Tâche de classification {VRAI} vs {FAUX}

Etudiants : Cazeres Mathieu (22200082), Martin-Chantereau Etienne (21909526), Moreaux Victor (22200010), Poiret Valentin (21609227)

Introduction Notebook

-> Importation librairies

In [ ]:
```
import pandas as pd

from google.colab import drive
drive.mount('/content/gdrive')

import sys

my_local_drive='/content/gdrive/MyDrive/ML'
# Ajout du path pour les librairies, fonctions et données
sys.path.append(my_local_drive)
# Se positionner sur le répertoire associé
%cd $my_local_drive
%pwd
```

```
Mounted at /content/gdrive
/content/gdrive/MyDrive/ML
```

Out[ ]:
```
'/content/gdrive/MyDrive/ML'
```

In [ ]:
```
import pandas as pd
# fonctions utilities (affichage, confusion, etc.)
from Fonction.MyNLPUtilities import *
# fonctions utilities (fonction de clean, import etc etc)
from Fonction.myFonction import *
from Fonction.AllModels import *
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

-> Chargement des données brutes

In [ ]:
```
Init_train = pd.read_csv('./Data_brut/HAI817_Projet_train.csv', sep=",")
Init_test = pd.read_csv('./Data_brut/HAI817_Projet_test.csv', sep=",")
data_brute = pd.concat([Init_train,Init_test], ignore_index = True)

print("data_brute avant drop de duplicate : \n", data_brute['our rating'].value_
data_brute = data_brute.drop_duplicates()
print("data après drop de duplicate : \n", data_brute['our rating'].value_counts


data_to_test = data_brute[~data_brute['our rating'].isin(['mixture'])]
data_to_test = data_to_test.drop(columns=['public_id', 'ID'])
print("data_to_test avant drop duplicates : \n", data_to_test['our rating'].valu
data_to_test = data_to_test.drop_duplicates()
print("datato_test après drop duplicates : \n", data_to_test['our rating'].value

display(data_to_test.describe())
```

```
data_brute avant drop de duplicate :
 false      893
true       421
mixture    414
other      148
Name: our rating, dtype: int64

data après drop de duplicate :
 false      815
true       411
mixture    374
other      127
Name: our rating, dtype: int64

data_to_test avant drop duplicates :
 false    815
true     411
other    127
Name: our rating, dtype: int64

datato_test après drop duplicates :
 false    801
true     407
other    126
Name: our rating, dtype: int64
```

	text	title	our rating
count	1334	1324	1334
unique	1323	1220	3
top	ADVERTISEMENT The fire that gutted a flank of ...	Marine Corps. Rebukes Pelosi: “WE DON’T WORK F...	false
freq	2	4	801

-> Initialisation des dataframes équilibrage

Lors de la visualisation les différentes données, nous nous sommes rendus compte que les données n'était pas du tout bien isolés je décide donc de tester tout les modèles en prenant différente classification : Texte, Titre, Texte+Titre pour ensuite sélectionner le meilleur de classification que je souhaite et le meilleur classifieur.

Je crée donc 3 dataframe correspondant à chaque type de classification.
J'équilibre les dataframes en fonctions du nombre de other en appliquant la formule suivante : nombre_de_True = len(Other)/2 nombre_de_False = len(Other)/2 nombre_de_Other = Nombre de other dans le jeu de données d'origine Au regard du faible nombre d'other dans le jeu de données nous décidons d'opérer un oversampling d'environ 20% sur les données classées others

In [ ]:
```
from pandas import DataFrame


def initDataToTest(df : DataFrame, columnToDrop : list[str], columnToCount : str
    data = df.copy()
    data = data.drop(columns=columnToDrop)
    print("data", valueToMerge ,"avant drop duplicates : \n", data[columnToCount
    data = data.drop_duplicates()
    print("data", valueToMerge ,"après drop duplicates : \n", data[columnToCount
    if size > 0 :
        data = balanceSample(data,size,valueToMerge)
    if valueOfReplacement is not None :
        data = data.replace(valueToMerge, valueOfReplacement)
    return data
```

❗ IMPORTANT ❗ Ceci est le code permettant de créer des jeux de données équilibrés, si vous souhaitez les recréers il faut recharger la cellule attention toutefois cela risque d'écraser les anciens dataframes si vous les enregistrer par la suite et faussera les résultats des classifieurs

In [ ]:
```
"""
print("dataFrame text only :")
df_text = pd.concat([initDataToTest(data_to_test[data_to_test['our rating'] == '
                             initDataToTest(data_to_test,['title'], ['our rating
df_text = balanceSample(df_text, 150, ['true/false', 'other'])
X_train_text  = df_text['text']
Y_train_text = df_text['our rating']


print("dataFrame title only :")
df_title = pd.concat([initDataToTest(data_to_test[data_to_test['our rating'] ==
                             initDataToTest(data_to_test,['text'], ['our rating'
df_title = balanceSample(df_title, 150, ['true/false', 'other'])
X_train_title = df_title['title']
Y_train_title = df_title['our rating']


print("dataFrame text_title concatenate only :")
df_text_titre_concat = data_to_test.copy()
df_text_titre_concat = df_text_titre_concat.fillna('')
df_text_titre_concat['title_text'] = df_text_titre_concat['title'].str.cat(df_te
df_text_and_title = pd.concat([initDataToTest(df_text_titre_concat[df_text_titre
                             initDataToTest(df_text_titre_concat,['text', 'title
```

```
df_text_and_title = balanceSample(df_text_and_title, 150, ['true/false', 'other'
X_train_text_and_title = df_text_and_title['title_text']
Y_train_text_and_title = df_text_and_title['our rating']
"""
```

```
dataFrame text only :
data ['other'] avant drop duplicates :
 our rating
other         126
dtype: int64

data ['other'] après drop duplicates :
 our rating
other         126
dtype: int64

data ['true', 'false'] avant drop duplicates :
 our rating
false         801
true          407
other         126
dtype: int64

data ['true', 'false'] après drop duplicates :
 our rating
false         800
true          400
other         126
dtype: int64

dataFrame title only :
data ['other'] avant drop duplicates :
 our rating
other         126
dtype: int64

data ['other'] après drop duplicates :
 our rating
other         125
dtype: int64

data ['true', 'false'] avant drop duplicates :
 our rating
false         801
true          407
other         126
dtype: int64

data ['true', 'false'] après drop duplicates :
 our rating
false         702
true          400
other         125
dtype: int64

dataFrame text_title concatenate only :
data ['other'] avant drop duplicates :
 our rating
other         126
dtype: int64

data ['other'] après drop duplicates :
 our rating
other         126
dtype: int64
```

```
data ['true', 'false'] avant drop duplicates :
 our rating
false         801
true          407
other         126
dtype: int64

data ['true', 'false'] après drop duplicates :
 our rating
false         801
true          407
other         126
dtype: int64
```

-> Enregistrement DataFrame à lancer seulement si de nouveau jeux de données sont crées

In [ ]:
```
df_text.to_csv("./Data_equilibre/True_And_False_VS_Other/balanced_df_TFO_text.cs
df_title.to_csv("./Data_equilibre/True_And_False_VS_Other/balanced_df_TFO_title.
df_text_and_title.to_csv("./Data_equilibre/True_And_False_VS_Other/balanced_df_T
```

-> Chargement des jeux de données équilibrés

In [ ]:
```
df_text = pd.read_csv("./Data_equilibre/True_And_False_VS_Other/balanced_df_TFO_
X_train_text  = df_text['text']
Y_train_text = df_text['our rating']
df_title = pd.read_csv("./Data_equilibre/True_And_False_VS_Other/balanced_df_TFO
X_train_title = df_title['title']
Y_train_title = df_title['our rating']
df_text_and_title = pd.read_csv("./Data_equilibre/True_And_False_VS_Other/balanc
X_train_text_and_title = df_text_and_title['title_text']
Y_train_text_and_title = df_text_and_title['our rating']
```

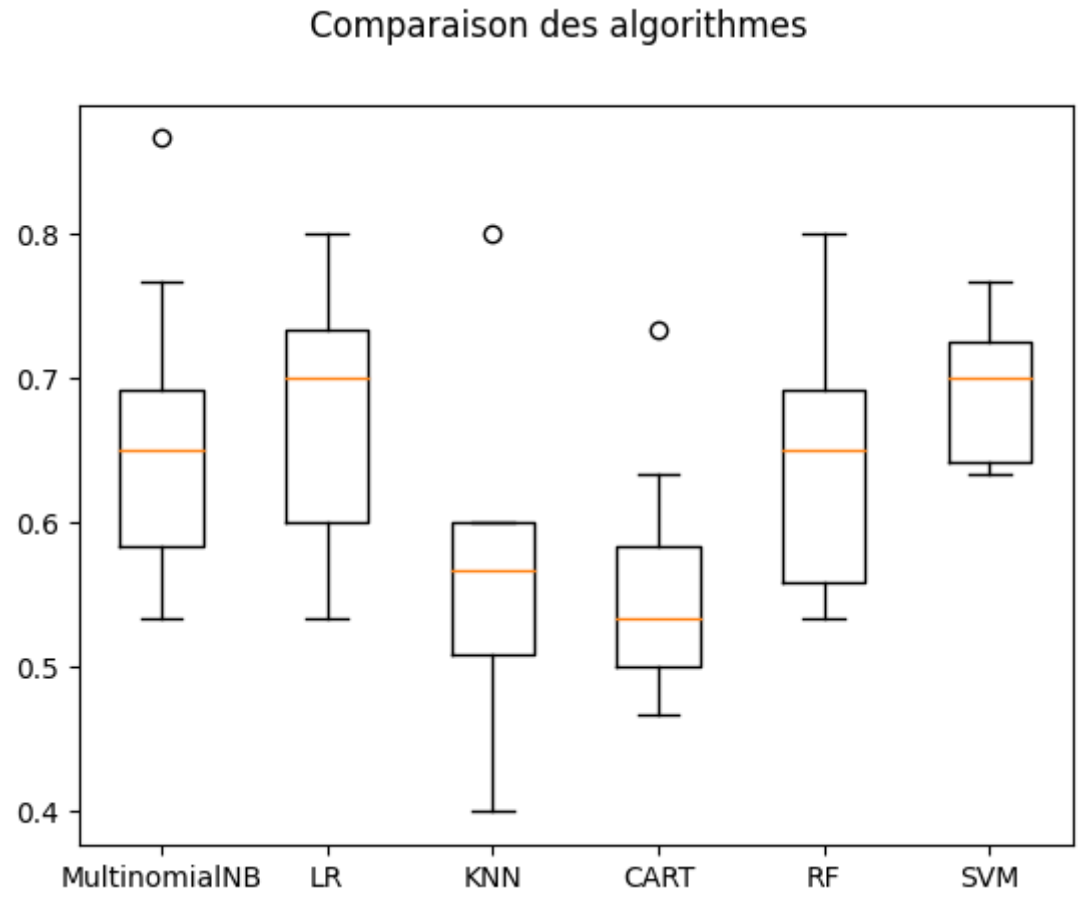
-> Première prédiction des modèles

Cela permet de partir sur une première estimation pour savoir lesquels sont les plus performants puis nous allons vérifier cela après les différents test

In [ ]:
```
print("Fonction Text")
testAllModel(X_train_text,Y_train_text,5)
```

```
Fonction Text
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\33683\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

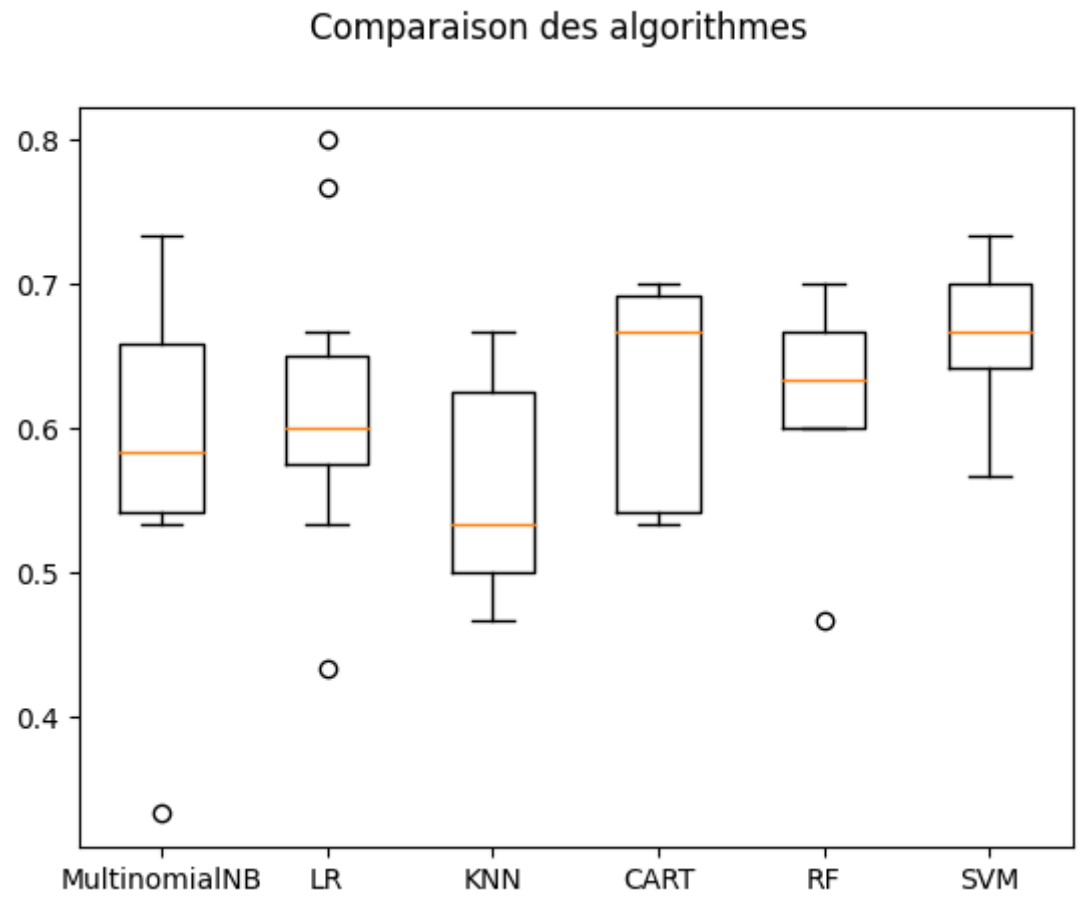
```
Evaluation de  MultinomialNB
MultinomialNB : 0.660 (0.095) in 0.534 s
Evaluation de  LR
LR : 0.677 (0.083) in 2.817 s
Evaluation de  KNN
KNN : 0.557 (0.109) in 1.219 s
Evaluation de  CART
CART : 0.550 (0.079) in 8.088 s
Evaluation de  RF
RF : 0.643 (0.086) in 12.640 s
Evaluation de  SVM
SVM : 0.690 (0.045) in 308.819 s

Le meilleur resultat :
Classifier :  SVM  accuracy : 0.690  (0.045)  en 308.819   s

Tous les résultats :

Classifier :  SVM  accuracy : 0.690  (0.045)  en 308.819   s
Classifier :  LR  accuracy : 0.677  (0.083)  en 2.817   s
Classifier :  MultinomialNB  accuracy : 0.660  (0.095)  en 0.534   s
Classifier :  RF  accuracy : 0.643  (0.086)  en 12.640   s
Classifier :  KNN  accuracy : 0.557  (0.109)  en 1.219   s
Classifier :  CART  accuracy : 0.550  (0.079)  en 8.088   s
```



In [ ]:
```
print("Fonction Titre")
testAllModel(X_train_title,Y_train_title,5)
```

```
Fonction Titre
Evaluation de  MultinomialNB
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\33683\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

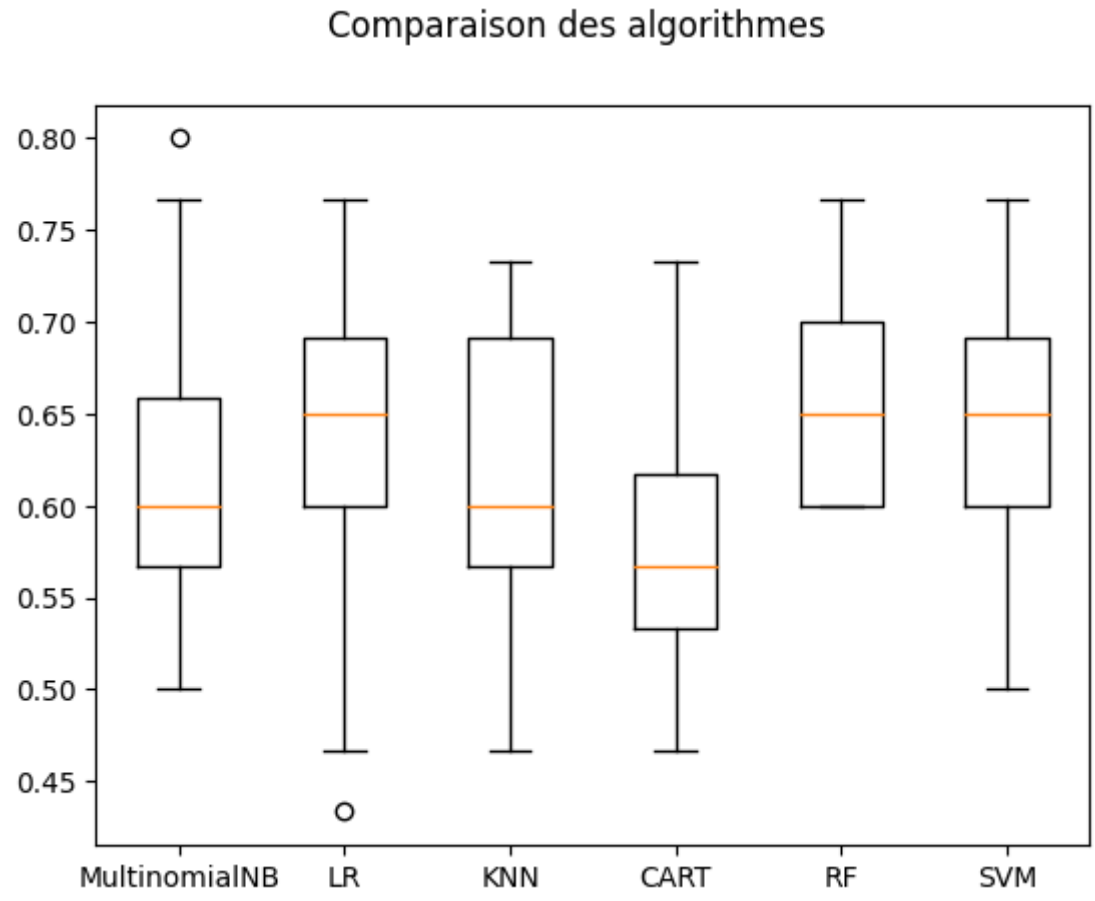
```
MultinomialNB : 0.587 (0.107) in 0.147 s
Evaluation de  LR
LR : 0.617 (0.101) in 0.443 s
Evaluation de  KNN
KNN : 0.560 (0.071) in 0.367 s
Evaluation de  CART
CART : 0.627 (0.071) in 1.949 s
Evaluation de  RF
RF : 0.623 (0.062) in 7.689 s
Evaluation de  SVM
SVM : 0.667 (0.045) in 1.894 s

Le meilleur resultat :
Classifier :  SVM  accuracy : 0.667  (0.045)  en 1.894   s

Tous les résultats :

Classifier :  SVM  accuracy : 0.667  (0.045)  en 1.894   s
Classifier :  CART  accuracy : 0.627  (0.071)  en 1.949   s
Classifier :  RF  accuracy : 0.623  (0.062)  en 7.689   s
Classifier :  LR  accuracy : 0.617  (0.101)  en 0.443   s
Classifier :  MultinomialNB  accuracy : 0.587  (0.107)  en 0.147   s
Classifier :  KNN  accuracy : 0.560  (0.071)  en 0.367   s
```



In [ ]:
```
print("Fonction titre et text ")
testAllModel(X_train_text_and_title, Y_train_text_and_title,5)
```

```
Fonction titre et text
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\33683\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

```
Evaluation de  MultinomialNB
MultinomialNB : 0.627 (0.090) in 0.522 s
Evaluation de  LR
LR : 0.623 (0.099) in 2.528 s
Evaluation de  KNN
KNN : 0.610 (0.084) in 0.923 s
Evaluation de  CART
CART : 0.577 (0.076) in 8.669 s
Evaluation de  RF
RF : 0.660 (0.059) in 12.323 s
Evaluation de  SVM
SVM : 0.640 (0.080) in 305.789 s

Le meilleur resultat :
Classifier :  RF  accuracy : 0.660  (0.059)  en 12.323   s

Tous les résultats :

Classifier :  RF  accuracy : 0.660  (0.059)  en 12.323   s
Classifier :  SVM  accuracy : 0.640  (0.080)  en 305.789   s
Classifier :  MultinomialNB  accuracy : 0.627  (0.090)  en 0.522   s
Classifier :  LR  accuracy : 0.623  (0.099)  en 2.528   s
Classifier :  KNN  accuracy : 0.610  (0.084)  en 0.923   s
Classifier :  CART  accuracy : 0.577  (0.076)  en 8.669   s
```



---

1. Test des Classifieurs avec les données Brutes

1.1 Test SVC

---> Texte

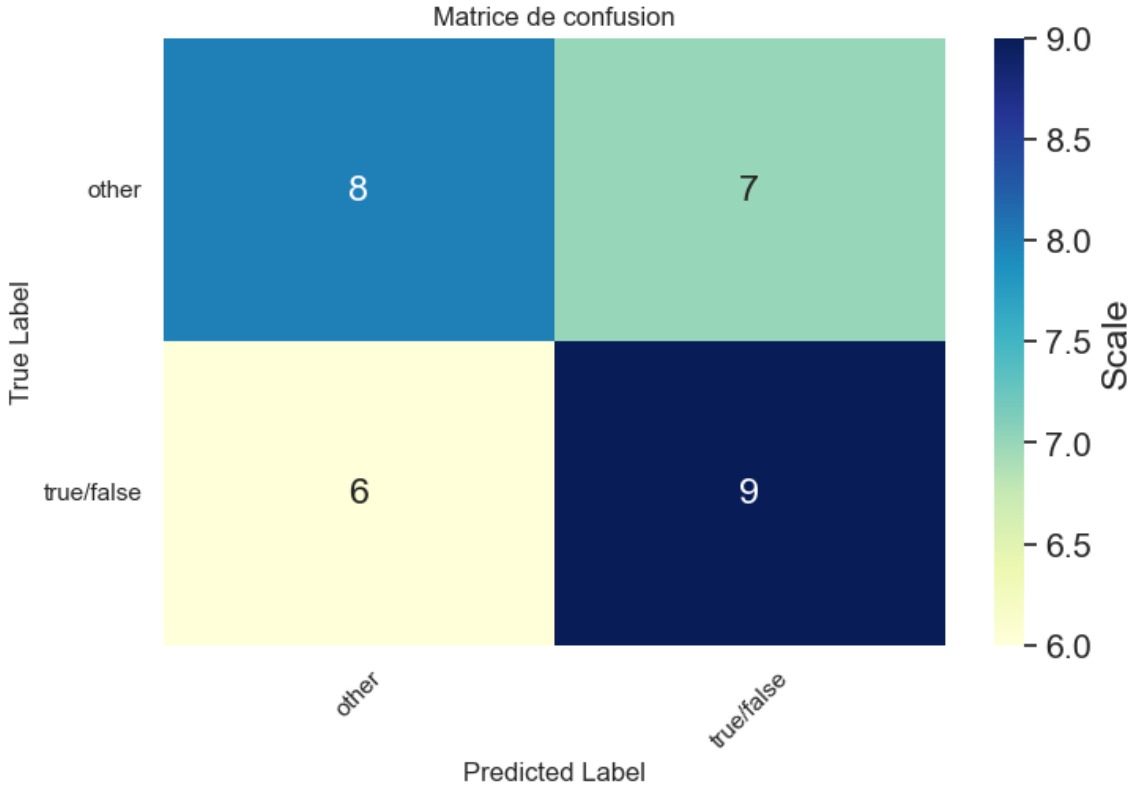
In [ ]:
```
print("Test SVC Text")
testSVC(X_train_text,Y_train_text, 5,'/True_And_False_VS_Other/Texte/data_svc_TF
```

```
Test SVC Text
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'svm']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'svm__C': [0.001, 0.01, 0.1, 1, 10], 's
vm__gamma': [0.001, 0.01, 0.1, 1], 'svm__kernel': ['linear', 'rbf', 'poly', 'si
gmoid']}
Fitting 5 folds for each of 2560 candidates, totalling 12800 fits
réalisé en  6501.005 s
Meilleur résultat : 0.703
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: True
        cleaner__getstemmer: False
        cleaner__removedigit: True
        svm__C: 10
        svm__gamma: 1
        svm__kernel: 'rbf'
        tfidf__lowercase: False
        tfidf__stop_words: None

Les premiers résultats :
      cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
951                        True                False                  True
620                        True                 True                 False
544                        True                 True                 False
528                        True                 True                 False
514                        True                 True                 False

     svm__C  svm__gamma svm__kernel  tfidf__lowercase tfidf__stop_words  \
951    10.0       1.000         rbf             False              None
620    10.0       0.100     sigmoid              True           english
544     1.0       0.100      linear              True           english
528     1.0       0.010      linear              True           english
514     1.0       0.001      linear             False           english

     accuracy
951  0.703333
620  0.700000
544  0.700000
528  0.700000
514  0.700000
```

In [ ]:
```
from sklearn.model_selection import train_test_split
import pickle
# Création d'un jeu d'apprentissage et de test
trainsize=0.9 # 90% pour le jeu d'apprentissage, il reste 10% du jeu de données
testsize= 0.1
seed=30
```

```
train_text,test_text,train_note,test_note=train_test_split(X_train_text,Y_train_

pipeline=Pipeline([
    ("cleaner", TextNormalizer(removedigit=True, getlemmatisation=True, getstemm
    ("tfidf", TfidfVectorizer(lowercase=False, stop_words=None)),
    ('svm', SVC(C=10, gamma=1, kernel='rbf',probability=True))
])
pipeline.fit(train_text,train_note)
filename='./Modele/True_And_False_VS_Other/TFO_svm_not_DE.pkl'
print("Sauvegarde du modèle dans ", filename)
pickle.dump(pipeline, open(filename, "wb"))


print ("Chargement du modèle \n")
# le chargement se fait via la fonction load
clf_loaded = pickle.load(open(filename, 'rb'))
# affichage du modèle sauvegardé
print (clf_loaded)


# test  avec les données qu'il a apprise c'est parfait woahhha c'est beau
y_pred = clf_loaded.predict(test_text)
# autres mesures et matrice de confusion
MyshowAllScores(test_note,y_pred)
```

```
Sauvegarde du modèle dans  ./Modele/TFO_svm_not_DE.pkl
Chargement du modèle

Pipeline(steps=[('cleaner',
                 TextNormalizer(getlemmatisation=True, removedigit=True)),
                ('tfidf', TfidfVectorizer(lowercase=False)),
                ('svm', SVC(C=10, gamma=1, probability=True))])
Accuracy : 0.567
Classification Report
              precision    recall  f1-score   support

       other    0.57143   0.53333   0.55172        15
  true/false    0.56250   0.60000   0.58065        15

    accuracy                        0.56667        30
   macro avg    0.56696   0.56667   0.56618        30
weighted avg    0.56696   0.56667   0.56618        30
```



---> Titre

In [ ]:
```
print("Test SVC Titre")
testSVC(X_train_title,Y_train_title, 5, '/True_And_False_VS_Other/Titre/data_svc
```

```
Test SVC Titre
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'svm']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'svm__C': [0.001, 0.01, 0.1, 1, 10], 's
vm__gamma': [0.001, 0.01, 0.1, 1], 'svm__kernel': ['linear', 'rbf', 'poly', 'si
gmoid']}
Fitting 5 folds for each of 2560 candidates, totalling 12800 fits
réalisé en  222.986 s
Meilleur résultat : 0.660
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: True
        cleaner__getstemmer: False
        cleaner__removedigit: False
        svm__C: 0.001
        svm__gamma: 0.001
        svm__kernel: 'poly'
        tfidf__lowercase: True
        tfidf__stop_words: 'english'

Les premiers résultats :
       cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
2296                       False                False                 False
2360                       False                False                 False
2200                       False                False                  True
2536                       False                False                 False
1112                        True                False                 False

      svm__C  svm__gamma svm__kernel  tfidf__lowercase tfidf__stop_words  \
2296   0.001        1.00        poly              True           english
2360   0.010        1.00        poly              True           english
2200  10.000        0.01        poly              True           english
2536  10.000        0.10        poly              True           english
1112   0.100        0.01        poly              True           english

      accuracy
2296      0.66
2360      0.66
2200      0.66
2536      0.66
1112      0.66


        ---> Texte et Titre
```

In [ ]:
```
print("Test SVC Text and Titre")
testSVC(X_train_text_and_title,Y_train_text_and_title, 5,'/True_And_False_VS_Oth
```

```
Test SVC Text and Titre
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'svm']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'svm__C': [0.001, 0.01, 0.1, 1, 10], 's
vm__gamma': [0.001, 0.01, 0.1, 1], 'svm__kernel': ['linear', 'rbf', 'poly', 'si
gmoid']}
Fitting 5 folds for each of 2560 candidates, totalling 12800 fits
réalisé en  6559.843 s
Meilleur résultat : 0.673
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: True
        cleaner__getstemmer: False
        cleaner__removedigit: False
        svm__C: 1
        svm__gamma: 0.001
        svm__kernel: 'linear'
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
       cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
1201                       True                False                 False
1153                       True                False                 False
1261                       True                False                 False
1169                       True                False                 False
1185                       True                False                 False

      svm__C  svm__gamma svm__kernel  tfidf__lowercase tfidf__stop_words  \
1201     1.0       1.000      linear              True              None
1153     1.0       0.001      linear              True              None
1261    10.0       0.100     sigmoid              True              None
1169     1.0       0.010      linear              True              None
1185     1.0       0.100      linear              True              None

      accuracy
1201  0.673333
1153  0.673333
1261  0.673333
1169  0.673333
1185  0.673333
```

1.2 Test RFC

---> Texte

In [ ]:
```
print("Test RFC Text")
testRFC(X_train_text,Y_train_text,5,'/True_And_False_VS_Other/Texte/data_RFC_TFC
```

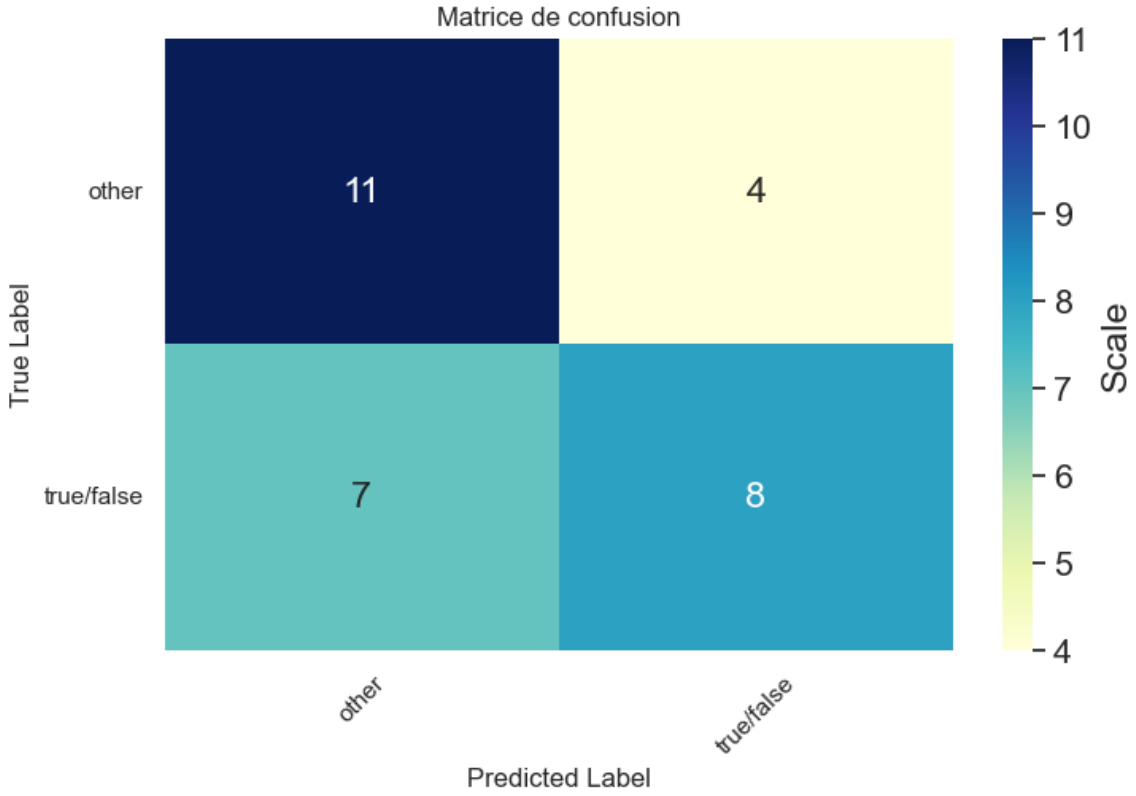
```
Test RFC Text
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'rfc']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'rfc__n_estimators': [500, 1200], 'rfc_
_max_depth': [25, 30], 'rfc__min_samples_split': [5, 10, 15], 'rfc__min_samples
_leaf': [1, 2]}
Fitting 5 folds for each of 768 candidates, totalling 3840 fits
réalisé en  3044.708 s
Meilleur résultat : 0.720
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: True
        cleaner__getstemmer: True
        cleaner__removedigit: False
        rfc__max_depth: 30
        rfc__min_samples_leaf: 2
        rfc__min_samples_split: 10
        rfc__n_estimators: 1200
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
      cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
181                       True                 True                 False
141                       True                 True                 False
507                      False                 True                 False
547                      False                 True                 False
541                      False                 True                 False

     rfc__max_depth  rfc__min_samples_leaf  rfc__min_samples_split  \
181              30                      2                      10
141              25                      2                      15
507              25                      2                       5
547              30                      1                      15
541              30                      1                      10

     rfc__n_estimators  tfidf__lowercase tfidf__stop_words  accuracy
181               1200              True              None  0.720000
141               1200              True              None  0.716667
507                500             False              None  0.713333
547                500             False              None  0.710000
541               1200              True              None  0.710000
```

In [ ]:

```
import pickle
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
# Création d'un jeu d'apprentissage et de test
trainsize=0.9 # 90% pour le jeu d'apprentissage, il reste 10% du jeu de données
testsize= 0.1
seed=30
train_text,test_text,train_note,test_note=train_test_split(X_train_text,Y_train_

pipeline=Pipeline([
    ("cleaner", TextNormalizer(removedigit=False, getlemmatisation=True, getstem
    ("tfidf", TfidfVectorizer(lowercase=True, stop_words='english')),
    ('rfc', RandomForestClassifier(max_depth=30,min_samples_leaf=2, min_samples_
])
pipeline.fit(train_text,train_note)
```

```
filename='./Modele/True_And_False_VS_Other/TFO_RFC_not_DE.pkl.pkl'
print("Sauvegarde du modèle dans ", filename)
pickle.dump(pipeline, open(filename, "wb"))



print ("Chargement du modèle \n")
# le chargement se fait via la fonction load
clf_loaded = pickle.load(open(filename, 'rb'))
# affichage du modèle sauvegardé
print (clf_loaded)


# test  avec les données qu'il a apprise c'est parfait woahhha c'est beau
y_pred = clf_loaded.predict(test_text)
# autres mesures et matrice de confusion
MyshowAllScores(test_note,y_pred)
```

```
Sauvegarde du modèle dans  ./Modele/True_And_False_VS_Other/TFO_RFC_not_DE.pkl.
pkl
Chargement du modèle

Pipeline(steps=[('cleaner',
                 TextNormalizer(getlemmatisation=True, getstemmer=True)),
                ('tfidf', TfidfVectorizer(stop_words='english')),
                ('rfc',
                 RandomForestClassifier(max_depth=30, min_samples_leaf=2,
                                        min_samples_split=10,
                                        n_estimators=1200))])
Accuracy : 0.633
Classification Report
              precision    recall  f1-score   support

       other    0.61111   0.73333   0.66667        15
  true/false    0.66667   0.53333   0.59259        15

    accuracy                        0.63333        30
   macro avg    0.63889   0.63333   0.62963        30
weighted avg    0.63889   0.63333   0.62963        30
```



---> Title

In [ ]:
```
print("Test RFC Title")
testRFC(X_train_title,Y_train_title,5,'/True_And_False_VS_Other/Titre/data_rfc_T
```

```
Test RFC Title
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'rfc']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'rfc__n_estimators': [500, 1200], 'rfc_
_max_depth': [25, 30], 'rfc__min_samples_split': [5, 10, 15], 'rfc__min_samples
_leaf': [1, 2]}
Fitting 5 folds for each of 768 candidates, totalling 3840 fits
réalisé en  1281.896 s
Meilleur résultat : 0.663
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: False
        cleaner__getstemmer: True
        cleaner__removedigit: False
        rfc__max_depth: 30
        rfc__min_samples_leaf: 1
        rfc__min_samples_split: 5
        rfc__n_estimators: 1200
        tfidf__lowercase: False
        tfidf__stop_words: 'english'

Les premiers résultats :
      cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
534                       False                 True                 False
542                       False                 True                 False
636                       False                False                  True
632                       False                False                  True
580                       False                False                  True

     rfc__max_depth  rfc__min_samples_leaf  rfc__min_samples_split  \
534              30                      1                       5
542              30                      1                      10
636              30                      1                      10
632              30                      1                      10
580              25                      1                       5

     rfc__n_estimators  tfidf__lowercase tfidf__stop_words  accuracy
534               1200             False           english  0.663333
542               1200             False           english  0.660000
636               1200              True           english  0.656667
632                500              True           english  0.656667
580               1200              True           english  0.656667


        ---> Texte et titre
```

In [ ]:
```
print("Test RFC Text and title")
testRFC(X_train_text_and_title,Y_train_text_and_title,5,'/True_And_False_VS_Othe
```

```
Test RFC Text and title
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'rfc']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'rfc__n_estimators': [500, 1200], 'rfc_
_max_depth': [25, 30], 'rfc__min_samples_split': [5, 10, 15], 'rfc__min_samples
_leaf': [1, 2]}
Fitting 5 folds for each of 768 candidates, totalling 3840 fits
réalisé en  3288.387 s
Meilleur résultat : 0.663
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: False
        cleaner__getstemmer: True
        cleaner__removedigit: False
        rfc__max_depth: 25
        rfc__min_samples_leaf: 1
        rfc__min_samples_split: 10
        rfc__n_estimators: 500
        tfidf__lowercase: False
        tfidf__stop_words: 'english'

Les premiers résultats :
      cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
490                       False                 True                 False
369                        True                False                 False
689                       False                False                 False
467                       False                 True                  True
753                       False                False                 False

     rfc__max_depth  rfc__min_samples_leaf  rfc__min_samples_split  \
490              25                      1                      10
369              30                      2                      10
689              25                      1                      15
467              30                      2                      10
753              30                      2                      10

     rfc__n_estimators  tfidf__lowercase tfidf__stop_words  accuracy
490                500             False           english  0.663333
369                500              True              None  0.650000
689                500              True              None  0.650000
467                500             False              None  0.650000
753                500              True              None  0.646667
```

1.3 Test LR

---> Texte

In [ ]:
```
print("Test LR text")
testLR(X_train_text,Y_train_text,5,'/True_And_False_VS_Other/Texte/data_LR_TFO_T
```

```
Test LR text
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'lr']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'lr__solver': ['newton-cg', 'lbfgs', 'l
iblinear'], 'lr__penalty': ['l2'], 'lr__C': [100, 10, 1.0, 0.1, 0.01]}
Fitting 5 folds for each of 480 candidates, totalling 2400 fits
réalisé en  941.495 s
Meilleur résultat : 0.710
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: True
        cleaner__getstemmer: True
        cleaner__removedigit: False
        lr__C: 1.0
        lr__penalty: 'l2'
        lr__solver: 'liblinear'
        tfidf__lowercase: True
        tfidf__stop_words: 'english'

Les premiers résultats :
     cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
94                        True                 True                 False
92                        True                 True                 False
86                        True                 True                 False
90                        True                 True                 False
88                        True                 True                 False

    lr__C lr__penalty lr__solver  tfidf__lowercase tfidf__stop_words  accuracy
94    1.0          l2  liblinear              False          english  0.710000
92    1.0          l2  liblinear               True          english  0.710000
86    1.0          l2  newton-cg              False          english  0.706667
90    1.0          l2      lbfgs              False          english  0.706667
88    1.0          l2      lbfgs               True          english  0.706667


        ---> Titre
```

In [ ]:
```
print("Test LR Titre")
testLR(X_train_title,Y_train_title,5,'/True_And_False_VS_Other/Titre/data_LR_TFC
```

```
Test LR Titre
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'lr']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'lr__solver': ['newton-cg', 'lbfgs', 'l
iblinear'], 'lr__penalty': ['l2'], 'lr__C': [100, 10, 1.0, 0.1, 0.01]}
Fitting 5 folds for each of 480 candidates, totalling 2400 fits
réalisé en  93.201 s
Meilleur résultat : 0.653
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: True
        cleaner__getstemmer: True
        cleaner__removedigit: True
        lr__C: 1.0
        lr__penalty: 'l2'
        lr__solver: 'newton-cg'
        tfidf__lowercase: True
        tfidf__stop_words: 'english'

Les premiers résultats :
     cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
26                        True                 True                  True
28                        True                 True                  True
30                        True                 True                  True
32                        True                 True                  True
34                        True                 True                  True

    lr__C lr__penalty lr__solver  tfidf__lowercase tfidf__stop_words  accuracy
26    1.0          l2  newton-cg             False          english  0.653333
28    1.0          l2      lbfgs              True          english  0.653333
30    1.0          l2      lbfgs             False          english  0.653333
32    1.0          l2  liblinear              True          english  0.653333
34    1.0          l2  liblinear             False          english  0.653333


        ---> Texte et Titre
```

In [ ]:
```
print("Test LR Texte et Titre")
testLR(X_train_text_and_title, Y_train_text_and_title,5,'/True_And_False_VS_Othe
```

```
Test LR Texte et Titre
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'lr']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'lr__solver': ['newton-cg', 'lbfgs', 'l
iblinear'], 'lr__penalty': ['l2'], 'lr__C': [100, 10, 1.0, 0.1, 0.01]}
Fitting 5 folds for each of 480 candidates, totalling 2400 fits
réalisé en  1079.282 s
Meilleur résultat : 0.663
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: False
        cleaner__getstemmer: True
        cleaner__removedigit: False
        lr__C: 100
        lr__penalty: 'l2'
        lr__solver: 'newton-cg'
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
      cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
305                       False                 True                 False
301                       False                 True                 False
309                       False                 True                 False
307                       False                 True                 False
303                       False                 True                 False

     lr__C lr__penalty lr__solver  tfidf__lowercase tfidf__stop_words  \
305  100.0          l2      lbfgs              True              None
301  100.0          l2  newton-cg              True              None
309  100.0          l2  liblinear              True              None
307  100.0          l2      lbfgs             False              None
303  100.0          l2  newton-cg             False              None

     accuracy
305  0.663333
301  0.663333
309  0.663333
307  0.663333
303  0.663333
```

1.4 Test CART

---> Texte

In [ ]:
```
print("Test CART, Texte")
testCART(X_train_text, Y_train_text,5,'/True_And_False_VS_Other/Texte/data_CART_
```

```
Test CART, Texte
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'CART']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'CART__max_depth': [10, 20, 30], 'CART_
_min_samples_split': [2, 5, 10], 'CART__min_samples_leaf': [1, 2, 4], 'CART__cr
iterion': ['gini', 'entropy']}
Fitting 5 folds for each of 1728 candidates, totalling 8640 fits
réalisé en  3389.543 s
Meilleur résultat : 0.677
Ensemble des meilleurs paramètres :
        CART__criterion: 'entropy'
        CART__max_depth: 10
        CART__min_samples_leaf: 1
        CART__min_samples_split: 5
        cleaner__getlemmatisation: False
        cleaner__getstemmer: True
        cleaner__removedigit: True
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
      CART__criterion  CART__max_depth  CART__min_samples_leaf  \
913          entropy               10                       1
702             gini               30                       2
1203         entropy               20                       1
1157         entropy               20                       1
290             gini               20                       1

      CART__min_samples_split  cleaner__getlemmatisation  cleaner__getstemmer
\
913                         5                      False                 True
702                         2                      False                False
1203                        5                      False                 True
1157                        2                       True                 True
290                         2                       True                 True

      cleaner__removedigit  tfidf__lowercase tfidf__stop_words  accuracy
913                   True              True              None  0.676667
702                  False             False           english  0.670000
1203                  True             False              None  0.670000
1157                 False              True              None  0.666667
290                   True             False           english  0.666667


        ---> Titre
```

In [ ]:
```
print("Test CART, Title")
testCART(X_train_title, Y_train_title,5,'/True_And_False_VS_Other/Titre/data_CAR
```

```
Test CART, Title
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'CART']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'CART__max_depth': [10, 20, 30], 'CART_
_min_samples_split': [2, 5, 10], 'CART__min_samples_leaf': [1, 2, 4], 'CART__cr
iterion': ['gini', 'entropy']}
Fitting 5 folds for each of 1728 candidates, totalling 8640 fits
réalisé en  283.017 s
Meilleur résultat : 0.657
Ensemble des meilleurs paramètres :
        CART__criterion: 'gini'
        CART__max_depth: 30
        CART__min_samples_leaf: 2
        CART__min_samples_split: 2
        cleaner__getlemmatisation: False
        cleaner__getstemmer: True
        cleaner__removedigit: True
        tfidf__lowercase: True
        tfidf__stop_words: 'english'

Les premiers résultats :
     CART__criterion  CART__max_depth  CART__min_samples_leaf  \
688            gini               30                       2
754            gini               30                       2
704            gini               30                       2
690            gini               30                       2
710            gini               30                       2

     CART__min_samples_split  cleaner__getlemmatisation  cleaner__getstemmer  \
688                        2                      False                 True
754                       10                      False                 True
704                        5                       True                 True
690                        2                      False                 True
710                        5                       True                 True

     cleaner__removedigit  tfidf__lowercase tfidf__stop_words  accuracy
688                  True              True           english  0.656667
754                  True             False           english  0.653333
704                  True              True           english  0.643333
690                  True             False           english  0.643333
710                 False             False           english  0.643333


        ---> Texte et Titre
```

In [ ]:
```
print("Test CART, Text and title")
testCART(X_train_text_and_title, Y_train_text_and_title,5,'/True_And_False_VS_Ot
```

```
Test CART, Text and title
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'CART']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'CART__max_depth': [10, 20, 30], 'CART_
_min_samples_split': [2, 5, 10], 'CART__min_samples_leaf': [1, 2, 4], 'CART__cr
iterion': ['gini', 'entropy']}
Fitting 5 folds for each of 1728 candidates, totalling 8640 fits
réalisé en  4087.149 s
Meilleur résultat : 0.650
Ensemble des meilleurs paramètres :
        CART__criterion: 'gini'
        CART__max_depth: 20
        CART__min_samples_leaf: 1
        CART__min_samples_split: 2
        cleaner__getlemmatisation: True
        cleaner__getstemmer: True
        cleaner__removedigit: True
        tfidf__lowercase: True
        tfidf__stop_words: 'english'

Les premiers résultats :
     CART__criterion  CART__max_depth  CART__min_samples_leaf  \
288            gini               20                       1
608            gini               30                       1
322            gini               20                       1
416            gini               20                       2
576            gini               30                       1

     CART__min_samples_split  cleaner__getlemmatisation  cleaner__getstemmer  \
288                        2                       True                 True
608                        5                       True                 True
322                        5                       True                 True
416                        5                       True                 True
576                        2                       True                 True

     cleaner__removedigit  tfidf__lowercase tfidf__stop_words  accuracy
288                  True              True           english  0.650000
608                  True              True           english  0.646667
322                  True             False           english  0.646667
416                  True              True           english  0.643333
576                  True              True           english  0.640000
```

1.5 Test KNN

---> Texte

In [ ]:
```
print("Test KNN text")
testKNeighborsClassifier(X_train_text, Y_train_text,5,'/True_And_False_VS_Other/
```

```
Test KNN text
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'KNN']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'KNN__n_neighbors': [1, 2, 3, 4, 5, 6,
7, 8, 9, 10, 11, 12, 13, 14], 'KNN__algorithm': ['auto', 'ball_tree', 'kd_tre
e', 'brute'], 'KNN__weights': ['uniform', 'distance'], 'KNN__metric': ['minkows
ki', 'euclidean', 'manhattan']}
Fitting 5 folds for each of 10752 candidates, totalling 53760 fits
réalisé en  25578.512 s
Meilleur résultat : 0.707
Ensemble des meilleurs paramètres :
        KNN__algorithm: 'auto'
        KNN__metric: 'minkowski'
        KNN__n_neighbors: 14
        KNN__weights: 'distance'
        cleaner__getlemmatisation: True
        cleaner__getstemmer: True
        cleaner__removedigit: False
        tfidf__lowercase: True
        tfidf__stop_words: 'english'

Les premiers résultats :
      KNN__algorithm KNN__metric  KNN__n_neighbors KNN__weights  \
868             auto   minkowski                14     distance
3558       ball_tree   minkowski                14     distance
4454       ball_tree   euclidean                14     distance
4452       ball_tree   euclidean                14     distance
7142         kd_tree   euclidean                14     distance

      cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
868                        True                 True                 False
3558                       True                 True                 False
4454                       True                 True                 False
4452                       True                 True                 False
7142                       True                 True                 False

      tfidf__lowercase tfidf__stop_words  accuracy
868               True           english  0.706667
3558             False           english  0.706667
4454             False           english  0.706667
4452              True           english  0.706667
7142             False           english  0.706667


        ---> Titre
```

In [ ]:
```
print("Test KNN title")
testKNeighborsClassifier(X_train_title, Y_train_title,5,'/True_And_False_VS_Othe
```

```
Test KNN title
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'KNN']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'KNN__n_neighbors': [1, 2, 3, 4, 5, 6,
7, 8, 9, 10, 11, 12, 13, 14], 'KNN__algorithm': ['auto', 'ball_tree', 'kd_tre
e', 'brute'], 'KNN__weights': ['uniform', 'distance'], 'KNN__metric': ['minkows
ki', 'euclidean', 'manhattan']}
Fitting 5 folds for each of 10752 candidates, totalling 53760 fits
réalisé en  1517.085 s
Meilleur résultat : 0.683
Ensemble des meilleurs paramètres :
        KNN__algorithm: 'auto'
        KNN__metric: 'minkowski'
        KNN__n_neighbors: 11
        KNN__weights: 'distance'
        cleaner__getlemmatisation: False
        cleaner__getstemmer: False
        cleaner__removedigit: False
        tfidf__lowercase: True
        tfidf__stop_words: 'english'

Les premiers résultats :
      KNN__algorithm KNN__metric  KNN__n_neighbors KNN__weights  \
4284      ball_tree   euclidean                11     distance
8764          brute   minkowski                11     distance
1596           auto   euclidean                11     distance
3388      ball_tree   minkowski                11     distance
6076        kd_tree   minkowski                11     distance

      cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
4284                      False                False                 False
8764                      False                False                 False
1596                      False                False                 False
3388                      False                False                 False
6076                      False                False                 False

      tfidf__lowercase tfidf__stop_words  accuracy
4284              True           english  0.683333
8764              True           english  0.683333
1596              True           english  0.683333
3388              True           english  0.683333
6076              True           english  0.683333
```

---> Texte et Titre

In [ ]:
```
print("Test KNN text and title")
testKNeighborsClassifier(X_train_text_and_title, Y_train_text_and_title,5,'/True
```

```
Test KNN text and title
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'KNN']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'KNN__n_neighbors': [1, 2, 3, 4, 5, 6,
7, 8, 9, 10, 11, 12, 13, 14], 'KNN__algorithm': ['auto', 'ball_tree', 'kd_tre
e', 'brute'], 'KNN__weights': ['uniform', 'distance'], 'KNN__metric': ['minkows
ki', 'euclidean', 'manhattan']}
Fitting 5 folds for each of 10752 candidates, totalling 53760 fits
réalisé en  22120.986 s
Meilleur résultat : 0.673
Ensemble des meilleurs paramètres :
        KNN__algorithm: 'auto'
        KNN__metric: 'minkowski'
        KNN__n_neighbors: 9
        KNN__weights: 'distance'
        cleaner__getlemmatisation: False
        cleaner__getstemmer: True
        cleaner__removedigit: False
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
      KNN__algorithm KNN__metric  KNN__n_neighbors KNN__weights  \
4151      ball_tree   euclidean                 9     distance
8629          brute   minkowski                 9     distance
6839        kd_tree   euclidean                 9     distance
6837        kd_tree   euclidean                 9     distance
9527          brute   euclidean                 9     distance

      cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
4151                      False                 True                 False
8629                      False                 True                 False
6839                      False                 True                 False
6837                      False                 True                 False
9527                      False                 True                 False

      tfidf__lowercase tfidf__stop_words  accuracy
4151             False              None  0.673333
8629              True              None  0.673333
6839             False              None  0.673333
6837              True              None  0.673333
9527             False              None  0.673333
```

1.6 Test MB

---> Texte

In [ ]:
```
print("MultinomialNB test Text")
testMNB(X_train_text, Y_train_text,5,'/True_And_False_VS_Other/Texte/data_MNB_TF
```

```
MultinomialNB test Text
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'MultinomialNB']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'MultinomialNB__alpha': [0.1, 0.5, 1.0,
2.0], 'MultinomialNB__fit_prior': [True, False], 'MultinomialNB__force_alpha':
[True, False]}
Fitting 5 folds for each of 512 candidates, totalling 2560 fits
réalisé en  1010.455 s
Meilleur résultat : 0.707
Ensemble des meilleurs paramètres :
        MultinomialNB__alpha: 1.0
        MultinomialNB__fit_prior: True
        MultinomialNB__force_alpha: True
        cleaner__getlemmatisation: True
        cleaner__getstemmer: False
        cleaner__removedigit: False
        tfidf__lowercase: False
        tfidf__stop_words: 'english'

Les premiers résultats :
      MultinomialNB__alpha  MultinomialNB__fit_prior  \
334                    1.0                     False
302                    1.0                      True
366                    1.0                     False
270                    1.0                      True
350                    1.0                     False

     MultinomialNB__force_alpha  cleaner__getlemmatisation  \
334                        True                       True
302                       False                       True
366                       False                       True
270                        True                       True
350                        True                      False

     cleaner__getstemmer  cleaner__removedigit  tfidf__lowercase  \
334                False                 False             False
302                False                 False             False
366                False                 False             False
270                False                 False             False
350                False                 False             False

    tfidf__stop_words  accuracy
334           english  0.706667
302           english  0.706667
366           english  0.706667
270           english  0.706667
350           english  0.700000


        ---> Texte et Titre
```

In [ ]:
```
print("MultinomialNB test Title")
testMNB(X_train_title, Y_train_title,5,'/True_And_False_VS_Other/Titre/data_MNB_
```

```
MultinomialNB test Title
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'MultinomialNB']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'MultinomialNB__alpha': [0.1, 0.5, 1.0,
2.0], 'MultinomialNB__fit_prior': [True, False], 'MultinomialNB__force_alpha':
[True, False]}
Fitting 5 folds for each of 512 candidates, totalling 2560 fits
réalisé en  73.155 s
Meilleur résultat : 0.657
Ensemble des meilleurs paramètres :
        MultinomialNB__alpha: 0.1
        MultinomialNB__fit_prior: True
        MultinomialNB__force_alpha: True
        cleaner__getlemmatisation: False
        cleaner__getstemmer: True
        cleaner__removedigit: False
        tfidf__lowercase: True
        tfidf__stop_words: 'english'

Les premiers résultats :
      MultinomialNB__alpha  MultinomialNB__fit_prior  \
20                     0.1                      True
54                     0.1                      True
86                     0.1                     False
84                     0.1                     False
116                    0.1                     False

     MultinomialNB__force_alpha  cleaner__getlemmatisation  \
20                         True                      False
54                        False                      False
86                         True                      False
84                         True                      False
116                       False                      False

     cleaner__getstemmer  cleaner__removedigit  tfidf__lowercase  \
20                  True                 False              True
54                  True                 False             False
86                  True                 False             False
84                  True                 False              True
116                 True                 False              True

    tfidf__stop_words  accuracy
20            english  0.656667
54            english  0.656667
86            english  0.656667
84            english  0.656667
116           english  0.656667
```

---> Texte et Titre

In [ ]:
```
print("MultinomialNB test Text and Title")
testMNB(X_train_text_and_title, Y_train_text_and_title,5,'/True_And_False_VS_Oth
```

```
MultinomialNB test Text and Title
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'MultinomialNB']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'MultinomialNB__alpha': [0.1, 0.5, 1.0,
2.0], 'MultinomialNB__fit_prior': [True, False], 'MultinomialNB__force_alpha':
[True, False]}
Fitting 5 folds for each of 512 candidates, totalling 2560 fits
réalisé en  987.406 s
Meilleur résultat : 0.677
Ensemble des meilleurs paramètres :
        MultinomialNB__alpha: 1.0
        MultinomialNB__fit_prior: True
        MultinomialNB__force_alpha: True
        cleaner__getlemmatisation: True
        cleaner__getstemmer: True
        cleaner__removedigit: False
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
      MultinomialNB__alpha  MultinomialNB__fit_prior  \
277                    1.0                      True
295                    1.0                      True
327                    1.0                     False
293                    1.0                      True
343                    1.0                     False

     MultinomialNB__force_alpha  cleaner__getlemmatisation  \
277                        True                      False
295                       False                       True
327                        True                       True
293                       False                       True
343                        True                      False

     cleaner__getstemmer  cleaner__removedigit  tfidf__lowercase  \
277                 True                 False              True
295                 True                 False             False
327                 True                 False             False
293                 True                 False              True
343                 True                 False             False

    tfidf__stop_words  accuracy
277              None  0.676667
295              None  0.676667
327              None  0.676667
293              None  0.676667
343              None  0.676667
```

---

2. Enrichissement des Jeux de données et Nouveau test des Classifieurs

---

Après nos premiers résultats, nous nous rendons compte que la meilleur classification ce fait sur le texte. Afin d'essayer d'augmenter notre score nous essayons d'enrichir nos données en convertissant le jeu de données allemands du challenge à l'aide de l'API deepl cf le notebook Traduction dans le dossier Autre_Notebook pour la traduction. Nous concaténons ensuite nos jeu de données brut et le jeu de données allemandes et réequilibrons les jeux de données pour avoir le même nombre de true/false que de other cette fois nous ne testons sans oversampling.

In [ ]:
```
df_allemand =  pd.read_csv("./Data_brut/data_allemand_all.csv")
#display(df_allemand)
df_allemand = df_allemand.drop(columns=["ID", "Unnamed: 0"], axis=1)
df_allemand = df_allemand.replace("Other", "other")
#display(df_allemand)
df_allemand_text = initDataToTest(df_allemand, ["title"],["our rating"], ["other
```

❗ IMPORTANT ❗ Ceci est le code permettant de créer des jeux de données équilibrés avec les jeux de données allemandes, si vous souhaitez les recréers il faut recharger la cellule attention toutefois cela risque d'écraser les anciens dataframes si vous les enregistrer par la suite et faussera les résultats des classifieurs

In [ ]:
```
"""
import random
print("Valeur du dataset de base: ", data_to_test['our rating'].value_counts())
data_to_add = initDataToTest(data_to_test[~data_to_test['text'].isin(df_text['te
print("Manque une row dans le dataset :\n", data_to_add['our rating'].value_coun
print("Choix aléatoire d'une row à ajouter dans le data set")
data_to_random = data_to_test[~data_to_test['text'].isin(data_to_add['text'])]
random_row_index = random.randint(0, len(data_to_random) - 1)
random_row = data_to_random.iloc[random_row_index]
data_to_add = data_to_add.append(random_row, ignore_index=True)
print("Nouveau nombre de row : \n", data_to_add['our rating'].value_counts())
data_to_add = data_to_add.replace("false", "true/false")
print("Nouveau nombre de row : ", data_to_add['our rating'].value_counts())

"""
```

```
Valeur du dataset de base:  false    801
true     407
other    126
Name: our rating, dtype: int64
data ['true', 'false'] avant drop duplicates :
 our rating
false         726
true          331
dtype: int64

data ['true', 'false'] après drop duplicates :
 our rating
false         725
true          324
dtype: int64

Manque une row dans le dataset :
 true/false    54
Name: our rating, dtype: int64
Choix aléatoire d'une row à ajouter dans le data set
Nouveau nombre de row :
 true/false    54
false          1
Name: our rating, dtype: int64
Nouveau nombre de row :  true/false    55
Name: our rating, dtype: int64
```

-> Enregistrement DataFrame à lancer seulement si de nouveau jeux de données sont crées

In [ ]:
```
df_text_EN_and_DE = pd.concat([df_text, df_allemand_text, data_to_add])
df_text_EN_and_DE.to_csv("./Data_equilibre/True_And_False_VS_Other/balanced_df_T
```

-> Chargement des données Anglaise et Allemandes traduites

In [ ]:
```
df_text_EN_and_DE = pd.read_csv("./Data_equilibre/True_And_False_VS_Other/balanc
display(df_text_EN_and_DE)
print(df_text_EN_and_DE['our rating'].value_counts())
```

	Unnamed: 0.1	Unnamed: 0	text	our rating	title
0	0	0.0	(CNN) Even though the coronavirus pandemic con...	true/false	NaN
1	1	1.0	As we try to come to terms with the extent of ...	true/false	NaN
2	2	2.0	HOUSTON — In Texas, marijuana is illegal, and ...	true/false	NaN
3	3	3.0	Milwaukee County Executive Chris Abele high-fi...	true/false	NaN
4	4	4.0	Extremely hot days, when temperatures soar to ...	true/false	NaN
...	...	...	...	...	...
405	50	NaN	There was joy in the air Tuesday, as State Rep...	true/false	NaN
406	51	NaN	@chris_najdek Chris_najdek IG Cnajdek27@gmai...	true/false	NaN
407	52	NaN	The chant “lock her up” has finally seen fruit...	true/false	NaN
408	53	NaN	The Daily Star's FREE newsletter is spectacula...	true/false	NaN
409	54	NaN	This is HUGE! Pennsylvania Judge Patricia A. ...	true/false	BREAKING HUGE: Pennsylvania Judge Files Memora...

410 rows × 5 columns

```
true/false    205
other         205
Name: our rating, dtype: int64
```

---

2. Nouveau Test des Classifieurs avec les données Allemandes seulement avec le Texte

---

In [ ]:
```
X_train_text_DE = df_text_EN_and_DE['text']
Y_train_text_DE = df_text_EN_and_DE['our rating']
```

2.1 SVC

In [ ]:
```
print("Test SVC Text")
testSVC(X_train_text_DE,Y_train_text_DE, 5,'/True_And_False_VS_Other/Texte/data_
```

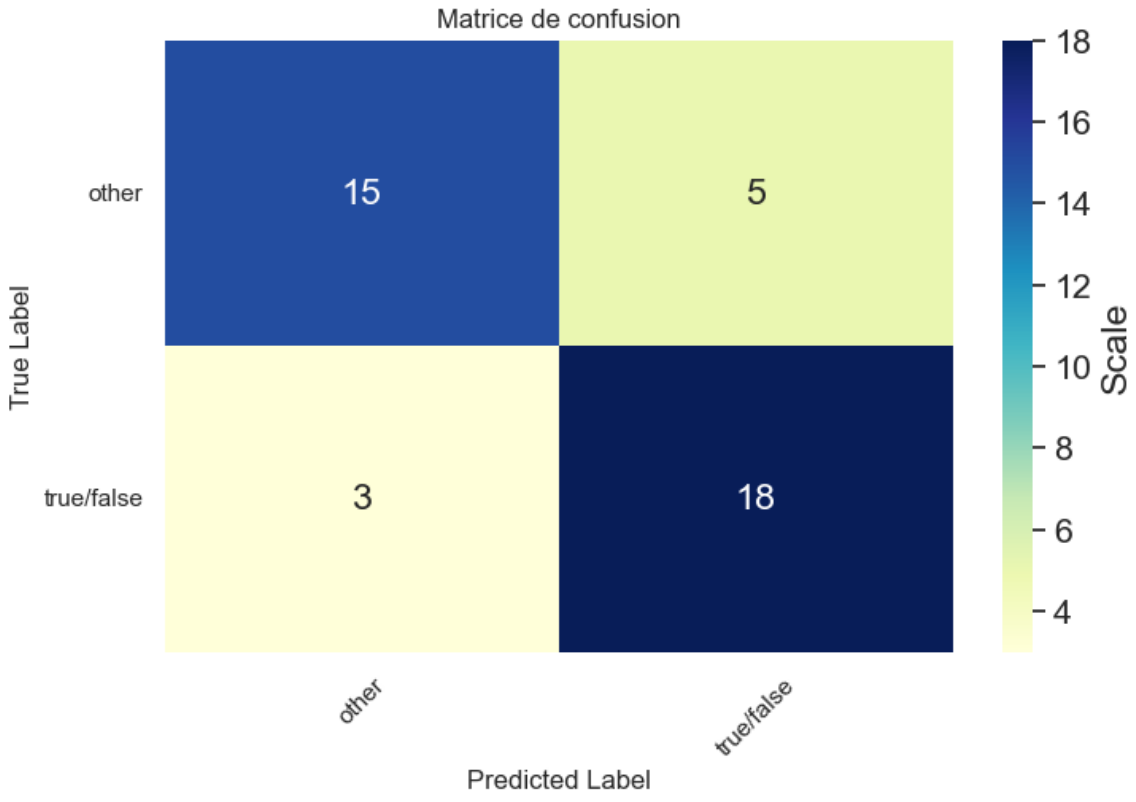
```
Test SVC Text
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'svm']
parameters :
{'cleaner__removedigit': [True, False], 'cleaner__getlemmatisation': [True, Fal
se], 'cleaner__getstemmer': [True, False], 'tfidf__stop_words': ['english', Non
e], 'tfidf__lowercase': [True, False], 'svm__C': [0.001, 0.01, 0.1, 1, 10], 'sv
m__gamma': [0.001, 0.01, 0.1, 1], 'svm__kernel': ['linear', 'rbf', 'poly', 'sig
moid']}
Fitting 5 folds for each of 2560 candidates, totalling 12800 fits
[nltk_data] Error loading omw-1.4: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>
réalisé en  6697.374 s
Meilleur résultat : 0.722
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: True
        cleaner__getstemmer: True
        cleaner__removedigit: False
        svm__C: 10
        svm__gamma: 0.1
        svm__kernel: 'rbf'
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
       cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
613                         True                 True                 False
615                         True                 True                 False
1893                       False                 True                 False
1895                       False                 True                 False
1253                        True                False                 False

      svm__C  svm__gamma svm__kernel  tfidf__lowercase tfidf__stop_words  \
613     10.0         0.1         rbf              True              None
615     10.0         0.1         rbf             False              None
1893    10.0         0.1         rbf              True              None
1895    10.0         0.1         rbf             False              None
1253    10.0         0.1         rbf              True              None

      accuracy
613   0.721951
615   0.721951
1893  0.719512
1895  0.719512
1253  0.714634
```

In [ ]:

```
from sklearn.model_selection import train_test_split
import pickle
# Création d'un jeu d'apprentissage et de test
trainsize=0.9 # 70% pour le jeu d'apprentissage, il reste 30% du jeu de données
testsize= 0.1
seed=30

train_title,test_title,train_note,test_note=train_test_split(X_train_text_DE,Y_t

pipeline=Pipeline([
    ("cleaner", TextNormalizer(removedigit=False, getlemmatisation=True, getstem
    ("tfidf", TfidfVectorizer(lowercase=True, stop_words=None)),
    ('svm', SVC(C=10, gamma=0.1, kernel='rbf',probability=True))
])
```

```
pipeline.fit(train_title,train_note)
filename='./Modele/True_And_False_VS_Other/TFO_svm_DE.pkl'
print("Sauvegarde du modèle dans ", filename)
pickle.dump(pipeline, open(filename, "wb"))


print ("Chargement du modèle \n")
# le chargement se fait via la fonction load
clf_loaded = pickle.load(open(filename, 'rb'))
# affichage du modèle sauvegardé
print (clf_loaded)


# test  avec les données qu'il a apprise c'est parfait woahhha c'est beau
y_pred = clf_loaded.predict(test_title)
# autres mesures et matrice de confusion
MyshowAllScores(test_note,y_pred)
```

```
Sauvegarde du modèle dans  ./Modele/True_And_False_VS_Other/TFO_svm_DE.pkl
Chargement du modèle

Pipeline(steps=[('cleaner',
                 TextNormalizer(getlemmatisation=True, getstemmer=True)),
                ('tfidf', TfidfVectorizer()),
                ('svm', SVC(C=10, gamma=0.1, probability=True))])
Accuracy : 0.805
Classification Report
              precision    recall  f1-score   support

       other    0.83333   0.75000   0.78947        20
  true/false    0.78261   0.85714   0.81818        21

    accuracy                        0.80488        41
   macro avg    0.80797   0.80357   0.80383        41
weighted avg    0.80735   0.80488   0.80418        41
```



2.2 RFC

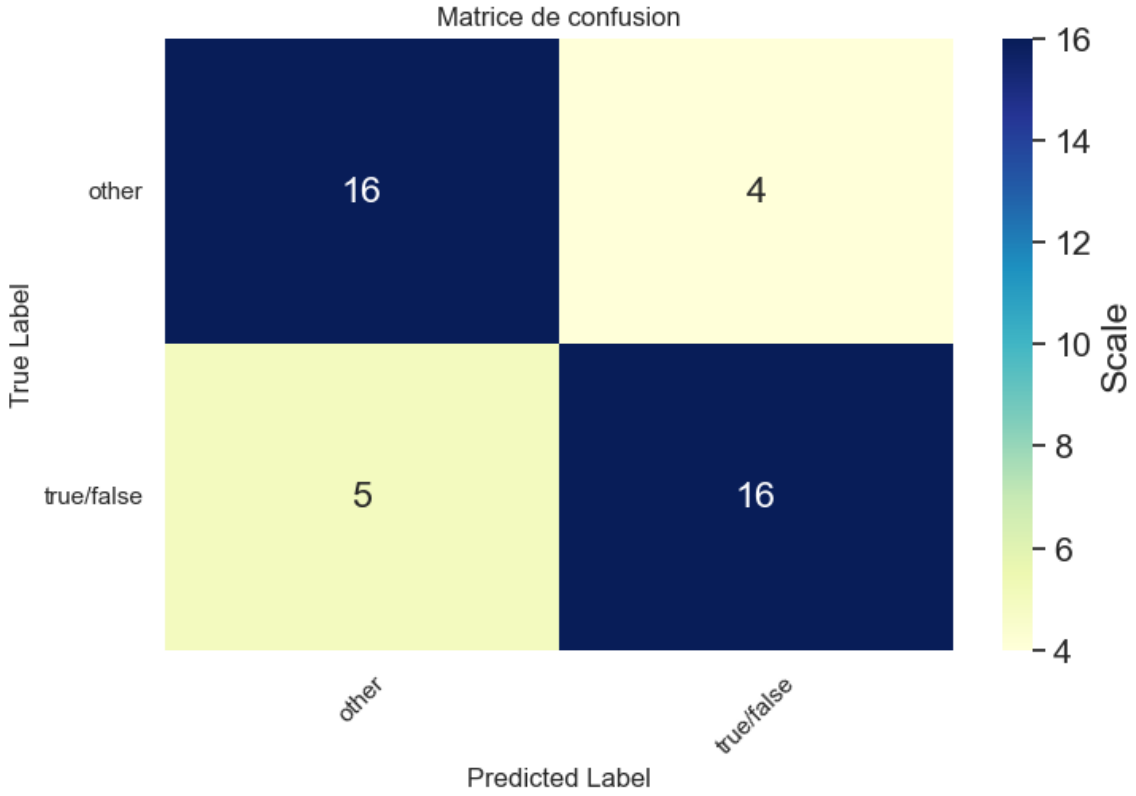
In [ ]:
```
print("Test RFC Text")
testRFC(X_train_text_DE,Y_train_text_DE,5,'/True_And_False_VS_Other/Texte/data_E
```

```
Test RFC Text
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'rfc']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'rfc__n_estimators': [500, 1200], 'rfc_
_max_depth': [25, 30], 'rfc__min_samples_split': [5, 10, 15], 'rfc__min_samples
_leaf': [1, 2]}
Fitting 5 folds for each of 768 candidates, totalling 3840 fits
réalisé en  3698.907 s
Meilleur résultat : 0.710
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: True
        cleaner__getstemmer: True
        cleaner__removedigit: False
        rfc__max_depth: 25
        rfc__min_samples_leaf: 2
        rfc__min_samples_split: 5
        rfc__n_estimators: 500
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
      cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
121                        True                True                 False
657                       False               False                  True
107                        True                True                 False
669                       False               False                  True
339                        True               False                 False

     rfc__max_depth  rfc__min_samples_leaf  rfc__min_samples_split  \
121              25                      2                       5
657              30                      2                      10
107              25                      1                      10
669              30                      2                      15
339              30                      1                       5

     rfc__n_estimators  tfidf__lowercase tfidf__stop_words  accuracy
121                500              True              None  0.709756
657                500              True              None  0.704878
107                500             False              None  0.704878
669               1200              True              None  0.704878
339                500             False              None  0.702439
```

In [ ]:
```
import pickle
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
# Création d'un jeu d'apprentissage et de test
trainsize=0.9 # 70% pour le jeu d'apprentissage, il reste 30% du jeu de données
testsize= 0.1
seed=30
train_title,test_title,train_note,test_note=train_test_split(X_train_text_DE,Y_t
```

```
pipeline=Pipeline([
    ("cleaner", TextNormalizer(removedigit=False, getlemmatisation=True, getstem
    ("tfidf", TfidfVectorizer(lowercase=True, stop_words='english')),
    ('rfc', RandomForestClassifier(max_depth=25,min_samples_leaf=2, min_samples_
])
pipeline.fit(train_title,train_note)
filename='./Modele/True_And_False_VS_Other/TFO_RFC_DE.pkl'
print("Sauvegarde du modèle dans ", filename)
pickle.dump(pipeline, open(filename, "wb"))



print ("Chargement du modèle \n")
# le chargement se fait via la fonction load
clf_loaded = pickle.load(open(filename, 'rb'))
# affichage du modèle sauvegardé
print (clf_loaded)


# test  avec les données qu'il a apprise c'est parfait woahhha c'est beau
y_pred = clf_loaded.predict(test_title)
# autres mesures et matrice de confusion
MyshowAllScores(test_note,y_pred)
```

```
Sauvegarde du modèle dans  ./Modele/True_And_False_VS_Other/TFO_RFC_DE.pkl
Chargement du modèle

Pipeline(steps=[('cleaner',
                 TextNormalizer(getlemmatisation=True, getstemmer=True)),
                ('tfidf', TfidfVectorizer(stop_words='english')),
                ('rfc',
                 RandomForestClassifier(max_depth=25, min_samples_leaf=2,
                                        min_samples_split=5,
                                        n_estimators=500))])
Accuracy : 0.780
Classification Report
              precision    recall  f1-score   support

       other    0.76190   0.80000   0.78049        20
  true/false    0.80000   0.76190   0.78049        21

    accuracy                        0.78049        41
   macro avg    0.78095   0.78095   0.78049        41
weighted avg    0.78142   0.78049   0.78049        41
```



2.3 LR

In [ ]:
```
print("Test LR text")
testLR(X_train_text_DE,Y_train_text_DE,5,'/True_And_False_VS_Other/Texte/data_EN
```

```
Test LR text
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'lr']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'lr__solver': ['newton-cg', 'lbfgs', 'l
iblinear'], 'lr__penalty': ['l2'], 'lr__C': [100, 10, 1.0, 0.1, 0.01]}
Fitting 5 folds for each of 480 candidates, totalling 2400 fits
réalisé en  1256.716 s
Meilleur résultat : 0.710
Ensemble des meilleurs paramètres :
        cleaner__getlemmatisation: False
        cleaner__getstemmer: True
        cleaner__removedigit: False
        lr__C: 100
        lr__penalty: 'l2'
        lr__solver: 'newton-cg'
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
      cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
301                      False                 True                 False
303                      False                 True                 False
305                      False                 True                 False
307                      False                 True                 False
309                      False                 True                 False

     lr__C lr__penalty lr__solver  tfidf__lowercase tfidf__stop_words  \
301  100.0          l2  newton-cg              True              None
303  100.0          l2  newton-cg             False              None
305  100.0          l2      lbfgs              True              None
307  100.0          l2      lbfgs             False              None
309  100.0          l2  liblinear              True              None

     accuracy
301  0.709756
303  0.709756
305  0.709756
307  0.709756
309  0.709756
```

2.4 CART

In [ ]:
```
print("Test CART, Texte")
testCART(X_train_text_DE,Y_train_text_DE,5,'/True_And_False_VS_Other/Texte/data_
```

```
Test CART, Texte
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'CART']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'CART__max_depth': [10, 20, 30], 'CART_
_min_samples_split': [2, 5, 10], 'CART__min_samples_leaf': [1, 2, 4], 'CART__cr
iterion': ['gini', 'entropy']}
Fitting 5 folds for each of 1728 candidates, totalling 8640 fits
réalisé en  4589.401 s
Meilleur résultat : 0.695
Ensemble des meilleurs paramètres :
        CART__criterion: 'gini'
        CART__max_depth: 20
        CART__min_samples_leaf: 1
        CART__min_samples_split: 5
        cleaner__getlemmatisation: True
        cleaner__getstemmer: False
        cleaner__removedigit: True
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
      CART__criterion  CART__max_depth  CART__min_samples_leaf  \
329             gini               20                       1
1612         entropy               30                       2
1580         entropy               30                       2
361             gini               20                       1
1468         entropy               30                       1

      CART__min_samples_split  cleaner__getlemmatisation  cleaner__getstemmer
\
329                         5                       True                False
1612                       10                       True                False
1580                        5                       True                False
361                        10                       True                False
1468                        2                      False                False

      cleaner__removedigit  tfidf__lowercase tfidf__stop_words  accuracy
329                   True              True              None  0.695122
1612                 False              True           english  0.678049
1580                 False              True           english  0.678049
361                   True              True              None  0.673171
1468                 False              True           english  0.673171
```

2.5 KNN

In [ ]:
```
print("Test KNN text")
testKNeighborsClassifier(X_train_text_DE,Y_train_text_DE,5,'/True_And_False_VS_C
```

```
Test KNN text
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'KNN']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'KNN__n_neighbors': [1, 2, 3, 4, 5, 6,
7, 8, 9, 10, 11, 12, 13, 14], 'KNN__algorithm': ['auto', 'ball_tree', 'kd_tre
e', 'brute'], 'KNN__weights': ['uniform', 'distance'], 'KNN__metric': ['minkows
ki', 'euclidean', 'manhattan']}
Fitting 5 folds for each of 10752 candidates, totalling 53760 fits
réalisé en  36830.463 s
Meilleur résultat : 0.712
Ensemble des meilleurs paramètres :
        KNN__algorithm: 'auto'
        KNN__metric: 'minkowski'
        KNN__n_neighbors: 4
        KNN__weights: 'distance'
        cleaner__getlemmatisation: True
        cleaner__getstemmer: True
        cleaner__removedigit: True
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
      KNN__algorithm KNN__metric  KNN__n_neighbors KNN__weights  \
6497        kd_tree   euclidean                 4     distance
227            auto   minkowski                 4     distance
2915      ball_tree   minkowski                 4     distance
2913      ball_tree   minkowski                 4     distance
1123           auto   euclidean                 4     distance

      cleaner__getlemmatisation  cleaner__getstemmer  cleaner__removedigit  \
6497                       True                 True                  True
227                        True                 True                  True
2915                       True                 True                  True
2913                       True                 True                  True
1123                       True                 True                  True

      tfidf__lowercase tfidf__stop_words  accuracy
6497              True              None  0.712195
227              False              None  0.712195
2915             False              None  0.712195
2913              True              None  0.712195
1123             False              None  0.712195


        2.6  MB
```

In [ ]:
```
print("MultinomialNB test Text")
testMNB(X_train_text_DE,Y_train_text_DE,5,'/True_And_False_VS_Other/Texte/data_E
```

```
MultinomialNB test Text
Application de gridsearch ...
pipeline : ['cleaner', 'tfidf', 'MultinomialNB']
parameters :
{'cleaner__getstemmer': [True, False], 'cleaner__removedigit': [True, False],
'cleaner__getlemmatisation': [True, False], 'tfidf__stop_words': ['english', No
ne], 'tfidf__lowercase': [True, False], 'MultinomialNB__alpha': [0.1, 0.5, 1.0,
2.0], 'MultinomialNB__fit_prior': [True, False], 'MultinomialNB__force_alpha':
[True, False]}
Fitting 5 folds for each of 512 candidates, totalling 2560 fits
réalisé en  1360.580 s
Meilleur résultat : 0.693
Ensemble des meilleurs paramètres :
        MultinomialNB__alpha: 0.1
        MultinomialNB__fit_prior: True
        MultinomialNB__force_alpha: True
        cleaner__getlemmatisation: True
        cleaner__getstemmer: False
        cleaner__removedigit: True
        tfidf__lowercase: True
        tfidf__stop_words: None

Les premiers résultats :
      MultinomialNB__alpha  MultinomialNB__fit_prior  \
105                    0.1                     False
41                     0.1                      True
73                     0.1                     False
9                      0.1                      True
0                      0.1                      True

     MultinomialNB__force_alpha  cleaner__getlemmatisation  \
105                       False                       True
41                        False                       True
73                         True                       True
9                          True                       True
0                          True                       True

     cleaner__getstemmer  cleaner__removedigit  tfidf__lowercase  \
105                False                  True              True
41                 False                  True              True
73                 False                  True              True
9                  False                  True              True
0                   True                  True              True

    tfidf__stop_words  accuracy
105              None  0.692683
41               None  0.692683
73               None  0.692683
9                None  0.692683
0             english  0.687805
```

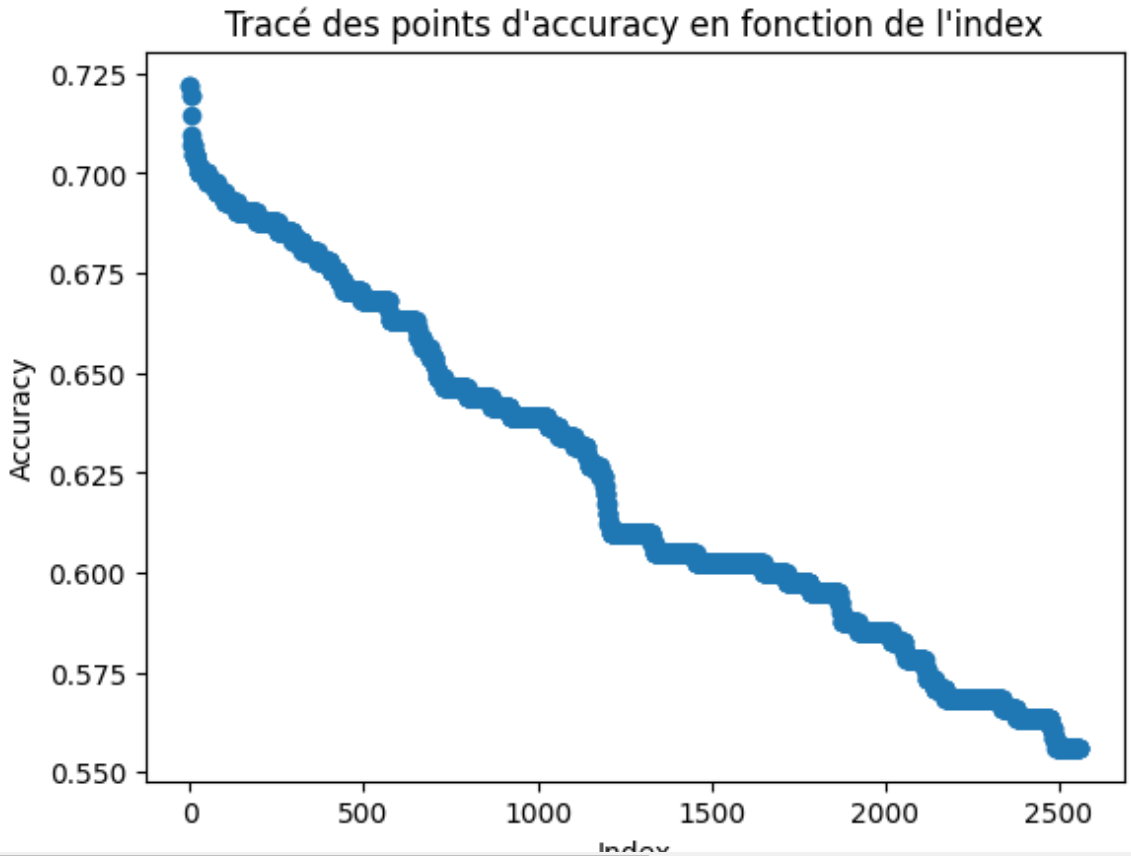
---

3. Affichage des données du meilleur modèle Courbe Roc, d'apprentissage ect...

Meilleur modèle SVC avec données allemandes

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
# Chargement des données
data = pd.read_csv('./Data_parametrage/True_And_False_VS_Other/Texte/data_EN_And


display(data.head())

df = data
# Récupérer les valeurs de x et de y
x = df.index.values
y = df['accuracy']

# Tracer les points
plt.scatter(x, y)

# Ajouter un titre et des labels d'axe
plt.title("Tracé des points d'accuracy en fonction de l'index")
plt.xlabel('Index')
plt.ylabel('Accuracy')

# Afficher le plot
plt.show()
```

	cleaner__getlemmatisation	cleaner__getstemmer	cleaner__removedigit	svm__C	svm__gamma
0	True	True	False	10.0	0.1
1	True	True	False	10.0	0.1
2	False	True	False	10.0	0.1
3	False	True	False	10.0	0.1
4	True	False	False	10.0	0.1



In [ ]:
```
from matplotlib import patches
from numpy import NaN
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
!pip install ruptures
import ruptures as rpt

# Charger les données
data = pd.read_csv('./Data_parametrage/True_And_False_VS_Other/Texte/data_EN_And

data.fillna('vide', inplace=True)
display(data.head())
# Sélectionner les colonnes à analyser
y_colonne = 'accuracy'

signal = data[y_colonne].values
model = "l2"
algo = rpt.Window(width=40, model=model, jump=1).fit(signal)
result = algo.predict(n_bkps=1)

fig, ax = plt.subplots(figsize=(10, 5))


#Tu choisis la liste des param ici a visualiser je te conseil de le faire en deu


params = ['tfidf__stop_words', 'svm__C','svm__kernel']

l = []
for param in params:
  p = data[param].value_counts(normalize=True)
```

```
  for i in range(len(p)):
    x = p.index.tolist()[i]
    l.append(str(param)+"="+str(x))


mydf = pd.DataFrame(columns=l,index=[0, 1, 2])
print(mydf)

# Plot the data points
ax.plot(data.index, data[y_colonne], 'x', color='black')
my_row=[]
for i, (start, end) in enumerate(zip([0] + result, result + [len(signal)])):
    segment = data.iloc[start:end]

    # Calculate the proportion of each unique value in the selected columns for
    param_props = []
    for param in params:
        param_value_counts = segment[param].value_counts(normalize=True)

        param_value_props = [f"{count*100:.2f}%" for count in param_value_counts

        param_value_legend = " / ".join([f"{param}={param_value} ({param_value_p
                                         for j, param_value in enumerate(param_v

        param_props.append(param_value_legend)

        for j, param_value in enumerate(param_value_counts.index):
          k=str(param)+"="+str(param_value)

          # print(k)
          param_value_counts_df = param_value_counts.reset_index()
          param_value_counts_df = param_value_counts_df.rename(columns={param: '
          # print(param_value_counts_df.loc[j, 'Parametre'])
          mydf.loc[i][k]=param_value_counts_df.loc[j, 'Parametre']

    # Join the legends for each parameter into one legend for the segment
    segment_legend = " / ".join(param_props)

    # Plot the regression line for this segment with the corresponding color ana
    sns.regplot(x=segment.index, y=y_colonne, data=segment, ax=ax, color=f'C{i+1
                label=f'Segment {i+1} ', scatter=False)
    #({segment_legend})'
    # Add text to show the start and end of each segment
    if(start != len(data[y_colonne])):
        ax.text(start, segment[y_colonne].min(), f'start: {start}', fontsize=8)
        if start not in my_row:
            my_row.append(start)
    if(end-1 != len(data[y_colonne])):
        ax.text(end, segment[y_colonne].max(), f'end: {end-1}', fontsize=8)
        if end-1 not in my_row:
            my_row.append(end-1)

    # Set the values of the corresponding row in mydf to the parameters in this




# d = pd.DataFrame(my_param, columns=nom_col)

# Set the axis labels and title
```

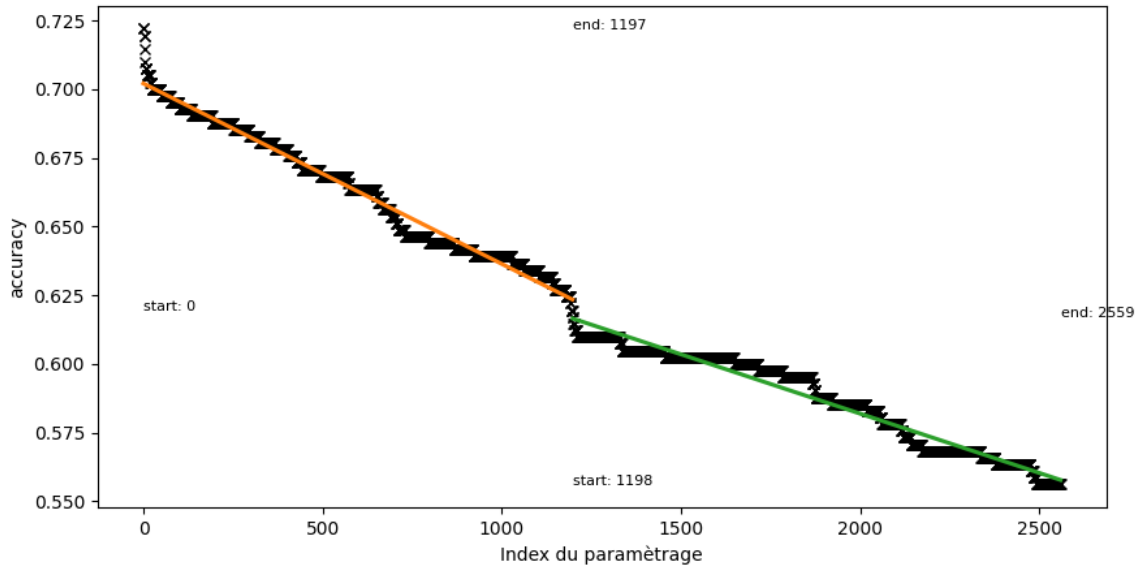
```
ax.set_xlabel('Index du paramètrage')
ax.set_ylabel(y_colonne)
# ax.set_title('Tracé de droites de régression avec ruptures des paramètrage dif

# Hide the current legend
ax.legend(loc='lower center', bbox_to_anchor=(0.5, -0.6), ncol=1)



plt.show()

fig.savefig('./Images/data_svm_TFO.png', dpi=300, bbox_inches='tight')

# créer le graphique
fig, ax = plt.subplots(figsize=(8, 6))
mydf.plot(kind='bar', ax=ax)

# ajouter des étiquettes
# ax.set_title('Proportion des paramètres sans impact pour le modèle LogisticReg
ax.set_xlabel('Segment')
ax.set_ylabel('Proportion des paramètres')
legend = ax.legend(ncol=1)

# afficher le graphique
plt.show()
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wh
eels/public/simple/
Requirement already satisfied: ruptures in /usr/local/lib/python3.9/dist-packag
es (1.1.7)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages
(from ruptures) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages
(from ruptures) (1.10.1)
```

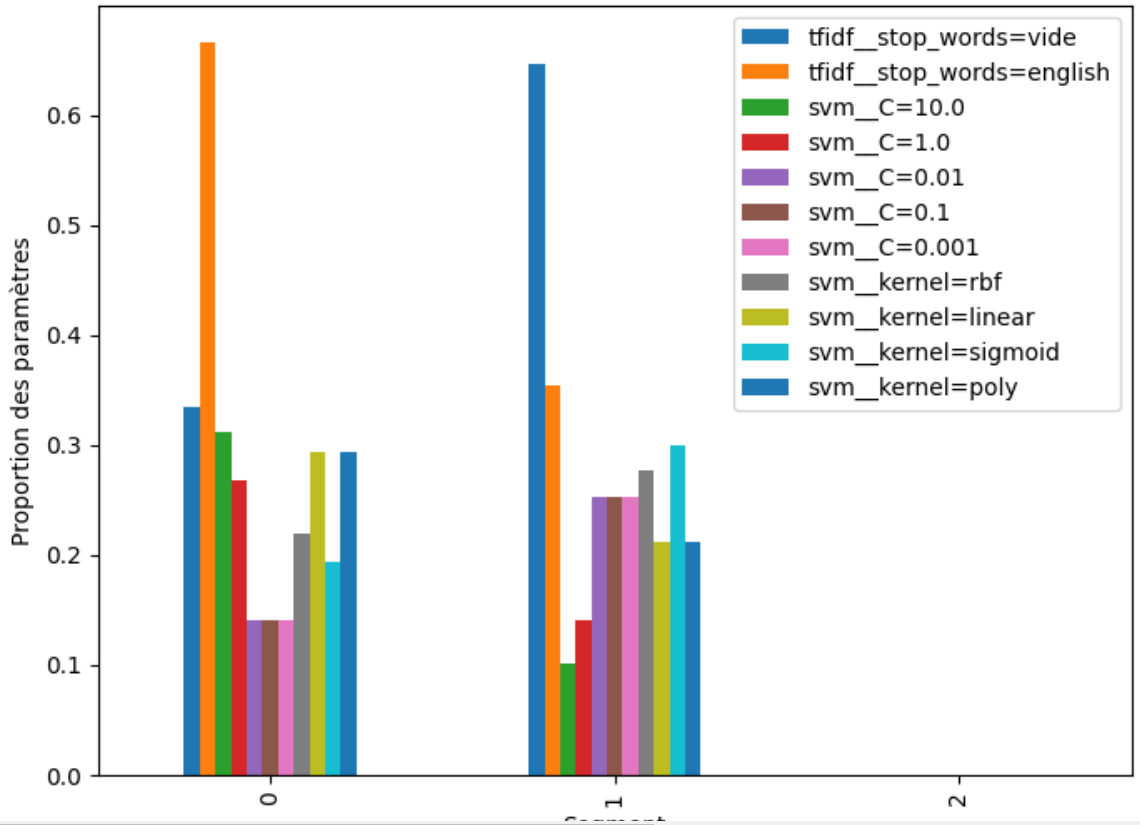
	cleaner__getlemmatisation	cleaner__getstemmer	cleaner__removedigit	svm__C	svm__gamma
0	True	True	False	10.0	0.1
1	True	True	False	10.0	0.1
2	False	True	False	10.0	0.1
3	False	True	False	10.0	0.1
4	True	False	False	10.0	0.1

```
  tfidf__stop_words=vide tfidf__stop_words=english svm__C=10.0 svm__C=1.0  \
0                    NaN                       NaN         NaN        NaN
1                    NaN                       NaN         NaN        NaN
2                    NaN                       NaN         NaN        NaN

  svm__C=0.01 svm__C=0.1 svm__C=0.001 svm__kernel=rbf svm__kernel=linear  \
0         NaN        NaN          NaN             NaN                NaN
1         NaN        NaN          NaN             NaN                NaN
2         NaN        NaN          NaN             NaN                NaN

  svm__kernel=sigmoid svm__kernel=poly
0                 NaN              NaN
1                 NaN              NaN
2                 NaN              NaN
```

```
WARNING:matplotlib.text:posx and posy should be finite values
WARNING:matplotlib.text:posx and posy should be finite values
```




```
WARNING:matplotlib.text:posx and posy should be finite values
WARNING:matplotlib.text:posx and posy should be finite values
```



Parametre non impactant

In [ ]:
```
from matplotlib import patches
from numpy import NaN
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
!pip install ruptures
import ruptures as rpt

# Charger les données
data = pd.read_csv('./Data_parametrage/True_And_False_VS_Other/Texte/data_EN_And

data.fillna('vide', inplace=True)
display(data.head())
# Sélectionner les colonnes à analyser
y_colonne = 'accuracy'

signal = data[y_colonne].values
model = "l2"
algo = rpt.Window(width=40, model=model, jump=1).fit(signal)
result = algo.predict(n_bkps=1)

fig, ax = plt.subplots(figsize=(10, 5))


#Tu choisis la liste des param ici a visualiser je te conseil de le faire en deu


params = ['cleaner__removedigit','cleaner__getlemmatisation','tfidf__lowercase',

l = []
for param in params:
  p = data[param].value_counts(normalize=True)
  for i in range(len(p)):
    x = p.index.tolist()[i]
    l.append(str(param)+"="+str(x))


mydf = pd.DataFrame(columns=l,index=[0, 1, 2])
print(mydf)

# Plot the data points
ax.plot(data.index, data[y_colonne], 'x', color='black')
my_row=[]
for i, (start, end) in enumerate(zip([0] + result, result + [len(signal)])):
    segment = data.iloc[start:end]

    # Calculate the proportion of each unique value in the selected columns for
    param_props = []
    for param in params:
        param_value_counts = segment[param].value_counts(normalize=True)

        param_value_props = [f"{count*100:.2f}%" for count in param_value_counts

        param_value_legend = " / ".join([f"{param}={param_value} ({param_value_p
                                         for j, param_value in enumerate(param_v

        param_props.append(param_value_legend)

        for j, param_value in enumerate(param_value_counts.index):
          k=str(param)+"="+str(param_value)

          # print(k)
```

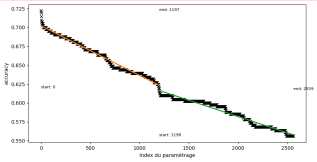
```
            param_value_counts_df = param_value_counts.reset_index()
            param_value_counts_df = param_value_counts_df.rename(columns={param: '
            # print(param_value_counts_df.loc[j, 'Parametre'])
            mydf.loc[i][k]=param_value_counts_df.loc[j, 'Parametre']

    # Join the legends for each parameter into one legend for the segment
    segment_legend = " / ".join(param_props)

    # Plot the regression line for this segment with the corresponding color ana
    sns.regplot(x=segment.index, y=y_colonne, data=segment, ax=ax, color=f'C{i+1
                label=f'Segment {i+1} ({segment_legend})', scatter=False)

    # Add text to show the start and end of each segment
    if(start != len(data[y_colonne])):
        ax.text(start, segment[y_colonne].min(), f'start: {start}', fontsize=8)
        if start not in my_row:
            my_row.append(start)
    if(end-1 != len(data[y_colonne])):
        ax.text(end, segment[y_colonne].max(), f'end: {end-1}', fontsize=8)
        if end-1 not in my_row:
            my_row.append(end-1)

# d = pd.DataFrame(my_param, columns=nom_col)

# Set the axis labels and title
ax.set_xlabel('Index du paramètrage')
ax.set_ylabel(y_colonne)
# ax.set_title('Tracé de droites de régression avec ruptures des paramètrage dif

# Hide the current legend
ax.legend(loc='lower center', bbox_to_anchor=(0.5, -0.6), ncol=1)


# créer le graphique
fig, ax = plt.subplots(figsize=(15, 10))
mydf.plot(kind='bar', ax=ax)

# ajouter des étiquettes
# ax.set_title('Proportion des paramètres sans impact pour le modèle LogisticReg
ax.set_xlabel('Segment')
ax.set_ylabel('Proportion des paramètres')
legend = ax.legend(ncol=1)

# afficher le graphique
plt.show()
```

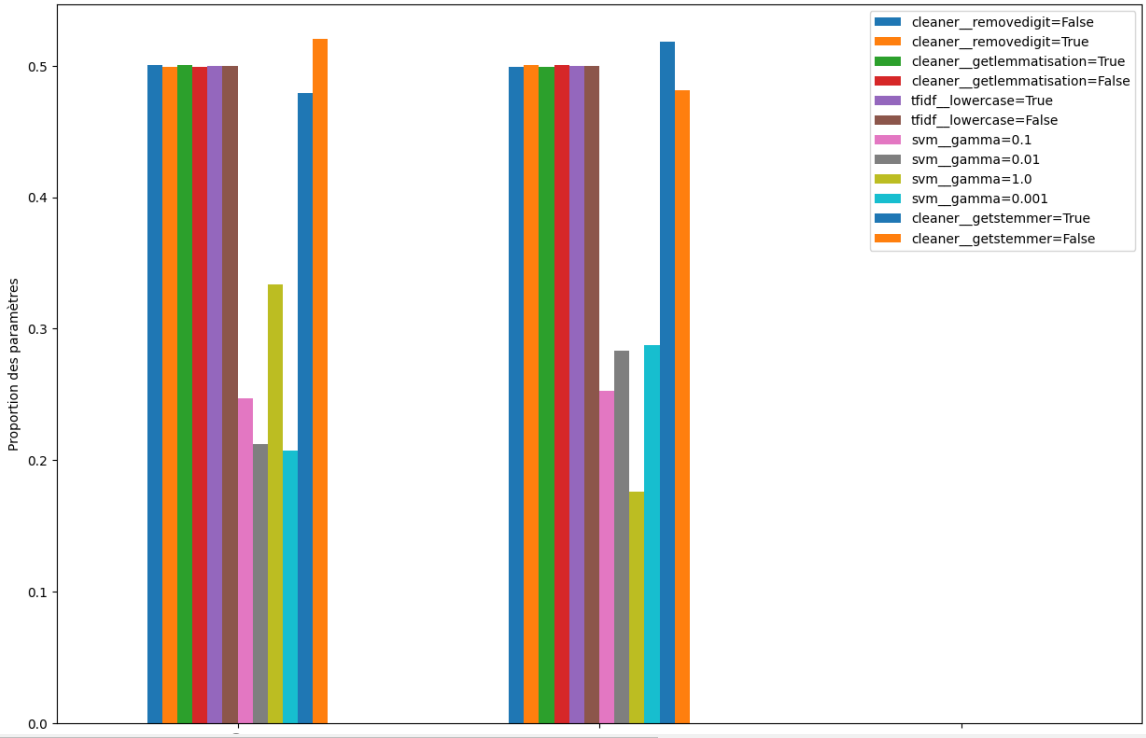
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wh
eels/public/simple/
Requirement already satisfied: ruptures in /usr/local/lib/python3.9/dist-packag
es (1.1.7)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages
(from ruptures) (1.10.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages
(from ruptures) (1.22.4)
```

	cleaner__getlemmatisation	cleaner__getstemmer	cleaner__removedigit	svm__C	svm__gamma
0	True	True	False	10.0	0.1
1	True	True	False	10.0	0.1
2	False	True	False	10.0	0.1
3	False	True	False	10.0	0.1
4	True	False	False	10.0	0.1

```
  cleaner__removedigit=False cleaner__removedigit=True  \
0                        NaN                        NaN
1                        NaN                        NaN
2                        NaN                        NaN

  cleaner__getlemmatisation=True cleaner__getlemmatisation=False  \
0                            NaN                             NaN
1                            NaN                             NaN
2                            NaN                             NaN

  tfidf__lowercase=True tfidf__lowercase=False svm__gamma=0.1 svm__gamma=0.01
\
0                   NaN                    NaN            NaN             NaN
1                   NaN                    NaN            NaN             NaN
2                   NaN                    NaN            NaN             NaN

  svm__gamma=1.0 svm__gamma=0.001 cleaner__getstemmer=True  \
0            NaN              NaN                       NaN
1            NaN              NaN                       NaN
2            NaN              NaN                       NaN

  cleaner__getstemmer=False
0                       NaN
1                       NaN
2                       NaN
```

```
WARNING:matplotlib.text:posx and posy should be finite values
WARNING:matplotlib.text:posx and posy should be finite values
```



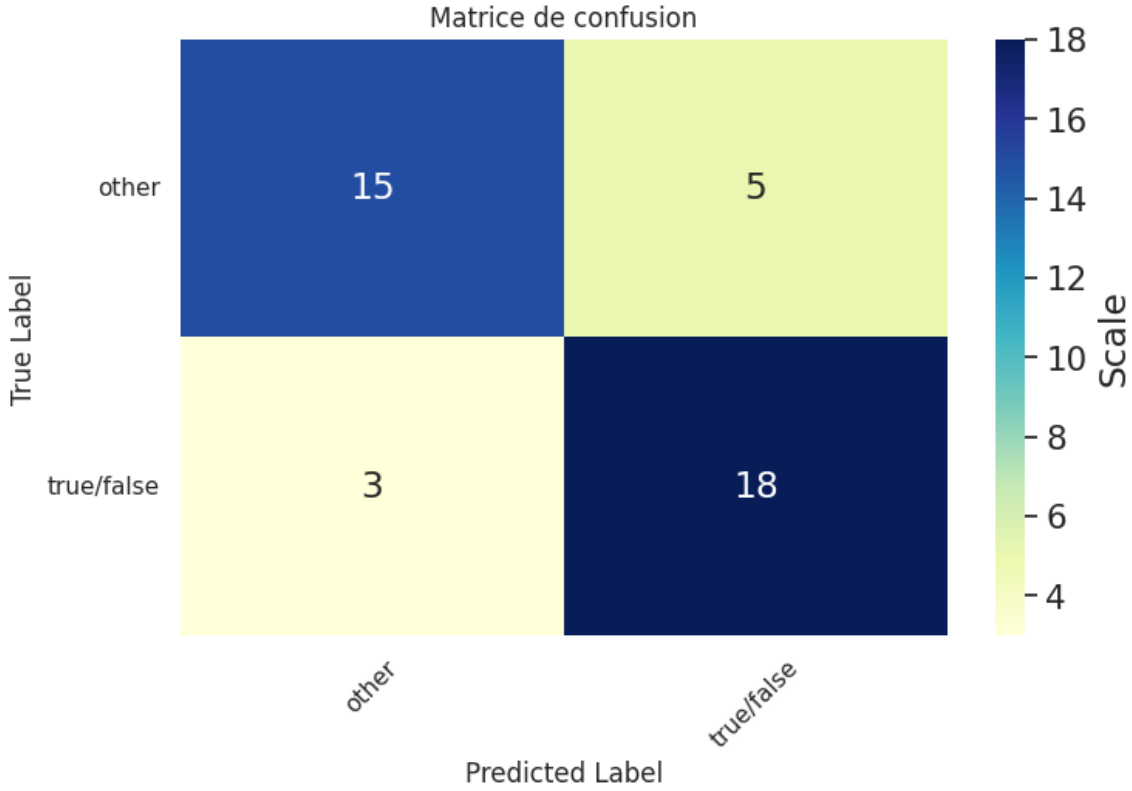


In [ ]:

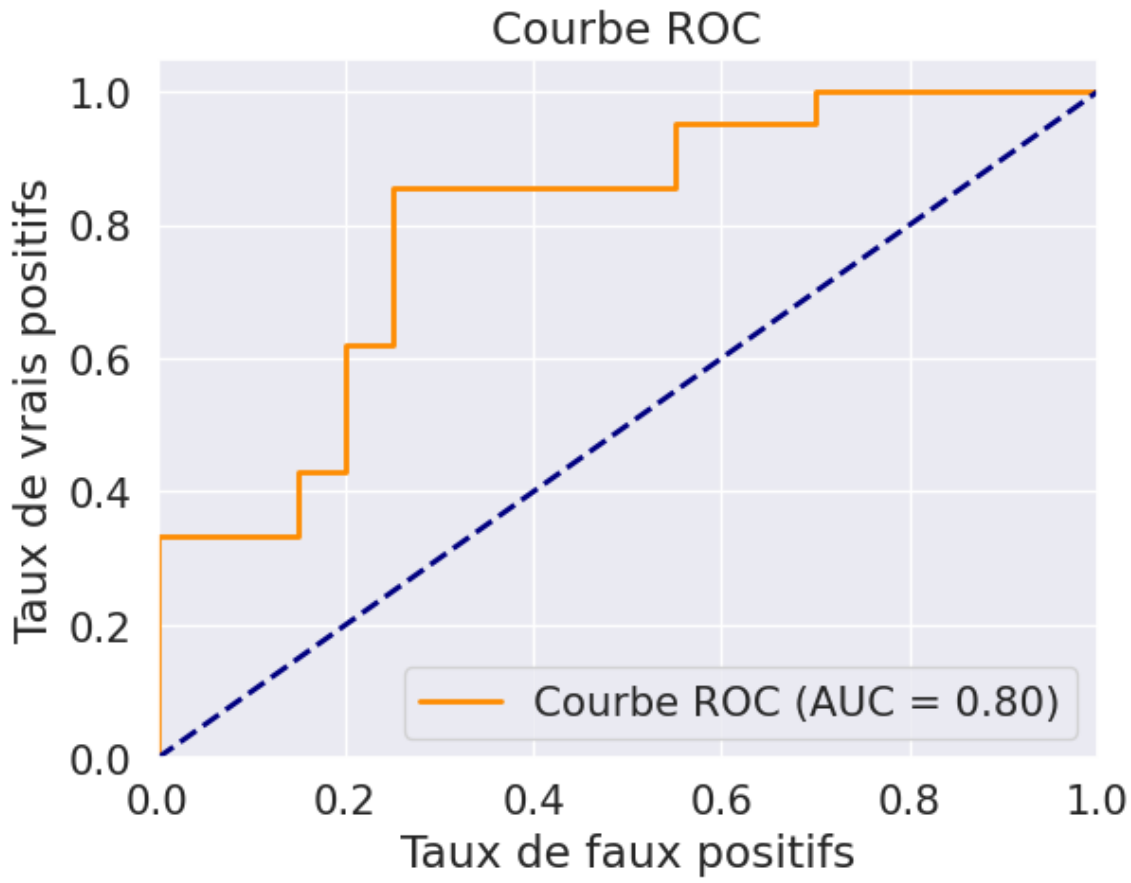
```
from sklearn.model_selection import train_test_split
import pickle
import pandas as pd
import numpy as np
from sklearn.manifold import TSNE
import plotly.express as px
# Création d'un jeu d'apprentissage et de test
trainsize=0.9 # 70% pour le jeu d'apprentissage, il reste 30% du jeu de données
testsize= 0.1
seed=30

train_title,test_title,train_note,test_note=train_test_split(X_train_text_DE,Y_t

pipeline=Pipeline([
    ("cleaner", TextNormalizer(removedigit=False, getlemmatisation=True, getstem
    ("tfidf", TfidfVectorizer(lowercase=True, stop_words=None)),
    ('svm', SVC(C=10, gamma=0.1, kernel='rbf',probability=True))
])
pipeline.fit(train_title,train_note)
filename='./Modele/True_And_False_VS_Other/TFO_svm_DE.pkl'
print("Sauvegarde du modèle dans ", filename)
pickle.dump(pipeline, open(filename, "wb"))


print ("Chargement du modèle \n")
# le chargement se fait via la fonction load
clf_loaded = pickle.load(open(filename, 'rb'))
# affichage du modèle sauvegardé
print (clf_loaded)


# test  avec les données qu'il a apprise c'est parfait woahhha c'est beau
y_pred = clf_loaded.predict(test_title)
# autres mesures et matrice de confusion
MyshowAllScores(test_note,y_pred)
```

```
# Calcul de la courbe ROC
y_pred_proba = clf_loaded.predict_proba(test_title)[:,1]

test = test_note.copy()
test = test.replace({'other': 0})
test = test.replace({'true/false': 1})
fpr, tpr, thresholds = roc_curve(test, y_pred_proba)
roc_auc = auc(fpr, tpr)

# Tracé de la courbe ROC
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='Courbe ROC (AUC = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Taux de faux positifs')
plt.ylabel('Taux de vrais positifs')
plt.title('Courbe ROC')
plt.legend(loc="lower right")
print("Courbe de Roc")
plt.show()


#Affichage Répartition des données
new_class_data = pd.DataFrame(np.column_stack((test_title, y_pred,test_note)), c

new_class_data = new_class_data.loc[~new_class_data['predictions'].isin(["false"

text_normalizer = TextNormalizer(getlemmatisation=True, removedigit=True, remove
cleaned_text = text_normalizer.fit_transform(new_class_data["text"])
tfidf = TfidfVectorizer(lowercase=False)
vector_tfidf = tfidf.fit_transform(cleaned_text)
tsne = TSNE(n_components=2, random_state=42)
projections = tsne.fit_transform(vector_tfidf.toarray())

# Ajoutez une colonne pour indiquer si la prédiction du modèle est correcte ou n
new_class_data["correct"] = (new_class_data["predictions"] == new_class_data["nc

# Tracez un graphique en utilisant Plotly pour représenter les projections obten
fig = px.scatter(x=projections[:,0], y=projections[:,1], color=new_class_data["c
print("Graphique de projections des prédictions vu le manque de données il est p
fig.show()
```

```
Sauvegarde du modèle dans  ./Modele/True_And_False_VS_Other/TFO_svm_DE.pkl
Chargement du modèle

Pipeline(steps=[('cleaner',
                 TextNormalizer(getlemmatisation=True, getstemmer=True)),
                ('tfidf', TfidfVectorizer()),
                ('svm', SVC(C=10, gamma=0.1, probability=True))])
Accuracy : 0.805
Classification Report
              precision    recall  f1-score   support

       other    0.83333   0.75000   0.78947        20
  true/false    0.78261   0.85714   0.81818        21

    accuracy                        0.80488        41
   macro avg    0.80797   0.80357   0.80383        41
weighted avg    0.80735   0.80488   0.80418        41
```




```
Courbe de Roc
```




```
Graphique de projections des prédictions vue le manque de données il est peut p
ertinent
```