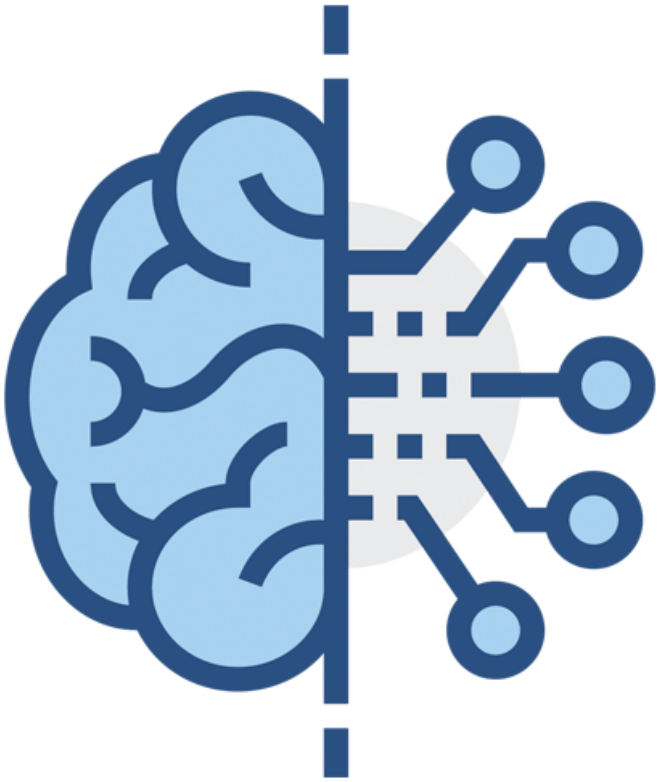
Université Abdelmalek Essaadi

Faculté des sciences et techniques Tanger

Logiciels & Systèmes Intelligents

Devoir 3 :



Encadré par : Mr. M'Hamed AIT KBIR

Réalisé par : Ahmed MELLOUK

Nora YOUSSFI

Omar RAGHLI

Remerciements :

Nous tenons à adresser nos sincères remerciements à notre professeur M'Hamed AIT KBIR pour sa disponibilité, pour sa lecture, suggestion et remarques et surtout pour sa confiance sans limite mise en nous tout au long de ce projet.

Nous vous prions de bien vouloir agréer le témoignage de notre plus vive reconnaissance et notre profond respect.

Table des matières :

Introduction :

Le Machine Learning ou apprentissage automatique est un domaine scientifique, et plus particulièrement une sous-catégorie de l'intelligence artificielle. Elle consiste à laisser des algorithmes découvrir des " patterns ", à savoir des motifs récurrents, dans les ensembles de données. Ces données peuvent être des chiffres, des mots, des images, des statistiques…

Tout ce qui peut être stocké numériquement peut servir de données pour le Machine Learning. En décelant les patterns dans ces données, les algorithmes apprennent et améliorent leurs performances dans l'exécution d'une tâche spécifique.

Pour résumer, les algorithmes de Machine Learning apprennent de manière autonome à effectuer une tâche ou à réaliser des prédictions à partir de données et améliorent leurs performances au fil du temps. Une fois entraîné, l'algorithme pourra retrouver les patterns dans de nouvelles données.

Parmi les méthodes utilisées pour l'apprentissage les réseaux neurones multicouches, et les réseaux RBF (Radial basis function), qu'on va les utiliser dans ce devoir.

Initialisation au devoir :

Le but de ce devoir est de développer deux applications pour implémenter les solutions apportées par les modèles de réseaux de neurones artificiels aux deux problèmes en bas.

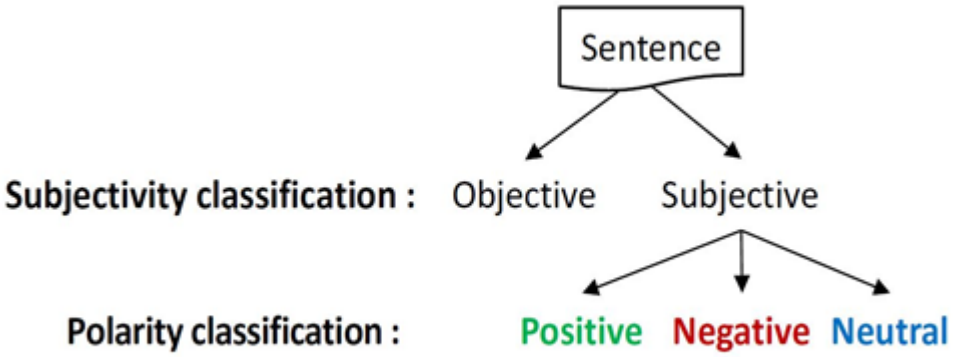
Premier problème :

Le premier problème concerne l'utilisation des réseaux multicouches pour l'analyse des sentiments des phrases issues d'une base d'exemples qui contient des phrases étiquetées avec un sentiment positif ou négatif.

Deuxième problème :

Le deuxième problème concerne l'utilisation des réseaux RBF (Radial basis function) pour l'approximation de la consommation énergétique d'une maison à partir d'un ensemble de données de prévision énergétique des appareils électroménagers.

Première application :



Objectif de l'exercice :

L'objectif de cet exercice est la réalisation d'un Perceptron multicouches capable d'analyser des sentiments des phrases issues

d’une base d’exemples qui contient des phrases étiquetées avec un sentiment positif ou négatif.

Ces sentiments sont extraits de la base de données IMDB, qui contient des opinions des films étiquetés par 1 si le commentaire est positif, et 0 si le commentaire est négatif.

	review	label
0	A very, very, very slow-moving, aimless movie ...	0
1	Not sure who was more lost - the flat characte...	0
2	Attempting artiness with black & white and cle...	0
3	Very little music or anything to speak of.	0
4	The best scene in the movie was when Gerardo i...	1
5	The rest of the movie lacks art, charm, meanin...	0
6	Wasted two hours.	0
7	Saw the movie today and thought it was a good ...	1
8	A bit predictable.	0
9	Loved the casting of Jimmy Buffet as the scien...	1

Figure 1 - Extrait des données

Solution Proposée :

Exploration des données :

La première étape dans l’analyse des sentiments est l’exploration des données, ou on essaye de bien comprendre notre dataset.

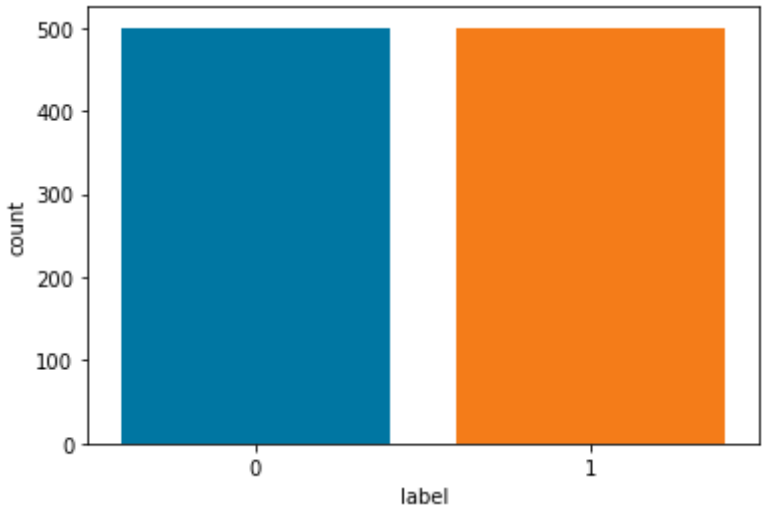


Figure 2 -Nombre de sentiment et commentaire

On observe qu'on a autant d'opinions négatives que positives.

Traitement des données :

La deuxième étape dans l'analyse des sentiments est le traitement des données.

L'objectif de cette étape est de connaître les facteurs qui influencent les sentiments de nos opinions et filtrer nos données en éliminant le bruit, qui sont les caractéristiques qui n'ont aucune influence sur le résultat.

a) Évaluer les mots les plus fréquent :

word	count		
.	905		
the	657		
,	649		
and	418		
a	413		
of	370		
is	338		
I	285		
to	247		
it	234	for	101
this	215	as	90
was	191	with	87
in	191	It	87
The	188	!	85
movie	177	are	78
's	160	This	76
film	159	on	74
that	158	you	72
n't	104	one	71
for	101	but	64

Figure 3 - Table des mots plus fréquent dans notre dataset

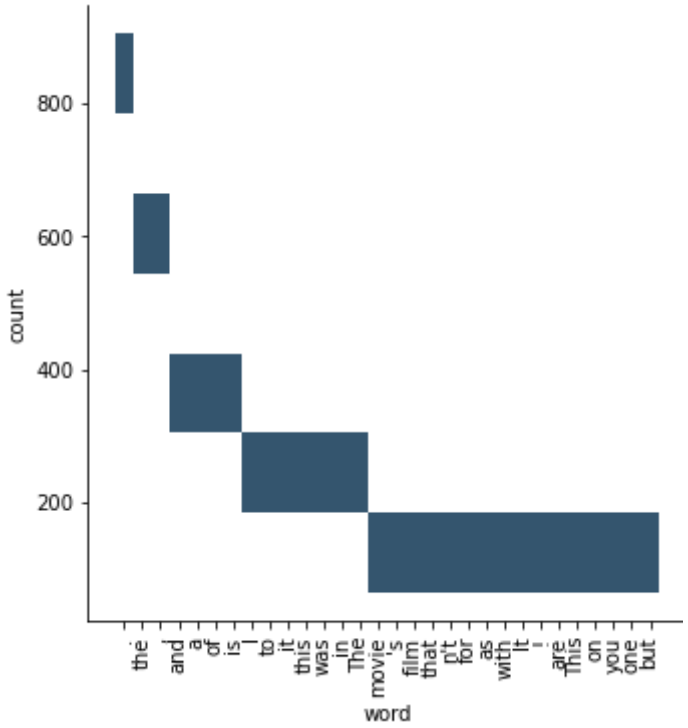


Figure 4 - Graphe de mots plus fréquent dans notre dataset

On observe que les mots les plus fréquents dans notre dataset sont des mots qui n'ont aucune influence sur le résultat, ie, des mots neutres.

tel que ‘the’,’is’,’if’, etc et des caractères spéciaux.

b) Éliminer les caractères spéciaux :

Pour éliminer les caractères spéciaux dans un commentaire, on utilisera la librairie regex.

```
import re #regex
```

Avec cette simple ligne, on peut filtrer un review et supprimer les caractères spéciaux.

```
phrase = re.compile('[^a-zA-Z]').sub(' ', phrase).lower().split() #suprimmer les caracteres speciaux
```

c) Elimination des stopwords:

L'élimination des stopwords est fait à l'aide du libraire nltk

```
from nltk.corpus import stopwords
```

Liste des mots avec les stopwords.

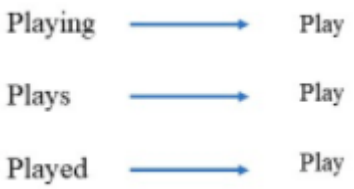
```
Tokenized text with stop words :
['Oh', 'man', ',', 'this', 'is', 'pretty', 'cool', '.', 'We', 'will', 'do', 'more', 'such', 'things', '.']
```

Liste des mots sans stopwords.

```
Tokenized text with out stop words :
['Oh', 'man', ',', 'pretty', 'cool', '.', 'We', 'things', '.']
```

d) Lemmatization:

La lemmatisation est l'opération de convertir les mots à leur racine, afin d'éviter la répétition de plusieurs mots ayant le même sens.



Tous ces étapes du traitement des données sont réalisées par la fonction suivante :

```
#Cleaning d'un commentaire
def clean(phrase):
    phrase = re.compile('[^a-zA-Z]').sub(' ', phrase).lower().split() #suprimmer les caracteres speciaux

    #lemmatization (greatly , greatness) => (great)
    #stopwords (suprimmer les mots neutres, tel que: in,out,i,am,if, ...)
    phrase = [nltk.WordNetLemmatizer().lemmatize(word) for word in phrase if not word in stopwords.words('english')]

    phrase=removeByLength(phrase) #filtrer par longueur
    phrase = ' '.join(phrase)

    return phrase
```

	clean_review	label
0	slow moving aimless movie distressed drifting ...	0
1	sure lost flat character audience nearly half ...	0
2	attempting artiness black white clever camera ...	0
3	little music anything speak	0
4	best scene movie gerardo trying find song keep...	1

Figure 5 - table des opinions traitées

Extraction des caractéristiques :

C'est le processus d'extraire les caractéristiques de notre dataset, et c'est avec ces caractéristiques qu'on sera capable de construire un vecteur qui va représenter chaque opinion.

Cette extraction est faite par la librairie :

```
#Librairie pour extraire les caracteristiques & creation du vecteur de carateristiques
from sklearn.feature_extraction.text import CountVectorizer as CV
```

Il existe plusieurs types de vecteur de caractéristiques, notre choix est le **CountVectorizer.**

Le Count Vectorizer compte le nombre de fois qu'un jeton apparaît dans le document et utilise cette valeur comme poids.

	aailiyah	abandoned	ability	abroad	absolutely	abstruse	abysmal	academy	accent	accessible	...	yet	young	younger	youthful	youtube	yun
0	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

1000 rows × 2677 columns

Figure 5 - Représentation des commentaires sous forme de vecteur

(1ère colonne : index des opinions)

Créer des échantillons d'Apprentissage et Test :

Cette étape consiste à séparer notre dataset en deux échantillons, un pour entraîner le réseau et l'autre pour tester la précision du réseau.

On utilisera la librairie suivante pour séparer créer ces échantillons.

```
from sklearn.model_selection import train_test_split
```

- On sépare nos données en deux listes :

```
# Données + classes cibles
data   = np.array(df.values[:,1:2733], dtype=np.float32) #Données d'entraînement (Input)
target = df.values[:,-1] #Résultats a prédire (Output)
```

Data : Contient les données d'entrées (commentaires)

Target : Contient les données de sorties (sentiments)

Créer le perceptron multicouche :

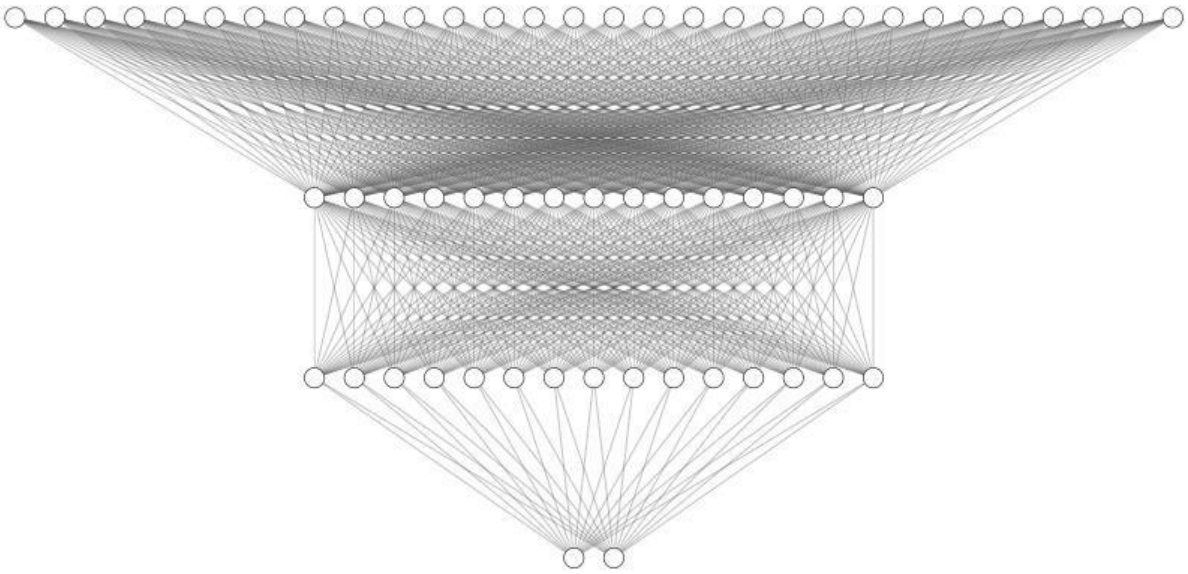
a. Architecture du réseau:

- **Le réseau est constitué de 4 couches**.

- **La couche d'entrée possède 2675 neurones** : le nombre de neurones dans cette couche représente la dimension du vecteur d'entrée

- La 2eme couche possède 1**5** neurones
- La 3eme couche possède 1**5** neurones

- **La couche de sortie possède 2 neurones** : le nombre de neurones dans cette couche représente la dimension du vecteur de sortie



b. *Fonction d'activation:*

Puisque dans notre cas, la valeur prédit sera une probabilité d'appartenance à une classe, (0 ou 1).

Ça sera convenable de choisir la fonction d'activation :

Sigmoid.

$$S(x) = \frac{1}{1 + e^{-x}}$$

c. *Taux d'activation :*

Le taux d'activation choisi a été **alpha=0.1**, puisqu'après un nombre d'itération, on a trouvé que c'était la valeur optimale pour avoir une précision maximum.

Apprentissage :

```
pmc = MultiLayerPerceptron(arch=[trainX.shape[1],15,15,2], alpha=0.1) #Instance du PMC

(errs, iter_fin) = pmc.fit(trainX, trainYC, iterations=500, bloc_size=5, error_min=0.00001, displayPeriod=20) #Apprentissage
```

```
Iteration: 0-500, Error: 0.316362
Iteration: 20-500, Error: 0.243143
Iteration: 40-500, Error: 0.183626
Iteration: 60-500, Error: 0.074495
Iteration: 80-500, Error: 0.025536
Iteration: 100-500, Error: 0.012021
Iteration: 120-500, Error: 0.007930
Iteration: 140-500, Error: 0.006242
Iteration: 160-500, Error: 0.005351
Iteration: 180-500, Error: 0.004792
Iteration: 200-500, Error: 0.004313
Iteration: 220-500, Error: 0.004015
Iteration: 240-500, Error: 0.003746
Iteration: 260-500, Error: 0.003530
Iteration: 280-500, Error: 0.003369
Iteration: 300-500, Error: 0.003237
Iteration: 320-500, Error: 0.003123
Iteration: 340-500, Error: 0.003048
Iteration: 360-500, Error: 0.002974
Iteration: 380-500, Error: 0.002880
Iteration: 400-500, Error: 0.002843
Iteration: 420-500, Error: 0.002773
Iteration: 440-500, Error: 0.002731
Iteration: 460-500, Error: 0.002695
Iteration: 480-500, Error: 0.002674
Iteration: 500-500, Error: 0.002620
Apprentissage Termine
```


Figure 6 - Evolution de l'erreur lors de l'apprentissage

Testing et Evaluation :

Le testing de notre réseau consiste par deux étapes :

1. Prévoir les résultats des opinions dans l'échantillon du test.

2. Comparer les résultats trouvés avec les résultats de base.

```
from sklearn import metrics

# Taux de la classification correcte (Comparaison entre sortie calculée et réelle)
metrics.accuracy_score(testYF, targetTestRF)
```

- **Taux de classification correcte :**

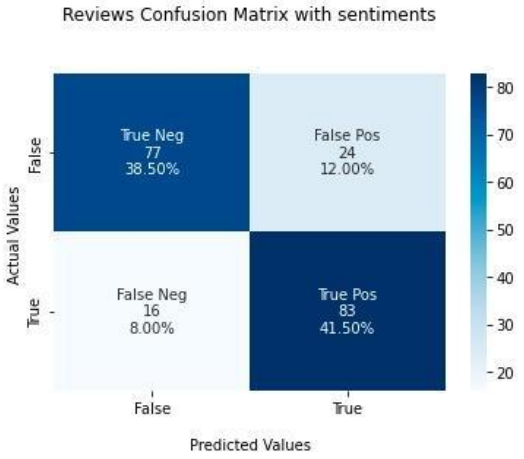
le taux de classification correcte pour ce pmc est: **75%**

```
from sklearn import metrics

# Taux de la classification correcte (Comparaison entre sortie calculée et réelle)
metrics.accuracy_score(testYF, targetTestRF)
```

```
0.75
```

- **Matrice de confusion :**



Deuxième application :

Description des données :

L'ensemble de données est à 10 min pendant environ 4,5 mois. Les conditions de température et d'humidité de la maison ont été surveillées avec un réseau de capteurs sans fil ZigBee. Chaque nœud sans fil a transmis les conditions de température et d'humidité environ 3,3 min. Ensuite, les données sans fil ont été moyennées sur des périodes de 10 minutes. Les données énergétiques ont été enregistrées toutes les 10 minutes avec des compteurs d'énergie m-bus. La météo de la station météorologique de l'aéroport la plus proche (aéroport de Chievres, Belgique) a été téléchargée à partir d'un ensemble de données publiques de Reliable Prognosis (rp5.ru)

et fusionnée avec les ensembles de données expérimentaux à l'aide de la colonne date et heure.

Informations sur les attributs :

- Date heure année-mois-jour heure:minute:second.
- Appareils, consommation d'énergie en Wh.
- Lumières, consommation d'énergie des luminaires de la maison en Wh.
- T1, Température dans la cuisine, en Celsius.
- RH_1, Humidité dans la cuisine, en %.
- T2, Température dans le salon, en Celsius.
- RH_2, Humidité dans le salon, en %.
- T3, Température dans la buanderie.
- RH_3, Humidité dans la buanderie, en %.
- T4, Température dans le bureau, en Celsius.
- RH_4, Humidité dans le bureau, en %.
- T5, Température dans la salle de bain, en Celsius.
- RH_5, Humidité dans la salle de bain, en %.
- T6, Température à l'extérieur du bâtiment (côté nord), en Celsius.
- RH_6, Humidité à l'extérieur du bâtiment (côté nord), en %.
- T7, Température salle de repassage, en Celsius.
- RH_7, Humidité salle de repassage, en %.
- T8, Température chambre ado 2, en Celsius.
- RH_8, Humidité chambre ado 2, en %.
- T9, Température chambre parents, en Celsius.
- RH_9, Humidité dans la chambre des parents, en %.
- To, Température extérieure (depuis la station météo de Chievres), en Celsius.
- Pression (depuis la station météo de Chievres), en mm Hg.
- RH_out, Humidité extérieure (depuis la station météo de Chievres), en %.
- Vitesse du vent (depuis la station météo de Chievres), en m/s.
- Visibilité (depuis la station météo de Chievres), en km energy_target.
- Point de rosée (depuis la station météo de Chievres), °C.

- rv1, Variable aléatoire 1, non dimensionnelle.
- rv2, Variable aléatoire 2, non dimensionnelle.

Algorithme d'apprentissage adopté :

SVM (Support Vector Machine ou Machine à vecteurs de support) : Les SVMs sont une famille d'algorithmes d'apprentissage automatique qui permettent de résoudre des problèmes tant de classification que de régression ou de détection d'anomalie. Ils sont connus pour leurs solides garanties théoriques, leur grande flexibilité ainsi que leur simplicité d'utilisation même sans grande connaissance de data mining.

Les SVMs ont été développés dans les années 1990. Comme le montre la figure ci-dessous, leur principe est simple : ils ont pour but de séparer les données en classes à l'aide d'une frontière aussi « simple » que possible, de telle façon que la distance entre les différents groupes de données et la frontière qui les sépare soit maximale. Cette distance est aussi appelée « marge » et les SVMs sont ainsi qualifiés de « séparateurs à vaste marge », les « vecteurs de support » étant les données les plus proches de la frontière.

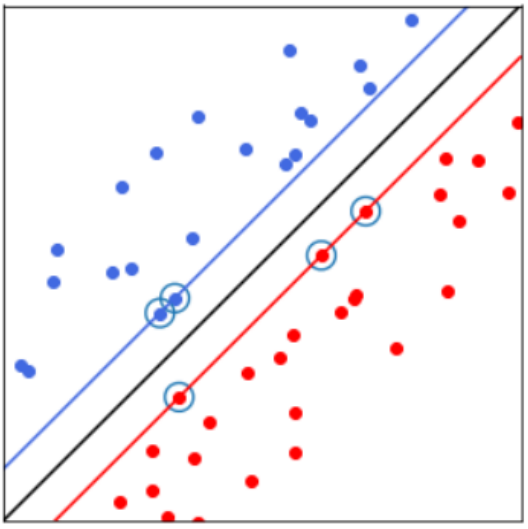


Figure 4 Une représentation du SVM.

Dans cet espace à deux dimensions, la « frontière » est la droite noire, les « vecteurs de support » sont les points entourés (les plus

proche de la frontière) et la « marge » est la distance entre la frontière et les droites bleue et rouge.

Cette notion de frontière suppose que les données soient linéairement séparables, ce qui est rarement le cas. Pour y pallier, les SVMs reposent souvent sur l'utilisation de « noyaux ». Ces fonctions mathématiques permettent de séparer les données en les projetant dans un feature space (un espace vectoriel de plus grande dimension, voir figure ci-dessous). La technique de maximisation de marge permet, quant à elle, de garantir une meilleure robustesse face au bruit – et donc un modèle plus généralisable.

Les SVMs permettent de projeter les données dans une espace de plus grande dimension via une fonction noyau pour les séparer linéairement.

Les SVMs sont utilisés dans une variété d'applications (bioinformatique, recherche d'informations, vision par ordinateur, finance, etc.) notamment parce qu'à la différence des réseaux de neurones, on peut les utiliser sans comprendre leur fonctionnement : il existe des jeux d'hyperparamètres par défaut – pour la classification, la régression ou la détection d'anomalie – qui fonctionnent dans l'immense majorité des cas. C'est un de leurs principaux avantages. Ces hyperparamètres sont, par ailleurs, en nombre très réduit : ils se limitent au choix de la technique de régularisation (de type lasso ou encore régularisation RKHS*, une méthode spécifique aux SVMs) et au choix du noyau (noyaux polynomiaux, Sobolev, RBF**…). Concernant les algorithmes SVMs, citons le kernel ridge regression pour la régression ou le one class SVM pour la détection d'anomalie.

Mais c'est quoi un kernel RBF ?

Les noyaux RBF sont la forme de noyaux la plus généralisée et sont l'un des noyaux les plus largement utilisés en raison de sa similitude avec la distribution gaussienne. La fonction noyau RBF pour deux points X_1 et X_2 calcule la similarité ou leur proximité. Ce noyau peut être représenté mathématiquement comme suit :

$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2\sigma^2})$$

Dans notre application, on a adopté deux approches, la première concerne l'algorithme SVM, avec l'importation d'une bibliothèque qui fait le rôle du kernel RBF, et dans une deuxième approche on a adopté l'algorithme Kmeans avec le kernel RBF, mais cette fois-ci en utilisant une classe et une fonction RBF développée par nous-même.

Qu'est-ce que Kmeans ?

K-means (ou K-moyennes) : C'est l'un des algorithmes de clustering les plus répandus. Il permet d'analyser un jeu de données caractérisées par un ensemble de descripteurs, afin de regrouper les données "similaires" en groupes (ou clusters).

La similarité entre deux données peut être inférée grâce à la "distance" séparant leurs descripteurs ; ainsi deux données très similaires sont deux données dont les descripteurs sont très proches. Cette définition permet de formuler le problème de partitionnement des données comme la recherche de K "données prototypes", autour desquelles peuvent être regroupées les autres données.

Ces données prototypes sont appelés centroïdes ; en pratique l'algorithme associe chaque donnée à son centroïde le plus proche, afin de créer des clusters. D'autre part, les moyennes des descripteurs des données d'un cluster, définissent la position de leur centroïde dans l'espace des descripteurs : ceci est à l'origine du nom de cet algorithme (K-moyennes ou K-means en anglais).

Après avoir initialisé ses centroïdes en prenant des données au hasard dans le jeu de données, K-means alterne plusieurs fois ces deux étapes pour optimiser les centroïdes et leurs groupes :

1. Regrouper chaque objet autour du centroïde le plus proche.

2. Replacer chaque centroïde selon la moyenne des descripteurs de son groupe.

L’architecture de cet algorithme est comme suivie :

Entrée:

Nombre de groupements/clusters à former (K)La bases des exemples d’apprentissage.

Début:

Prendre aléatoirement K exemples de la base des exemples d’apprentissage comme vecteurs prototypes.

Répéter:

Affecter chaque exemple au vecteur prototype le plus proche-Recalculer le vecteur prototype lié à chaque groupement Jusqu’à convergence (prototypes fixes ou nombre d’itérations maximum atteint).

Fin

Solution proposée :

Importation des données :

On doit d’abord lire le fichier contenant le dataset :

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...	T9	RH_9	T_out	Press_mm_
0	2016-01-11 17:00:00	60	30	19.890000	47.596667	19.200000	44.790000	19.790000	44.730000	19.000000	...	17.033333	45.5300	6.600000	73
1	2016-01-11 17:10:00	60	30	19.890000	46.693333	19.200000	44.722500	19.790000	44.790000	19.000000	...	17.066667	45.5600	6.483333	73
2	2016-01-11 17:20:00	50	30	19.890000	46.300000	19.200000	44.626667	19.790000	44.933333	18.926667	...	17.000000	45.5000	6.366667	73
3	2016-01-11 17:30:00	50	40	19.890000	46.066667	19.200000	44.590000	19.790000	45.000000	18.890000	...	17.000000	45.4000	6.250000	73
4	2016-01-11 17:40:00	60	40	19.890000	46.333333	19.200000	44.530000	19.790000	45.000000	18.890000	...	17.000000	45.4000	6.133333	73
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
19730	2016-05-27 17:20:00	100	0	25.566667	46.560000	25.890000	42.025714	27.200000	41.163333	24.700000	...	23.200000	46.7900	22.733333	75
19731	2016-05-27 17:30:00	90	0	25.500000	46.500000	25.754000	42.080000	27.133333	41.223333	24.700000	...	23.200000	46.7900	22.600000	75
19732	2016-05-27 17:40:00	270	10	25.500000	46.596667	25.628571	42.768571	27.050000	41.690000	24.700000	...	23.200000	46.7900	22.466667	75
19733	2016-05-27 17:50:00	420	10	25.500000	46.990000	25.414000	43.036000	26.890000	41.290000	24.700000	...	23.200000	46.8175	22.333333	75
19734	2016-05-27 18:00:00	430	10	25.500000	46.600000	25.264286	42.971429	26.823333	41.156667	24.700000	...	23.200000	46.8450	22.200000	75

19735 rows × 29 columns

Extrait de notre dataset

On décrit notre dataset avec la fonction describe() :

Décrire notre dataset permet de visualiser les valeurs calculées ci-dessous :

	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	...	
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	...	1973
mean	97.694958	3.801875	21.686571	40.259739	20.341219	40.420420	22.267611	39.242500	20.855335	39.026904	...	1
std	102.524891	7.935988	1.606066	3.979299	2.192974	4.069813	2.006111	3.254576	2.042884	4.341321	...	
min	10.000000	0.000000	16.790000	27.023333	16.100000	20.463333	17.200000	28.766667	15.100000	27.660000	...	1
25%	50.000000	0.000000	20.760000	37.333333	18.790000	37.900000	20.790000	36.900000	19.530000	35.530000	...	1
50%	60.000000	0.000000	21.600000	39.656667	20.000000	40.500000	22.100000	38.530000	20.666667	38.400000	...	1
75%	100.000000	0.000000	22.600000	43.066667	21.500000	43.260000	23.290000	41.760000	22.100000	42.156667	...	2
max	1080.000000	70.000000	26.260000	63.360000	29.856667	56.026667	29.236000	50.163333	26.200000	51.090000	...	2

8 rows × 28 columns

On visualise la corrélation entre la variable :

On utilise le diagramme de corrélation pour sélectionner les meilleures caractéristiques pour le modèle.

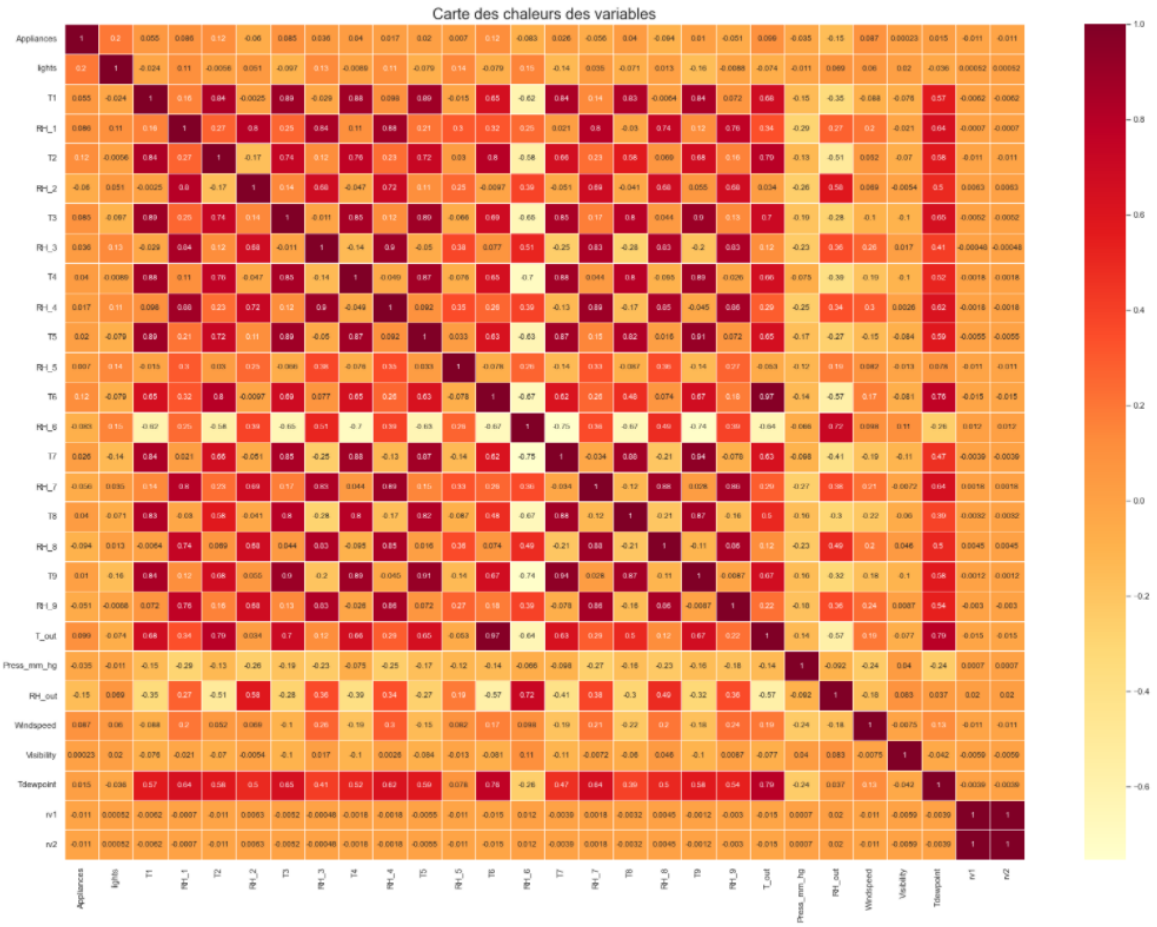


Figure 1 Carte des chaleurs des variables avant Selection.

Feature selection :

Pour la sélection des fonctionnalités, nous mettons en œuvre l'élimination à l'aide d'une corrélation soutenue par l'intuition métier.

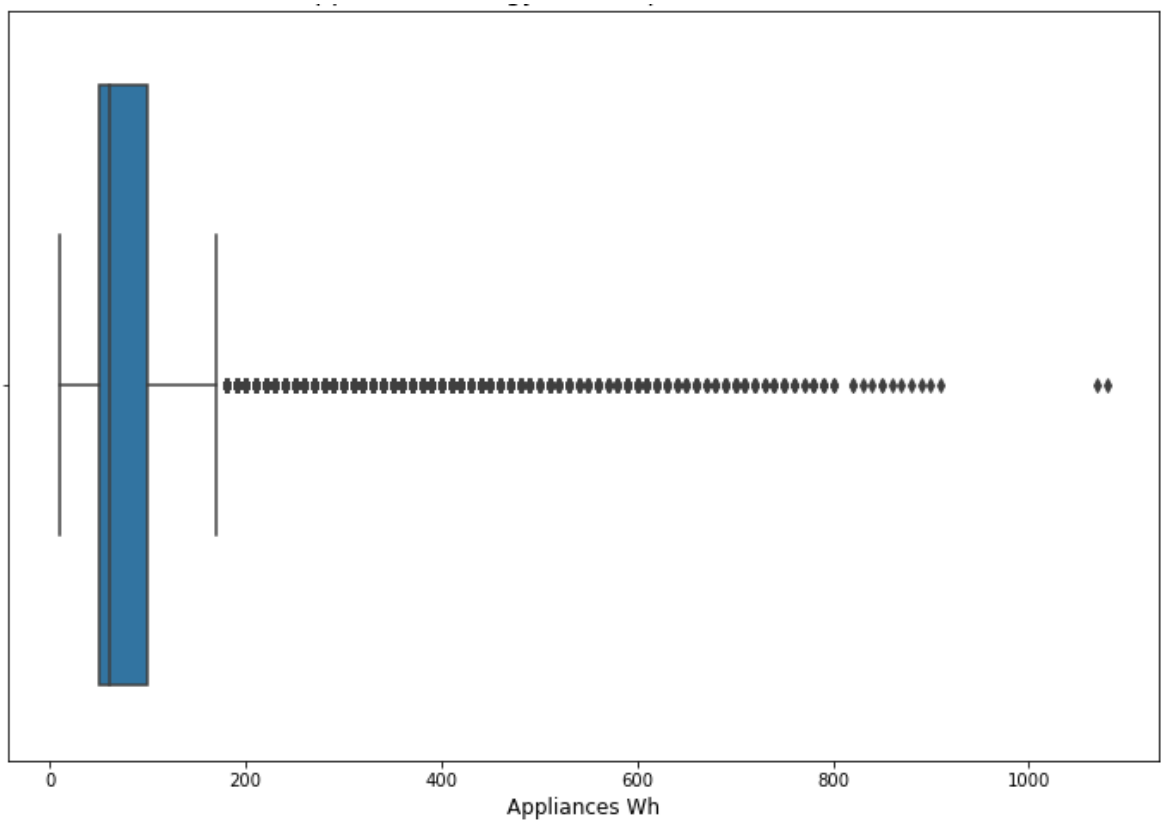


Figure Description de l'Appliance.

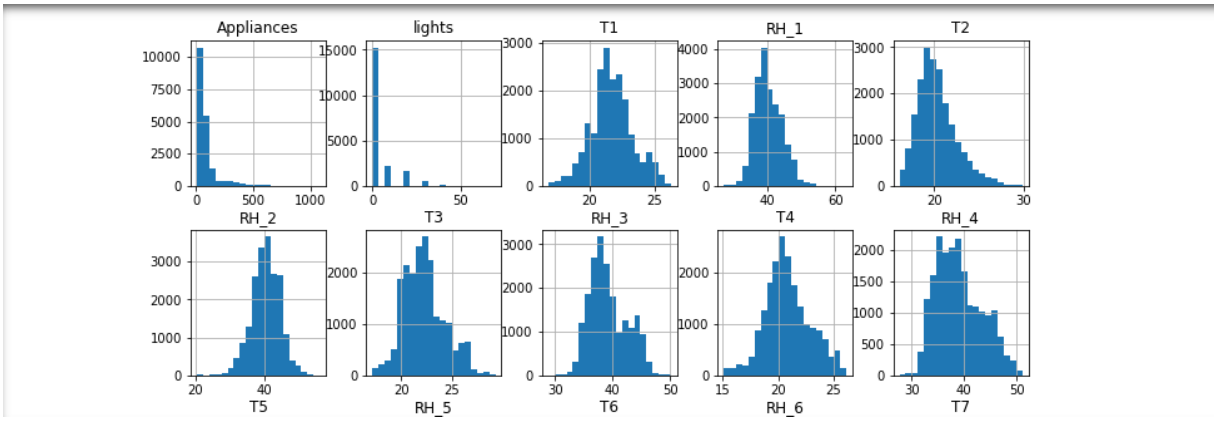


Figure Description de quelque colonne du tableau.

Voici la corrélation des variables sélectionné :

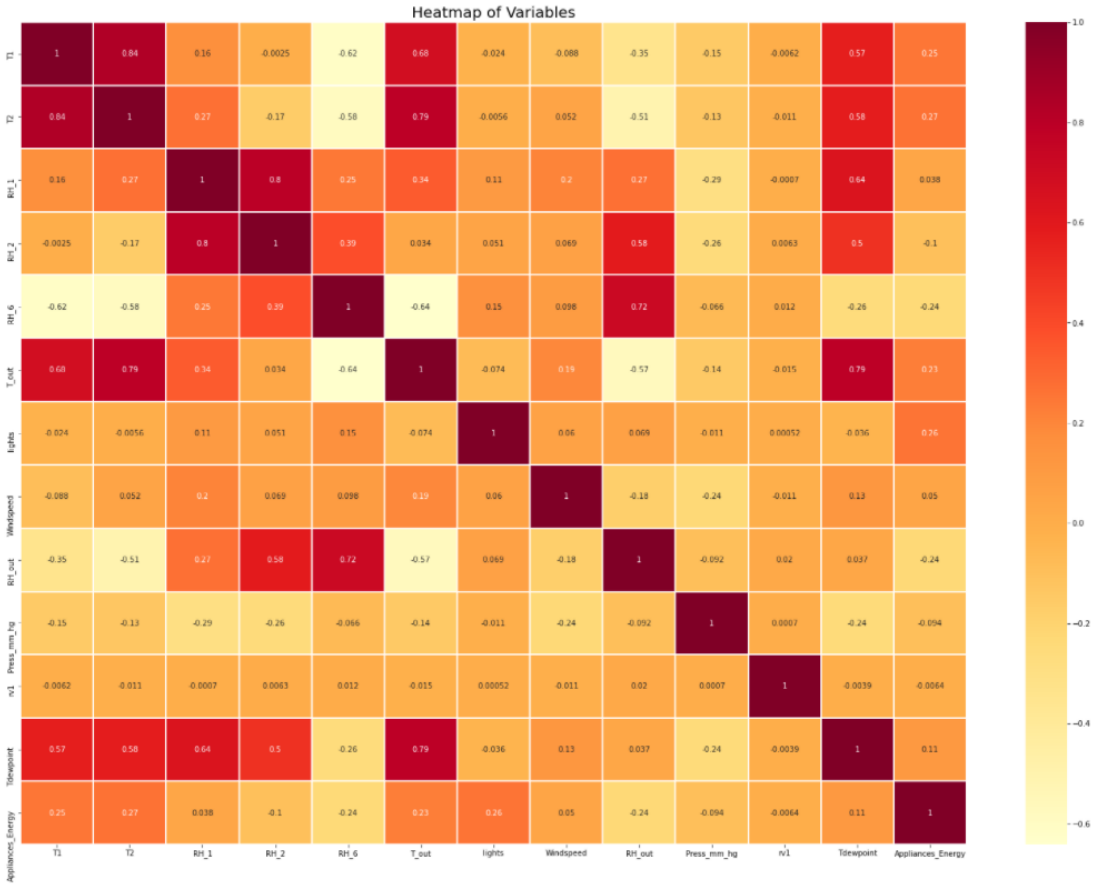


Figure 1 Carte des chaleurs des variables après la Selection.

Répartition des données en des données d'entrainement et test :

On divise les données comme suit : 70% pour l'entrainement et on laisse 30% pour le test.

```
Shape of xTrain Set (13814, 12)
Shape of yTrain Set (13814, 1)

Shape of xTest Set (5921, 12)
Shape of yTest Set (5921, 1)
```

Répartition selon la norme cross-validation split (6 ou 5) :

La technique dite "k-fold cross-validation", permet de diviser la base des exemples d'apprentissage en k échantillons. Dans le cas simple les échantillons de même taille. k-1 groupements sont utilisés et le dernier groupe pour l'évaluation. Cette

procédure est répétée pour tous les autres groupes, la performance est la moyenne des k scores.

```
Shape of xTrain Set (17762, 12)
Shape of yTrain Set (17762, 1)

Shape of xTest Set (1973, 12)
Shape of yTest Set (1973, 1)
```

Jetons un coup d'œil au décompte des classes pour vérifier la balance :

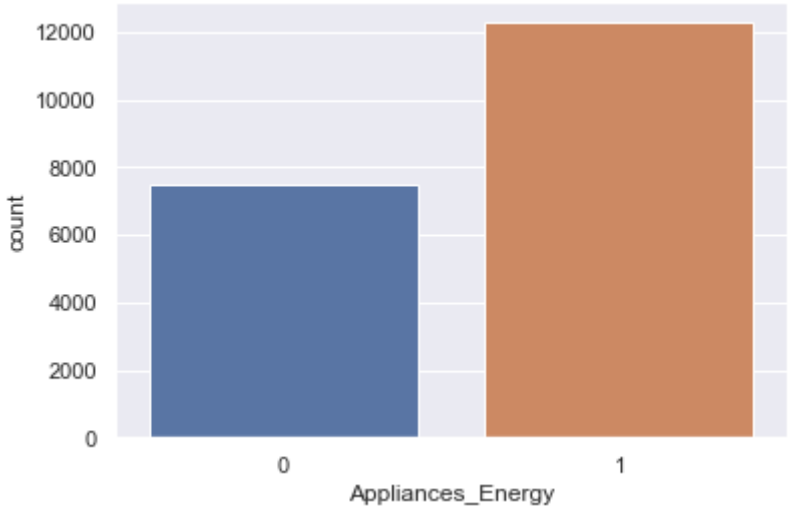


Figure Comparaison entre les Appliance qui ont des valeur inférieur et supérieur a 60

SVM :

Ici on veut créer une fonction pour prendre le noyau en entrée et exécuter le modèle et fournir des métriques pour tout type de SVM.

On vas utiliser RBF Kernel :

```
********** Result for Train-Test Split **********

 Kernel:  rbf
Train score: 67.9238 %
Test score: 68.1135 %

Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.30      0.41      2192
           1       0.69      0.91      0.78      3729

    accuracy                           0.68      5921
   macro avg       0.67      0.60      0.59      5921
weighted avg       0.67      0.68      0.64      5921

Confusion Matrix:
[[ 651 1541]
 [ 347 3382]]
********** Result for Cross-Validation **********

 Kernel:  rbf
Train score: 67.8921 %
Test score: 73.2894 %

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       527
           1       0.73      1.00      0.85      1446

    accuracy                           0.73      1973
   macro avg       0.37      0.50      0.42      1973
weighted avg       0.54      0.73      0.62      1973

Confusion Matrix:
[[   0  527]
 [   0 1446]]
```

Autre solution :

Une autre approche pour résoudre ce problème concerne l'utilisation de l'algorithme de Kmeans et un kernel basé sur le RBF.

Ici on vient de définir la fonction RBF qui calcule les distances, et la fonction Kmeans qui implémente l'algorithme et qui prend en paramètre le kernel de RBF.

```
nbreClasses = 2
def rbf(x, c, s):
    distance=np.linalg.norm(np.array(x)-np.array(c))
    return 1/np.exp(-distance/(2*s**2))

def kmeans(X, k, itersmax=100):

    clusters=X[np.random.choice(range(len(X)), k, replace=False)]
    converged=False

    iter=0
    while(not converged)and(iter<itersmax):

        cluster_list=[[]for i in range(len(clusters))]
        for x in X:
            distances=[]
            for c in clusters:
                distances.append(np.linalg.norm(np.array(x)-np.array(c)))
            cluster_list[int(np.argmin(distances))].append(x)

        cluster_list=list((filter(None, cluster_list)))

        prevClusters=clusters.copy()
        clusters=[]

        for j in range(len(cluster_list)):
            clusters.append(np.mean(cluster_list[j], axis=0))

        diff=np.abs(np.sum(prevClusters)-np.sum(clusters))
        print('Test KMEANS :', diff)
        converged=(diff==0)

        iter+=1

    stds=[np.std(x) for x in cluster_list]
    return np.array(clusters), stds
```

RBFNet :

On va clairement l'implémentation de la classe RBFNet, dont on définit 4 méthodes __init__() pour l'initialisation, rbf states, fit et predict pour la prédiction.

```
class RBFNet(object):

    def __init__(self, k=2, lr=0.01, epochs=100, rbf=rbf, withstds=True):
        self.k=k
        self.lr=lr
        self.epochs=epochs
        self.rbf=rbf
        self.withstds=withstds

        self.w=np.random.randn(nbreClasses,self.k)
        self.b=np.random.randn(nbreClasses)

    def rbf_states(self, X, clusters, stds):
        rbfs=[]
        for x in X:
            rbfs.append([rbf(x, c, s)for(c, s)in zip(clusters, stds)])
        return np.array(rbfs)

    def fit(self, X, y):
```

Ici on exploite l'algorithme Kmeans qui utilise le kernel RBF sur les données de consommation d'énergie qu'on a déjà préparées :

```
Test KMEANS : 0.48500470663930173
Test KMEANS : 0.6632897546405729
Test KMEANS : 0.01847916228143731
Test KMEANS : 0.04089782356459182
Test KMEANS : 0.12006921601641807
Test KMEANS : 0.07432387701919652
Test KMEANS : 0.23720533155938028
Test KMEANS : 0.25823769296039245
Test KMEANS : 0.1522490124043543
Test KMEANS : 0.42742749209719477
Test KMEANS : 0.44625869565061294
Test KMEANS : 0.6430677219723293
Test KMEANS : 1.3749755676981295
Test KMEANS : 0.7254513386469625
Test KMEANS : 0.8525628855459217
Test KMEANS : 1.045980654336745
Test KMEANS : 0.5935836839053081
Test KMEANS : 0.3308880679142021
Test KMEANS : 0.35820627579960274
Test KMEANS : 0.009202139135595644
Test KMEANS : 0.34207948926996323
Test KMEANS : 0.18439775687875226
Test KMEANS : 0.35143127266928786
Test KMEANS : 0.2211312556428311
Test KMEANS : 0.2809858843465918
Test KMEANS : 0.3333405706543999
Test KMEANS : 0.0593824393581599
Test KMEANS : 0.17076920898762182
Test KMEANS : 0.28324237298511434
Test KMEANS : 0.19561636437356356
Test KMEANS : 0.08247539278090699
Test KMEANS : 0.013897276374336798
Test KMEANS : 0.010933459430816583
Test KMEANS : 0.030967683760536602
Test KMEANS : 0.019608273101766827
Test KMEANS : 0.018718250656093005
Test KMEANS : 0.0
0 RBF training : mean error :0.41
20 RBF training : mean error :0.39
40 RBF training : mean error :0.39
60 RBF training : mean error :0.39
80 RBF training : mean error :0.39
100 RBF training : mean error :0.39
120 RBF training : mean error :0.39
140 RBF training : mean error :0.39
160 RBF training : mean error :0.39
180 RBF training : mean error :0.39
```

Après avoir prédire l'approximation énergétique après l'apprentissage, les résultats de l'apprentissage et du test sont les suivants :

```
Evaluation :
Exemples Test : (5921, 12)
---------------------------------- :
--------------------   Prediction   ------------------
[0 0 0 ... 1 1 0]

--------------------   true   ------------------
[1 0 1 ... 0 0 1]
              precision    recall  f1-score   support

           0       0.29      0.42      0.34      2192
           1       0.54      0.40      0.45      3729

    accuracy                           0.40      5921
   macro avg       0.41      0.41      0.40      5921
weighted avg       0.44      0.40      0.41      5921
```

Conclusion :

Le Machine Learning est massivement utilisé pour la Data Science et l'analyse de données. Il permet de développer, de tester et d'appliquer des algorithmes d'analyse prédictive sur différents types de données afin de prédire le futur.

En automatisant le développement de modèle analytique, le Machine Learning permet d'accélérer l'analyse de données et de la rendre plus précise. Il permet d'assigner aux machines des tâches au cœur de l'analyse de données comme la classification, le clustering ou la détection d'anomalie.

Les algorithmes ingèrent les données et délivrent des inférences statistiques, et peuvent s'améliorer de manière autonome au fil du temps. Lorsqu'ils détectent un changement dans les données, ils sont capables de prendre des décisions sans intervention humaine.

Pour l'heure, un humain reste toutefois nécessaire pour passer en revue les résultats des analyses produites par les algorithmes de Machine Learning. Son rôle est de donner du sens à ces résultats, ou encore de s'assurer que les données traitées par l'algorithme ne soient ni biaisées ni altérées.

Bibliographie :

https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences

https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction

https://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support

https://dataanalyticspost.com/Lexique/k-means-ou-k-moyennes/