des valeurs à utiliser pour l'imputation : pour chaque
#  observation, la valeur sur l'axe 2 du centre correspondant
estimated = np.zeros(n_missing_samples)
indices = range(n_missing_samples)
for i in indices:
    estimated[i] = centers[nearest[i]-1,1]

data_imputed = data_missing.copy()   # initialisation de data_imputed
# imputation avec les valeurs obtenues
data_imputed[missing_samples, 1] = estimated
# calcul de l'erreur moyenne d'imputation
mean_squared_error(data[missing_samples,1],data_imputed[missing_samples,1])
```

Et nous pouvons à nouveau visualiser le nuage de points :

```
plt.scatter(data_imputed[:,0], data_imputed[:,1], marker='+', c=missing_samples)
plt.scatter(centers[:,0], centers[:,1], edgecolors='k', s=300, marker='*', label="Centres", c='r')
plt.legend()
plt.show()
```

L'erreur moyenne peut parfois être supérieure à celle obtenue lors de l'imputation par la moyenne ou par la médiane, même si les données forment des groupes « naturels », bien identifiés par la classification automatique. Les données complètes et le choix des données manquantes étant issus de tirages aléatoires, pour d'autres tirages il est possible qu'une autre des méthodes d'imputation donne de meilleurs résultats en termes d'erreur quadratique moyenne.

**Question 5:**

*Que faudrait-il changer dans la génération des données pour que l'imputation par les centres fonctionne mieux que l'imputation par la moyenne ou la médiane ?*

### 4. Imputation par les *k* plus proches voisins, pour des données générées

Cette méthode est directement disponible dans scikit-learn depuis la version 0.22 (veillez à bien installer une version récente). L'utilisation est simple, on appelle `KNNImputer()` avec les paramètres d'appel ci-dessous :

- `missing_values` : codage des valeurs manquantes dans le tableau de données. Ce codage peut être fait de différentes façons, par ex. par « NaN » (*Not a Number*), `np.nan`, parfois par `0` ou par des valeurs qui sont éloignées du domaine de variation de la variable concernée, par ex. `-999`. En général, le même code est employé pour les valeurs manquantes dans tout le tableau de données.
- `n_neighbors` : le nombre de voisins à considérer (par défaut, 5).
- `weights` : la pondération des voisins (uniforme ou dépendant de la distance entre le point et ses voisins).
- `metric` : la métrique à utiliser (distance euclidienne par défaut).
- `copy` : si `False`, la matrice de données est écrasée (si cela est possible) par les données transformées (par défaut `True`).

(se référer à la documentation pour plus de détails)

```
from sklearn.impute import KNNImputer
# imputation sur la base des 3 plus proches voisins sans données manquantes
#  obtenus à partir de distances calculées avec les seules données présentes
data_imputed = KNNImputer(missing_values=np.nan, n_neighbors=3).fit_transform(data_missing)
# évaluation des résultats
mean_squared_error(data[missing_samples,1],data_imputed[missing_samples,1])
```

Et comme précédemment, nous pouvons réaliser la visualisation du jeu de données après imputation:

```
plt.scatter(data_imputed[:,0], data_imputed[:,1], marker='+', c=missing_samples)
plt.show()
```

**5. Imputation pour les données « textures » projetées sur les 2 premiers axes principaux**

L'objectif de cette section est de traiter des données dont la distribution est plus asymétrique et donc pour lesquelles l'écart entre la médiane et la moyenne est plus fort. Pour cela, nous nous servirons de la projection des données « textures » sur les deux premiers axes principaux. Les données ainsi obtenues sont bidimensionnelles et peuvent être facilement visualisées.

Pour rappel, ces données correspondent à 5500 observations décrites par 40 variables. Chaque observation appartient à une des 11 classes de textures ; chaque classe est représentée par 500 observations. Les données sont issues de https://www.elen.ucl.ac.be/neural-nets/Research/Projects/ELENA/databases/REAL/texture/. Nous appliquerons *K-means* à ces données, avec `n_clusters = 11`, et examinerons dans quelle mesure les groupes issus de la classification automatique se rapprochent des classes présentes.

Si les données ne sont pas dans le répertoire de travail il est nécessaire de les chercher d'abord :

```
$ wget -nc http://cedric.cnam.fr/~crucianm/src/texture.dat
```

**Question 6 :**

Projetez ces données sur les deux premiers axes principaux, choisissez aléatoirement 25% de valeurs manquantes sur le premier axe (les projections sur les deux premiers axes sont des vecteurs à 2 dimensions) et appliquez l'imputation par la moyenne, ensuite par la médiane.

**Question 7:**

*Comparez les résultats obtenus avec la médiane à ceux obtenus avec la moyenne.*

**Question 8:**

*Appliquer aux données « textures » l'imputation par les centres des groupes vous semble être un choix pertinent ? L'imputation par les k plus proches voisins ?*