Classification d'assertions selons leur valeurs de véracité

Lien avec GoogleDrive (Facultatif)

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

```
my_local_drive = '/content/gdrive/My Drive/Colab Notebooks/Projet_Machine_Learning/
sys.path.append(my_local_drive)
%cd $my_local_drive
%ls -l
```

Lien du fichier de données

```
path = 'TRUE vs FALSE/output_200/output_247T-266F.csv'
# path = 'TRUE_FALSE vs MIXTURE/output_400-400-800.csv'
# path = 'TRUE vs FALSE vs MIXTURE/output_500-500-500.csv'

# Dans le cas de TRUE&FALSE vs MIXTURE : Mettre ce booleen a True, sinon False
joinTrueAndFalse = False
```

Importation des libraires utiles pour le notebook

```
# Importation des différentes librairies utiles pour le notebook

!pip install langdetect
!pip install contractions

#Sickit learn met régulièrement à jour des versions et
#indique des futurs warnings.
#ces deux lignes permettent de ne pas les afficher.
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

import pandas as pd
import seaborn as sns
import seaborn as sb
import matplotlib.pyplot as plt
import sys
import numpy as np
import sklearn
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_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.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn import preprocessing
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
from sklearn.pipeline import Pipeline

#Sickit learn met régulièrement à jour des versions et indique des futurs warnings.
#ces deux lignes permettent de ne pas les afficher.
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

```
Requirement already satisfied: langdetect in d:\logiciels\python\lib\site-pack
Requirement already satisfied: six in d:\logiciels\python\lib\site-packages (f
WARNING: You are using pip version 21.0.1; however, version 22.1 is available.
You should consider upgrading via the 'd:\logiciels\python\python.exe -m pip i
WARNING: You are using pip version 21.0.1; however, version 22.1 is available.
You should consider upgrading via the 'd:\logiciels\python\python.exe -m pip i
Requirement already satisfied: contractions in d:\logiciels\python\lib\site-pa
Requirement already satisfied: textsearch>=0.0.21 in d:\logiciels\python\lib\s
Requirement already satisfied: anyascii in d:\logiciels\python\lib\site-packag
Requirement already satisfied: pyahocorasick in d:\logiciels\python\lib\site-p
```

Lecture des données

```
from MyNLPUtilities import *

df = pd.read_csv(path, sep=',', encoding="utf-8")

if joinTrueAndFalse:
    df['ratingName'].replace("TRUE", "TRUEorFALSE", inplace=True)
    df['ratingName'].replace("FALSE", "TRUEorFALSE", inplace=True)
```

```
display(df.head())
```

	id	text	date	truthRating	ratingNam
0	http://data.gesis.org/claimskg/claim_review/0f...	'All non-US citizens, illegal or not, will be ...	2009-07-28	1	Fals
1	http://data.gesis.org/claimskg/claim_review/4d...	'There's no rationing in any of these bills.'	2009-08-09	1	Fals
2	http://data.gesis.org/claimskg/claim_review/3f...	Says that under his tax plan, seniors making l...	2008-09-30	3	Tru
3	http://data.gesis.org/claimskg/claim_review/fa...	'I have never said that I don't wear flag pins...	2008-04-16	1	Fals
4	http://data.gesis.org/claimskg/claim_review/9a...	'Two-thirds of our economy is a consumer econo...	2008-01-24	3	Tru

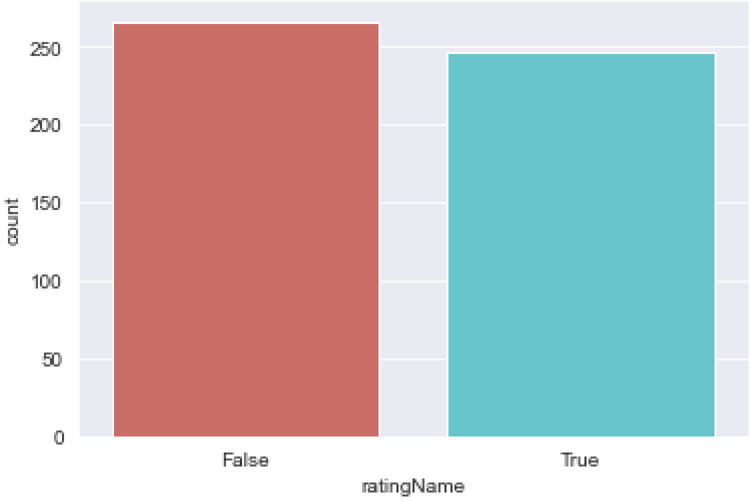
Analyse du jeu de données

```
import plotly.express as px
from sklearn.decomposition import PCA

print ("Nombre d'occurrences par classe : \n", df['ratingName'].value_counts())
print ("Shape: " + str(df.shape))
```

```
Nombre d'occurrences par classe :
 False    266
True     247
Name: ratingName, dtype: int64
Shape: (513, 14)
```

```
def create_distribution(dataFile):
    return sb.countplot(x='ratingName', data=dataFile, palette='hls')
create_distribution(df)
```

```
<AxesSubplot:xlabel='ratingName', ylabel='count'>
```



```
fig=px.scatter(df, x = 'ratingName',  y="date")
fig.show()

fig=px.scatter(df, x = 'ratingName',  y="keywords")
fig.show()
```

Formatage du jeu de données

```
#Définition variables apprentissage et de prédiction

# Extraction des variables
Xsource = df.source
Xtext = df.text
Xheadline = df.headline
Xauthor = df.author
Xentities = df.named_entities_claim
Xkeywords = df.keywords

#Concaténation du texte à traiter
X = Xtext + Xheadline + Xsource + Xauthor + Xentities + Xkeywords

#Variable de prédiction
y = df['ratingName']
```

Normalisation du datatext

```
#Définition de la fonction MyCleanText pour le pré-traitement du texte
import re
import string
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk import word_tokenize
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')
stop_words = set(stopwords.words('english'))

def MyCleanText(X,
 lowercase=True, # mettre en minuscule
 removestopwords=False, # supprimer les stopwords
 removedigit=False, # supprimer les nombres
 getstemmer=False, # conserver la racine des termes
 getlemmatisation=False # lematisation des termes
 ):

 sentence=str(X)
 # suppression des caractères spéciaux
 sentence = re.sub(r'[^\w\s]',' ', sentence)
 # suppression de tous les caractères uniques
 sentence = re.sub(r'\s+[a-zA-Z]\s+', ' ', sentence)
 # substitution des espaces multiples par un seul espace
 sentence = re.sub(r'\s+', ' ', sentence, flags=re.I)
 # découpage en mots
 tokens = word_tokenize(sentence)
 if lowercase:
  tokens = [token.lower() for token in tokens]

 # suppression ponctuation
 table = str.maketrans('', '', string.punctuation)
 words = [token.translate(table) for token in tokens]
 # suppression des tokens non alphabetique ou numerique
 words = [word for word in words if word.isalnum()]

 # suppression des tokens numerique
 if removedigit:
  words = [word for word in words if not word.isdigit()]

 # suppression des stopwords
 if removestopwords:
  words = [word for word in words if not word in stop_words]

 # lemmatisation
 if getlemmatisation:
  lemmatizer=WordNetLemmatizer()
  words = [lemmatizer.lemmatize(word)for word in words]

 # racinisation
 if getstemmer:
  ps = PorterStemmer()
  words=[ps.stem(word) for word in words]

 sentence= ' '.join(words)
 return sentence


#Définir la classe TextNormalizer qui effectue les prétraitements sur les données
from sklearn.base import BaseEstimator, TransformerMixin
class TextNormalizer(BaseEstimator, TransformerMixin):
 def __init__(self,
 removestopwords=False, # suppression des stopwords
 lowercase=False,# passage en minuscule
 removedigit=False, # supprimer les nombres
 getstemmer=False,# racinisation des termes
 getlemmatisation=False # lemmatisation des termes
 ):
   self.lowercase=lowercase
   self.getstemmer=getstemmer
   self.removestopwords=removestopwords
   self.getlemmatisation=getlemmatisation
   self.removedigit=removedigit

 # Nettoyage du texte
 def transform(self, X, **transform_params):
  X=X.copy() # pour conserver le fichier d'origine
  return [MyCleanText(text,lowercase=self.lowercase, getstemmer=self.getstemmer, re

 def fit(self, X, y=None, **fit_params):
   return self

 def fit_transform(self, X, y=None, **fit_params):
   return self.fit(X).transform(X)

 def get_params(self, deep=True):
   return { 'lowercase':self.lowercase, 'getstemmer':self.getstemmer, 'removestopwo

 def set_params (self, **parameters):
   for parameter, value in parameters.items():
    setattr(self,parameter,value)
   return self




# création d'un objet de la classe TextNormalizer
text_normalizer = TextNormalizer(removestopwords=True,
                                lowercase=True,
                                removedigit=False,
                                getstemmer=True,
                                getlemmatisation=False)
```

```
# application du fit.transform pour appliquer les pré traitements
X_cleaned = text_normalizer.fit_transform(X)
```

```
    [nltk_data] Downloading package wordnet to
    [nltk_data]     C:\Users\Martin\AppData\Roaming\nltk_data...
    [nltk_data]   Package wordnet is already up-to-date!
    [nltk_data] Downloading package stopwords to
    [nltk_data]     C:\Users\Martin\AppData\Roaming\nltk_data...
    [nltk_data]   Package stopwords is already up-to-date!
    [nltk_data] Downloading package punkt to
    [nltk_data]     C:\Users\Martin\AppData\Roaming\nltk_data...
    [nltk_data]   Package punkt is already up-to-date!
```

Création d'un jeu d'apprentissage et de tests

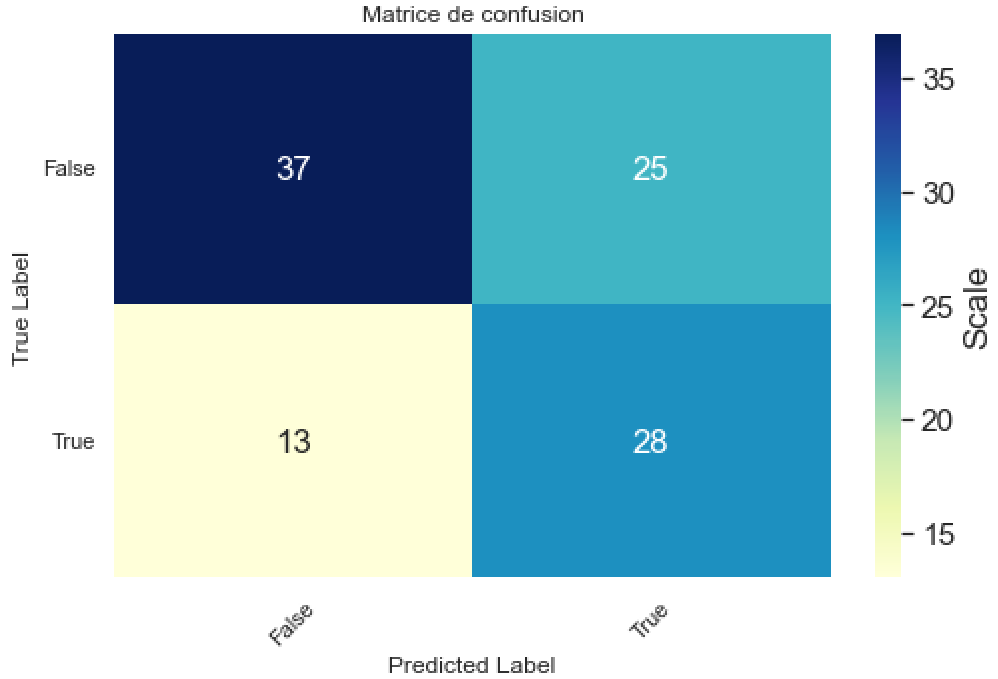
```
# Transformation du texte en données utilisables par les classifieurs
tf = TfidfVectorizer()
X_transformed = tf.fit_transform(X_cleaned).toarray()

# Séparation du jeu de données
trainsize = 0.8
testsize = 0.2
seed = 30

X_train,X_test,y_train,y_test=train_test_split(X_transformed, y, train_size=trainsi
```

Essai sur les différents classifieurs

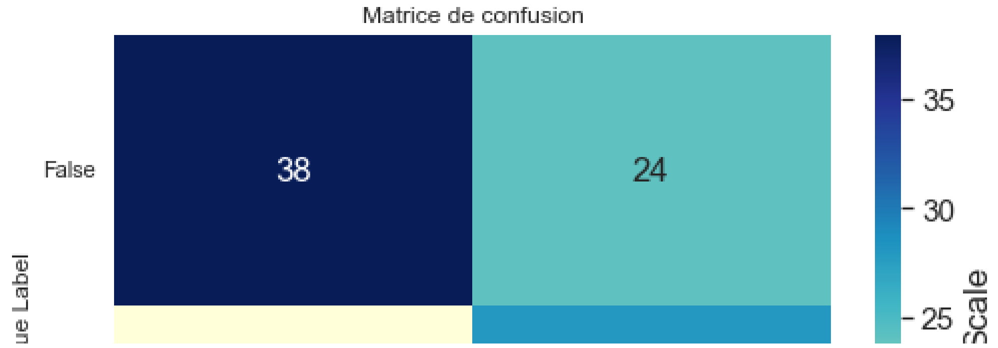
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn import model_selection

models = []
models.append(('MultinomialNB',MultinomialNB()))
models.append(('LR', LogisticRegression(solver='lbfgs')))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('RF', RandomForestClassifier()))
models.append(('SVM', SVC()))


seed = 7
results = []
names = []
scoring='accuracy'
```

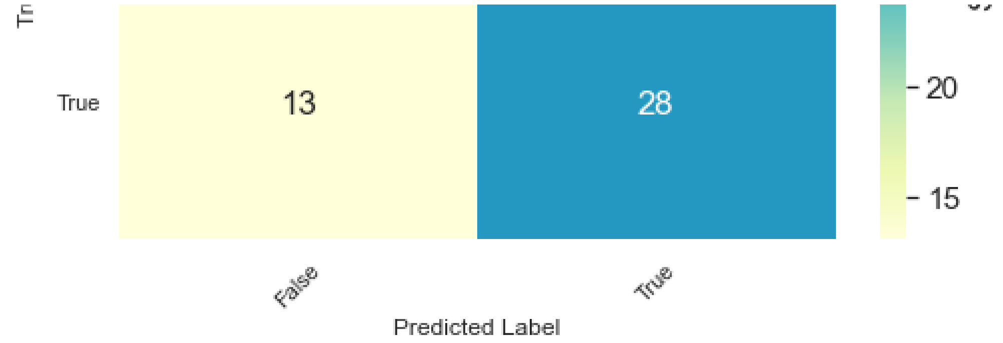
```
for name,model in models:
    print("*--*--*--*--*--*--*--*--*--* "+len(name)*" " +" *--*--*--*--*--*--*--
    print("*--*--*--*--*--*--*--*--*--* "+name+" *--*--*--*--*--*--*--*--*--*
    print("*--*--*--*--*--*--*--*--*--* "+len(name)*" " +" *--*--*--*--*--*--*--
    clf = model
    clf.fit(X_train, y_train)
    result = clf.predict(X_test)

    y_pred = clf.predict(X_test)
    MyshowAllScores(y_test,y_pred)
```

```
*--*--*--*--*--*--*--*--*--*--*               *--*--*--*--*--*--*--*--*--*--
*--*--*--*--*--*--*--*--*--*--* MultinomialNB *--*--*--*--*--*--*--*--*--*--
*--*--*--*--*--*--*--*--*--*--*               *--*--*--*--*--*--*--*--*--*--
Accuracy : 0.631
Classification Report
              precision    recall  f1-score   support

       False    0.74000   0.59677   0.66071        62
        True    0.52830   0.68293   0.59574        41

    accuracy                        0.63107       103
   macro avg    0.63415   0.63985   0.62823       103
weighted avg    0.65573   0.63107   0.63485       103
```

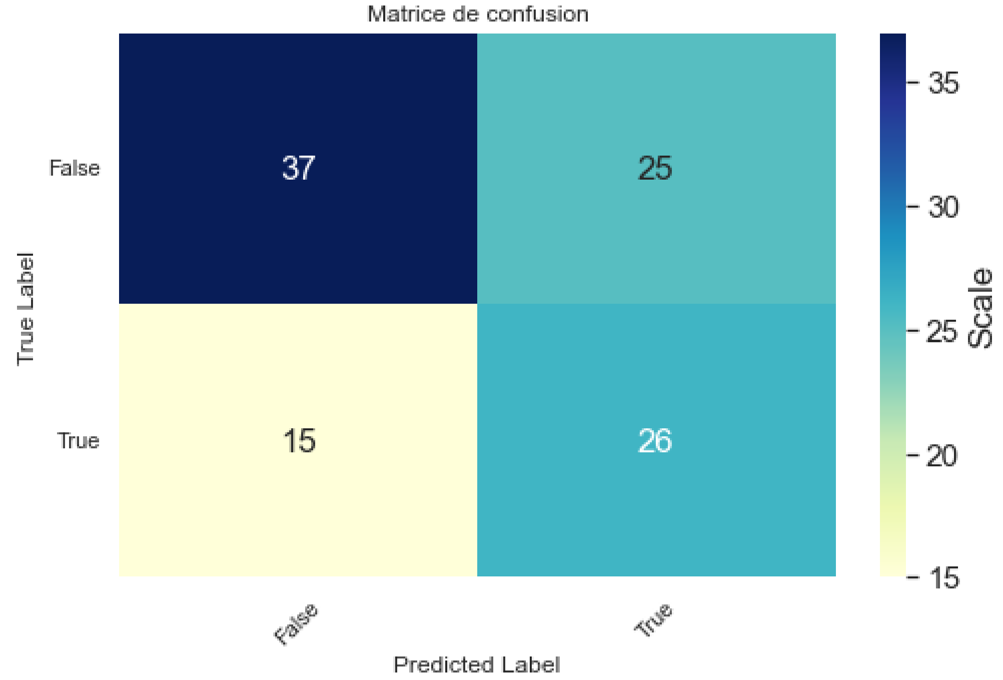


```
*--*--*--*--*--*--*--*--*--*--*    *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--* LR *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--*    *--*--*--*--*--*--*--*--*--*--*
Accuracy : 0.641
Classification Report
              precision    recall  f1-score   support

       False    0.74510   0.61290   0.67257        62
        True    0.53846   0.68293   0.60215        41

    accuracy                        0.64078       103
   macro avg    0.64178   0.64792   0.63736       103
weighted avg    0.66284   0.64078   0.64454       103
```

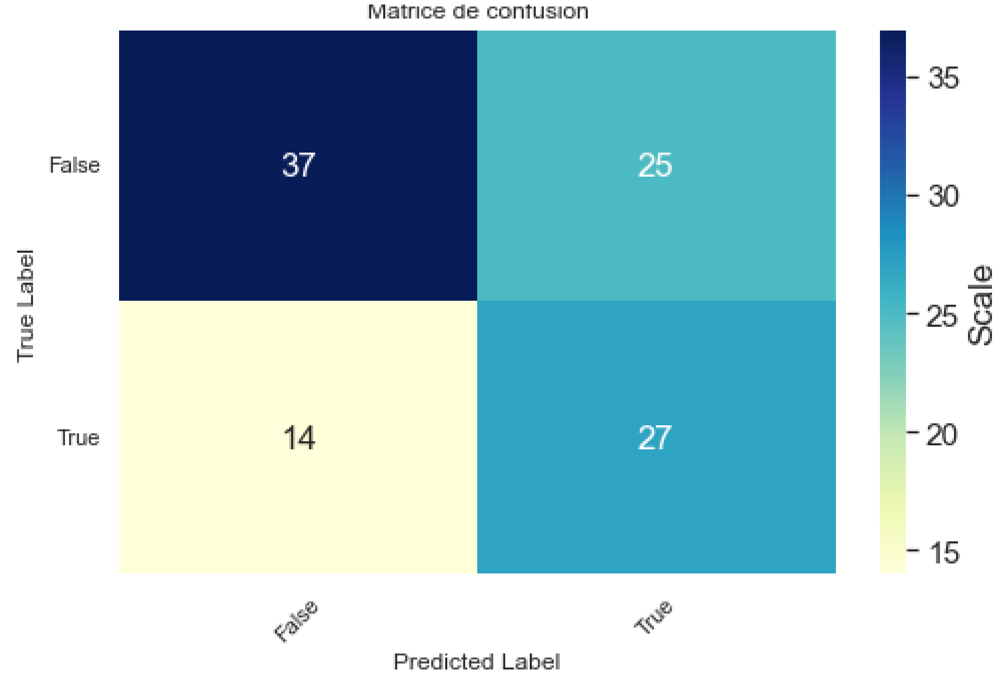




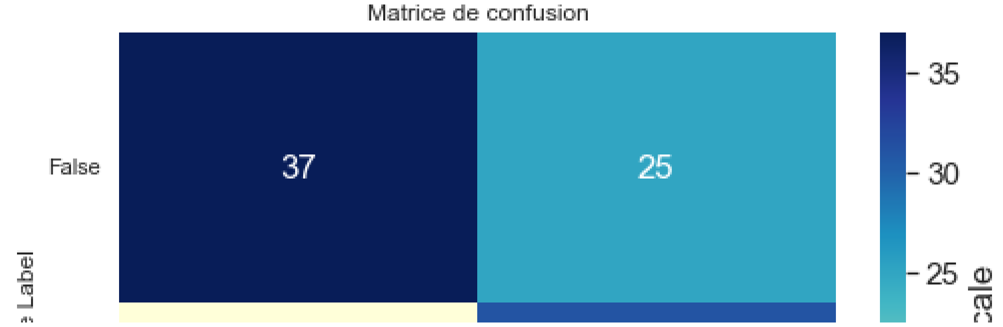
```
*--*--*--*--*--*--*--*--*--*--*     *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--* KNN *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--*     *--*--*--*--*--*--*--*--*--*--*
Accuracy : 0.612
Classification Report
              precision    recall  f1-score   support

       False    0.71154   0.59677   0.64912        62
        True    0.50980   0.63415   0.56522        41

    accuracy                        0.61165       103
   macro avg    0.61067   0.61546   0.60717       103
weighted avg    0.63124   0.61165   0.61572       103
```



```
*--*--*--*--*--*--*--*--*--*--*      *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--* CART *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--*      *--*--*--*--*--*--*--*--*--*--*
Accuracy : 0.621
Classification Report
              precision    recall  f1-score   support

       False    0.72549   0.59677   0.65487        62
        True    0.51923   0.65854   0.58065        41

    accuracy                        0.62136       103
   macro avg    0.62236   0.62766   0.61776       103
weighted avg    0.64339   0.62136   0.62532       103
```



```
*--*--*--*--*--*--*--*--*--*--*    *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--* RF *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--*    *--*--*--*--*--*--*--*--*--*--*
Accuracy : 0.660
Classification Report
              precision    recall  f1-score   support

       False    0.78723   0.59677   0.67890        62
        True    0.55357   0.75610   0.63918        41

    accuracy                        0.66019       103
   macro avg    0.67040   0.67644   0.65904       103
weighted avg    0.69422   0.66019   0.66309       103
```



Comparaison des classifieurs

Accuracy

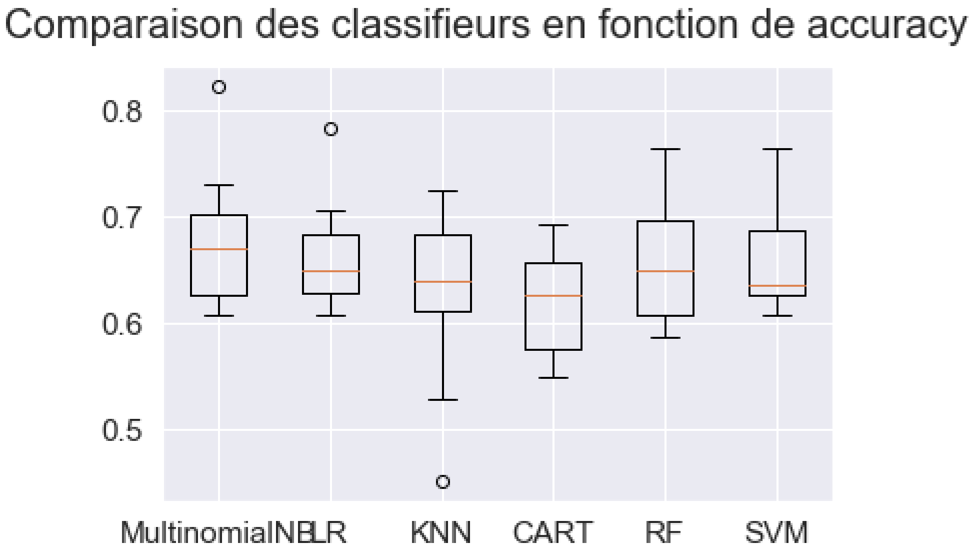
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

score = 'accuracy'
allresults = []
```

```
results = []
names = []

for name,model in models:
 # cross validation en 10 fois
 kfold = KFold(n_splits=10, random_state=seed, shuffle=True)

 print ("Evaluation de ",name)
 start_time = time.time()

 # application de la classification
 cv_results = cross_val_score(model, X_transformed, y, cv=kfold, scoring=score)

 thetime=time.time() - start_time
 result=Result(name,cv_results.mean(),cv_results.std(),thetime)
 allresults.append(result)

 # pour affichage
 results.append(cv_results)
 names.append(name)

allresults=sorted(allresults, key=lambda result: result.scoremean, reverse=True)

 # affichage résultats

print ('\n Tous les résultats :')
for result in allresults:
 print ('Classifier : ',result.name,
 ' %s : %0.3f' %(score,result.scoremean),
 ' (%0.3f)'%result.stdresult,
 ' en %0.3f '%result.timespent,' s')

print ('\nLe meilleur resultat : ')
best_accuracy = allresults[0]
print ('Classifier : ',allresults[0].name, ' %s : %0.3f' %(score,allresults[0].scor
       ' (%0.3f)'%allresults[0].stdresult, ' en %0.3f '%allresults[0].timespent,' s
```

```
Evaluation de  MultinomialNB
Evaluation de  LR
Evaluation de  KNN
Evaluation de  CART
Evaluation de  RF
Evaluation de  SVM

 Tous les résultats :
Classifier :  MultinomialNB  accuracy : 0.678  (0.062)  en 0.198   s
Classifier :  LR  accuracy : 0.665  (0.049)  en 0.647   s
Classifier :  SVM  accuracy : 0.659  (0.047)  en 6.203   s
Classifier :  RF  accuracy : 0.657  (0.057)  en 4.986   s
Classifier :  KNN  accuracy : 0.629  (0.081)  en 0.257   s
Classifier :  CART  accuracy : 0.618  (0.047)  en 2.709   s

Le meilleur resultat :
Classifier :  MultinomialNB  accuracy : 0.678  (0.062)  en 0.198   s
```

```
import matplotlib.pyplot as plt

fig = plt.figure()
fig.suptitle('Comparaison des classifieurs en fonction de accuracy')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
```

```
[Text(1, 0, 'MultinomialNB'),
 Text(2, 0, 'LR'),
 Text(3, 0, 'KNN'),
 Text(4, 0, 'CART'),
 Text(5, 0, 'RF'),
 Text(6, 0, 'SVM')]
```



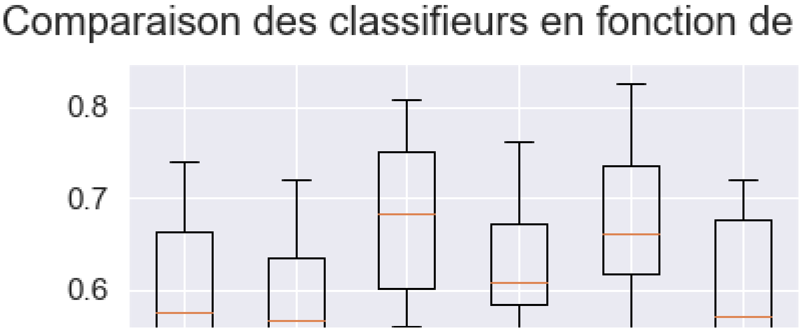
Recall

```
score = 'recall'
allresults = []
results = []
names = []

for name,model in models:
 # cross validation en 10 fois
 kfold = KFold(n_splits=10, random_state=seed, shuffle=True)

 print ("Evaluation de ",name)
 start_time = time.time()

 # application de la classification
 cv_results = cross_val_score(model, X_transformed, y, cv=kfold, scoring=score)

 thetime=time.time() - start_time
 result=Result(name,cv_results.mean(),cv_results.std(),thetime)
 allresults.append(result)
 # pour affichage
 results.append(cv_results)
 names.append(name)

allresults=sorted(allresults, key=lambda result: result.scoremean, reverse=True)

 # affichage des résultats

print ('\nTous les résultats :')
for result in allresults:
 print ('Classifier : ',result.name,
 ' %s : %0.3f' %(score,result.scoremean),
 ' (%0.3f)'%result.stdresult,
 ' en %0.3f '%result.timespent,' s')

print ('\nLe meilleur resultat : ')
best_recall = allresults[0]
print ('Classifier : ',allresults[0].name, ' %s : %0.3f' %(score,allresults[0].scor
       ' (%0.3f)'%allresults[0].stdresult, ' en %0.3f '%allresults[0].timespent,' s
```

```
Evaluation de  MultinomialNB
Evaluation de  LR
Evaluation de  KNN
Evaluation de  CART
Evaluation de  RF
Evaluation de  SVM

Tous les résultats :
Classifier :  KNN  recall : 0.678  (0.086)  en 0.277   s
Classifier :  RF  recall : 0.670  (0.091)  en 5.409   s
Classifier :  CART  recall : 0.622  (0.074)  en 2.855   s
Classifier :  LR  recall : 0.582  (0.081)  en 0.538   s
Classifier :  SVM  recall : 0.578  (0.101)  en 6.249   s
Classifier :  MultinomialNB  recall : 0.576  (0.104)  en 0.225   s

Le meilleur resultat :
Classifier :  KNN  recall : 0.678  (0.086)  en 0.277   s
```

```
import matplotlib.pyplot as plt

fig = plt.figure()
fig.suptitle('Comparaison des classifieurs en fonction de recall')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
```

```
[Text(1, 0, 'MultinomialNB'),
 Text(2, 0, 'LR'),
 Text(3, 0, 'KNN'),
 Text(4, 0, 'CART'),
 Text(5, 0, 'RF'),
 Text(6, 0, 'SVM')]
```



Essai des meilleurs classifieurs en variant les hyper paramètres

```
params = {
 'GaussianNB' :
    [{'var_smoothing': np.logspace(0,-9, num=100)}],

 'RF':[{'bootstrap': [True, False],
    'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}],

 'LR' : [{'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'C' : np.logspace(-4, 4, 20),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
    'max_iter' : [100, 1000,2500, 5000]
    }],

  'CART' : [{'max_depth': [2, 3, 5, 10, 21],
    'min_samples_leaf': [5, 10,16 ,20, 50, 100],
    'criterion': ["gini", "entropy"]}],

  'SVM' : [{'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'p

  'KNN' : [{ 'n_neighbors' : [5,7,9,11,13,15],
               'weights' : ['uniform','distance'],
               'metric' : ['minkowski','euclidean','manhattan']}],

  'MultinomialNB' : [{
    'alpha': np.linspace(0.5, 1.5, 6),
    'fit_prior': [True, False]}]
}

params_lite = {
    'GaussianNB' :
        [{'var_smoothing': np.logspace(0,-9, num=100)}],

    'RF':[{'n_estimators': [4, 6, 9],
           'max_features': ['log2', 'sqrt','auto'],
           'criterion': ['entropy', 'gini'],
           'max_depth': [2, 3, 5, 10],
           'min_samples_split': [2, 3, 5],
           'min_samples_leaf': [1,5,8]
           }],

    'LR' : [{'C' : [0.001,0.01,0.1,1,10,100]}],

    'CART' : [{'max_depth': [1,2,3,4,5,6,7,8,9,10],
               'criterion': ['gini', 'entropy'],
               'min_samples_leaf': [1,2,3,4,5,6,7,8,9,10]}],

    'SVM' : [{'C': [0.001, 0.01, 0.1, 1, 10],
              'gamma' : [0.001, 0.01, 0.1, 1],
              'kernel': ['linear','rbf']}],

    'KNN' : [{ 'n_neighbors' : [5,7,9,11,13,15],
               'weights' : ['uniform','distance'],
               'metric' : ['minkowski','euclidean','manhattan']}],

    'MultinomialNB' : [{
        'alpha': np.linspace(0.5, 1.5, 6),
        'fit_prior': [True, False]}]
}


from sklearn.model_selection import GridSearchCV

print("*--*--*--*--*--*--*--*--*--*--* "+len(best_accuracy.name)*" " +" *--*--*--*-
print("*--*--*--*--*--*--*--*--*--*--* "+best_accuracy.name+" *--*--*--*--*--*--
print("*--*--*--*--*--*--*--*--*--*--* "+len(best_accuracy.name)*" " +" *--*--*--*-
for name,model in models:
    if best_accuracy.name == name:
        themodel = model
        theparams = params_lite[name]
gd_sr = GridSearchCV(estimator=themodel,
                     param_grid=theparams,
                     scoring='accuracy',
                     cv=5,
                     n_jobs=-1,
                     return_train_score=True)

gd_sr.fit(X_train, y_train)

print ('Accuracy :')
print ('meilleur score ',gd_sr.best_score_,'\n')
print ('meilleurs paramètres', gd_sr.best_params_,'\n')
print ('meilleur estimateur',gd_sr.best_estimator_,'\n')
```

```
gd_sr = GridSearchCV(estimator=themodel,
                     param_grid=theparams,
                     scoring='recall',
                     cv=5,
                     n_jobs=-1,
                     return_train_score=True)

gd_sr.fit(X_train, y_train)

print ('Recall :')
print ('meilleur score ',gd_sr.best_score_,'\n')
print ('meilleurs paramètres', gd_sr.best_params_,'\n')
print ('meilleur estimateur',gd_sr.best_estimator_,'\n')
```

```
*--*--*--*--*--*--*--*--*--*--*               *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--* MultinomialNB *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--*               *--*--*--*--*--*--*--*--*--*--*
Accuracy :
meilleur score  0.6414634146341464

meilleurs paramètres {'alpha': 0.5, 'fit_prior': False}

meilleur estimateur MultinomialNB(alpha=0.5, fit_prior=False)

Recall :
meilleur score  0.6452961672473868

meilleurs paramètres {'alpha': 0.5, 'fit_prior': True}

meilleur estimateur MultinomialNB(alpha=0.5)
```

```
if best_accuracy.name != best_recall.name:
    print("*--*--*--*--*--*--*--*--*--*--* "+len(best_recall.name)*" " +" *--*--*--
    print("*--*--*--*--*--*--*--*--*--*--* "+best_recall.name+" *--*--*--*--*--*--*
    print("*--*--*--*--*--*--*--*--*--*--* "+len(best_recall.name)*" " +" *--*--*--
    for name,model in models:
        if best_recall.name == name:
            themodel = model
            theparams = params_lite[name]

    gd_sr = GridSearchCV(estimator=themodel,
                         param_grid=theparams,
                         scoring='accuracy',
                         cv=5,
                         n_jobs=-1,
                         return_train_score=True)

    gd_sr.fit(X_train, y_train)

    print ('Accuracy :')
    print ('meilleur score ',gd_sr.best_score_,'\n')
    print ('meilleurs paramètres', gd_sr.best_params_,'\n')
    print ('meilleur estimateur',gd_sr.best_estimator_,'\n')
    gd_sr = GridSearchCV(estimator=themodel,
                         param_grid=theparams,
                         scoring='recall',
                         cv=5,
                         n_jobs=-1,
                         return_train_score=True)

    gd_sr.fit(X_train, y_train)

    print ('Recall :')
    print ('meilleur score ',gd_sr.best_score_,'\n')
    print ('meilleurs paramètres', gd_sr.best_params_,'\n')
    print ('meilleur estimateur',gd_sr.best_estimator_,'\n')
```

```
*--*--*--*--*--*--*--*--*--*--*     *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--* KNN *--*--*--*--*--*--*--*--*--*--*
*--*--*--*--*--*--*--*--*--*--*     *--*--*--*--*--*--*--*--*--*--*
Accuracy :
meilleur score  0.6097560975609756

meilleurs paramètres {'metric': 'minkowski', 'n_neighbors': 7, 'weights': 'uni

meilleur estimateur KNeighborsClassifier(n_neighbors=7)

Recall :
meilleur score  0.9269454123112659

meilleurs paramètres {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'uni

meilleur estimateur KNeighborsClassifier(metric='manhattan')
```

Creation du pipe

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

print ('Création du pipeline \n')
for name,model in models:
    if best_accuracy.name == name:
        pipeline = Pipeline([('scl', StandardScaler()), ('clf', model)])
        break

Xsource = df.source
Xtext = df.text
Xheadline = df.headline
Xauthor = df.author
Xentities = df.named_entities_claim
Xkeywords = df.keywords
```