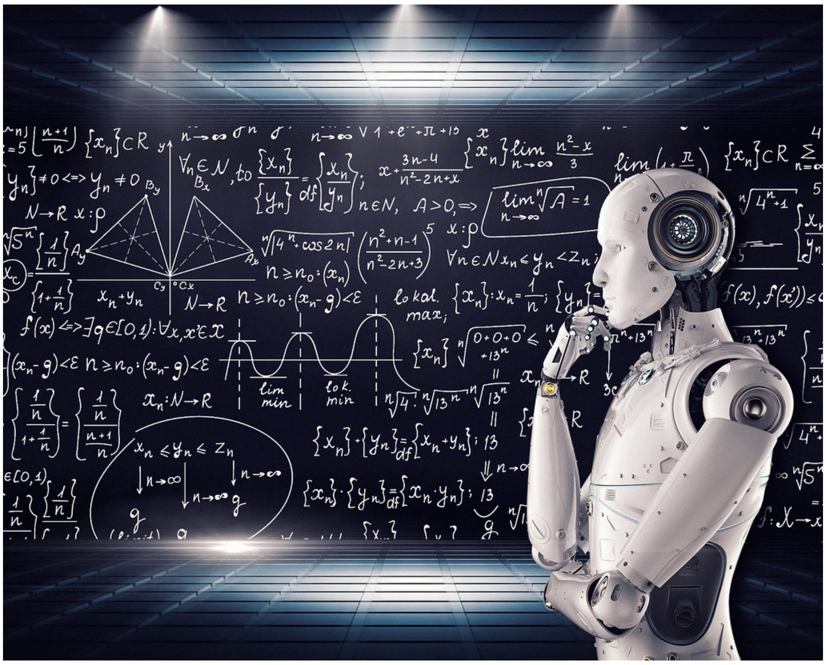
ROYAUME DU MAROC
UNIVERSITE ABDELMALEK ESSAADI
FACULTE DES SCIENCES ET TECHNIQUES
TANGER

المملكة المغربية
جامعة عبد المالك السعدي
كلية العلوم والتقنيات
طنجة

Logiciels et systèmes intelligents (LSI)
Département Informatique



Rapport de mini projet de module

IA

Réseaux de neurones artificiels

Réalisé par :
Mahjoubi redwane
Outaleb Asmaa
Ait abbou Samir

Encadré par : MR.M'hamed AIT KBIR

Année Universitaire 2021-2022

1. réalisée par laTex

Remerciement

C'est avec un réel plaisir que nous adressons les plus sincères remerciements à notre chère professeur **MR.M'hamed AIT KBIR** ,

pour leurs conseils précieux, leur soutien et leur compréhension à ses étudiants.

Merci infiniment

Table des matières

Table des figures

Introduction générale

Les réseaux de neurones, communément appelés des réseaux de neurones artificiels sont des imitations simples des fonctions d'un neurone dans le cerveau humain pour résoudre des problématiques d'apprentissage de la machine (Machine Learning)

Le neurone est une unité qui est exprimée généralement par une fonction sigmoïde.

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

FIGURE 1 – une fonction sigmoïd

Chapitre 1

Exercice 1 :

1.1 BUT :

Utilisation des réseaux multi-couches pour l'analyse des sentiments des phrases issues d'une base d'exemples qui contient des phrases étiquetées avec un sentiment positif ou négatif, voir la base d'exemples : https ://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences

1.2 Solution :

Importer les packages requis :

1- L'importation des données :

Dans un premier temps on importe les bibliothèques dont on aura besoin :

Entrée [3]:

```
import pandas as pd
import string
import numpy as np
#imports
import string # from some string manipulation tasks
import nltk # natural language toolkit
import re # regex
from string import punctuation # solving punctuation problems
from nltk.corpus import stopwords # stop words in sentences
from nltk.stem import WordNetLemmatizer # For stemming the sentence
from nltk.stem import SnowballStemmer # For stemming the sentence

#from contractions import contractions_dict # to solve contractions
#from autocorrect import Speller #correcting the spellings
from spellchecker import SpellChecker

import matplotlib.pyplot as plt
import seaborn as sns
import pylab as pl

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

# Import sklearn
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import scale
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn import metrics
import seaborn as sns  # Bibliothèque pour la visualisation des données
```

Pandas:C'est une bibleotheque qui nous permettre de recuperer les donnees d'un fichier texte dans ce cas on a utiliser une fonction read_table dont ces parametres on a : le nom de notre fichier, delimiter ,header et nom des colonnes de notre table pour diviser les donnees en deux colonnes Review: commentaires et sentiment: exprimer par 0 ou 1

Entrée [6]:
```
train_data = pd.read_table('imdb_labelled.txt',
                  delimiter='\t',
                  header=None,
                  names=['Review', 'Sentiment'])
train_data
```

Out[6]:

	Review	Sentiment
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
...	...	...
743	I just got bored watching Jessice Lange take h...	0
744	Unfortunately, any virtue in this film's produ...	0
745	In a word, it is embarrassing.	0
746	Exceptionally bad!	0
747	All in all its an insult to one's intelligence...	0

748 rows × 2 columns

La fonction head() permettre d'afficher les cinqs premiers lignes de l'objet train-data.

Entrée [7]:
```
train_data.head()
```

Out[7]:

	Review	Sentiment
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

Value_counts() permet de renvoyer dans l'ordre decroissant le nombre de valeurs pour chaque sentiment c'est a dire 386 element avec un sentiment egale 1 et 362 element pour un sentiment de valeur 0

Entrée [8]:
```
train_data['Sentiment'].value_counts()
```

Out[8]:
```
1    386
0    362
Name: Sentiment, dtype: int64
```

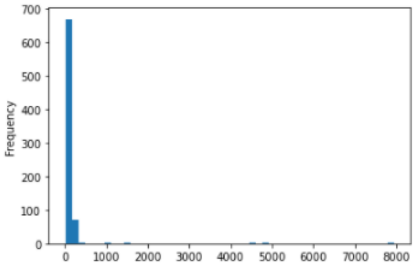
Ici on creer un tableau de longeur dont on met la longeur de chaque commentaire

Entrée [9]:
```
train_data['Length'] = train_data['Review'].apply(len)
```

plot() pour tracer l'histogramme de la longueur des commentaires

Entrée [10]:
```
train_data['Length'].plot(kind = 'hist' , bins = 50)
```

Out[10]: <AxesSubplot:ylabel='Frequency'>

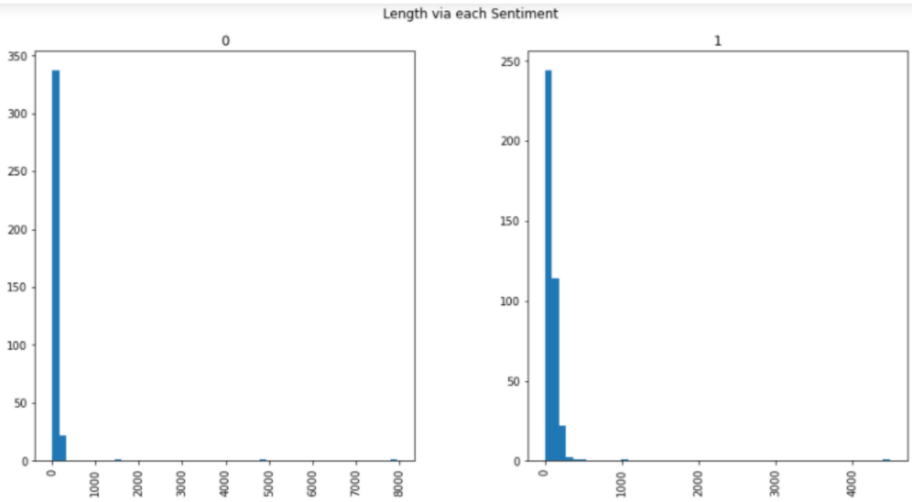


Maintenant on a tracer les histogrammes pour chaqu'un des sentiments

Entrée [11]:
```
ax = train_data.hist(column = 'Length', by = 'Sentiment', bins = 50 , figsize = (14,7));
pl.suptitle('Length via each Sentiment')
```

Out[11]: Text(0.5, 0.98, 'Length via each Sentiment')

Length via each Sentiment



La tokenisation est un moyen qui nous permettre de diviser les chaines en une listes de mots on utilisons Toolkit un langage qui a des fonctions pour la tokenisation on 2 types a utiliser pour la tokenisation l'une pour convertir la phrase entiere en une liste et l'autre pour convertir des mots séparés en jetons

Entrée [10]:
```
def word_tokenize(text):
    """
    :param text:
    :return: list of words
    """
    return nltk.word_tokenize(text)
```

Entrée [11]:
```
train_data['Review'].apply(sentence_tokenize)
```

Out[11]:
```
0      [A very, very, very slow-moving, aimless movie...
1      [Not sure who was more lost - the flat charact...
2      [Attempting artiness with black & white and cl...
3           [Very little music or anything to speak of.]
4      [The best scene in the movie was when Gerardo ...
                             ...
743    [I just got bored watching Jessice Lange take ...
744    [Unfortunately, any virtue in this film's prod...
745                     [In a word, it is embarrassing.]
746                                 [Exceptionally bad!]
747    [All in all its an insult to one's intelligenc...
Name: Review, Length: 748, dtype: object
```

Entrée [12]:
```
train_data['Review'].apply(word_tokenize)
```

Out[12]:
```
0      [A, very, ,, very, ,, very, slow-moving, ,, ai...
1      [Not, sure, who, was, more, lost, -, the, flat...
2      [Attempting, artiness, with, black, &, white, ...
3      [Very, little, music, or, anything, to, speak,...
4      [The, best, scene, in, the, movie, was, when, ...
                             ...
743    [I, just, got, bored, watching, Jessice, Lange...
744    [Unfortunately, ,, any, virtue, in, this, film...
745             [In, a, word, ,, it, is, embarrassing, .]
746                             [Exceptionally, bad, !]
747    [All, in, all, its, an, insult, to, one, 's, i...
```

Entrée [12]:
```
def sentence_tokenize(text):
    """
    take string input and return a list of sentences.
    use nltk.sent_tokenize() to split the sentences.
    """
    return nltk.sent_tokenize(text)
```

Entrée [13]:
```
def word_tokenize(text):
    """
    :param text:
    :return: list of words
    """
    return nltk.word_tokenize(text)
```

Ici on teste la fonction sentence_tokenize() :

Entrée [14]:
```
train_data['Review'].apply(sentence_tokenize)
```

```
---------------------------------------------------------------------------
LookupError                               Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12488/216522243.py in <module>
----> 1 train_data['Review'].apply(sentence_tokenize)

~\anaconda3\lib\site-packages\pandas\core\series.py in apply(self, func, convert_dtype, args, **kwargs)
   4355         dtype: float64
   4356         """
-> 4357         return SeriesApply(self, func, convert_dtype, args, kwargs).apply()
   4358
   4359     def _reduce(

~\anaconda3\lib\site-packages\pandas\core\apply.py in apply(self)
   1041             return self.apply_str()
   1042
-> 1043         return self.apply_standard()
```

Ici on teste la fonction word_tokenize() :

Entrée [51]:
```
train_data['Review'].apply(word_tokenize)
```

Out[51]:
```
0      [A, very, ,, very, ,, very, slow-moving, ,, ai...
1      [Not, sure, who, was, more, lost, -, the, flat...
2      [Attempting, artiness, with, black, &, white, ...
3      [Very, little, music, or, anything, to, speak,...
4      [The, best, scene, in, the, movie, was, when, ...
                             ...
743    [I, just, got, bored, watching, Jessice, Lange...
744    [Unfortunately, ,, any, virtue, in, this, film...
745            [In, a, word, ,, it, is, embarrassing, .]
746                       [Exceptionally, bad, !]
747    [All, in, all, its, an, insult, to, one, 's, i...
Name: Review, Length: 748, dtype: object
```

Dans cette etape on a assuré que chaque lettre est en minuscules afin que le modèle fonctionne de manière équivalente

Entrée [52]:
```
def to_lower(text):
        """
        :param text:
        :return:
            Converted text to lower case as in, converting "Hello" to "hello" or "HELLO" to "hello".
        """
        return text.lower()
```

Testons la fonction to_lower() :

Entrée [51]:
```
train_data['Review'].apply(word_tokenize)
```

Out[51]:
```
0      [A, very, ,, very, ,, very, slow-moving, ,, ai...
1      [Not, sure, who, was, more, lost, -, the, flat...
2      [Attempting, artiness, with, black, &, white, ...
3      [Very, little, music, or, anything, to, speak,...
4      [The, best, scene, in, the, movie, was, when, ...
                             ...
743    [I, just, got, bored, watching, Jessice, Lange...
744    [Unfortunately, ,, any, virtue, in, this, film...
745            [In, a, word, ,, it, is, embarrassing, .]
746                       [Exceptionally, bad, !]
747    [All, in, all, its, an, insult, to, one, 's, i...
Name: Review, Length: 748, dtype: object
```

Dans cette etape on a assuré que chaque lettre est en minuscules afin que le modèle fonctionne de manière équivalente

Entrée [52]:
```
def to_lower(text):
        """
        :param text:
        :return:
            Converted text to lower case as in, converting "Hello" to "hello" or "HELLO" to "hello".
        """
        return text.lower()
```

Testons la fonction to_lower() :

Entrée [53]:
```
train_data['Review'].apply(to_lower)
```

Out[53]:
```
0      a very, very, very slow-moving, aimless movie ...
1      not sure who was more lost - the flat characte...
2      attempting artiness with black & white and cle...
3           very little music or anything to speak of.
4      the best scene in the movie was when gerardo i...
                             ...
743    i just got bored watching jessice lange take h...
744    unfortunately, any virtue in this film's produ...
745                      in a word, it is embarrassing.
746                                  exceptionally bad!
747    all in all its an insult to one's intelligence...
Name: Review, Length: 748, dtype: object
```

Maintenat on a supprimé tous les numéros.

Entrée [54]:
```
def remove_numbers(text):
        """
        take string input and return a clean text without numbers.
        Use regex to discard the numbers.
        """
        output = ''.join(c for c in text if not c.isdigit())
        return output
```

maintenant on teste si la fonction remove_numbers() marche très bien :

Entrée [55]:
```
z = pd.Series(['a1', 'b2e', 'a3'])
z.apply(remove_numbers)
```

Out[55]:
```
0     a
1    be
2     a
dtype: object
```

et maintenant supprimer la ponctuation, car elle n'a aucun sens pour l'analyse des sentiments.

Entrée [56]:
```
def remove_punct(text):

        return ''.join(c for c in text if c not in punctuation)
```

supprimer les mots vides. Les mots vides sont des mots qui n'ont pas de sens et n'aident pas beaucoup dans l'analyse des sentiments.

Entrée [57]:
```
def remove_stopwords(sentence):
        """
        removes all the stop words like "is,the,a, etc."
        """
        stop_words = stopwords.words('english')
        return ' '.join([w for w in nltk.word_tokenize(sentence) if not w in stop_words])
```

Testons la fonction remove_stopwords :

Entrée [58]:
```
print(train_data['Review'][9:11])
train_data['Review'][9:11].apply(remove_stopwords)
```

```
9     Loved the casting of Jimmy Buffet as the scien...
10                 And those baby owls were adorable.
Name: Review, dtype: object
```

Out[58]:
```
9     Loved casting Jimmy Buffet science teacher .
10                         And baby owls adorable .
Name: Review, dtype: object
```

2- Le nettoyage des données :

La fonction preprocess() dont on va faire appel a toutes les fonctions qu'on a définit en haut

Entrée [59]:
```
def preprocess(text):
        lower_text = to_lower(text)
        sentence_tokens = sentence_tokenize(lower_text)
        word_list = []
        for each_sent in sentence_tokens:
            clean_text = remove_numbers(each_sent)
            clean_text1 = remove_punct(clean_text)
            clean_text2 = remove_stopwords(clean_text1)
            word_tokens = word_tokenize(clean_text2)
            for i in word_tokens:
                word_list.append(i)
        return word_list
```

Entrée [60]:
```
sample_data = train_data['Review'].head(5)
print(sample_data)
sample_data.apply(preprocess)
```

```
0    A very, very, very slow-moving, aimless movie ...
1    Not sure who was more lost - the flat characte...
2    Attempting artiness with black & white and cle...
3         Very little music or anything to speak of.
4    The best scene in the movie was when Gerardo i...
Name: Review, dtype: object
```

Out[60]:
```
0    [slowmoving, aimless, movie, distressed, drift...
1    [sure, lost, flat, characters, audience, nearl...
2    [attempting, artiness, black, white, clever, c...
3                     [little, music, anything, speak]
4    [best, scene, movie, gerardo, trying, find, so...
Name: Review, dtype: object
```

Type *Markdown* and LaTeX: α^2

3- Séparation des données :

Et maintenant la séparation des données dont on va utiliser deux tableux l'un pour les positifs tokens et l'autre pour negatifs tokens

Entrée [61]:
```
ps=nltk.stem.porter.PorterStemmer()

revs = train_data['Review'].copy() #liste des phrases
senti= train_data['Sentiment'].copy() #liste des sentiments

i=0
positiveTokens= [] # tokens du review  positif
negativeTokens= [] # tokens du review  negatif


#separerles  positifs et negatifs tokens
for rev in revs:
    if senti[i]==0:
        negativeTokens.append(preprocess(rev)) #tableau de negatif tokens
    else:
        positiveTokens.append(preprocess(rev)) #tableau de positif tokens
    i+=1

positiveTokens=(np.concatenate((positiveTokens), axis=0)) #  list de tout les positifs tokens
negativeTokens=(np.concatenate((negativeTokens), axis=0)) # list de tout les negatifs tokens
```

Notre objectif ici est de calculer la fréquence d'un mot pour bien déterminer est-ce qu'il est positif ou négatif :

Entrée [62]:
```
WordFreq=[]

#calculater frequence d'un mot (Number of occurences of a word in Tokens)
def wordFrequency(word,array):
    wordFreq=np.count_nonzero(array==word)
    return wordFreq

#-------------------- VISUALISATION------------------------
#create dataframe of [word,posFreq,negFreq]
for word in list(set(np.concatenate((positiveTokens,negativeTokens), axis=0))):
    WordFreq.append([word,wordFrequency(word,positiveTokens),wordFrequency(word,negativeTokens)]) #[word,posFreq,negFreq]
wordFreqDF = pd.DataFrame(WordFreq, columns=['word','positive Freq', 'negative Freq'])
wordFreqDF.head()
#-------------------------------------------------------------
```

Out[62]:

	word	positive Freq	negative Freq
0	meaning	1	2
1	artless	0	1
2	rpg	0	1
3	jealousy	1	0
4	bakery	0	1

4- Convertir les lignes en vecteurs [PosF, NegF] :

Dans cette etape on va convertir les commentaires sous forme des vecteurs qui précisent par la suite si ces commentaires sont positifs ou négatifs à partir de la comparaison entre la fréquence des mots positifs et négatifs dans ce commentaire :

Entrée [63]:
```
DataSet=[]

#calculate row of dataset [review,PosF,NegF,sentiment]
def phraseFreq(phrase,sentiment):
    Posfreq=0
    Negfreq=0
    for word in preprocess(phrase):
        Posfreq+=wordFrequency(word,positiveTokens) #la somme des frequences positifs
        Negfreq+=wordFrequency(word,negativeTokens) #la somme des frequences negatifs

    return [phrase,Posfreq,Negfreq,sentiment]
```

```
#convert review(input) to vector(PosF,NegF)
def review2vec(review):
    Posfreq=0
    Negfreq=0
    for word in preprocess(phrase):
        Posfreq+=wordFrequency(word,positiveTokens)
        Negfreq+=wordFrequency(word,negativeTokens)
    return [Posfreq,Negfreq]


def createDataSet():
    i=0
    for rev in revs:
        DataSet.append(phraseFreq(rev,senti[i]))
        i+=1

createDataSet()

DataSet=pd.DataFrame(DataSet, columns=['review','PosF', 'NegF','sentiment'])
DataSet.head(100)
```

Out[63]:

	review	PosF	NegF	sentiment
0	A very, very, very slow-moving, aimless movie ...	94	104	0
1	Not sure who was more lost - the flat characte...	25	34	0
2	Attempting artiness with black & white and cle...	140	214	0
3	Very little music or anything to speak of.	16	24	0
4	The best scene in the movie was when Gerardo i...	123	118	1
...	...	...	...	...
95	Worst hour and a half of my life!Oh my gosh!	3	21	0
96	I had to walk out of the theatre for a few min...	7	18	0
97	I hate movies like that.	40	39	0
97	I hate movies like that.	40	39	0
98	Yeah, the movie pretty much sucked.	95	123	0
99	THERE IS NO PLOT OR STORYLINE!!	5	26	0

100 rows × 4 columns

Entrée [64]:

```
# Données + classes cibles
data   = np.array(DataSet.values[:,1:3], dtype=np.float32)
target = DataSet.values[:,-1]
print(data[0],target[0])
```

```
[ 94. 104.] 0
```

5- Division des données :

Et maintenant on divisons les donnees en deux ,donnee d'entrainement et données de test représentent 10 des données dans notre cas :

Entrée [65]:

```
#Partition aléatoire de l'échantillon
# 10%=100 exemples pour le test
(trainX, testX, trainY, testY) = train_test_split(data, target, test_size=0.1)

len(testY)
```

Out[65]: 75

La transformation des étiquettes ou des sentiments en des vecteurs binaires :

Entrée [66]:

```
# Transformer l'étiquette(sentiments) en un vecteur binaire : 3 --> (0,0,0,1,0,0,0,0,0,0)
trainYC = np.array(list(map(lambda x: [1,0] if x == 1 else [0,1], trainY)))
testYC = np.array(list(map(lambda x: [1,0] if x == 1 else [0,1], testY)))
```

```
# review => network => [0.82,0.18]
```

Out[66]:
```
array([[1, 0],
       [0, 1],
       [0, 1],
       ...,
       [1, 0],
       [0, 1],
       [1, 0]])
```

6- Les réseaux neurones multicouches (PMC) :

Et maintenant la classe principale qui utilise oou bien implemente l'algorithme des reseaux neurones multicouche et qui définit plusieur methodes : init() :fonction d'initialisation sigmoid() et dsigmoid() qui concerne la fonction d'activation d'un neurone forward_pass() pour le Calcul et la mémorisation de l'état de tous les neurones du réseau predict pour le calcul de la sortie du réseau associée à une entrée X (les états des autres neurones ne sont pas mémorisés), quadratic_loss() pour le calcul de l'erreur quadratique moyenne compute_gradient() pour le Calcul des gradients locaux update_with_bloc() pour la Mise à jour par rapport à l'erreur moyenne (relative à un bloc d'exemples) fit() pour l'apprentissage.

Entrée [67]:
```
class MultiLayerPerceptron:

    def __init__(self, arch , alpha = 0.1):
        # poids + biais
        self.W = {}
        self.B = {}

        # Taux d'adaptation
        self.alpha = alpha

        # Architecture :nbre de couches et nombre de neurones par couche
        self.arch = arch

        # Initialisation des poids: valeurs issues d'une distribution normale
        for i in np.arange(1,len(self.arch)):
            # Poids
```


```
        # Initialisation des poids: valeurs issues d'une distribution normale
        for i in np.arange(1,len(self.arch)):
            # Poids
```


```
            w = np.random.randn(self.arch[i], self.arch[i-1])
            self.W[i] = w/np.sqrt(self.arch[i])
            # Bias
            b = np.random.randn(self.arch[i],1)
            self.B[i] = b/np.sqrt(self.arch[i])


    def sigmoid(self, x):
        return 1.0/(1 + np.exp(-x))


    def dsigmoid(self, x): # x correspond ici à sigmoid(uj(t)), voir le cours
        return x * (1 - x)


    #Calcul et mémorisation de l'état de tous les neurones du réseau
    def forward_pass(self, x):
        a = np.atleast_2d(x).T

        stats = {}
        stats[0] = a
        for layer in np.arange(1, len(self.arch)):
            a = self.sigmoid(np.dot(self.W[layer], a) + self.B[layer])
            stats[layer] = a
        return stats


    #Sortie du réseau associée à une entrée X (les états des autres neurones ne sont pas mémorisés)
    def predict(self, X):
        a = np.atleast_2d(X).T
        for layer in np.arange(1, len(self.arch)):
            a = self.sigmoid(np.dot(self.W[layer], a) + self.B[layer])
        return a
```

```
    #Sortie du réseau associée à une entrée X (les états des autres neurones ne sont pas mémorisés)
    def predict(self, X):
        a = np.atleast_2d(X).T
        for layer in np.arange(1, len(self.arch)):
            a = self.sigmoid(np.dot(self.W[layer], a) + self.B[layer])
        return a


    #Calcul de l'erreur quadratique moyenne
    def quadratic_loss(self, X, Y):
        Y = np.atleast_2d(Y).T
        predictions = self.predict(X)
        n = X.shape[0]
        loss = (1/n) * 0.5 * np.sum((predictions - Y) ** 2)
        return loss


    #Calcul des gradients locaux
    def compute_gradient(self, x, y):

        L = len(self.arch) - 1 # indice de la couche de sortie
        # Gradients
        Gw = {}
        Gb = {}
        A = self.forward_pass(x)
        # Les vecteurs delta
        D = {}
        y = np.atleast_2d(y).T
        deltaL = (A[L] - y) * self.dsigmoid(A[L])
        D[L] = deltaL # Pour la sortie

        # Calculer les vecteurs delta des autres couches en utilisants les vecteurs delta de la couche suivante
        for l in np.arange(L-1, 0, -1):
            D[l] = (self.W[l+1].T.dot(D[l+1])) * self.dsigmoid(A[l])
```

Maintenant on entraine notre modèle à travers l'initialisation et l'apprentissage :

Entrée [68]:

```
# Initialisation et apprentissage
# trainX.shape[1]
# testY
pmc = MultiLayerPerceptron(arch=[trainX.shape[1],15,15,2], alpha=0.1)
(errs, iter_fin) = pmc.fit(trainX, trainYC, iterations=500, bloc_size=5, error_min=0.00001, displayPeriod=20)
```

```
C:\Users\nessm\AppData\Local\Temp/ipykernel_13800/1717125781.py:25: RuntimeWarning: overflow encountered in exp
  return 1.0/(1 + np.exp(-x))
```

```
Itération: 0-500, Erreur: 0.301148
Itération: 20-500, Error: 0.114453
Itération: 40-500, Error: 0.127220
Itération: 60-500, Error: 0.126362
Itération: 80-500, Error: 0.119776
Itération: 100-500, Error: 0.104387
Itération: 120-500, Error: 0.117441
Itération: 140-500, Error: 0.125834
Itération: 160-500, Error: 0.098903
Itération: 180-500, Error: 0.115985
Itération: 200-500, Error: 0.105444
Itération: 220-500, Error: 0.102406
Itération: 240-500, Error: 0.109162
Itération: 260-500, Error: 0.104526
Itération: 280-500, Error: 0.113659
Itération: 300-500, Error: 0.109934
Itération: 320-500, Error: 0.109928
Itération: 340-500, Error: 0.105818
Itération: 360-500, Error: 0.108382
Itération: 380-500, Error: 0.112309
Itération: 400-500, Error: 0.110223
Itération: 420-500, Error: 0.102715
```

On teste notre modèle :

Entrée [69]:
```
# Test pour un exemple
# data.shape[0]
randIndex = np.random.randint(0,data.shape[0]-1,1)[0]
# print('Exemple : '+str(randIndex)+', classe réelle : '+str(target[randIndex]))
print(testX[7])
print(testY[7])
# # print(data[randIndex])
print('Sortie prédite : \n'+str(pmc.predict(testX[7]))+')' )
# testY
```

```
[95. 75.]
1
Sortie prédite :
[[0.91744561]
 [0.09266412]])
```

```
C:\Users\nessm\AppData\Local\Temp/ipykernel_13800/1717125781.py:25: RuntimeWarning: overflow encountered in exp
  return 1.0/(1 + np.exp(-x))
```

On effectue une comparaison entre la sortie calculée et la sortie réelle :

Entrée [1]:
```
targetTestR = ['']*(np.array(testY).shape[0])

# targetTestR
for index in range(testX.shape[0]):
    o = np.round(pmc.predict(testX[index]),0)[:,0].astype(int)
    if((o==np.array([1,0])).all()):
        targetTestR[index] = 1
    elif((o==np.array([0,1])).all()):
        targetTestR[index] = 0
```

```
# Sortie calculée et sortie réelle pour la base de test
targetTestRF=list(map(lambda x: '1' if x == 1 else '0', targetTestR))
# print(targetTestR)
testYF=list(map(lambda x: '1' if x == 1 else '0', testY))
print(testYF)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12488/2185910414.py in <module>
----> 1 targetTestR = ['']*(np.array(testY).shape[0])
      2
      3 # targetTestR
      4 for index in range(testX.shape[0]):
      5     o = np.round(pmc.predict(testX[index]),0)[:,0].astype(int)

NameError: name 'np' is not defined
```

Et en fin , on mesure la performance à travers le calcul du taux de classification correcte :

Entrée [2]:
```
# Taux de la classification correcte
metrics.accuracy_score(testYF, targetTestRF)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12488/1300573210.py in <module>
      1 # Taux de la classification correcte
----> 2 metrics.accuracy_score(testYF, targetTestRF)

NameError: name 'metrics' is not defined
```

Chapitre 2

Exercice 2 :

2.1 BUT :

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

2.2 Solution :

Dans un premier temps on importe les bibliothèques dont on aura besoin :

Entrée [107]:

```
import os
from fnmatch import fnmatch
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split,KFold
from sklearn import preprocessing
import seaborn as sns
from sklearn.metrics import confusion_matrix,classification_report
%matplotlib inline
from sklearn.svm import SVC
import warnings
warnings.filterwarnings('ignore')
```

On utilise la bibliothèque pandas pour récupérer les données à partir d'un fichier csv qu'on le met soit dans le meme dossier que notre notebook ou on initialise la fonction read_csv() par le chemin dont on a notre fichier.

Entrée [108]:

```
df_Energy=pd.read_csv('energydata_complete.csv')
df_Energy.head(3)
```

Out[108]:

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...	T9	RH_9	T_out	Press_mm_hg	RH_out	Windspe
0	2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	...	17.033333	45.53	6.600000	733.5	92.0	7.0000
1	2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	...	17.066667	45.56	6.483333	733.6	92.0	6.6666
2	2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	...	17.000000	45.50	6.366667	733.7	92.0	6.3333

3 rows × 29 columns

Ici on veut vérifier s'il y a des valeurs manquantes.

Entrée [109]:

```
#Check for Missing Value

df_Energy.isnull().sum()
```

Out[109]:
```
date            0
Appliances      0
lights          0
T1              0
RH_1            0
T2              0
RH_2            0
T3              0
RH_3            0
T4              0
RH_4            0
T5              0
RH_5            0
T6              0
RH_6            0
T7              0
RH_7            0
T8              0
RH_8            0
T9              0
RH_9            0
T_out           0
Press_mm_hg     0
RH_out          0
Windspeed       0
Visibility      0
Tdewpoint       0
rv1             0
rv2             0
dtype: int64
```

On décrit note dataset en calculant les valeurs visualisées sur le tableau suivant :

On décrit note dataset en calculant les valeurs visualisées sur le tableau suivant :

Entrée [110]:
```
df_Energy.describe()
```

Out[110]:

	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	...	19735
mean	97.694958	3.801875	21.686571	40.259739	20.341219	40.420420	22.267611	39.242500	20.855335	39.026904	...	19
std	102.524891	7.935988	1.606066	3.979299	2.192974	4.069813	2.006111	3.254576	2.042884	4.341321	...	2
min	10.000000	0.000000	16.790000	27.023333	16.100000	20.463333	17.200000	28.766667	15.100000	27.660000	...	14
25%	50.000000	0.000000	20.760000	37.333333	18.790000	37.900000	20.790000	36.900000	19.530000	35.530000	...	18
50%	60.000000	0.000000	21.600000	39.656667	20.000000	40.500000	22.100000	38.530000	20.666667	38.400000	...	19
75%	100.000000	0.000000	22.600000	43.066667	21.500000	43.260000	23.290000	41.760000	22.100000	42.156667	...	20
max	1080.000000	70.000000	26.260000	63.360000	29.856667	56.026667	29.236000	50.163333	26.200000	51.090000	...	24

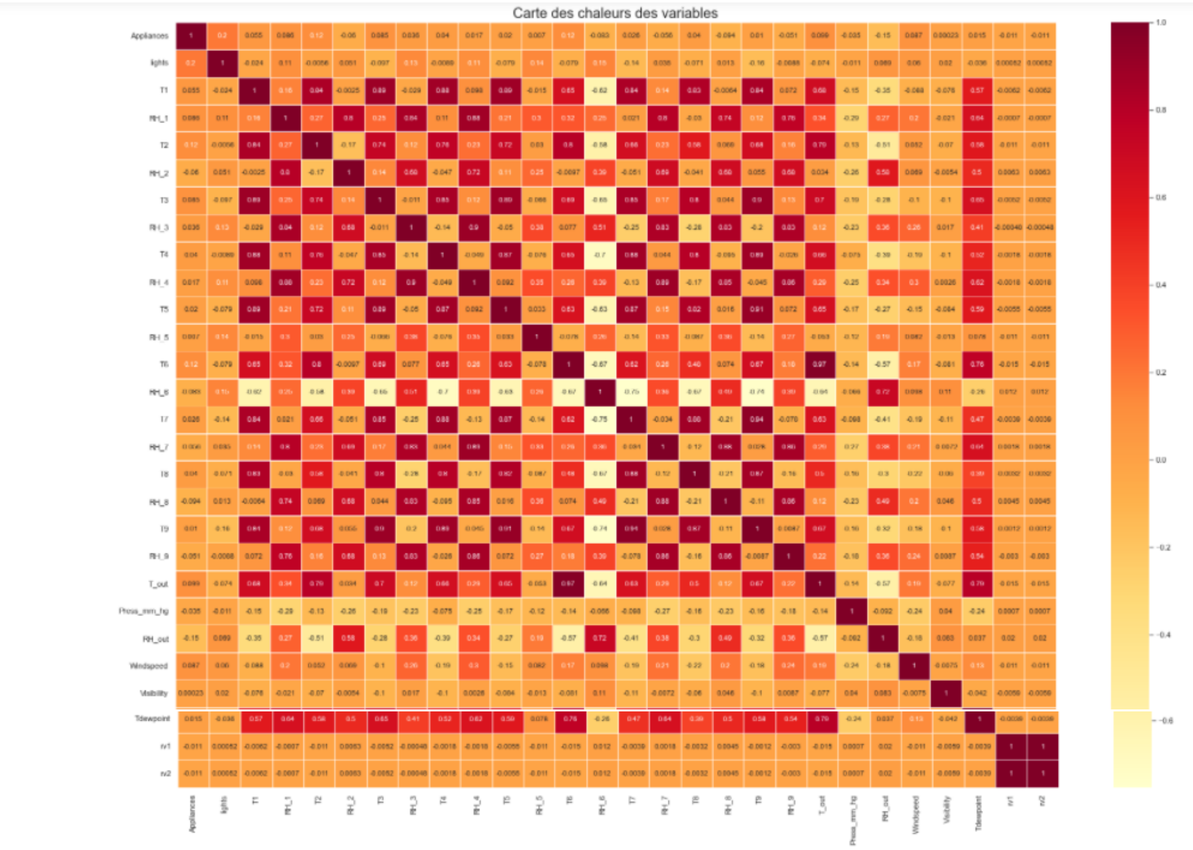
8 rows × 28 columns

2- Correlation plot

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

Entrée [111]:
```
#Correlation:

fig = plt.figure(figsize=(28,20))

axis = sns.heatmap(df_Energy.corr(), cmap= 'YlOrRd', linewidth=1, linecolor='white', annot=True)
axis.set_title('Carte des chaleurs des variables', fontsize=20)
```

Out[111]: `Text(0.5, 1.0, 'Carte des chaleurs des variables')`



3- Feature selection :

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

Entrée [112]:
```
corr_matrix = df_Energy.corr().abs()

#the matrix is symmetric so we need to extract upper triangle matrix without diagonal (k = 1)

sol = (corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
                 .stack()
                 .sort_values(ascending=False))
sol = sol.to_frame()
sol.columns=['corr']
sol[sol['corr'] >  0.88]
```

Out[112]:

		corr
rv1	rv2	1.000000
T6	T_out	0.974787
T7	T9	0.944776
T5	T9	0.911055
T3	T9	0.901324
RH_3	RH_4	0.898978
RH_4	RH_7	0.894301
T1	T3	0.892402
T4	T9	0.889439
T3	T5	0.888169
T1	T5	0.885247
RH_7	RH_8	0.883984
T7	T8	0.882123
RH_1	RH_4	0.880359

Entrée [113]:
```
df_Energy.columns
```

Out[113]:
```
Index(['date', 'Appliances', 'lights', 'T1', 'RH_1', 'T2', 'RH_2', 'T3',
       'RH_3', 'T4', 'RH_4', 'T5', 'RH_5', 'T6', 'RH_6', 'T7', 'RH_7', 'T8',
       'RH_8', 'T9', 'RH_9', 'T_out', 'Press_mm_hg', 'RH_out', 'Windspeed',
       'Visibility', 'Tdewpoint', 'rv1', 'rv2'],
      dtype='object')
```

Vous trouverez ci-dessous les 10 caractéristiques finales ainsi que la variable cible "Appliances" que nous avons sélectionnées pour l'analyse ultérieure.

Entrée [114]:
```
df_final = df_Energy[['T1',
                      'T2',
                      'RH_1',
                      'RH_2',
                      'RH_6',
                      'T_out',
                      'lights',
                      'Windspeed',
                      'RH_out',
                      'Appliances',
                      'Press_mm_hg',
                      'rv1',
                      'Tdewpoint']]

df_final['Appliances_Energy'] = np.where(df_final['Appliances']>= 60, 1, 0)
df_final.drop(columns=['Appliances'],axis=1,inplace=True)
```

Entrée [115]:
```
df_final.head()
```

Out[115]:

	T1	T2	RH_1	RH_2	RH_6	T_out	lights	Windspeed	RH_out	Press_mm_hg	rv1	Tdewpoint	Appliances_Energy
0	19.89	19.2	47.596667	44.790000	84.256667	6.600000	30	7.000000	92.0	733.5	13.275433	5.3	1
1	19.89	19.2	46.693333	44.722500	84.063333	6.483333	30	6.666667	92.0	733.6	18.606195	5.2	1
2	19.89	19.2	46.300000	44.626667	83.156667	6.366667	30	6.333333	92.0	733.7	28.642668	5.1	0
3	19.89	19.2	46.066667	44.590000	83.423333	6.250000	40	6.000000	92.0	733.8	45.410389	5.0	0
4	19.89	19.2	46.333333	44.530000	84.893333	6.133333	40	5.666667	92.0	733.9	10.084097	4.9	1

4- L'algorithme adopté :

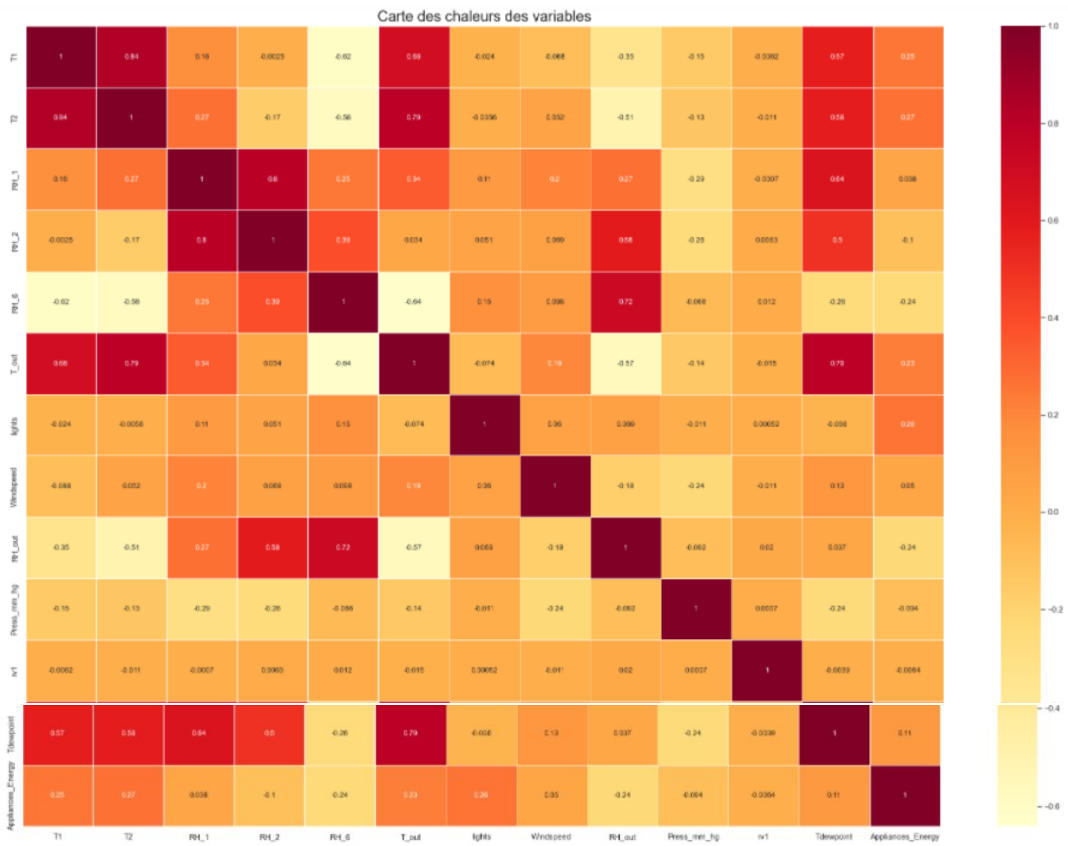
Dans l'apprentissage automatique , le noyau de fonction de base radiale , ou noyau RBF , est une fonction de noyau populaire utilisée dans divers algorithmes d'apprentissage noyaux. En particulier, il est couramment utilisé dans la classification des machines à vecteurs de support .

Ici on a choisi de travailler avec les machines à vecteurs de support, ou support vector machine (SVM), qui sont des modèles de machine learning supervisés centrés sur la résolution de problèmes de discrimination et de régression mathématiques.

Voici notre carte des chaleurs finale variables que nous allons utiliser avec le modèle [SVM, Decision Tress and Boosting].

Entrée [116]:
```
fig = plt.figure(figsize=(28,20))

axis = sns.heatmap(df_final.corr(), cmap= 'YlOrRd', linewidth=1, linecolor='white', annot=True)
axis.set_title('Carte des chaleurs des variables', fontsize=20)
```

Out[116]: Text(0.5, 1.0, 'Carte des chaleurs des variables')



5- Train Test Split :

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

Entrée [117]:
```
x = df_final.drop(columns=['Appliances_Energy'],axis=1)

y = df_final[['Appliances_Energy']]

xTrain, xTest, yTrain, yTest = train_test_split(x,y, test_size = 0.3, random_state = 0)
```

Entrée [118]:
```
print('Shape of xTrain Set', xTrain.shape)
print('Shape of yTrain Set', yTrain.shape)

print('')

print('Shape of xTest Set', xTest.shape)
print('Shape of yTest Set', yTest.shape)
```

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

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

6- Cross-Validation Split :

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.

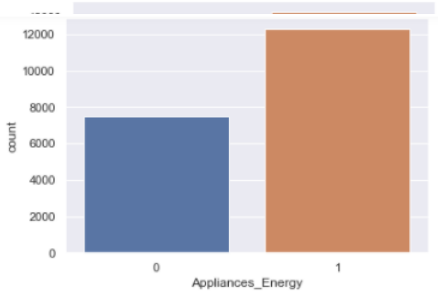
Entrée [119]:
```
cv=KFold(n_splits=10)

for train_index, test_index in cv.split(x):
    xTrain_cv,xTest_cv=x.iloc[train_index],x.iloc[test_index]
    yTrain_cv,yTest_cv=y.iloc[train_index],y.iloc[test_index]

print('Shape of xTrain Set', xTrain_cv.shape)
print('Shape of yTrain Set', yTrain_cv.shape)

print('')

print('Shape of xTest Set', xTest_cv.shape)
print('Shape of yTest Set', yTest_cv.shape)
```

```
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 :

Entrée [120]:
```
sns.countplot(df_final['Appliances_Energy'])
```

Out[120]:
```
<AxesSubplot:xlabel='Appliances_Energy', ylabel='count'>
```



7- 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 va effectuer une comparaison entre les différents noyaux : 1- Linéaire. 2- RBF : qui est demandé dans cette application. 3- Sigmoid.

Et on va afficher par la suite des résultats pour le train-test split et le cross-validation pour chaque kernel des kernels suivants.

Entrée [121]:
```
def runModelSVM(k,xTrain,yTrain,xTest,yTest):

    svc_clf = SVC(kernel=k)
    svc_clf.fit(xTrain,yTrain)
    y_pred=svc_clf.predict(xTest)

    print(' Kernel: ',k)
    print('Train score: {:.4f} %'.format(svc_clf.score(xTrain, yTrain)*100))
    print('Test score: {:.4f} %'.format(svc_clf.score(xTest, yTest)*100))
    print('')
    print('Classification Report:')
    print(classification_report(yTest,y_pred))
    print('Confusion Matrix:')
    print(confusion_matrix(yTest,y_pred))
```

Linear kernel

Entrée [122]:
```
print('********************** Result for Train-Test Split **********************\n')
runModelSVM('linear',xTrain,yTrain,xTest,yTest)
print('********************** Result for Cross-Validation **********************\n')
runModelSVM('linear',xTrain_cv,yTrain_cv,xTest_cv,yTest_cv)
```

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

 Kernel:  linear
Train score: 70.5950 %
Test score: 70.0051 %

              precision    recall  f1-score   support

           0       0.60      0.55      0.58      2192
           1       0.75      0.79      0.77      3729

    accuracy                           0.70      5921
   macro avg       0.68      0.67      0.67      5921
weighted avg       0.70      0.70      0.70      5921

Confusion Matrix:
[[1204  988]
 [ 788 2941]]
********************** Result for Cross-Validation **********************

 Kernel:  linear
Train score: 69.8514 %
Test score: 73.2894 %

Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.03      0.06       527
           1       0.74      0.99      0.84      1446

    accuracy                           0.73      1973
   macro avg       0.62      0.51      0.45      1973
weighted avg       0.67      0.73      0.64      1973

Confusion Matrix:
[[  18  509]
 [  18 1428]]
```

RBF Kernel

Entrée [82]:
```
print('********************** Result for Train-Test Split **********************\n')
runModelSVM('rbf',xTrain,yTrain,xTest,yTest)
print('********************** Result for Cross-Validation **********************\n')
runModelSVM('rbf',xTrain_cv,yTrain_cv,xTest_cv,yTest_cv)
```

```
********************** 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]]
```

Sigmoid Kernel

Entrée [83]:
```
print('********************* Result for Train-Test Split *********************\n')
runModelSVM('sigmoid',xTrain,yTrain,xTest,yTest)
print('********************* Result for Cross-Validation *********************\n')
runModelSVM('sigmoid',xTrain_cv,yTrain_cv,xTest_cv,yTest_cv)
```

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

 Kernel:  sigmoid
Train score: 61.8503 %
Test score: 62.9792 %

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      2192
           1       0.63      1.00      0.77      3729

    accuracy                           0.63      5921
   macro avg       0.31      0.50      0.39      5921
weighted avg       0.40      0.63      0.49      5921

Confusion Matrix:
[[   0 2192]
 [   0 3729]]
********************* Result for Cross-Validation *********************

 Kernel:  sigmoid
Train score: 60.9560 %
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]]
```

8- Decision Trees :

Pour le Train-Test Split. Full Length Tree :

Entrée [84]:
```
from sklearn.tree import DecisionTreeClassifier

dtree=DecisionTreeClassifier(criterion='gini')

dtree.fit(xTrain,yTrain)
y_pred=dtree.predict(xTest)

print('Train score: {:.4f} %'.format(dtree.score(xTrain,yTrain)*100))
print('Test score: {:.4f} %'.format(dtree.score(xTest, yTest)*100))
```

```
Train score: 100.0000 %
Test score: 77.8247 %
```

Expérimentation de la taille. Tailler pour éviter le surajustement.

L'élagage nous aide à éviter le surajustement.

Généralement, il est préférable d'avoir un modèle simple, cela évite les problèmes de surajustement. Toute division supplémentaire qui n'ajoute pas de valeur significative n'en vaut pas la peine. Nous pouvons éviter le surajustement en modifiant les paramètres comme :

max_leaf_nodes min_samples_leaf profondeur max Paramètres d'élagage max_leaf_nodes : réduire le nombre de nœuds feuilles. min_samples_leaf : limite la taille de la feuille d'échantillon. La taille minimale de l'échantillon dans les nœuds terminaux peut être fixée à 30, 100, 300 ou 5 % du total max_depth. Réduire la profondeur de l'arbre pour construire un arbre généralisé Régler la profondeur de l'arbre à 3, 5, 10 selon après vérification sur les données de test.

Entrée [85]:
```
from sklearn.tree import DecisionTreeClassifier
best_score=0

for n in range(1,20):
    for m in [10,15,20,25,30,35,40,50]:
        for l in range(2,30):
            dtree = DecisionTreeClassifier(criterion = 'gini', max_depth=n,max_leaf_nodes=l,min_samples_leaf=m)
            dtree.fit(xTrain,yTrain)
            score=dtree.score(xTest,yTest)
            if(score>best_score):
                best_score=score
                best_parameters={'max_depth':n,'min_samples_leaf':m,'max_leaf_nodes':l}

print(best_parameters)
```

```
{'max_depth': 10, 'min_samples_leaf': 10, 'max_leaf_nodes': 27}
```

Entrée [86]:
```
d_tree=DecisionTreeClassifier(max_depth=9,criterion='gini',min_samples_leaf=10,max_leaf_nodes=27)

d_tree.fit(xTrain,yTrain)
y_pred=d_tree.predict(xTest)

print('Train score: {:.4f} %'.format(d_tree.score(xTrain,yTrain)*100))
print('Test score: {:.4f} %'.format(d_tree.score(xTest, yTest)*100))

print('Classification Report:')
print(classification_report(yTest,y_pred))
print('Confusion Matrix:')
print(confusion_matrix(yTest,y_pred))
```

```
Train score: 74.3304 %
Test score: 72.6229 %
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.52      0.58      2192
           1       0.75      0.85      0.80      3729

    accuracy                           0.73      5921
   macro avg       0.71      0.68      0.69      5921
weighted avg       0.72      0.73      0.72      5921

Confusion Matrix:
[[1133 1059]
 [ 562 3167]]
```

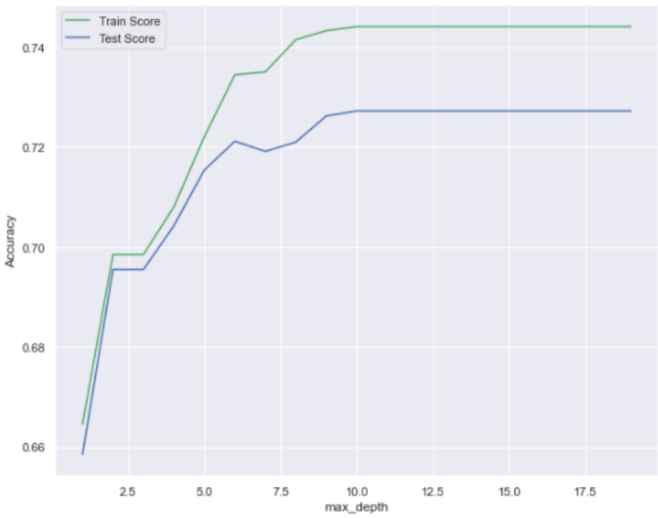
Entrée [87]:
```
from pprint import pprint
pprint(dtree.get_params())
```

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 19,
 'max_features': None,
 'max_leaf_nodes': 29,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 50,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

Ici on affiche la courbe d'apprentissage par rapport à la profondeur maximale :

Entrée [28]:
```
from sklearn.tree import DecisionTreeClassifier

sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in range(1,20):
            dtree = DecisionTreeClassifier(criterion = 'gini', max_depth=n,min_samples_leaf=10,max_leaf_nodes=27)
            dtree.fit(xTrain,yTrain)
            train_score_array.append(dtree.score(xTrain,yTrain))
            test_score_array.append(dtree.score(xTest, yTest))

x_axis = range(1,20)
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('max_depth')
plt.ylabel('Accuracy')
plt.legend()
```

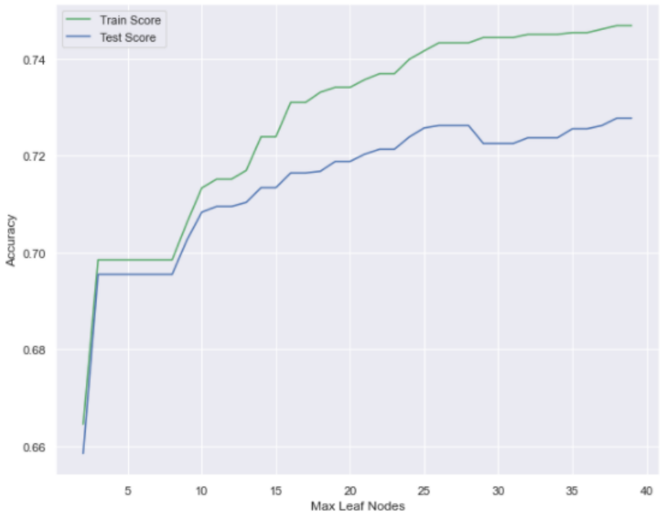
Out[28]:
```
<matplotlib.legend.Legend at 0x28c1d371dc0>
```



Ici on affiche la courbe d'apprentissage par rapport à Max Leaf Nodes.

Entrée [88]:
```
from sklearn.tree import DecisionTreeClassifier

sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in range(2,40):
            dtree = DecisionTreeClassifier(criterion = 'gini', max_depth=9,min_samples_leaf=10,max_leaf_nodes=n)
            dtree.fit(xTrain,yTrain)
            train_score_array.append(dtree.score(xTrain,yTrain))
            test_score_array.append(dtree.score(xTest, yTest))

x_axis = range(2,40)
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('Max Leaf Nodes')
plt.ylabel('Accuracy')
plt.legend()
```

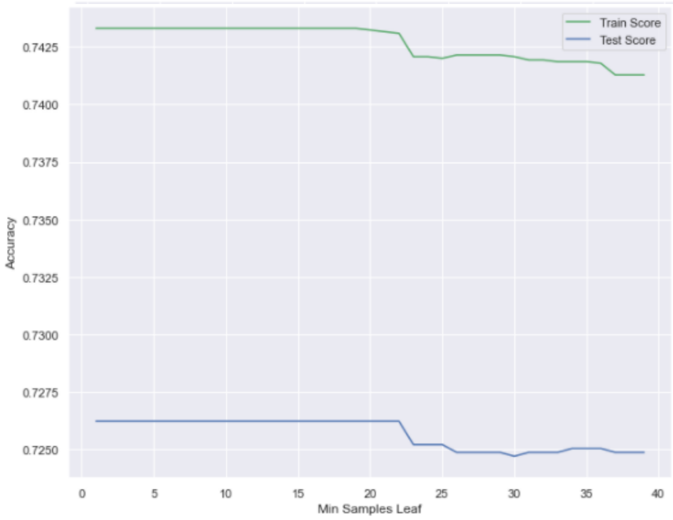
Out[88]: <matplotlib.legend.Legend at 0x28c0c090fa0>



Ici on affiche la courbe d'apprentissage par rapport à Min Samples Leaf.

Entrée [90]:
```
from sklearn.tree import DecisionTreeClassifier

sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in range(1,40):
            dtree = DecisionTreeClassifier(criterion = 'gini', max_depth=9,min_samples_leaf=n,max_leaf_nodes=27)
            dtree.fit(xTrain,yTrain)
            train_score_array.append(dtree.score(xTrain,yTrain))
            test_score_array.append(dtree.score(xTest, yTest))

x_axis = range(1,40)
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('Min Samples Leaf')
plt.ylabel('Accuracy')
plt.legend()
```

Out[90]: <matplotlib.legend.Legend at 0x28c0d182eb0>



Pour la Cross_Validation Split : Full Length Tree :

Entrée [91]:
```
from sklearn.tree import DecisionTreeClassifier

dtree_cv=DecisionTreeClassifier(criterion='gini')

dtree_cv.fit(xTrain_cv,yTrain_cv)
y_pred_cv=dtree_cv.predict(xTest_cv)

print('Train score: {:.4f} %'.format(dtree_cv.score(xTrain_cv,yTrain_cv)*100))
print('Test score: {:.4f} %'.format(dtree_cv.score(xTest_cv, yTest_cv)*100))
```

```
Train score: 100.0000 %
Test score: 65.4840 %
```

Expérimentation de la taille. Tailler pour éviter le surajustement.

L'élagage nous aide à éviter le surajustement.

Généralement, il est préférable d'avoir un modèle simple, cela évite les problèmes de surajustement. Toute division supplémentaire qui n'ajoute pas de valeur significative n'en vaut pas la peine. Nous pouvons éviter le surajustement en modifiant les paramètres comme :

max_leaf_nodes min_samples_leaf profondeur max Paramètres d'élagage max_leaf_nodes : réduire le nombre de nœuds feuilles. min_samples_leaf : limite la taille de la feuille d'échantillon. La taille minimale de l'échantillon dans les nœuds terminaux peut être fixée à 30, 100, 300 ou 5 % du total max_depth Réduire la profondeur de l'arbre pour construire un arbre généralisé Régler la profondeur de l'arbre à 3, 5, 10 selon après vérification sur les données de test.

Entrée [92]:
```
from sklearn.tree import DecisionTreeClassifier
best_score=0

for n in range(1,20):
    for m in [10,15,20,25,30,35,40,50]:
        for l in range(2,30):
            dtree_cv = DecisionTreeClassifier(criterion = 'gini', max_depth=n,max_leaf_nodes=l,min_samples_leaf=m)
            dtree_cv.fit(xTrain_cv,yTrain_cv)
            score=dtree_cv.score(xTest_cv,yTest_cv)
            if(score>best_score):
                best_score=score
                best_parameters={'max_depth':n,'min_samples_leaf':m,'max_leaf_nodes':l}

print(best_parameters)
```

```
{'max_depth': 6, 'min_samples_leaf': 10, 'max_leaf_nodes': 28}
```

Entrée [93]:
```
dtree_cv=DecisionTreeClassifier(max_depth=5,criterion='gini',min_samples_leaf=10,max_leaf_nodes=28)

dtree_cv.fit(xTrain_cv,yTrain_cv)
y_pred_cv=dtree_cv.predict(xTest_cv)

print('Train score: {:.4f} %'.format(dtree_cv.score(xTrain_cv,yTrain_cv)*100))
print('Test score: {:.4f} %'.format(dtree_cv.score(xTest_cv, yTest_cv)*100))

print('Classification Report:')
print(classification_report(yTest_cv,y_pred_cv))
print('Confusion Matrix:')
print(confusion_matrix(yTest_cv,y_pred_cv))
```

```
Train score: 72.7564 %
Test score: 73.1374 %
Classification Report:
              precision    recall  f1-score   support

           0       0.29      0.00      0.01       527
           1       0.73      1.00      0.84      1446

    accuracy                           0.73      1973
   macro avg       0.51      0.50      0.43      1973
weighted avg       0.61      0.73      0.62      1973

Confusion Matrix:
[[   2  525]
 [   5 1441]]
```

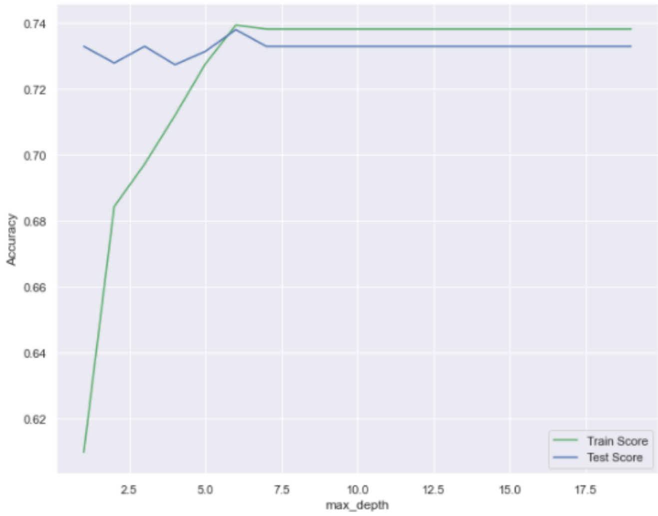
Entrée [33]:
```
from pprint import pprint
pprint(dtree_cv.get_params())
```

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 5,
 'max_features': None,
 'max_leaf_nodes': 28,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 10,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

Ici on affiche la courbe d'apprentissage avec respect to Max depth :

Entrée [34]:
```
from sklearn.tree import DecisionTreeClassifier

sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in range(1,20):
            dtree_cv = DecisionTreeClassifier(criterion = 'gini', max_depth=n,min_samples_leaf=10,max_leaf_nodes=28)
            dtree_cv.fit(xTrain_cv,yTrain_cv)
            train_score_array.append(dtree_cv.score(xTrain_cv,yTrain_cv))
            test_score_array.append(dtree_cv.score(xTest_cv, yTest_cv))

x_axis = range(1,20)
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('max_depth')
plt.ylabel('Accuracy')
plt.legend()
```

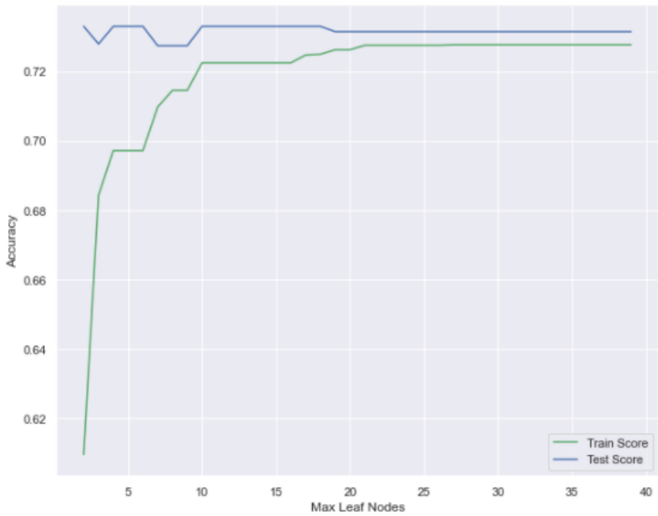
Out[34]:
```
<matplotlib.legend.Legend at 0x28c1d6fe580>
```



Ici on affiche la courbe d'apprentissage avec respect to Max Leaf Nodes :

Entrée [35]:
```
from sklearn.tree import DecisionTreeClassifier

sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in range(2,40):
            dtree_cv = DecisionTreeClassifier(criterion = 'gini', max_depth=5,min_samples_leaf=10,max_leaf_nodes=n)
            dtree_cv.fit(xTrain_cv,yTrain_cv)
            train_score_array.append(dtree_cv.score(xTrain_cv,yTrain_cv))
            test_score_array.append(dtree_cv.score(xTest_cv, yTest_cv))

x_axis = range(2,40)
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('Max Leaf Nodes')
plt.ylabel('Accuracy')
plt.legend()
```

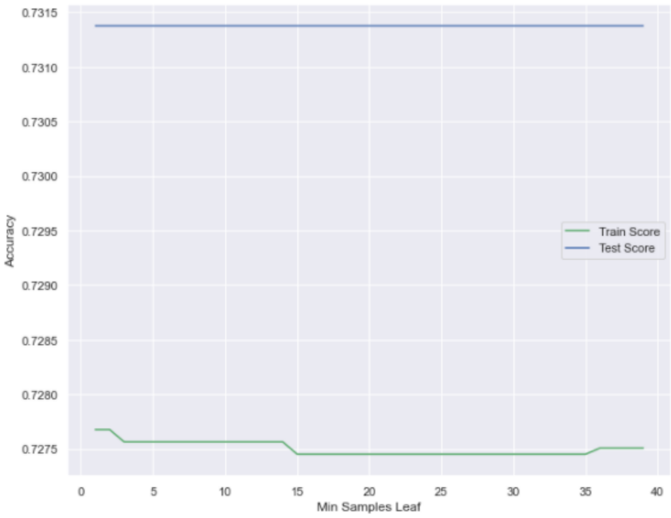
Out[35]: <matplotlib.legend.Legend at 0x28c1d770dc0>



Ici on affiche la courbe d'apprentissage avec respect to Min Samples Leaf :

Entrée [36]:
```
from sklearn.tree import DecisionTreeClassifier

sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in range(1,40):
            dtree_cv = DecisionTreeClassifier(criterion = 'gini', max_depth=5,min_samples_leaf=n,max_leaf_nodes=28)
            dtree_cv.fit(xTrain_cv,yTrain_cv)
            train_score_array.append(dtree_cv.score(xTrain_cv,yTrain_cv))
            test_score_array.append(dtree_cv.score(xTest_cv, yTest_cv))

x_axis = range(1,40)
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('Min Samples Leaf')
plt.ylabel('Accuracy')
plt.legend()
```

Out[36]:
```
<matplotlib.legend.Legend at 0x28c1d7ee610>
```



9- Boosting :

For Train-Test-Split :

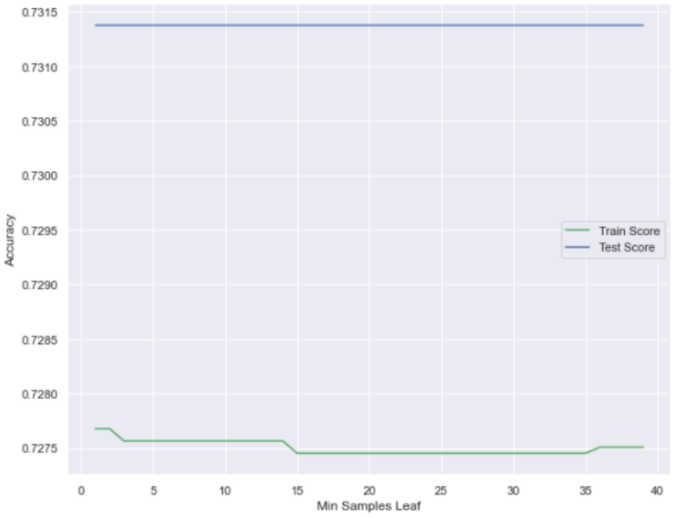
Entrée [38]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier()

xgb.fit(xTrain,yTrain)
y_pred=xgb.predict(xTest)

print('Train score: {:.4f} %'.format(xgb.score(xTrain,yTrain)*100))
print('Test score: {:.4f} %'.format(xgb.score(xTest, yTest)*100))
```

```
[04:27:24] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Train score: 91.9792 %
Test score: 80.9154 %
```

Entrée [39]:
```
pprint(xgb.get_params())
```

```
{'base_score': 0.5,
 'booster': 'gbtree',
 'colsample_bylevel': 1,
 'colsample_bynode': 1,
 'colsample_bytree': 1,
 'enable_categorical': False,
 'gamma': 0,
 'gpu_id': -1,
 'importance_type': None,
 'interaction_constraints': '',
 'learning_rate': 0.300000012,
 'max_delta_step': 0,
 'max_depth': 6,
 'min_child_weight': 1,
 'missing': nan,
 'monotone_constraints': '()',
 'n_estimators': 100,
 'n_jobs': 4,
 'num_parallel_tree': 1,
```

Experimentation avec Pruning pour XGBoost:



Entrée [41]:
```
xgb=XGBClassifier(max_depth=8,n_estimators=200,learning_rate=0.1)

xgb.fit(xTrain,yTrain)
y_pred=xgb.predict(xTest)

print('Train score: {:.4f} %'.format(xgb.score(xTrain,yTrain)*100))
print('Test score: {:.4f} %'.format(xgb.score(xTest, yTest)*100))

print('Classification Report:')
print(classification_report(yTest,y_pred))
print('Confusion Matrix:')
print(confusion_matrix(yTest,y_pred))
```

```
[04:36:09] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Train score: 94.7083 %
Test score: 81.0505 %
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.72      0.74      2192
           1       0.84      0.86      0.85      3729

    accuracy                           0.81      5921
   macro avg       0.80      0.79      0.79      5921
weighted avg       0.81      0.81      0.81      5921

Confusion Matrix:
[[1576  616]
 [ 506 3223]]
```

Entrée [42]:
```
from pprint import pprint
pprint(xgb.get_params())
```

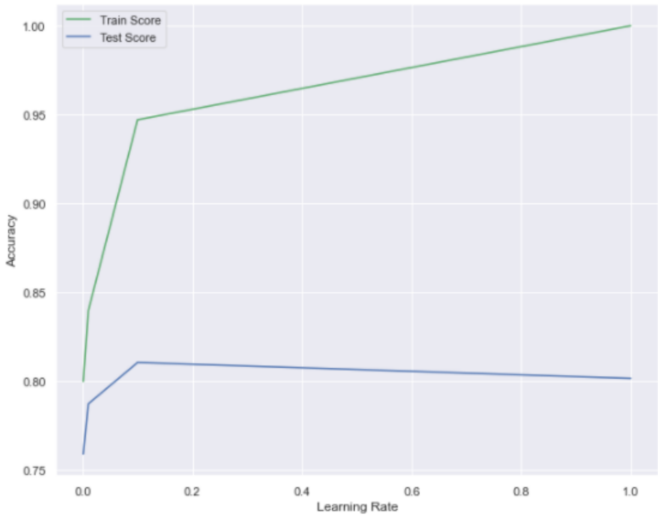
```
{'base_score': 0.5,
 'booster': 'gbtree',
 'colsample_bylevel': 1,
 'colsample_bynode': 1,
 'colsample_bytree': 1,
 'enable_categorical': False,
 'gamma': 0,
 'gpu_id': -1,
 'importance_type': None,
 'interaction_constraints': '',
 'learning_rate': 0.1,
 'max_delta_step': 0,
 'max_depth': 8,
 'min_child_weight': 1,
 'missing': nan,
 'monotone_constraints': '()',
 'n_estimators': 200,
 'n_jobs': 4,
 'num_parallel_tree': 1,
 'objective': 'binary:logistic',
 'predictor': 'auto',
 'random_state': 0,
 'reg_alpha': 0,
 'reg_lambda': 1,
 'scale_pos_weight': 1,
 'subsample': 1,
 'tree_method': 'exact',
 'use_label_encoder': True,
 'validate_parameters': 1,
 'verbosity': None}
```

Ici on affiche la courbe d'apprentissage avec respect to learning rate :

Entrée [43]:
```
sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in [0.001,0.01,0.1,1]:
            xgb = XGBClassifier( max_depth=8,learning_rate=n,n_estimators=200)
            xgb.fit(xTrain,yTrain)
            train_score_array.append(xgb.score(xTrain,yTrain))
            test_score_array.append(xgb.score(xTest, yTest))

x_axis = [0.001,0.01,0.1,1]
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('Learning Rate')
plt.ylabel('Accuracy')
plt.legend()
```

```
[04:36:38] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:36:43] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:36:50] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:36:54] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[43]:
```
<matplotlib.legend.Legend at 0x28c1da90c10>
```

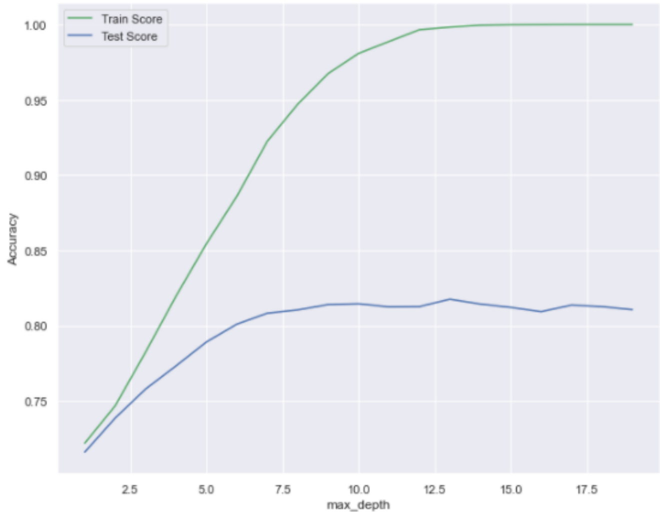


Ici on affiche la courbe d'apprentissage avec respect to max_depth :

Entrée [44]:
```
sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in range(1,20):
            xgb = XGBClassifier( max_depth=n,learning_rate=0.1,n_estimators=200)
            xgb.fit(xTrain,yTrain)
            train_score_array.append(xgb.score(xTrain,yTrain))
            test_score_array.append(xgb.score(xTest, yTest))

x_axis = range(1,20)
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('max_depth')
plt.ylabel('Accuracy')
plt.legend()
```

```
[04:37:26] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:37:27] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:37:28] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:37:30] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:37:32] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:37:35] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:37:38] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:37:42] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:37:47] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

```
Out[44]: <matplotlib.legend.Legend at 0x28c1dafd370>
```



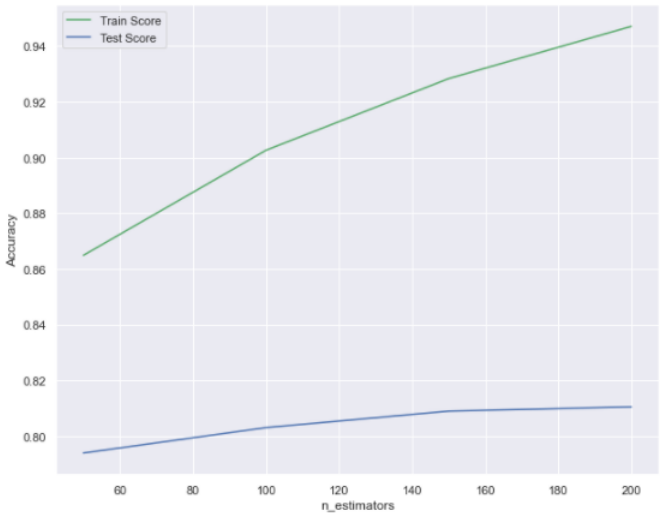
Ici on affiche la courbe d'apprentissage avec respect to n-estimators :

```
Entrée [45]: sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in [50,100,150,200]:
            xgb = XGBClassifier( max_depth=8,learning_rate=0.1,n_estimators=n)
            xgb.fit(xTrain,yTrain)
            train_score_array.append(xgb.score(xTrain,yTrain))
            test_score_array.append(xgb.score(xTest, yTest))

x_axis = [50,100,150,200]
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('n_estimators')
plt.ylabel('Accuracy')
plt.legend()
```

```
[04:41:25] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:41:26] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:41:29] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:41:32] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

```
Out[45]: <matplotlib.legend.Legend at 0x28c1db79310>
```


```
Out[45]: <matplotlib.legend.Legend at 0x28c1db79310>
```

Pour la Cross-Validation :

Entrée [46]:
```
from xgboost import XGBClassifier
xgb_cv = XGBClassifier()

xgb_cv.fit(xTrain_cv,yTrain_cv)
y_pred_cv=xgb_cv.predict(xTest_cv)

print('Train score: {:.4f} %'.format(xgb_cv.score(xTrain_cv,yTrain_cv)*100))
print('Test score: {:.4f} %'.format(xgb_cv.score(xTest_cv, yTest_cv)*100))
```

```
[04:46:11] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Train score: 91.3748 %
Test score: 72.3264 %
```

Entrée [47]:
```
pprint(xgb_cv.get_params())
```

```
{'base_score': 0.5,
 'booster': 'gbtree',
 'colsample_bylevel': 1,
 'colsample_bynode': 1,
 'colsample_bytree': 1,
 'enable_categorical': False,
 'gamma': 0,
 'gpu_id': -1,
 'importance_type': None,
 'interaction_constraints': '',
 'learning_rate': 0.300000012,
 'max_delta_step': 0,
 'max_depth': 6,
 'min_child_weight': 1,
 'missing': nan,
 'monotone_constraints': '()',
 'n_estimators': 100,
 'n_jobs': 4,
 'num_parallel_tree': 1,
 'objective': 'binary:logistic',
 'predictor': 'auto',
```

Experimentation avec Pruning pour XGBoost:

Entrée [48]:
```
best_score=0

for n in [0.001,0.01,0.1,1,10,100]:
    for m in [50,100,150,200]:
        for l in range(1,10):
            xgb_cv = XGBClassifier(learning_rate=n,n_estimators=m,max_depth=l)
            xgb_cv.fit(xTrain_cv,yTrain_cv)
            score=xgb_cv.score(xTest_cv, yTest_cv)
            if(score>best_score):
                best_score=score
                best_parameters = {'learning_rate': n,'n_estimators':m,'max_depth':l}

print(best_parameters)
```

```
[04:46:18] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objecti
'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[04:46:18] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[04:46:19] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[04:46:19] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[04:46:20] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[04:46:21] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[04:46:22] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
```

Entrée [60]:
```
xgb_cv=XGBClassifier(max_depth=3,n_estimators=50,learning_rate=1)

xgb_cv.fit(xTrain_cv,yTrain_cv)
y_pred_cv=xgb_cv.predict(xTest_cv)

print('Train score: {:.4f} %'.format(xgb_cv.score(xTrain_cv,yTrain_cv)*100))
print('Test score: {:.4f} %'.format(xgb_cv.score(xTest_cv, yTest_cv)*100))

print('Classification Report:')
print(classification_report(yTest_cv,y_pred_cv))
print('Confusion Matrix:')
print(confusion_matrix(yTest_cv,y_pred_cv))
```

```
[04:56:34] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Train score: 81.9559 %
Test score: 73.6442 %
Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.09      0.15       527
           1       0.75      0.97      0.84      1446

    accuracy                           0.74      1973
   macro avg       0.64      0.53      0.50      1973
weighted avg       0.69      0.74      0.66      1973

Confusion Matrix:
[[  46  481]
 [  39 1407]]
```

Entrée [61]:
```
from pprint import pprint
pprint(xgb_cv.get_params())
```

```
{'base_score': 0.5,
 'booster': 'gbtree',
 'colsample_bylevel': 1,
 'colsample_bynode': 1,
 'colsample_bytree': 1,
 'enable_categorical': False,
 'gamma': 0,
 'gpu_id': -1,
 'importance_type': None,
 'interaction_constraints': '',
 'learning_rate': 1,
 'max_delta_step': 0,
 'max_depth': 3,
 'min_child_weight': 1,
 'missing': nan,
 'monotone_constraints': '()',
 'n_estimators': 50,
 'n_jobs': 4,
 'num_parallel_tree': 1,
 'objective': 'binary:logistic',
 'predictor': 'auto',
 'random_state': 0,
 'reg_alpha': 0,
 'reg_lambda': 1,
 'scale_pos_weight': 1,
 'subsample': 1,
 'tree_method': 'exact',
 'use_label_encoder': True,
 'validate_parameters': 1,
 'verbosity': None}
```

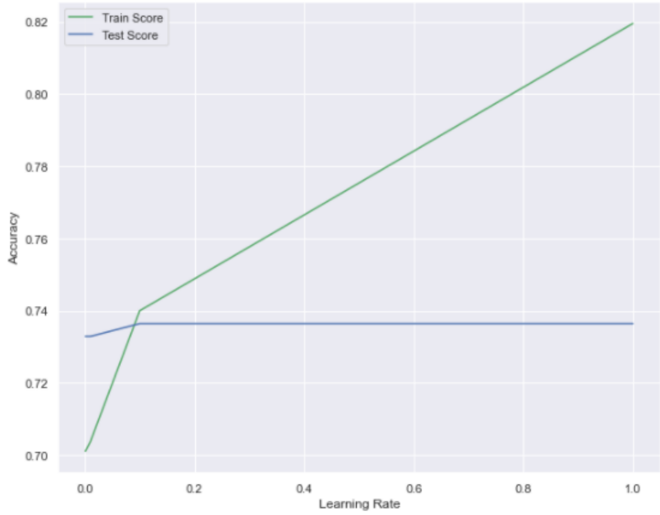
Ici on affiche la courbe d'apprentissage avec respect to learning rate :

Entrée [62]:
```
sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in [0.001,0.01,0.1,1]:
            xgb_cv = XGBClassifier( max_depth=3,learning_rate=n,n_estimators=50)
            xgb_cv.fit(xTrain_cv,yTrain_cv)
            train_score_array.append(xgb_cv.score(xTrain_cv,yTrain_cv))
            test_score_array.append(xgb_cv.score(xTest_cv, yTest_cv))

x_axis = [0.001,0.01,0.1,1]
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('Learning Rate')
plt.ylabel('Accuracy')
plt.legend()
```

```
[04:56:44] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:44] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:45] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:45] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[62]:
```
<matplotlib.legend.Legend at 0x28c1e36d910>
```

```
Out[62]: <matplotlib.legend.Legend at 0x28c1e36d910>
```

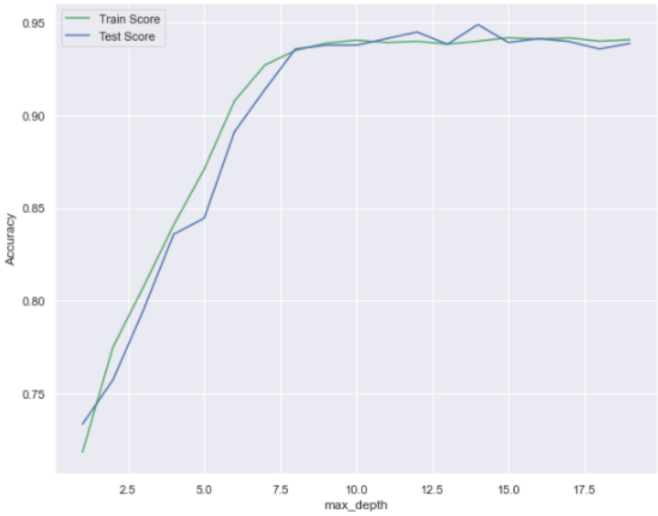


Ici on affiche la courbe d'apprentissage avec respect to max_depth :

```
Entrée [63]: sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in range(1,20):
            xgb_cv = XGBClassifier( max_depth=n,learning_rate=1,n_estimators=50)
            xgb_cv.fit(xTrain,yTrain)
            train_score_array.append(xgb_cv.score(xTrain_cv,yTrain_cv))
            test_score_array.append(xgb_cv.score(xTest_cv, yTest_cv))

x_axis = range(1,20)
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('max_depth')
plt.ylabel('Accuracy')
plt.legend()
```

```
[04:56:50] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:51] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:51] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:51] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:52] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:53] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:54] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:55] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:56] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:56:58] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
```

Out[63]: <matplotlib.legend.Legend at 0x28c1e3e0eb0>

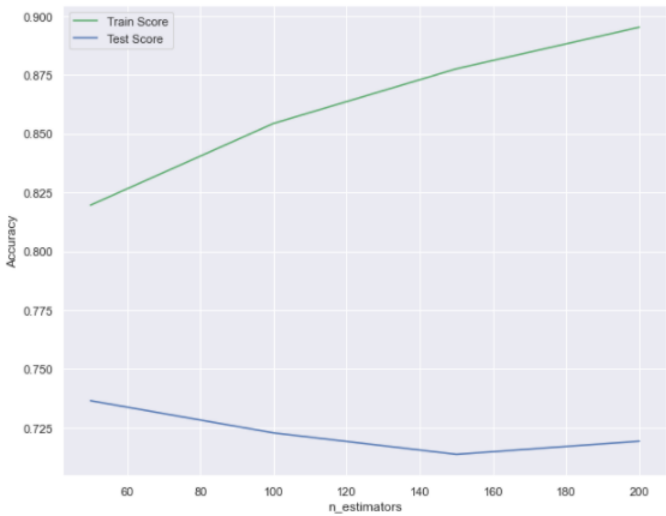


Ici on affiche la courbe d'apprentissage avec respect to n-estimators :

Entrée [64]:
```
sns.set(rc={'figure.figsize':(10,8)})

train_score_array = []
test_score_array = []

for n in [50,100,150,200]:
            xgb_cv = XGBClassifier( max_depth=3,learning_rate=1,n_estimators=n)
            xgb_cv.fit(xTrain_cv,yTrain_cv)
            train_score_array.append(xgb_cv.score(xTrain_cv,yTrain_cv))
            test_score_array.append(xgb_cv.score(xTest_cv, yTest_cv))

x_axis = [50,100,150,200]
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('n_estimators')
plt.ylabel('Accuracy')
plt.legend()
```

```
[04:58:26] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:58:26] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:58:27] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[04:58:29] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[64]: <matplotlib.legend.Legend at 0x28c1ebd48e0>



Conclusion

Les réseaux de neurones artificiels, inspirés du comportement du cerveau humain, permettent de créer de l'intelligence artificielle. Notamment appliqués en datamining principalement à travers l'apprentissage non supervisé, ils servent à prédire, à identifier et à classifier les données. L'apprentissage, moteur essentiel du système, leur permet d'assimiler un traitement d'information à travers une fonction et de le reproduire pour les données qui lui seront ensuite présentées.

références

https ://www.tensorflow.org/api$_d$ocs/python/tf/keras/Sequential
https : //developpaper.com/python−uses−neural−networks−for−simple−text−classification/
https : //scikit−learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
https : //www.shanelynn.ie/python − pandas − read − csv − load − data − from − csv − files/
https : //towardsdatascience.com/an−easy−tutorial−about−sentiment−analysis−with−deep − learning − and − keras − 2bf52b9cba91
https : //curiousily.com/posts/sentiment − analysis − with − tensorflow − 2 − and − keras − using − python/