Resultat_Projet_rapport_Truefalse

May 15, 2020

1 HMIN232M - Automatic fact-checking

Sabri BENBRAHIM - 21604014
Bénédicte DAYNAC - 21605192
Yann DUFRESNE - 20055179
Llivia LANGEVIN - 21604582

Imports globaux préalables

[0]:
```
import nltk as nltk

# pour colab
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

from nltk import sent_tokenize
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem.lancaster import LancasterStemmer
from nltk.stem import WordNetLemmatizer

# pour colab
!pip install contractions

import contractions

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
import scipy
import unicodedata
from time import time

import sklearn
```

```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.utils import resample

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

from scipy.stats import randint

#remove warnings
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

#python version 2 or 3
import sys
if sys.version_info[0] < 3:
    print("python2")
else:
    print("python3")

#python architecture 32 or 64 bits
import platform
```

```
print(platform.architecture()[0])

#run garbage collector
import gc
gc.collect()

#free memory size
import psutil
print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
 ↪1024)))
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt 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 wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
Requirement already satisfied: contractions in /usr/local/lib/python3.6/dist-
packages (0.0.24)
Requirement already satisfied: textsearch in /usr/local/lib/python3.6/dist-
packages (from contractions) (0.0.17)
Requirement already satisfied: Unidecode in /usr/local/lib/python3.6/dist-
packages (from textsearch->contractions) (1.1.1)
Requirement already satisfied: pyahocorasick in /usr/local/lib/python3.6/dist-
packages (from textsearch->contractions) (1.4.0)
python3
64bit
freememory=21.844 Go
```

Fonctions elémentaires récupérées

[0]:
```
#remove html (clean)
from bs4 import BeautifulSoup
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# replave contractions (clean)
import contractions
def replace_contractions(text):
    return contractions.fix(text)

# replace numbers to words (token)
import inflect
def numbers_to_words(tokens):
    result=[]
    p = inflect.engine()
    for t in tokens:
```

```
        if t.isdigit():
            t = p.number_to_words(t)
        result.append(t)
    return result

#merge words from array (vector)
def concatenate_list_data(list):
    result= ''
    for element in list:
        result += ' '+str(element)
    return result

print("ready")
```

```
ready
```

1.1 Importation du jeu de données

[0]:
```
dataFile="data-truefalse.csv" # data-truefalse-1.csv data-truefalse-2.csv␣
↪data-truefalse-3.csv data-truefalse.csv data-mixture.csv

XColumnName= "claimReview_claimReviewed"
yColumnName= "true_false_mixture"

print("Chargement CSV: ",dataFile)
dfOrigin=pd.read_csv(dataFile, sep='\t')

# ! necessaire pour mixture, a mettre en commentaire pour truefalse !
#dfOrigin.loc[dfOrigin[yColumnName] == -1, yColumnName] = 1

display(dfOrigin.head())
print("dfOrigin taille:",dfOrigin.shape,'\n')

gc.collect()
print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
↪1024)))
```

```
Chargement CSV:  data-truefalse.csv
```

	claimReview_source_id	...	true_false_mixture
0	africacheck001	...	-1
1	africacheck002	...	-1
2	africacheck003	...	-1
3	africacheck004	...	-1
4	africacheck005	...	-1

```
[5 rows x 13 columns]
```

```
dfOrigin taille: (18061, 13)

freememory=21.844 Go
```

1.2 Nettoyage et prétraitements

1.2.1 sur les claims

suppression caractères non utf8, html, formes anglaise contractées

[0]:
```
def preclean_data(df):

    # print("preclean claims")

    # clean encodage
    index = df.index.values
    for i in index:
        if not pd.isnull(df.at[i]):
            df.at[i]=unicodedata.normalize('NFKD', df.at[i]).
 ↪encode('ascii','ignore').decode('utf-8','ignore')
            # c=unicodedata.normalize('NFKD', unicode(str(df.at[i])).
 ↪encode('ascii','ignore') #python2

    # remove html
    for i in index:
        if not pd.isnull(df.at[i]):
            df.at[i]=strip_html(df.at[i])

    # replace contractions
    for i in index:
        if not pd.isnull(df.at[i]):
            df.at[i]=replace_contractions(df.at[i])

    return df
```

[0]:
```
claimsClean=dfOrigin[XColumnName].copy()

print("Avant:")
print(claimsClean.head(),'\n')

claimsClean = preclean_data(claimsClean)

print("Apres Clean:")
print(claimsClean.head(),'\n')
print("claimsClean taille:",claimsClean.shape,'\n')

gc.collect()
print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
 ↪1024)))
```

```
Avant:
0    If you sit more than 11 hours a day, there's a…
1    Don't touch soya foods as they're very bad for…
2    Nigeria accounts for about one-quarter of the …
3    Out of every [10 women] that died during child…
4    Four claims on female sexual and reproductive …
Name: claimReview_claimReviewed, dtype: object

Apres Clean:
0    If you sit more than 11 hours a day, there is …
1    do not touch soya foods as they are very bad f…
2    Nigeria accounts for about one-quarter of the …
3    Out of every [10 women] that died during child…
4    Four claims on female sexual and reproductive …
Name: claimReview_claimReviewed, dtype: object

claimsClean taille: (18061,)

freememory=21.812 Go
```

1.2.2 sur les tokens

passage en minuscule, convertion des chiffres en lettres, suppression des caractères spéciaux

[0]:
```python
def token_data(df):

    # print("clean tokens")

    df = df.astype('object')
    index = df.index.values
    for i in index:
        tokensClaimsResult=[]
        if not pd.isnull(df.at[i]):
            phrases = sent_tokenize(df.at[i])
            for p in phrases:
                tokens = word_tokenize(p)
                # minuscule
                tokens = [t.lower() for t in tokens]
                # replace number to letters
                tokens = numbers_to_words(tokens)
                # remove non-alpha signs
                tokens = [t for t in tokens if t.isalpha()]
                for t in tokens:
                    tokensClaimsResult.append(t)
            df.at[i]=tokensClaimsResult
        else:
            df.at[i]=[]
```

```
    return df
```

[0]:
```
claimsTokens=claimsClean.copy()

print("Avant:")
print(claimsTokens.head(),'\n')

claimsTokens = token_data(claimsTokens)

print("Apres Token:")
print(claimsTokens.head(),'\n')
print("claimsTokens taille:",claimsTokens.shape,'\n')

gc.collect()
print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
 ↪1024)))
```

```
Avant:
0    If you sit more than 11 hours a day, there is …
1    do not touch soya foods as they are very bad f…
2    Nigeria accounts for about one-quarter of the …
3    Out of every [10 women] that died during child…
4    Four claims on female sexual and reproductive …
Name: claimReview_claimReviewed, dtype: object

Apres Token:
0    [if, you, sit, more, than, eleven, hours, a, d…
1    [do, not, touch, soya, foods, as, they, are, v…
2    [nigeria, accounts, for, about, of, the, estim…
3    [out, of, every, ten, women, that, died, durin…
4    [four, claims, on, female, sexual, and, reprod…
Name: claimReview_claimReviewed, dtype: object

claimsTokens taille: (18061,)

freememory=21.794 Go
```

1.3 Merge des tokens

[0]:
```
def merge_tokens(df):

    # print("merge tokens")

    claimsToStr=[]
    index = df.index.values
    for i in index:
```

```
        line=""
        if df.at[i] != []:
            for w in df.at[i]:
                line+=" "+w
        df.at[i]=line.strip()

    return df
```

[0]:
```
claimsMergeInit=claimsTokens.copy()

print("Avant:")
print(claimsMergeInit.head(),'\n')

claimsMergeInit = merge_tokens(claimsMergeInit)

print("Apres Merge:")
print(claimsMergeInit.head(),'\n')
print("claimsMerge taille:",claimsMergeInit.shape,'\n')

gc.collect()
print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
 ↪1024)))
```

```
Avant:
0    [if, you, sit, more, than, eleven, hours, a, d…
1    [do, not, touch, soya, foods, as, they, are, v…
2    [nigeria, accounts, for, about, of, the, estim…
3    [out, of, every, ten, women, that, died, durin…
4    [four, claims, on, female, sexual, and, reprod…
Name: claimReview_claimReviewed, dtype: object

Apres Merge:
0    if you sit more than eleven hours a day there …
1    do not touch soya foods as they are very bad f…
2    nigeria accounts for about of the estimated to…
3    out of every ten women that died during childb…
4    four claims on female sexual and reproductive …
Name: claimReview_claimReviewed, dtype: object

claimsMerge taille: (18061,)

freememory=21.792 Go
```

1.4 Resultat sur un premier classifieur

[0]:
```
def do_classifier(dfComplet, column, dfEval, trace):

    t0 = time()

    # Vectorisation
    vectorizerT = TfidfVectorizer(min_df=2)
    vectorT = vectorizerT.fit_transform(dfEval)

    gc.collect()
    if trace:
        print("Vocabulary:")
        i=0
        limit=50
        for key, value in vectorizerT.vocabulary_.items():
            print (key, end=', ')
            i+=1
            if i>= limit:
                break
        print("\nTfidVector taille: ",vectorT.shape)
        print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024␣
↪* 1024)),"\n")

    # Jeux d'apprentissage
    X=vectorT.toarray()
    y=dfComplet[column].copy()
    gc.collect()
    if trace:
        print ("X taille: ",X.shape)
        print ("y taille: ",y.shape)
        print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024␣
↪* 1024)),"\n")

    validation_size=0.25
    testsize= 1-validation_size
    X_train,X_test,y_train,y_test=train_test_split(X,
                                                   y,
                                                   train_size=validation_size,
                                                   random_state=20,
                                                   test_size=testsize)
    gc.collect()
    if trace:
        print ("X_train taille: ",X_train.shape)
        print ("X_test taille: ",X_test.shape)
        print ("y_train taille: ",y_train.shape)
        print ("y_test taille: ",y_test.shape)
```

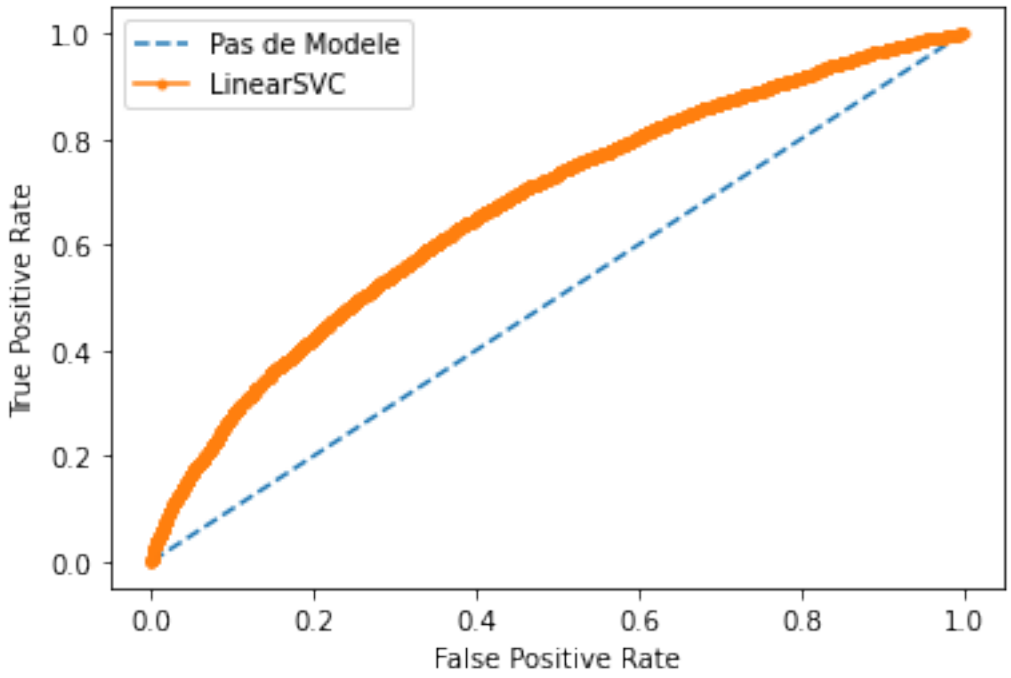
```
        print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024␣
↪* 1024)),"\n")

    # Classifieurs par defaut

    # GaussianNB
    #clfN="GaussianNB"
    #print(clfN)
    #clf = GaussianNB()

    # LinearSVC
    clfN="LinearSVC"
    print(clfN)
    #clf = LinearSVC()
    clf = CalibratedClassifierCV(LinearSVC())

    #LogisticRegression
    #clfN='LogisticRegression'
    #print(clfN)
    #clf = LogisticRegression()
    #param= {}
    #clf.set_params(**param)

    # Execution et resultats
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision,recall,fscore,support=score(y_test,y_pred,average='macro')

    print("Accuracy: %.3f%%" % (accuracy * 100.0))
    print("F1-score moyen: %.3f%%\n" % (fscore * 100.0))
    print ('Matrice de confusion:\n', confusion_matrix(y_test, y_pred),'\n')
    print (classification_report(y_test, y_pred))

    # Roc Curve
    ns_probs = [0 for _ in range(len(y_test))]
    lr_probs = clf.predict_proba(X_test)
    lr_probs = lr_probs[:, 1]
    ns_auc = roc_auc_score(y_test, ns_probs)
    lr_auc = roc_auc_score(y_test, lr_probs)
    ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
    lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)

    print('Sans modèle : ROC AUC =%.3f' % (ns_auc))
    print('Avec',clfN,' : ROC AUC =%.3f' % (lr_auc))
    plt.plot(ns_fpr, ns_tpr, linestyle='--', label='Pas de Modele')
    plt.plot(lr_fpr, lr_tpr, marker='.', label=clfN)
```

```
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()

    print("Réalisé en %.1fs" % (time() - t0))

    del vectorT, X, y, X_train, X_test, y_train, y_test
    del ns_probs, lr_probs, ns_auc, lr_auc, ns_fpr, ns_tpr, lr_fpr, lr_tpr
    print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
↪1024)),"\n")

    return fscore
```

[0]:
```
# Première valeur de score de référence
scoreInit = do_classifier(dfOrigin, yColumnName, claimsMergeInit, True)
```

```
Vocabulary:
if, you, sit, more, than, eleven, hours, day, there, is, fifty, chance, will,
die, within, the, next, three, years, do, not, touch, foods, as, they, are,
very, bad, for, breast, cancer, nigeria, accounts, about, of, estimated, to,
million, women, worldwide, out, every, ten, that, died, during, childbirth,
four, had, serious,
TfidVector taille:  (18061, 11617)
freememory=21.789 Go

X taille:  (18061, 11617)
y taille:  (18061,)
freememory=20.489 Go

X_train taille:  (4515, 11617)
X_test taille:  (13546, 11617)
y_train taille:  (4515,)
y_test taille:  (13546,)
freememory=18.921 Go

LinearSVC
Accuracy: 74.635%
F1-score moyen: 48.497%

Matrice de confusion:
 [[9880  150]
 [3286  230]]

              precision    recall  f1-score   support

          -1       0.75      0.99      0.85     10030
```

```
              1       0.61      0.07      0.12      3516

    accuracy                           0.75     13546
   macro avg       0.68      0.53      0.48     13546
weighted avg       0.71      0.75      0.66     13546

Sans modèle : ROC AUC =0.500
Avec LinearSVC  : ROC AUC =0.672
```




```
Réalisé en 6.5s
freememory=18.607 Go
```

1.5 Etude sur la suppression des Stop words

[0]:
```python
def stop_data(df, trace):

    if trace:
        print("remove stopwords")

    stop_words = set(stopwords.words('english'))
    # stop_words = set(stopwords.words('french')) #fr?
    index = df.index.values
    for i in index:
        if df.at[i] != []:
```

```
            withoutStopWordsClaims=[]
            withoutStopWordsClaims = [w for w in df.at[i] if not w in␣
 ↪stop_words]
            df.at[i]=withoutStopWordsClaims

    return df
```

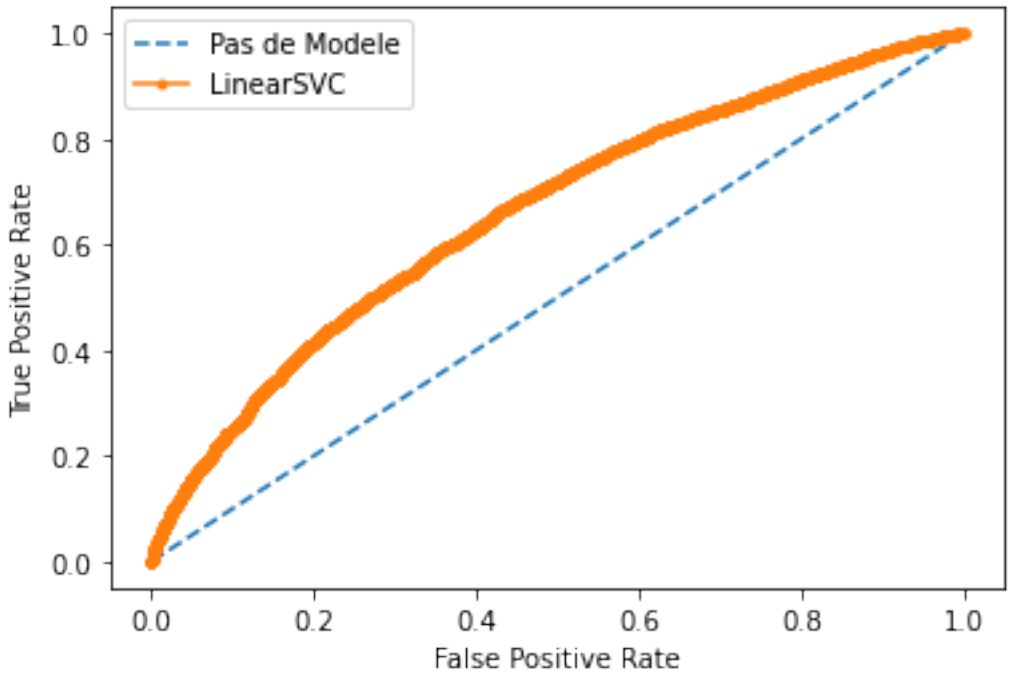
[0]:
```
claimsStop=claimsTokens.copy()

print("Avant:")
print(claimsStop.head(),'\n')

claimsStop = stop_data(claimsStop, True)

print("Apres Stop:")
print(claimsStop.head(),'\n')
print("claimsStop taille:",claimsStop.shape,'\n')

gc.collect()
print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
 ↪1024)))
```

```
Avant:
0    [if, you, sit, more, than, eleven, hours, a, d…
1    [do, not, touch, soya, foods, as, they, are, v…
2    [nigeria, accounts, for, about, of, the, estim…
3    [out, of, every, ten, women, that, died, durin…
4    [four, claims, on, female, sexual, and, reprod…
Name: claimReview_claimReviewed, dtype: object

remove stopwords
Apres Stop:
0    [sit, eleven, hours, day, fifty, chance, die, …
1            [touch, soya, foods, bad, breast, cancer]
2    [nigeria, accounts, estimated, million, circum…
3    [every, ten, women, died, childbirth, four, se…
4    [four, claims, female, sexual, reproductive, h…
Name: claimReview_claimReviewed, dtype: object

claimsStop taille: (18061,)

freememory=17.194 Go
```

[0]:
```
StopWords= False # False True

claimsMergeStop = merge_tokens(claimsStop.copy())
scoreStop = do_classifier(dfOrigin, yColumnName,claimsMergeStop, False)
```

```python
# Memorisation des parametres en fonction de l'évolution du score
if scoreStop >= scoreInit:
    StopWords= True
```

```
LinearSVC
Accuracy: 74.480%
F1-score moyen: 46.514%

Matrice de confusion:
 [[9942   88]
 [3369  147]]

              precision    recall  f1-score   support

          -1       0.75      0.99      0.85     10030
           1       0.63      0.04      0.08      3516

    accuracy                           0.74     13546
   macro avg       0.69      0.52      0.47     13546
weighted avg       0.72      0.74      0.65     13546

Sans modèle : ROC AUC =0.500
Avec LinearSVC  : ROC AUC =0.661
```



```
Réalisé en 5.3s
freememory=8.911 Go
```

Etude sur la Lemmatisation

[0]:
```python
def lemm_data(df, trace):

    if trace:
        print("lemmatize tokens")

    lemmatizer = WordNetLemmatizer()
    index = df.index.values
    for i in index:
        if df.at[i] != []:
            df.at[i]=[lemmatizer.lemmatize(word,pos='v') for word in df.at[i]]

    return df
```

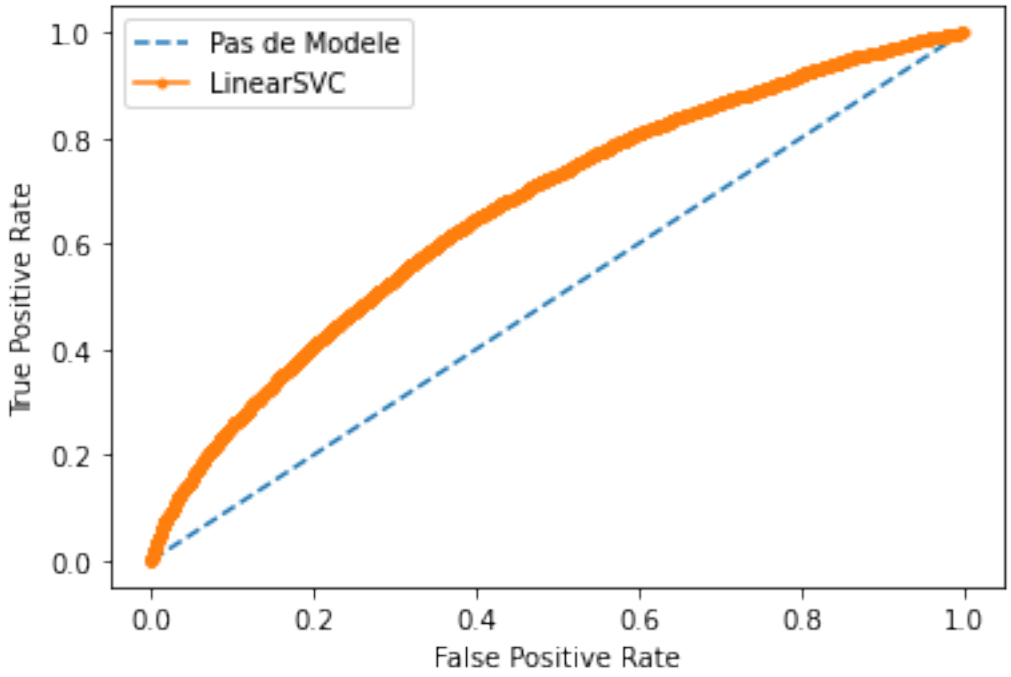
[0]:
```python
claimsLem=claimsTokens.copy()

print("Avant:")
print(claimsLem.head(),'\n')

claimsLem = lemm_data(claimsLem, True)

print("Apres Lemm:")
print(claimsLem.head(),'\n')
print("claimsLem taille:",claimsLem.shape,'\n')

gc.collect()
print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
 ↪1024)))
```

```
Avant:
0    [if, you, sit, more, than, eleven, hours, a, d…
1    [do, not, touch, soya, foods, as, they, are, v…
2    [nigeria, accounts, for, about, of, the, estim…
3    [out, of, every, ten, women, that, died, durin…
4    [four, claims, on, female, sexual, and, reprod…
Name: claimReview_claimReviewed, dtype: object

lemmatize tokens
Apres Lemm:
0    [if, you, sit, more, than, eleven, hours, a, d…
1    [do, not, touch, soya, foods, as, they, be, ve…
2    [nigeria, account, for, about, of, the, estima…
3    [out, of, every, ten, women, that, die, during…
```

```
4    [four, claim, on, female, sexual, and, reprodu…
Name: claimReview_claimReviewed, dtype: object

claimsLem taille: (18061,)

freememory=6.161 Go
```

[0]:
```python
Lemmatiz= False # False True

claimsMergeLem = merge_tokens(claimsLem.copy())
scoreLem = do_classifier(dfOrigin, yColumnName, claimsMergeLem, False)

# Memorisation des parametres en fonction de l'évolution du score
if StopWords:
    if scoreLem >= scoreStop:
        Lemmatiz= True
else:
    if scoreLem >= scoreInit:
        Lemmatiz= True
```

```
LinearSVC
Accuracy: 74.509%
F1-score moyen: 46.790%

Matrice de confusion:
 [[9935   95]
 [3358  158]]

              precision    recall  f1-score   support

          -1       0.75      0.99      0.85     10030
           1       0.62      0.04      0.08      3516

    accuracy                           0.75     13546
   macro avg       0.69      0.52      0.47     13546
weighted avg       0.72      0.75      0.65     13546

Sans modèle : ROC AUC =0.500
Avec LinearSVC  : ROC AUC =0.666
```




```
Réalisé en 4.8s
freememory=9.702 Go
```

[0]:
```
# Nettoyage complet paramétrable sans merge
def fullclean_data(column, stop, lemm, trace):

    result = preclean_data(column)
    result = token_data(result)
    if stop:
        result = stop_data(result, trace)
    if lemm:
        result = lemm_data(result, trace)

    return result
```

1.6 Etude sur l'ajout d'extras

1.6.1 Creation des jeux de données avec ajouts de colonnes

[0]:
```
def addtokens_2in1(df1,df2):

    index = df1.index.values
    for i in index:
        line=[]
```

```
        if df1.at[i] != []:
            for w in df1.at[i]:
                line.append(w)
        if df2.at[i] != []:
            for w in df2.at[i]:
                line.append(w)
        df1.at[i]=line

    return df1
```

[0]:
```
def addExtras(df, column, extraType, stop, lemm):

    if extraType == "addauthor" or extraType == "addall":
        print("Wait...")
        print("Clean author extras")
        extra1=df['creativeWork_author_name'].copy()
        extra2=df['extra_author_categories'].copy()
        extra1 = fullclean_data(extra1, stop, lemm, False)
        extra2 = fullclean_data(extra2, stop, lemm, False)

    if extraType == "addall":
        print("Clean other extras")
        extra3=df['extra_claimReview_claimReviewed_entity'].copy()
        extra4=df['extra_claimReview_claimReviewed_categories'].copy()
        extra5=df['extra_keywords_entity'].copy()
        extra6=df['extra_keywords_categories'].copy()
        extra7=df['extra_tags'].copy()
        #extra8=df['extra_title'].copy()
        extra3 = fullclean_data(extra3, stop, lemm, False)
        extra4 = fullclean_data(extra4, stop, lemm, False)
        extra5 = fullclean_data(extra5, stop, lemm, False)
        extra6 = fullclean_data(extra6, stop, lemm, False)
        extra7 = fullclean_data(extra7, stop, lemm, False)
        #extra8 = fullclean_data(extra8, stop, lemm, False)

    if extraType == "addauthor" or extraType == "addall":
        print("Extras cleaned")

    dfResult = df[column].copy()
    dfResult = fullclean_data(dfResult, stop, lemm, True)

    if extraType == "addauthor" or extraType == "addall":
        print("Add author extras")
        dfResult=addtokens_2in1(dfResult,extra1)
        dfResult=addtokens_2in1(dfResult,extra2)
        del extra1, extra2
```

```
    if extraType == "addall":
        print("Add other extras")
        dfResult=addtokens_2in1(dfResult,extra3)
        dfResult=addtokens_2in1(dfResult,extra4)
        dfResult=addtokens_2in1(dfResult,extra5)
        dfResult=addtokens_2in1(dfResult,extra6)
        dfResult=addtokens_2in1(dfResult,extra7)
        #dfResult=addtokens_2in1(dfResult,extra8)
        del extra3, extra4, extra5, extra6, extra7 #, extra8

    if extraType == "addauthor" or extraType == "addall":
        print("Extras added")

    print(extraType,"done")

    return dfResult
```

[0]:
```
includeExtra="addnone" # addnone addauthor addall

dfAjout1 = addExtras(dfOrigin, XColumnName, "addnone", StopWords, Lemmatiz)
print(dfAjout1.head())
print("dfAjout1 taille:",dfAjout1.shape,'\n')

print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
↪1024)),"\n")

dfAjout2 = addExtras(dfOrigin, XColumnName, "addauthor", StopWords, Lemmatiz)
print(dfAjout2.head())
print("dfAjout2 taille:",dfAjout2.shape,'\n')

print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
↪1024)),"\n")

dfAjout3 = addExtras(dfOrigin, XColumnName, "addall", StopWords, Lemmatiz)
print(dfAjout3.head())
print("dfAjout3 taille:",dfAjout3.shape,'\n')

print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
↪1024)))
```

```
addnone done
0    [if, you, sit, more, than, eleven, hours, a, d…
1    [do, not, touch, soya, foods, as, they, are, v…
2    [nigeria, accounts, for, about, of, the, estim…
3    [out, of, every, ten, women, that, died, durin…
4    [four, claims, on, female, sexual, and, reprod…
Name: claimReview_claimReviewed, dtype: object
```

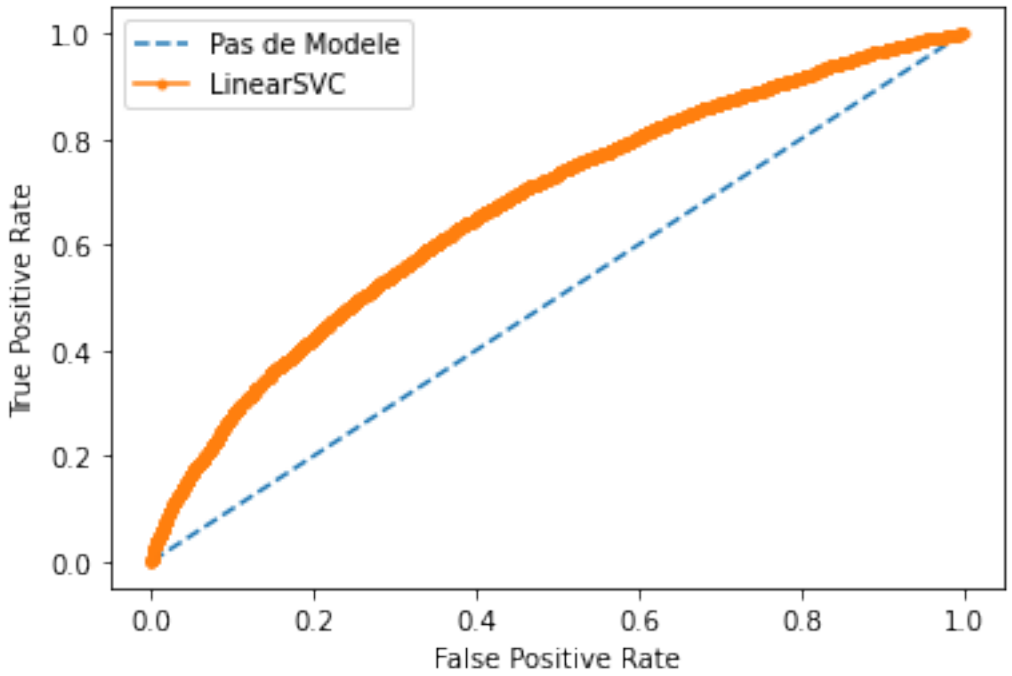
```
dfAjout1 taille: (18061,)

freememory=18.577 Go

Wait…
Clean author extras
Extras cleaned
Add author extras
Extras added
addauthor done
0    [if, you, sit, more, than, eleven, hours, a, d…
1    [do, not, touch, soya, foods, as, they, are, v…
2    [nigeria, accounts, for, about, of, the, estim…
3    [out, of, every, ten, women, that, died, durin…
4    [four, claims, on, female, sexual, and, reprod…
Name: claimReview_claimReviewed, dtype: object
dfAjout2 taille: (18061,)

freememory=18.530 Go

Wait…
Clean author extras
Clean other extras
Extras cleaned
Add author extras
Add other extras
Extras added
addall done
0    [if, you, sit, more, than, eleven, hours, a, d…
1    [do, not, touch, soya, foods, as, they, are, v…
2    [nigeria, accounts, for, about, of, the, estim…
3    [out, of, every, ten, women, that, died, durin…
4    [four, claims, on, female, sexual, and, reprod…
Name: claimReview_claimReviewed, dtype: object
dfAjout3 taille: (18061,)

freememory=18.377 Go
```

1.6.2 Evaluation des changements suite aux ajouts

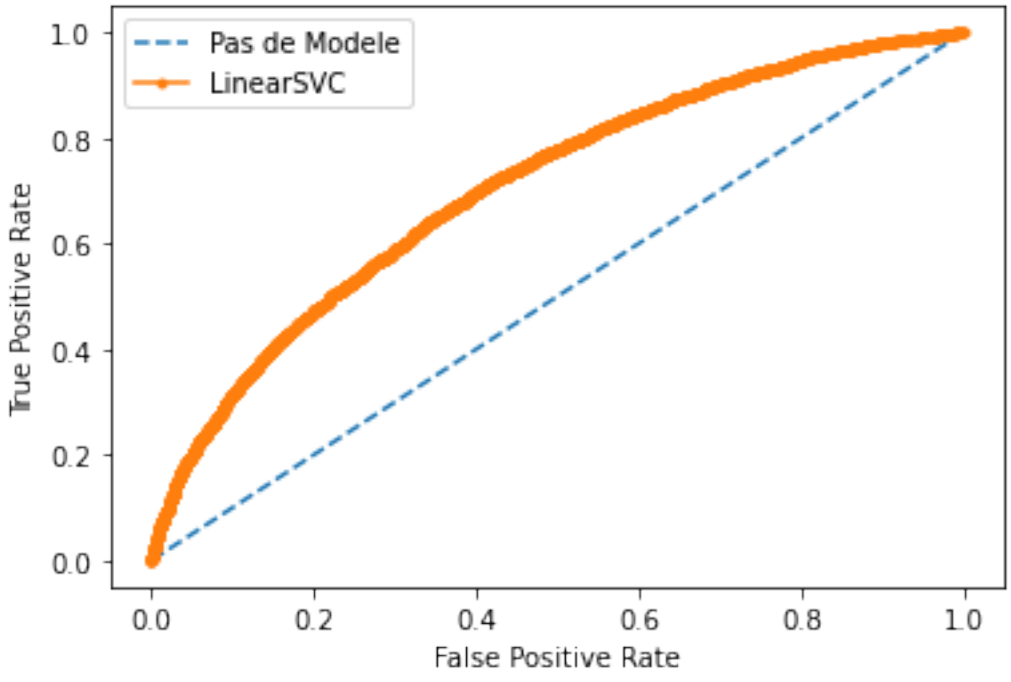
[0]:
```
claimsMergeAjout1 = merge_tokens(dfAjout1.copy())
print("addnone")
scoreAjout1 = do_classifier(dfOrigin, yColumnName, claimsMergeAjout1, False)

claimsMergeAjout2 = merge_tokens(dfAjout2.copy())
print("addauthor")
scoreAjout2 = do_classifier(dfOrigin, yColumnName, claimsMergeAjout2, False)
```

```
claimsMergeAjout3 = merge_tokens(dfAjout3.copy())
print("addall")
scoreAjout3 = do_classifier(dfOrigin, yColumnName, claimsMergeAjout3, False)
```

```
addnone
LinearSVC
Accuracy: 74.635%
F1-score moyen: 48.497%

Matrice de confusion:
 [[9880  150]
 [3286  230]]

              precision    recall  f1-score   support

          -1       0.75      0.99      0.85     10030
           1       0.61      0.07      0.12      3516

    accuracy                           0.75     13546
   macro avg       0.68      0.53      0.48     13546
weighted avg       0.71      0.75      0.66     13546

Sans modèle : ROC AUC =0.500
Avec LinearSVC  : ROC AUC =0.672
```

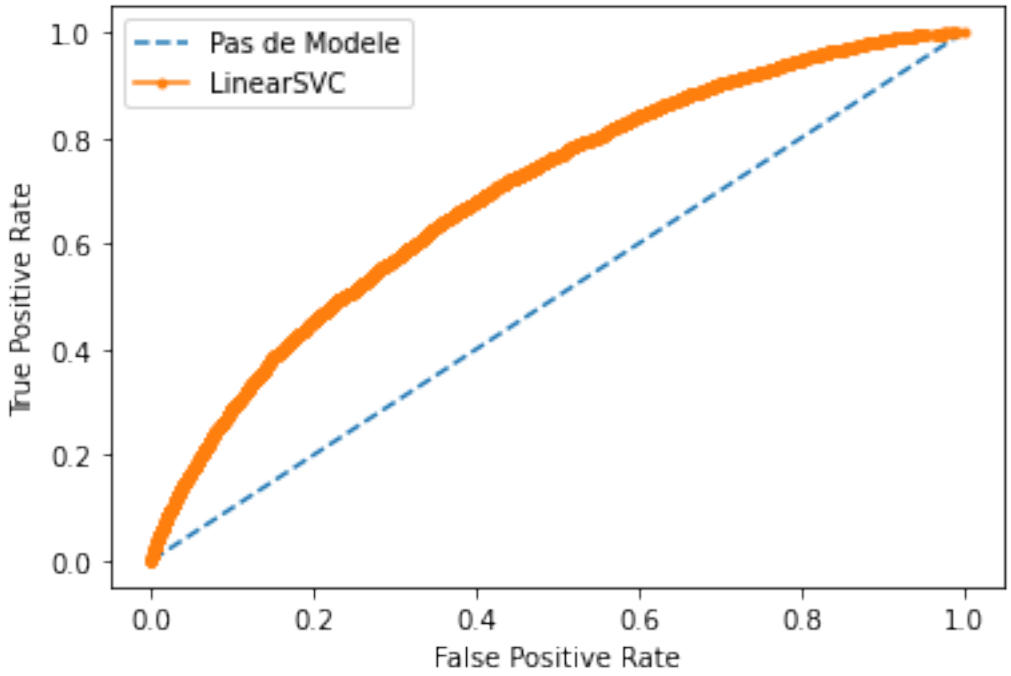


```
Réalisé en 5.4s
freememory=18.376 Go

addauthor
LinearSVC
Accuracy: 75.329%
F1-score moyen: 53.367%

Matrice de confusion:
 [[9750  280]
 [3062  454]]

              precision    recall  f1-score   support

          -1       0.76      0.97      0.85     10030
           1       0.62      0.13      0.21      3516

    accuracy                           0.75     13546
   macro avg       0.69      0.55      0.53     13546
weighted avg       0.72      0.75      0.69     13546

Sans modèle : ROC AUC =0.500
Avec LinearSVC  : ROC AUC =0.706
```




```
Réalisé en 6.4s
```

```
freememory=17.853 Go

addall
LinearSVC
Accuracy: 74.716%
F1-score moyen: 51.126%

Matrice de confusion:
 [[9766  264]
 [3161  355]]

              precision    recall  f1-score   support

          -1       0.76      0.97      0.85     10030
           1       0.57      0.10      0.17      3516

    accuracy                           0.75     13546
   macro avg       0.66      0.54      0.51     13546
weighted avg       0.71      0.75      0.67     13546

Sans modèle : ROC AUC =0.500
Avec LinearSVC  : ROC AUC =0.697
```




```
Réalisé en 8.9s
freememory=16.527 Go
```

[0]:
```
def NextDf(df0):

    dfN = df0.copy()

    dfN = dfN.drop('creativeWork_author_name', 1)
    dfN = dfN.drop('extra_author_categories', 1)
    dfN = dfN.drop('extra_claimReview_claimReviewed_entity', 1)
    dfN = dfN.drop('extra_claimReview_claimReviewed_categories', 1)
    dfN = dfN.drop('extra_keywords_entity', 1)
    dfN = dfN.drop('extra_keywords_categories', 1)
    dfN = dfN.drop('extra_tags', 1)
    dfN = dfN.drop('extra_title', 1)

    return dfN
```

[0]:
```
# Memorisation des parametres en fonction de l'évolution du score
if scoreAjout2>scoreAjout1:
    includeExtra = "addauthor"

if includeExtra == "addauthor":
    if scoreAjout3>scoreAjout2:
        includeExtra = "addall"
else:
    if scoreAjout3>scoreAjout1:
        includeExtra = "addall"

# Nouveau dataframe de travail
dfNext = NextDf(dfOrigin)

if includeExtra == "addnone":
    finalColumn = pd.DataFrame(claimsMergeAjout1)
    finalColumn.columns = [XColumnName]
    dfNext.update(finalColumn)

if includeExtra == "addauthor":
    finalColumn = pd.DataFrame(claimsMergeAjout2)
    finalColumn.columns = [XColumnName]
    dfNext.update(finalColumn)

if includeExtra == "addall":
    finalColumn = pd.DataFrame(claimsMergeAjout3)
    finalColumn.columns = [XColumnName]
    dfNext.update(finalColumn)
```

```
del dfOrigin, dfAjout1, dfAjout2, dfAjout3, claimsMergeAjout1,␣
 ↪claimsMergeAjout2, claimsMergeAjout3

print("Apres Add:")
display(dfNext.head())
print("dfNext taille:",dfNext.shape,'\n')

gc.collect()
print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
 ↪1024)))
```

```
Apres Add:
```

	claimReview_source_id	...	true_false_mixture
0	africacheck001	...	-1
1	africacheck002	...	-1
2	africacheck003	...	-1
3	africacheck004	...	-1
4	africacheck005	...	-1

```
[5 rows x 5 columns]


dfNext taille: (18061, 5)

freememory=16.643 Go
```

1.7 Etude sur le rééquilibrage du rating

[0]:
```
print("Avant resampling:")
print(dfNext[yColumnName].value_counts())
```

```
Avant resampling:
-1    13371
 1     4690
Name: true_false_mixture, dtype: int64
```

1.7.1 Creation des jeux de données resamplés

[0]:
```
def resampleDf(df, column, method):

    if method == "noresampling":
        dfResult = df.copy()

    else:
        count_big, count_small = df[column].value_counts()
        ordered_val = df[column].value_counts().index.tolist()
```

```
        dfBig = df[df[column] == ordered_val[0]]
        dfSmall = df[df[column] == ordered_val[1]]

        if method == "downsampling": # from dataframe
            dfBigDown = dfBig.sample(count_small, random_state=20)
            dfResult = pd.concat([dfBigDown, dfSmall])
            del dfBigDown

        elif method == "upsampling": # from dataframe
            dfSmallUp = dfSmall.sample(count_big, replace=True, random_state=20)
            dfResult = pd.concat([dfBig, dfSmallUp])
            del dfSmallUp

        elif method == "downresampl": # from scikitlearn
            dfBigDown = resample(dfBig, n_samples=count_small, random_state=20)
            dfResult = pd.concat([dfBigDown, dfSmall])
            del dfBigDown

        elif method == "upresampl": # from scikitlearn
            dfSmallUp = resample(dfSmall, replace=True, n_samples=count_big,␣
 ↪random_state=20)
            dfResult = pd.concat([dfBig, dfSmallUp])
            del dfSmallUp

        dfResult.to_csv('result.csv',sep='\t', index=False)
        dfResult=pd.read_csv('result.csv', sep='\t')
        del dfBig, dfSmall

    print("Apres",method,":")
    print(dfResult[column].value_counts())

    return dfResult
```

[0]:
```
methodResampl = "noresampling" # noresampling downsampling upsampling␣
 ↪downresampl upresampl
methodes=["noresampling", "downsampling", "upsampling", "downresampl",␣
 ↪"upresampl"]
dfResample = []

for i in range(len(methodes)):
    dfResample.append(resampleDf(dfNext, yColumnName, methodes[i]))
    print("dfResample",i,"taille:",dfResample[i].shape,"\n")
    print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
 ↪1024)),"\n")
```

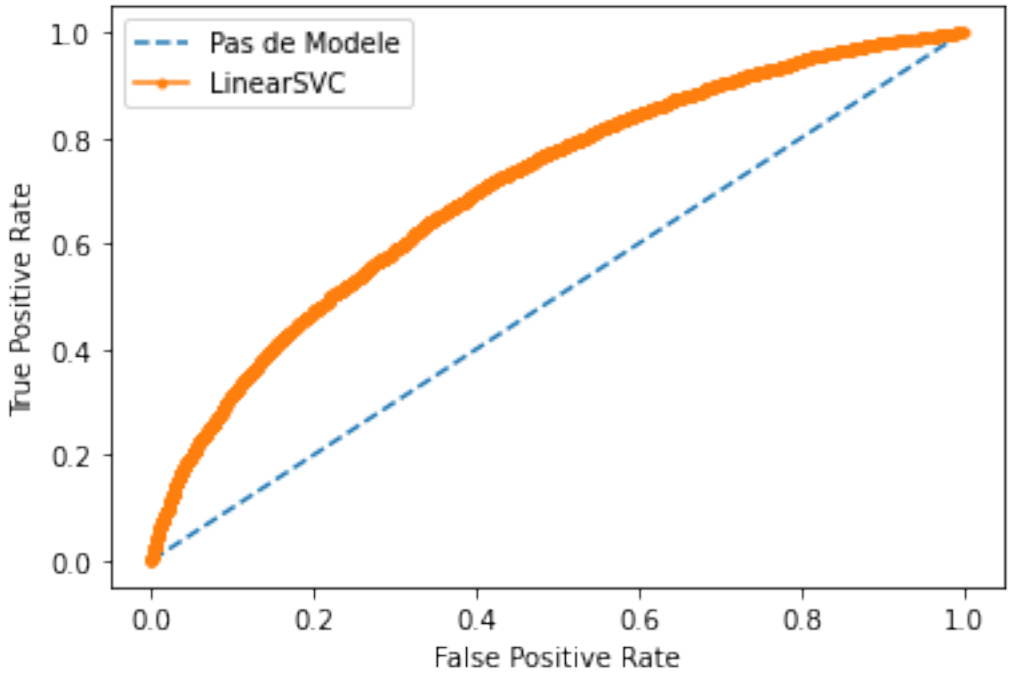
```
Apres noresampling :
-1    13371
```

```
 1     4690
Name: true_false_mixture, dtype: int64
dfResample 0 taille: (18061, 5)

freememory=16.643 Go

Apres downsampling :
-1    4690
 1    4690
Name: true_false_mixture, dtype: int64
dfResample 1 taille: (9380, 5)

freememory=16.639 Go

Apres upsampling :
-1    13371
 1    13371
Name: true_false_mixture, dtype: int64
dfResample 2 taille: (26742, 5)

freememory=16.623 Go

Apres downresampl :
-1    4690
 1    4690
Name: true_false_mixture, dtype: int64
dfResample 3 taille: (9380, 5)

freememory=16.626 Go

Apres upresampl :
-1    13371
 1    13371
Name: true_false_mixture, dtype: int64
dfResample 4 taille: (26742, 5)

freememory=16.614 Go
```

1.7.2 Evaluation des changements suite aux resamples

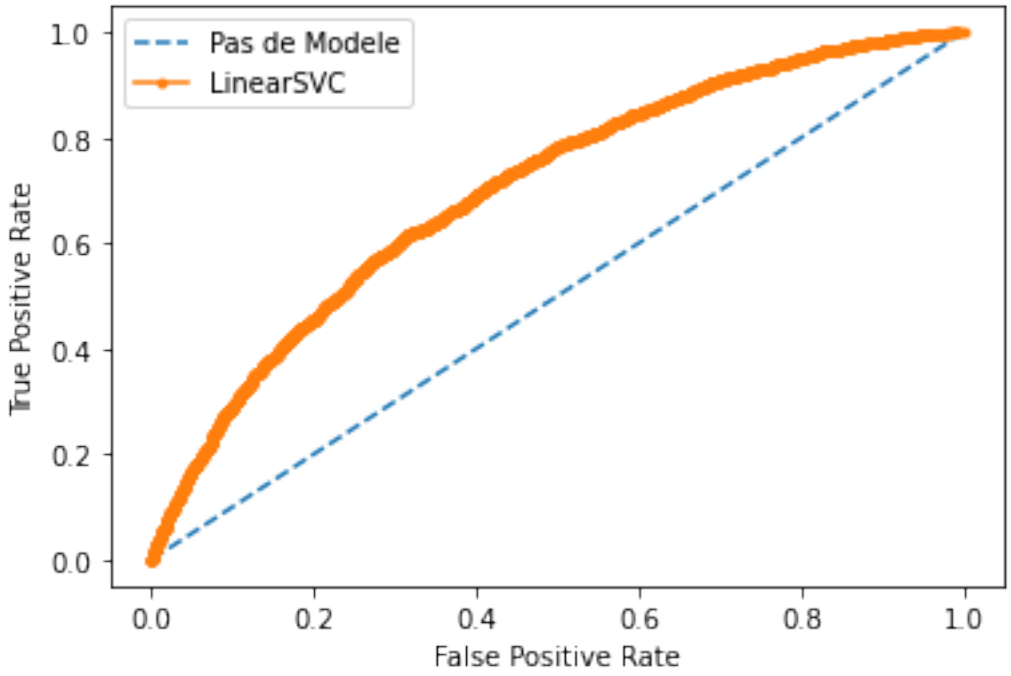
[0]:
```
scores = []

for i in range(len(methodes)):
    print(methodes[i])
    scores.append(do_classifier(dfResample[i], yColumnName,␣
 ↪dfResample[i][XColumnName].copy(), False))
```

```
noresampling
LinearSVC
Accuracy: 75.329%
F1-score moyen: 53.367%

Matrice de confusion:
 [[9750  280]
 [3062  454]]

              precision    recall  f1-score   support

          -1       0.76      0.97      0.85     10030
           1       0.62      0.13      0.21      3516

    accuracy                           0.75     13546
   macro avg       0.69      0.55      0.53     13546
weighted avg       0.72      0.75      0.69     13546

Sans modèle : ROC AUC =0.500
Avec LinearSVC  : ROC AUC =0.706
```



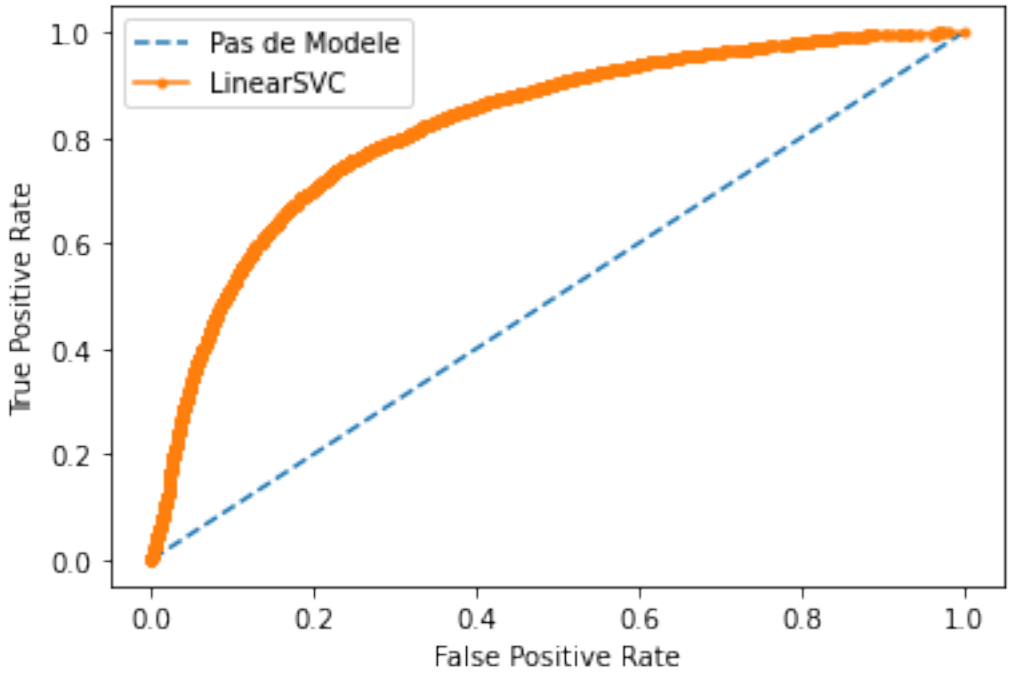

```
Réalisé en 5.9s
freememory=16.610 Go

downsampling
```

```
LinearSVC
Accuracy: 64.748%
F1-score moyen: 64.686%

Matrice de confusion:
 [[2424 1066]
 [1414 2131]]

              precision    recall  f1-score   support

          -1       0.63      0.69      0.66      3490
           1       0.67      0.60      0.63      3545

    accuracy                           0.65      7035
   macro avg       0.65      0.65      0.65      7035
weighted avg       0.65      0.65      0.65      7035

Sans modèle : ROC AUC =0.500
Avec LinearSVC  : ROC AUC =0.703
```



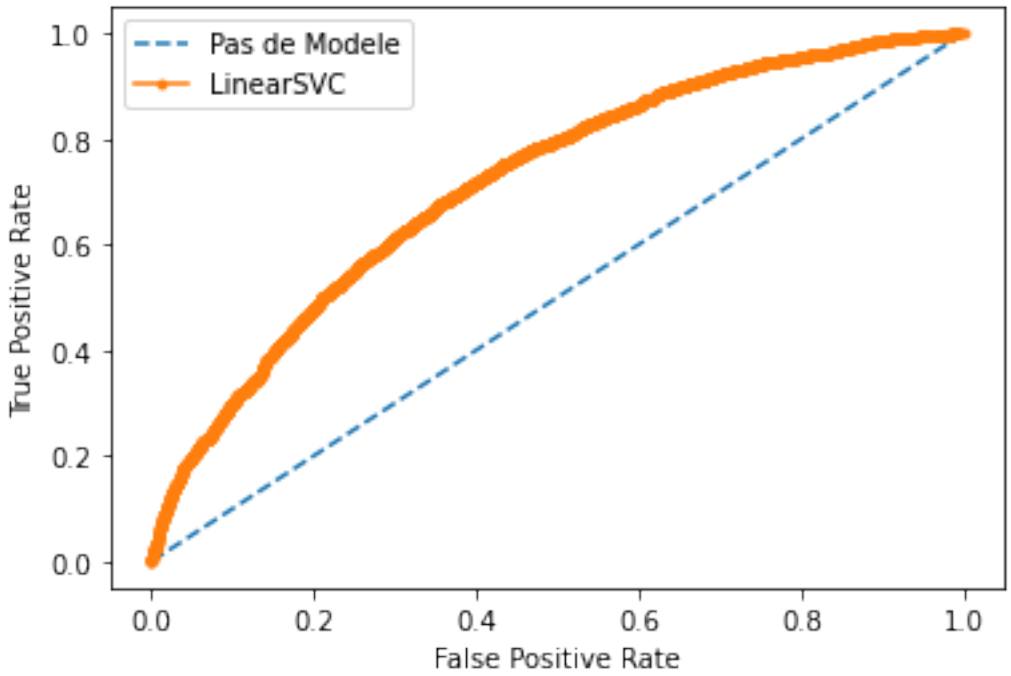

```
Réalisé en 3.0s
freememory=16.610 Go

upsampling
LinearSVC
```

```
Accuracy: 75.325%
F1-score moyen: 75.318%

Matrice de confusion:
 [[7382 2637]
 [2312 7726]]

              precision    recall  f1-score   support

          -1       0.76      0.74      0.75     10019
           1       0.75      0.77      0.76     10038

    accuracy                           0.75     20057
   macro avg       0.75      0.75      0.75     20057
weighted avg       0.75      0.75      0.75     20057

Sans modèle : ROC AUC =0.500
Avec LinearSVC  : ROC AUC =0.823
```



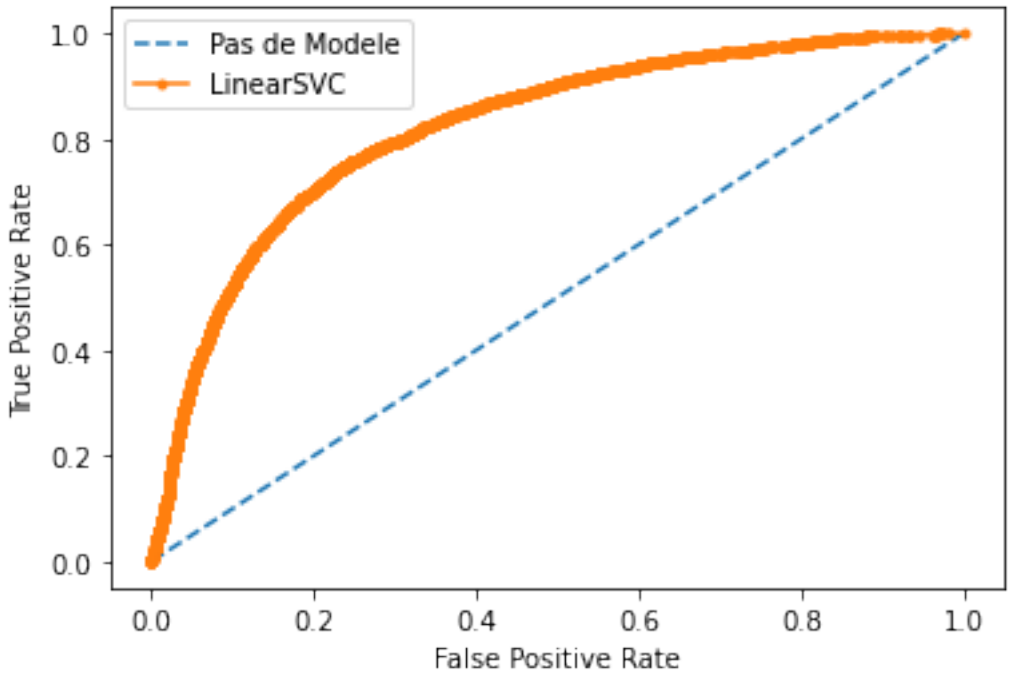

```
Réalisé en 9.8s
freememory=15.180 Go

downresampl
LinearSVC
Accuracy: 65.657%
```

```
F1-score moyen: 65.650%

Matrice de confusion:
 [[2361 1129]
 [1287 2258]]

              precision    recall  f1-score   support

          -1       0.65      0.68      0.66      3490
           1       0.67      0.64      0.65      3545

    accuracy                           0.66      7035
   macro avg       0.66      0.66      0.66      7035
weighted avg       0.66      0.66      0.66      7035

Sans modèle : ROC AUC =0.500
Avec LinearSVC  : ROC AUC =0.718
```




```
Réalisé en 3.1s
freememory=15.180 Go

upresampl
LinearSVC
Accuracy: 75.325%
F1-score moyen: 75.318%
```

```
Matrice de confusion:
 [[7382 2637]
 [2312 7726]]

              precision    recall  f1-score   support

          -1       0.76      0.74      0.75     10019
           1       0.75      0.77      0.76     10038

    accuracy                           0.75     20057
   macro avg       0.75      0.75      0.75     20057
weighted avg       0.75      0.75      0.75     20057

Sans modèle : ROC AUC =0.500
Avec LinearSVC  : ROC AUC =0.823
```




```
Réalisé en 9.4s
freememory=15.180 Go
```

[0]:
```
bestmean = 0
index = 0
for i in range(len(scores)):
    if scores[i] >= bestmean:
```

```
        methodResampl = methodes[i]
        bestmean = scores[i]
        index = i

dfFinal = dfResample[index].copy()

del dfNext, dfResample

print("Apres Resample:")
display(dfFinal.head())
print("dfFinal taille:",dfFinal.shape,'\n')

gc.collect()
print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
↪1024)))
```

```
Apres Resample:
```

	claimReview_source_id	...	true_false_mixture
0	africacheck001	...	-1
1	africacheck002	...	-1
2	africacheck003	...	-1
3	africacheck004	...	-1
4	africacheck005	...	-1

```
[5 rows x 5 columns]


dfFinal taille: (26742, 5)

freememory=15.182 Go
```

1.8 Recherche des meilleurs paramètres des classifieurs

[0]:
```
def log_results(fileName, stop, lemm, addExtra, reSample, clf, param, score, t):
    fichier = open("log.txt", "a")
    fichier.write("\nJeu de données: %s\n" % (fileName))
    if stop: s="True"
    else: s="False"
    fichier.write("Stop words: %s\n" % (s))
    if lemm: l="True"
    else: l="False"
    fichier.write("Lemmatisation: %s\n" % (l))
    fichier.write("Ajout d'extras: %s\n" % (addExtra))
    fichier.write("Méthode resample: %s\n" % (reSample))
    fichier.write("Classifieur: %s\n" % (clf))
    fichier.write("Paramètres: %s\n" % (param))
    fichier.write("Score: %.3f%%\n" % (score * 100.0))
```

```
    fichier.write("Réalisé en %.1fs\n" % (t))
    fichier.close()
```

[0]:
```
def do_gridsearch(name, estimClf, gridParam, X, y):

    print(name,": wait...")
    t0 = time()
    scoring = 'accuracy'

    gd_sr = GridSearchCV(estimator=estimClf,
                         param_grid=gridParam,
                         scoring=scoring,
                         cv=5,
                         #n_jobs=-1,
                         iid=True,
                         return_train_score=True)

    gd_sr.fit(X, y)

    log_results(dataFile, StopWords, Lemmatiz, includeExtra, methodResampl,
                 name, gd_sr.best_params_, gd_sr.best_score_,  time() - t0)

    print ("Meilleurs paramètres: %s" % (gd_sr.best_params_))
    print ("Meilleur score: %.3f%%" % (gd_sr.best_score_ * 100.0))
    print ("Réalisé en %.1fs\n" % (time() - t0))
    print ("Meilleur estimateur",gd_sr.best_estimator_,'\n')

    gc.collect()
    print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
 ↪1024)),"\n")

    return gd_sr.best_params_
```

[0]:
```
jobsTodo= [
    {
        'clf' : 'GaussianNB',
        'clfAbrev' : 'GNB',
        'estimClf' : GaussianNB(),
        'grid_param' : {}
    },
    {
        'clf' : 'MultinomialNB',
        'clfAbrev' : 'MNB',
        'estimClf' : MultinomialNB(),
        'grid_param' : {}
    },
#    { # trop long
```

```
#         'clf' : 'KNeighborsClassifier',
#         'clfAbrev' : 'KNC',
#         'estimClf' : KNeighborsClassifier(),
#         'grid_param' : {
#             'n_neighbors': list(range(8,16)),
#             'weights': ['uniform','distance'],
#             'metric': ['minkowski','euclidean','manhattan']
#         }
#     },
#     { # trop long
#         'clf' : 'SVC',
#         'clfAbrev' : 'SVC',
#         'estimClf' : SVC(),
#         'grid_param' : {
#             'C': [0.01, 0.1, 0.5, 1, 2, 10],
#             'gamma' : [0.001, 0.01, 0.1],
#             'kernel': ['linear','rbf']
#         }
#     },
    {
        'clf' : 'LinearSVC',
        'clfAbrev' : 'LSVC',
        'estimClf' : LinearSVC(),
        'grid_param' : {
            'C': [0.01, 0.1, 0.4, 0.5, 0.6, 1, 9, 10, 11, 100],
            'max_iter': [1000]
        }
    },
    {
        'clf' : 'LogisticRegression',
        'clfAbrev' : 'LR',
        'estimClf' : LogisticRegression(),
        'grid_param' : {
            'C' : [2,3,4,5,6,7,8],
            'max_iter': [500, 1000]
        },
    },
    {
        'clf' : 'DecisionTreeClassifier',
        'clfAbrev' : 'DTC',
        'estimClf' : DecisionTreeClassifier(),
        'grid_param' : {
            'max_depth': [7,8,9,10,11,12],
            'criterion': ['gini', 'entropy'],
            'min_samples_leaf': [2,3,4,5,6,7,8]
        }
    },
```

```
    {
        'clf' : 'RandomForestClassifier',
        'clfAbrev' : 'RFC',
        'estimClf' : RandomForestClassifier(),
        'grid_param' : {
            'criterion': ['entropy', 'gini'],
            'max_depth': [6, 9, 12],
            'max_features': ['log2', 'sqrt','auto'],
            'min_samples_leaf': [1, 5, 8],
            'min_samples_split': [2, 3, 5],
            'n_estimators': [6, 9, 12]
        }
    },
    {
        'clf' : 'SGDClassifier',
        'clfAbrev' : 'SGDC',
        'estimClf' : SGDClassifier(),
        'grid_param' : {
             'loss': ['log','hinge'],
             'penalty': ['l1','l2']
        }
    }
]
```

[0]:
```
print("Preparation aux classifieurs...")

vectorizerT = TfidfVectorizer(min_df=2)
vectorT = vectorizerT.fit_transform(dfFinal[XColumnName].copy())

X=vectorT.toarray()
y=dfFinal[yColumnName].copy()

validation_size=0.25
testsize= 1-validation_size
X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               train_size=validation_size,
                                               random_state=20,
                                               test_size=testsize)

gc.collect()
print("freememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
 ↪1024)),"\n")

models = []
for j in jobsTodo:
```

```
    params = do_gridsearch(j['clf'], j['estimClf'], j['grid_param'], X_train,␣
 ↪y_train)
    models.append((j['clf'], j['clfAbrev'], j['estimClf'], params))

del vectorT, X_train, X_test, y_train, y_test
```

```
Preparation aux classifieurs…
freememory=15.182 Go

GaussianNB : wait…
Meilleurs paramètres: {}
Meilleur score: 67.868%
Réalisé en 12.1s

Meilleur estimateur GaussianNB(priors=None, var_smoothing=1e-09)

freememory=13.807 Go

MultinomialNB : wait…
Meilleurs paramètres: {}
Meilleur score: 72.461%
Réalisé en 3.0s

Meilleur estimateur MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

freememory=13.799 Go

LinearSVC : wait…
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
```

```
  "the number of iterations.", ConvergenceWarning)
Meilleurs paramètres: {'C': 0.5, 'max_iter': 1000}
Meilleur score: 74.974%
Réalisé en 43.3s

Meilleur estimateur LinearSVC(C=0.5, class_weight=None, dual=True,
fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)

freememory=13.799 Go

LogisticRegression : wait…
Meilleurs paramètres: {'C': 8, 'max_iter': 500}
Meilleur score: 75.079%
Réalisé en 521.7s

Meilleur estimateur LogisticRegression(C=8, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=500,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)

freememory=13.799 Go

DecisionTreeClassifier : wait…
Meilleurs paramètres: {'criterion': 'gini', 'max_depth': 12, 'min_samples_leaf':
2}
Meilleur score: 62.408%
Réalisé en 2605.7s

Meilleur estimateur DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
criterion='gini',
                       max_depth=12, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')

freememory=13.803 Go

RandomForestClassifier : wait…
Meilleurs paramètres: {'criterion': 'entropy', 'max_depth': 12, 'max_features':
'auto', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 12}
Meilleur score: 63.635%
```

```
Réalisé en 2150.4s

Meilleur estimateur RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
class_weight=None,
                       criterion='entropy', max_depth=12, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=12,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)

freememory=13.804 Go

SGDClassifier : wait…
Meilleurs paramètres: {'loss': 'hinge', 'penalty': 'l2'}
Meilleur score: 74.106%
Réalisé en 290.4s

Meilleur estimateur SGDClassifier(alpha=0.0001, average=False,
class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)

freememory=13.807 Go
```

1.9 Recherche du meilleur classifieur paramétré

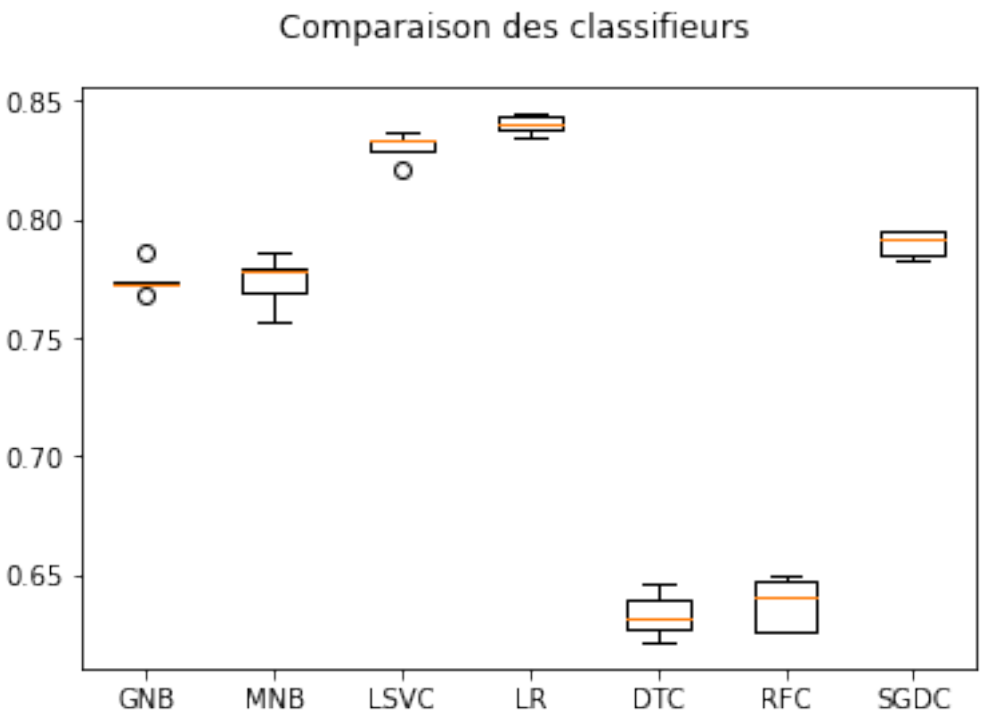
[0]:
```python
results = []
abrevs = []
scores = []
scoring = 'accuracy'
print("wait...\n")
for name,abrev,model,param in models:
    model.set_params(**param)
    kfold = KFold(n_splits=5, shuffle=True, random_state=3)
    cv_results = cross_val_score(model, X, y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    abrevs.append(abrev)
    scores.append((name,model,param,cv_results.mean()))
    msg = "%s: %0.3%%f (%.3f)" %(name, cv_results.mean()*100, cv_results.std())
    print(msg)
gc.collect()
```

```
print("\nfreememory=%2.3f Go" %(psutil.virtual_memory().free/(1024 * 1024 *␣
↪1024)))
```

```
wait…

GaussianNB: %77.488627 (0.006)
MultinomialNB: %77.395073 (0.010)
LinearSVC: %83.101483 (0.006)
LogisticRegression: %84.013906 (0.004)
DecisionTreeClassifier: %63.316127 (0.009)
RandomForestClassifier: %63.787304 (0.010)
SGDClassifier: %78.988118 (0.005)

freememory=11.358 Go
```

[0]:
```
fig = plt.figure()
fig.suptitle('Comparaison des classifieurs')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(abrevs)

clfName = ""
bestParam = ""
bestmean = 0
for name,model,param,mean in scores:
    if mean > bestmean:
        clfName = name
        clfMethod = model
        bestParam = param
        bestmean = mean

msg = "Meilleur résultat: %s(%s) score:%.3f%%" %(clfName, bestParam,␣
↪bestmean*100)
print(msg)
```

```
Meilleur résultat: LogisticRegression({'C': 8, 'max_iter': 500}) score:84.014%
```



1.10 Evaluation finale avec l'ensemble des meilleurs choix retenus

[0]:
```
def perform_final():
    t0 = time()
    dfOrigin=pd.read_csv(dataFile, sep='\t')

    # ! necessaire pour mixture, a mettre en commentaire pour truefalse !
    #dfOrigin.loc[dfOrigin[yColumnName] == -1, yColumnName] = 1

    dfAjout = addExtras(dfOrigin, XColumnName, includeExtra, StopWords,␣
↪Lemmatiz)
    claimsMerge = merge_tokens(dfAjout.copy())
    dfNext = NextDf(dfOrigin)
    finalColumn = pd.DataFrame(claimsMerge)
    finalColumn.columns = [XColumnName]
    dfNext.update(finalColumn)
    dfFinal = resampleDf(dfNext, yColumnName, methodResampl)
    vectorizerT = TfidfVectorizer(min_df=2)
    vectorT = vectorizerT.fit_transform(dfFinal[XColumnName].copy())
    X=vectorT.toarray()
    y=dfFinal[yColumnName].copy()
    kfold = KFold(n_splits=5, shuffle=True, random_state=3)
    clfMethod.set_params(**bestParam)
```

```
    scoring = 'accuracy'
    cv_results = cross_val_score(clfMethod, X, y, cv=kfold, scoring=scoring)
    log_results(dataFile, StopWords, Lemmatiz, includeExtra, methodResampl,
                  clfName, bestParam, cv_results.mean(),  time() - t0)
    msg = "Score= %.3f%% (%.3f)" % (cv_results.mean()*100, cv_results.std())
    return msg
```

[0]:
```
print("dataFile= '%s'" % (dataFile)) # data-truefalse-1.csv data-truefalse-2.
 ↪csv data-truefalse-3.csv data-truefalse.csv
print("XColumnName= '%s'" % (XColumnName))
print("yColumnName= '%s'" % (yColumnName))
print("StopWords=",StopWords) # False True
print("Lemmatiz=",Lemmatiz)  # False True
print("includeExtra= '%s'" % (includeExtra)) # addnone addauthor addall
print("methodResampl= '%s'" % (methodResampl)) # noresampling downsampling␣
 ↪upsampling downresampl upresampl
print("clfName= '%s'" % (clfName))
print("clfMethod= ",clfMethod)
print("bestParam= ",bestParam,"\n")
resulFinal = perform_final()
print("\n",resulFinal)
```

```
dataFile= 'data-truefalse.csv'
XColumnName= 'claimReview_claimReviewed'
yColumnName= 'true_false_mixture'
StopWords= False
Lemmatiz= False
includeExtra= 'addauthor'
methodResampl= 'upresampl'
clfName= 'LogisticRegression'
clfMethod=  LogisticRegression(C=8, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=500,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
bestParam=  {'C': 8, 'max_iter': 500}

Wait…
Clean author extras
Extras cleaned
Add author extras
Extras added
addauthor done
Apres upresampl :
-1    13371
 1    13371
```

```
Name: true_false_mixture, dtype: int64

 Score= 84.014% (0.004)
```

1.10.1 Save des datas

[0]:
```
dfFinal.to_csv('result.csv',sep='\t', index=False)
print("Save to CSV")
```

```
Save to CSV
```

1.11 Fin

[0]:
```
assert False, "breakpoint"
```

```
      ␣
↪---------------------------------------------------------------------------

       AssertionError                            Traceback (most recent call␣
↪last)

       <ipython-input-177-cea0b2969840> in <module>()
   ----> 1 assert False, "breakpoint"


       AssertionError: breakpoint
```

Sauvegarde des resultats

[0]:
```
# avec LinearSVC comme classifieur par defaut
# F1 score en critere de recherche et accuracy pour conclure
dataFile= 'data-truefalse.csv'
XColumnName= 'claimReview_claimReviewed'
yColumnName= 'true_false_mixture'
StopWords= False
Lemmatiz= False
includeExtra= 'addauthor'
methodResampl= 'upresampl'
clfName= 'LogisticRegression'
clfMethod= LogisticRegression()
bestParam= {'C': 8, 'max_iter': 500}
print(perform_final()) # pour relancer
#Score= 84.014% (0.004)
```

```
Wait…
Clean author extras
Extras cleaned
```

```
Add author extras
Extras added
addauthor done
Apres upresampl :
-1    13371
 1    13371
Name: true_false_mixture, dtype: int64
Score= 84.014% (0.004)
```

[0]:
```
dataFile= 'data-mixture.csv'
XColumnName= 'claimReview_claimReviewed'
yColumnName= 'true_false_mixture'
StopWords= False
Lemmatiz= False
includeExtra= 'addauthor'
methodResampl= 'upresampl'
clfName= 'LogisticRegression'
clfMethod= LogisticRegression()
bestParam=  {'C': 8, 'max_iter': 500}
print(perform_final()) # pour relancer
#Score= 78.791% (0.003)
```