#**CLASSIFICATION : TRUE/FALSE VS OTHER :**

Membres: Hadjoudja Bachir (21811363), Zeggar Rym (21909615), Bendahmane Rania (21811387), Labiad Youcef (21710780).

```
#les imports utilisés dans ce notebook
import sys
from numpy import vstack
import pandas as pd
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.data import random_split
from torch import Tensor
from torch.nn import Linear
from torch.nn import ReLU
from torch.nn import Sigmoid
from torch.nn import Module
from torch.optim import SGD
from torch.nn import BCELoss
from torch.nn.init import kaiming_uniform_
from torch.nn.init import xavier_uniform_
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from pandas import read_csv
from sklearn.feature_extraction.text import TfidfVectorizer
from  sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
import pickle
import string

import nltk
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk import word_tokenize
from sklearn.pipeline import Pipeline
```

autorisation

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

```
Drive already mounted at /content/gdrive/; to attempt to forcibly
remount, call drive.mount("/content/gdrive/", force_remount=True).
```

chemin spécifique Google Drive

```
my_local_drive='/content/gdrive/My Drive/Colab Notebooks'
# 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
%ls

%pwd
```

```
/content/gdrive/My Drive/Colab Notebooks
 avecscaler.pkl
 Classification_de_données_textuelles2023.ipynb
'Copie de TRUE FALSE_vs_OTHER.ipynb'
 Dataset/
 firstmodel.pkl
'Ingénierie_des_données_textuelles2023 (1).ipynb'
 Ingénierie_des_données_textuelles2023.ipynb
 MyNLPUtilities.py
 newsTrain2.csv
 newsTrain_-_newsTrain.csv
 penguins.csv
 penguins.csv.1
 pkl_modelNB.sav
 Premières_Classifications.ipynb
'Projet ML FakeNEWS_TRUE FALSE_TEXT.ipynb'
'Projet ML FakeNEWS_TRUE FALSE_TEXT+TITRE.ipynb'
'Projet ML FakeNEWS_TRUE FALSE_TEXT+TITRE_TOPIC_MODELLING.ipynb'
'Projet ML FakeNEWS_TRUE FALSE_TITRE.ipynb'
 __pycache__/
 ReviewsLabelled.csv
 ReviewsLabelled.csv.1
 ReviewsLabelled.csv.2
 ReviewsLabelled.csv.3
 ReviewsLabelled.csv.4
 ReviewsLabelled.csv.5
 SentimentModel.pkl
 StopWordsFrench.csv
 StopWordsFrench.csv.1
 StopWordsFrench.csv.2
 StopWordsFrench.csv.3
 StopWordsFrench.csv.4
 Topics_extraction.ipynb
 TP1_HAI817I.ipynb
 TP2_HAI817I.ipynb
'TRUE FALSE_vs_OTHER.ipynb'
 Visualisation_Donnees_2D_3D.ipynb
```

{"type":"string"}

La fonction qui sera utilisée pour les prétraitements: MyCleanText

- Mettre le texte en minuscule
- Se débarasser des stopwords
- Se débarasser des nombres
- Stemmatisation
- Lemmatisation ..

```python
#.................................................Fonction MyCleanText .......................................................................
# mettre en minuscule
#enlever les stopwords
#se debarasser des nombres
#stemmatisation
#lemmatisation
#.............................................................................................................................................


nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')
#liste des stopwords en anglais
stop_words = set(stopwords.words('english'))

def MyCleanText(X,
                lowercase=False, #mettre en minuscule
                removestopwords=False, #supprimer les stopwords
                removedigit=False, #supprimer les nombres
                getstemmer=False, #conserver la racine des termes
                getlemmatisation=False #lemmatisation des termes
                ):
  #conversion du texte d'entrée en chaîne de caractères
    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)

    # decoupage 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
```

```
[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!
```

- La classe TextNormalizer qui contiendra la fonction MyCleanText.
- Fit_transform de mon corpus propre.

```
#.................................................Etape 1 :
prétraitement du
texte .....................................................................
..................................
#..................................................Class
TextNormalizer  ......................................................
......................
#fit_transform de mon corpus propre
#.....................................................................
.....................................................................
..........
```

```python
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

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

    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,
            'removestopwords':self.removestopwords,
            'getlemmatisation':self.getlemmatisation,
            'removedigit':self.removedigit
        }

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

##**Etape 1 : Préparer les données**

- Load et preparer les données à partir des 2 fichiers csv

- Sélectionner que les lignes où on a True, False ou Other
- Après en créant une nouvelle colonne "regrouped" si la valeur de la colonne rating est true ou bien false on mettra TRUE/FALSE sinon on laisse OTHER

```
#Ici je cherche à séléctionner que les labels TRUE/FALSE et OTHER,
donc les LIGNES qui contiennent au rating TRUE,FALSE et OTHER
uniquement, le reste on enlève

dftrain = pd.read_csv("/content/gdrive/MyDrive/Colab
Notebooks/newsTrain2.csv", names=['id','text','title','rating'],
header=0,sep=',', encoding='utf8')
dftrain.reset_index(drop = True, inplace = True)

dftrain2 = pd.read_csv("/content/gdrive/MyDrive/Colab
Notebooks/newsTrain_-_newsTrain.csv",
names=['id','text','title','rating'], header=0,sep=',',
encoding='utf8')
dftrain2.reset_index(drop = True, inplace = True)


# concaténer les deux dataframes en ajoutant les lignes du deuxième à
la fin du premier
dftrain = pd.concat([dftrain, dftrain2], ignore_index=True)
dftrain = dftrain.loc[dftrain['rating'].isin(['TRUE','FALSE',
'other'])]

#On crée une colonne regroupe qui va mettre dans les lignes là où a
true ou bien false la valeur TRUE/FALSE et OTHER ça laisse
dftrain['regrouped'] = dftrain['rating'].apply(lambda x:'TRUE/FALSE'
if x in ['TRUE', 'FALSE'] else 'OTHER')

#Quelques affichages pour aider à mieux visualiser nos données
print("Echantillon de mon dataset \n")
print(dftrain.sample(n=10))
print("\n")
print("Quelques informations importantes \n")
dftrain.info()
```

```
Echantillon de mon dataset

            id                                               text  \
2374  d8c5eecd  Teachers who leave the profession for other jo...
986   d1741354  News| [email protected] “If you won’t lead, th...
832   97b3e15c  Denying 2000 years of the Medieval Warm Period...
2425  7f8bf578  President Trump has sometimes claimed that sci...
1451  2963ac03  General Colin Powell’s Chief of Staff, Col. Wi...
2356  e5a08193  Image copyright Getty Images  Workers on zero-...
2064   2695016  He did none of those things. I’ve reviewed all...
2205  9f10a8a9  It was an accurate and judicious answer, so na...
993   88a75bcc  A 30-year old man from Kentucky underwent some...
```

```
1799  d2a52dd6                          AMA Lied – How Many Died?

                                                          title rating
regrouped
2374  Relentless' workload forcing 'desperate' teach...  other
OTHER
986   IT'S OFFICIAL: Brexit Britain WILL thrive out ...  FALSE
TRUE/FALSE
832   Gov't Seeks to Control 'Disorderly' Internet P...  FALSE
TRUE/FALSE
2425  Climate Change Is Complex. We've Got Answers t...   TRUE
TRUE/FALSE
1451  General Colin Powell's Chief of Staff drops th...  FALSE
TRUE/FALSE
2356  Workplace reforms 'will protect gig economy wo...  FALSE
TRUE/FALSE
2064  Getting on with the job this week – Scottish N...   TRUE
TRUE/FALSE
2205                                A 62% Top Tax Rate?  other
OTHER
993   Months after being denied media credentials fo...  FALSE
TRUE/FALSE
1799  Impeachment Lawyers opened Trump's second Impe...  FALSE
TRUE/FALSE


Quelques informations importantes

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1812 entries, 0 to 2527
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   id         1812 non-null   object
 1   text       1812 non-null   object
 2   title      1784 non-null   object
 3   rating     1812 non-null   object
 4   regrouped  1812 non-null   object
dtypes: object(5)
memory usage: 84.9+ KB
```

Le jeu de données étant déséquilibré, on a pensé à appliquer le downsampling pour équilibrer nos données. on séléctionne des lignes aléatoirement de TRUE/FALSE de telle sorte que le nombre de lignes de TRUE/FALSE soit = au nbr de lignes de Other. et on mélange le DataFrame.

```
#On applique du sous-échantillonnage (downsampling) : car on a plus de
FALSE (578) que des TRUE (211)

# Séparer les classes en deux dataframes
```

```
df_false_true = dftrain[dftrain['regrouped'] == 'TRUE/FALSE']
df_other = dftrain [dftrain['regrouped'] == 'OTHER']

# Sous-échantillonner la classe majoritaire (FALSE) pour obtenir un
nombre égal d'échantillons pour chaque classe
df_subsampled = df_false_true.sample(n=len(df_other), random_state=42)

# Concaténer les deux dataframes
dftrain = pd.concat([df_subsampled, df_other])

# Mélanger aléatoirement les données
dftrain = dftrain.sample(frac=1, random_state=42)

X_text=dftrain.iloc[0:,1:2]
X_title=dftrain.iloc[0:,2:3]

print("le texte est")
display(X_text)
print("le titre est")
display(X_title)

y=dftrain.iloc[0:,-1]
print("le y est")
display(y)
print("la taille de X_text est",X_text.shape)
print("la taille de y_train est " ,y.shape)
print("les valeurs de TRUE/FALSE et OTHER maintenant sont
" ,y.value_counts())
```

```
le texte est

                                                   text
947   War-torn eastern regions of Ukraine have no la...
2224  TIJUANA, Mexico — It’s the image from the unfo...
1307  Today, Congresswoman Maxine Waters D-CA, Chair...
798   Meghan Markle will use the furore over her int...
320   Further proof that Democrats are the greatest ...
...                                                 ...
1160  The scale of Antarctica is startling. Miles of...
570   Coronavirus may be sexually transmitted and ca...
1200  Like what?  Helen Harwatt is a researcher trai...
2190                    Tumeric kills cancer not patient
391   WASHINGTON, DC — The Pentagon has issued an in...

[468 rows x 1 columns]

le titre est

                                                  title
947   Look No Further, The Best Doctor Strange in th...
```

```
2224  A discussion of ‘smokers’ black lungs’ started...
1307  Democratic Lawmaker introduces bill to rename ...
798   Newton Emerson: Swiss model offers food for th...
320   Democrats Introduce Bill To ‘Euthanize Seniors...
...                                                 ...
1160               Miles of Ice Collapsing Into the Sea
570   Universal Credit leaves working families worse...
1200              If Everyone Ate Beans Instead of Beef
2190  Vermont state trooper revived with Narcan afte...
391   Pentagon Confirms Coronavirus Accidently Got I...

[468 rows x 1 columns]

le y est

947     TRUE/FALSE
2224    TRUE/FALSE
1307    TRUE/FALSE
798          OTHER
320     TRUE/FALSE
           ...
1160    TRUE/FALSE
570          OTHER
1200         OTHER
2190         OTHER
391     TRUE/FALSE
Name: regrouped, Length: 468, dtype: object

la taille de X_text est (468, 1)
la taille de y_train est  (468,)
les valeurs de TRUE et FALSE maintenant sont  TRUE/FALSE    234
OTHER         234
Name: regrouped, dtype: int64
```

On divise notre grand X en jeu de données d'apprentissage et de test (20% de test).

```
X=dftrain.iloc[0:, 1:4]
print(X)

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size = 0.2,
random_state=10)
print("X_train is",X_train)
print("y_train is",y_train)
print("X_test is",X_test)
print("y_test is",y_test)
```

```
                                                   text  \
947   War-torn eastern regions of Ukraine have no la...
2224  TIJUANA, Mexico — It’s the image from the unfo...
1307  Today, Congresswoman Maxine Waters D-CA, Chair...
```

```
798   Meghan Markle will use the furore over her int...
320   Further proof that Democrats are the greatest ...
...                                                 ...
1160  The scale of Antarctica is startling. Miles of...
570   Coronavirus may be sexually transmitted and ca...
1200  Like what?  Helen Harwatt is a researcher trai...
2190                   Tumeric kills cancer not patient
391   WASHINGTON, DC – The Pentagon has issued an in...

                                                  title rating
947   Look No Further, The Best Doctor Strange in th...  FALSE
2224  A discussion of ‘smokers’ black lungs’ started...   TRUE
1307  Democratic Lawmaker introduces bill to rename ...  FALSE
798   Newton Emerson: Swiss model offers food for th...  other
320   Democrats Introduce Bill To ‘Euthanize Seniors...  FALSE
...                                                 ...    ...
1160               Miles of Ice Collapsing Into the Sea   TRUE
570   Universal Credit leaves working families worse...  other
1200             If Everyone Ate Beans Instead of Beef  other
2190  Vermont state trooper revived with Narcan afte...  other
391   Pentagon Confirms Coronavirus Accidently Got I...  FALSE

[468 rows x 3 columns]
X_train is                                                      text  \
2006  Historians may look to 2015 as the year when s...
1834  Coronavirus may be sexually transmitted and ca...
530   Contractors bidding for work with the governme...
564   More CO2 would actually help the planet , says...
340   To say out-loud that you find the results of t...
...                                                 ...
1775  Last week, in the days leading up to Sanders’ ...
208   This is one in a series of articles taken from...
562   Food parcels arriving at the Community Service...
676   On Tuesday, radio show host John Fredricks sta...
1925  A South African pastor, Alfred Ndlovu has died...

                                                  title rating
2006                                                NaN  other
1834  Universal Credit leaves working families worse...  other
530   Firms bidding for government contracts asked i...  other
564   Mitt Romney transfers $1 million left over fro...  other
340   Reasons why the 2020 presidential election is ...  other
...                                                 ...    ...
1775       Bernie makes it official: It’s Biden or bust  other
208   European royals killing naked children for fun...  other
562   Adam Castillejo is still free of the virus mor...  other
676                           Warren Statement on Boeing  other
1925  Joy Covey: Amazon pioneer and high tech rock s...  other

[374 rows x 3 columns]
```

```
y_train is 2006    OTHER
1834    OTHER
530     OTHER
564     OTHER
340     OTHER
        ...
1775    OTHER
208     OTHER
562     OTHER
676     OTHER
1925    OTHER
Name: regrouped, Length: 374, dtype: object
X_test is                                                    text  \
459   Thank you to Universities UK UUK for hosting u...
1519  Can any government statistics on COVID-19 deat...
925   MOSCOW — Russian President Vladimir Putin over...
1857  Please enable cookies on your web browser in o...
2386  PUPILS aged just five have been accused of sex...
...                                                 ...
1778  With a smile on her face, City Clerk Susana Me...
2205  It was an accurate and judicious answer, so na...
1568  Barack Obama, a former President of the US, wa...
66    Pennsylvania rejects 372,000 mail-in ballots, ...
2355  Rises in National Insurance Contributions NICS...

                                                 title rating
459   USCIS Announces Final Rule Enforcing Long-Stan...   TRUE
1519  The CDC Confesses to Lying About COVID-19 Deat...  FALSE
925          Short breaks damage young people’s futures  other
1857  Denying 2000 years of the Medieval Warm Period...  FALSE
2386  Pervs' aged five School sex crime claims trebl...  other
...                                                 ...    ...
1778  Trump administration asks Supreme Court to str...   TRUE
2205                              A 62% Top Tax Rate?  other
1568  Former President Barack Obama arrested for ESP...  FALSE
66    Pennsylvania rejects 372,000 mail-in ballots, ...  FALSE
2355  Budget 2017: National Insurance rate rise crit...  other

[94 rows x 3 columns]
y_test is 459     TRUE/FALSE
1519    TRUE/FALSE
925          OTHER
1857    TRUE/FALSE
2386         OTHER
           ...
1778    TRUE/FALSE
2205         OTHER
1568    TRUE/FALSE
66      TRUE/FALSE
```

```
2355         OTHER
Name: regrouped, Length: 94, dtype: object
```

##**Etape 2 : Classification selon la colonne TEXT :**

Tester avec plusieurs classifieurs classiques.

```python
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
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_fscore_support as score
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.ensemble import RandomForestClassifier

# Importation des différentes librairies utiles pour le notebook
#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 matplotlib.pyplot as plt
import sys
import pandas as pd
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
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
#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)

from sklearn.metrics._plot.confusion_matrix import
ConfusionMatrixDisplay
from sklearn import metrics
from sklearn.metrics import confusion_matrix , ConfusionMatrixDisplay
from sklearn.metrics import classification_report

from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
import time
```

La fonction MyshowAllScores prend le y_test et le y_predict, affiche l'accuracy et le classification report avec la matrice de confusion.

```python
# fonction qui affiche le classification report et la matrice de
confusion
def MyshowAllScores(y_test,y_pred):
  classes= np.unique(y_test)
  print("Accuracy : %0.3f"%(accuracy_score(y_test,y_pred)))
  print("Classification Report")
  print(classification_report(y_test,y_pred,digits=5))
  cnf_matrix = confusion_matrix(y_test,y_pred)
  disp=ConfusionMatrixDisplay(cnf_matrix,display_labels=classes)
  disp.plot()
```

Vu qu'on va travailler sur la colonne text, on va séléctionner cette dernière depuis le X_train et X_test pour apprendre et tester après.

```
X_train_text=X_train['text']
X_train_text.reset_index(drop = True, inplace = True)
X_test_text=X_test['text']
X_test_text.reset_index(drop = True, inplace = True)
```

Ici, c'est une étape importante, on va tester différents classifieurs, pour chacun des classifieurs, on va appliquer le prétraitement + Vectorisation TfIdf, et on applique une cross_val_score avec un Kfold de 10 fois, par la suite on stocke dans une liste all_results la moyenne des accuracy + l'écart type et on la trie par ordre décroissant de moyenne d'accuracy et d'écart type. on remarque que les 3 meilleurs sont SVM,LR et RF qu'on va séléctionner pour leur appliquer le GridSearch sur les paramètres des prétraitements + leurs hyperparamètres pour pouvoir choisir le meilleur.

```
np.random.seed(42)  # Set the random seed for NumPy

score = 'accuracy'
seed = 7
allresults = []
results = []
names = []


# Liste des modèles à tester
models = [
    ('MultinomialNB', MultinomialNB()),
    ('LogisticRegression', LogisticRegression(random_state=42)),
    ('KNN', KNeighborsClassifier()),
    ('CART', DecisionTreeClassifier(random_state=42)),
    ('RF', RandomForestClassifier(random_state=42)),
    ('SVM', SVC(random_state=42))
]


# Création d'un pipeline pour chaque modèle
pipelines = []
for name,model in models:
    pipeline = Pipeline([
        ('normalize', TextNormalizer()),
        ('tfidf', TfidfVectorizer()),
        (name,model)
    ])
    pipelines.append((name,pipeline))

all_results=[]
scores=[]
names=[]
for p in pipelines:
    print(p[1])
    # cross validation en 10 fois
```

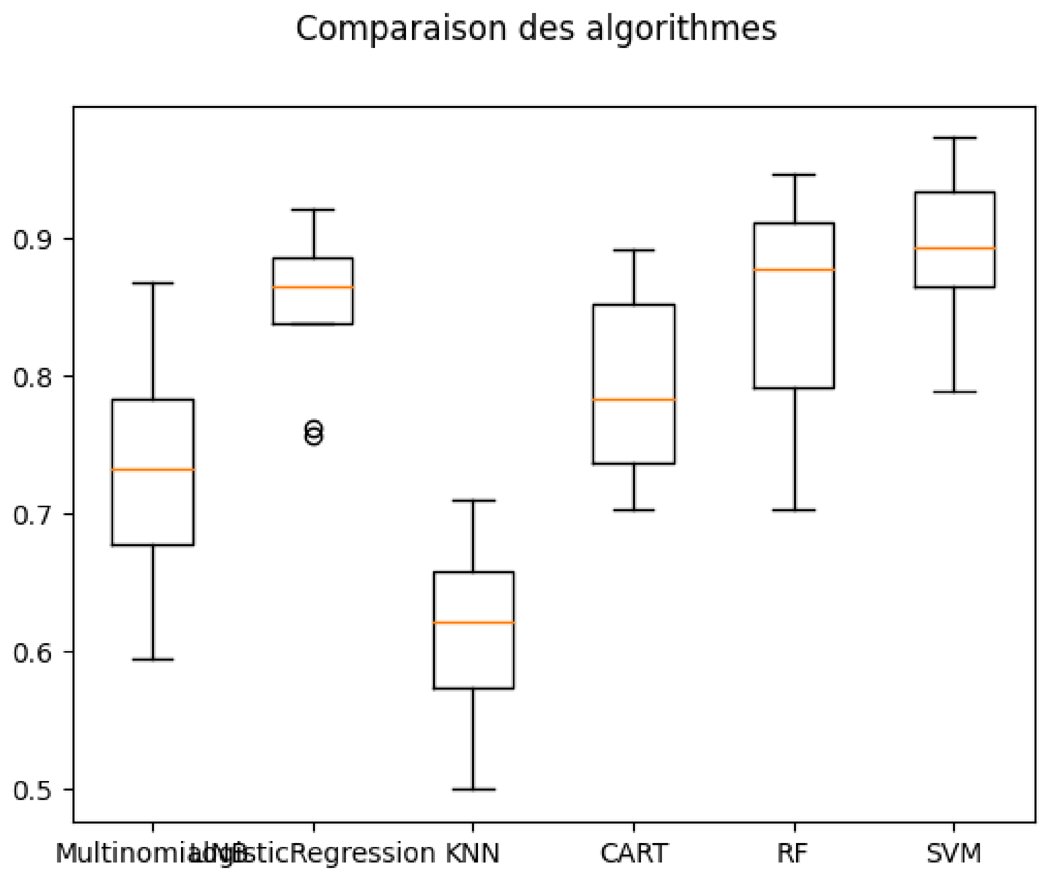
```
    kfold = KFold(n_splits=10,random_state=seed,shuffle=True)
    start_time = time.time()
    # application de la classification
    cv_results = cross_val_score(p[1],X_train_text,y_train, cv=kfold,
scoring=score)
    scores.append(cv_results)
    names.append(p[0])
    all_results.append((p[0],cv_results.mean(),cv_results.std()))
    end_time = time.time()


print("all resultats", all_results)

all_results = sorted(all_results, key=lambda x: (-x[1], -x[2]))
print("all resultats", all_results)
```

```
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('MultinomialNB', MultinomialNB())])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('LogisticRegression',
LogisticRegression(random_state=42))])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('KNN', KNeighborsClassifier())])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('CART', DecisionTreeClassifier(random_state=42))])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('RF', RandomForestClassifier(random_state=42))])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('SVM', SVC(random_state=42))])
all resultats [('MultinomialNB', 0.7273115220483642,
0.08390022989848737), ('LogisticRegression', 0.8527738264580369,
0.05393319572545874), ('KNN', 0.620199146514936, 0.06317935636792352),
('CART', 0.7891891891891893, 0.06474231016745102), ('RF',
0.8475106685633002, 0.0793750031815335), ('SVM', 0.893172119487909,
0.05576555995138434)]
all resultats [('SVM', 0.893172119487909, 0.05576555995138434),
('LogisticRegression', 0.8527738264580369, 0.05393319572545874),
('RF', 0.8475106685633002, 0.0793750031815335), ('CART',
0.7891891891891893, 0.06474231016745102), ('MultinomialNB',
0.7273115220483642, 0.08390022989848737), ('KNN', 0.620199146514936,
0.06317935636792352)]
```

On affiche les accuracy de chaque classifieur, on remarque la médiane (en rouge) de chaque et l'écart type aussi.

```python
import matplotlib.pyplot as plt
fig = plt.figure()
fig.suptitle('Comparaison des algorithmes')
ax = fig.add_subplot(111)
plt.boxplot(scores)
ax.set_xticklabels(names)
```

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



Choisir les meilleurs paramètres et hyperparamètres pour SVM, RF et LR :

On a un pipeline pour chaque prétraitement différent, on essaye pas mal (miniscule, lemmatisation, miniscule + lemmatisation..) et on stocke le fit_transorm de nos X_train, X_test sur les pipelines dans des listes qui vont contenir tous les fit_transform des pipelines pour chaque classifieur, par la suite on parcourt ces listes là, on itère dessus, et chaque élement de la liste (train) va passer par le GridSearch et puis on predict sur son corresapondant dans liste (test).

```
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from tabulate import tabulate

np.random.seed(42)  # Set the random seed for NumPy

# le plus simple est de faire un test sur differents pipelines.
# pipeline de l'utilisation de CountVectorizer sur le texte avec
differents pre-traitements
CV_brut = Pipeline([('cleaner', TextNormalizer()),
                    ('count_vectorizer',
CountVectorizer(lowercase=False))])
CV_lowcase = Pipeline([('cleaner',
TextNormalizer(removestopwords=False,lowercase=True,

getstemmer=False,removedigit=False)),
                    ('count_vectorizer',
CountVectorizer(lowercase=False))])
CV_lowStop = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=False,removedigit=False)),
                    ('count_vectorizer',
CountVectorizer(lowercase=False))])

CV_lowStopstem = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=True,removedigit=False)),
                    ('count_vectorizer',
CountVectorizer(lowercase=False))])

# pipeline de l'utilisation de TfidfVectorizer avec differents pre-
traitements
TFIDF_brut = Pipeline ([('cleaner', TextNormalizer()),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])

TFIDF_lowcase = Pipeline([('cleaner',
TextNormalizer(removestopwords=False,lowercase=True,
```

```
getstemmer=False,removedigit=False)),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])
TFIDF_lowStop = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=False,removedigit=False)),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])

TFIDF_lowStopstem = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=True,removedigit=False)),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])


# Liste de tous les modeles à tester
all_models = [
    ("CV_brut", CV_brut),
    ("CV_lowcase", CV_lowcase),
    ("CV_lowStop", CV_lowStop),
    ("CV_lowStopstem",CV_lowStopstem),
    ("TFIDF_lowcase", TFIDF_lowcase),
    ("TFIDF_lowStop", TFIDF_lowStop),
    ("TFIDF_lowStopstem",TFIDF_lowStopstem),
    ("TFIDF_brut", TFIDF_brut)
]

#on crée des listes qui vont contenir les fit_transform des pipelines
sur le X_train et X_test
X_train_text_SVC = []
X_test_text_SVC = []

X_train_text_RandomForestClassifier = []
X_test_text_RandomForestClassifier = []

X_train_text_LogisticRegression = []
X_test_text_LogisticRegression = []

for name, pipeline in all_models :

X_train_text_SVC.append(pipeline.fit_transform(X_train_text).toarray()
)
    X_test_text_SVC.append(pipeline.transform(X_test_text).toarray())

X_train_text_RandomForestClassifier.append(pipeline.fit_transform(X_tr
```

```
ain_text).toarray())

X_test_text_RandomForestClassifier.append(pipeline.transform(X_test_te
xt).toarray())

X_train_text_LogisticRegression.append(pipeline.fit_transform(X_train_
text).toarray())

X_test_text_LogisticRegression.append(pipeline.transform(X_test_text).
toarray())


models = {
    'SVC': SVC(random_state=42),
    'LogisticRegression' : LogisticRegression(random_state=42),
    'RandomForestClassifier': RandomForestClassifier(random_state=42)
}


params = {'SVC': [{'C': [0.001, 0.01, 0.1, 1,2,5,7,10]},
             {'gamma': [0.001, 0.01, 0.1,0.2,0.3,0.5,0.7,1]},
             {'kernel': ['linear', 'rbf']}],
    'RandomForestClassifier': [{'n_estimators': [10, 50, 100, 200,
300]},
                              {'max_features': ['auto', 'sqrt',
'log2']}],
    'LogisticRegression': [{'penalty': ['l1', 'l2','elasticnet',
'none']},
                              {'C': [0.001, 0.01, 0.1, 1, 10, 100]},
                              {'fit_intercept': [True,False]},
                              {'solver': ['newton-cg', 'lbfgs',
'liblinear', 'sag', 'saga']},
                              {'max_iter': [100, 1000, 10000]}]
}
#On itère sur le dictionnaire des modèles
for model_name, model in models.items():
    score='accuracy'
    X_train_text = eval('X_train_text_' + model_name) #On crée une
variable X_train_text qui est dynamique qui est la liste du
fit_transform du X_train d'un classifieur
    X_test_text = eval('X_test_text_' + model_name) #On crée une
variable X_test_text qui est dynamique qui est la liste du
fit_transform du X_test d'un classifieur
    for i in range (len(X_train_text)): #on itère sur cette liste
      grid_search = GridSearchCV(model, params[model_name], n_jobs=-1,
verbose=1,scoring=score) #on applique le GridSearch
      print("grid search fait")
      grid_search.fit(X_train_text[i],y_train)
      print ('meilleur score %0.3f'%(grid_search.best_score_),'\n')
      print ('meilleur estimateur',grid_search.best_estimator_,'\n')
```

```
      y_pred = grid_search.predict(X_test_text[i]) #on predict
      MyshowAllScores(y_test,y_pred) #matrice de confusion report
classification

      print("Ensemble des meilleurs paramètres :")
      best_parameters = grid_search.best_estimator_.get_params()
      for param_dict in params[model_name]:
        for param_name, param_value in param_dict.items():
            print("\t%s: %r" % (param_name,
best_parameters[param_name]))
```

```
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.901

meilleur estimateur SVC(gamma=0.5, random_state=42)

Accuracy : 0.968
Classification Report
              precision    recall  f1-score   support

       OTHER    0.97500   0.95122   0.96296        41
  TRUE/FALSE    0.96296   0.98113   0.97196        53

    accuracy                        0.96809        94
   macro avg    0.96898   0.96618   0.96746        94
weighted avg    0.96821   0.96809   0.96804        94

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 0.5
	kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.901

meilleur estimateur SVC(gamma=0.5, random_state=42)

Accuracy : 0.968
Classification Report
              precision    recall  f1-score   support

       OTHER    0.97500   0.95122   0.96296        41
  TRUE/FALSE    0.96296   0.98113   0.97196        53

    accuracy                        0.96809        94
   macro avg    0.96898   0.96618   0.96746        94
weighted avg    0.96821   0.96809   0.96804        94
```

```
Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 0.5
	kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.901

meilleur estimateur SVC(gamma=0.5, random_state=42)

Accuracy : 0.968
Classification Report
              precision    recall  f1-score   support

       OTHER    0.97500   0.95122   0.96296        41
  TRUE/FALSE    0.96296   0.98113   0.97196        53

    accuracy                        0.96809        94
   macro avg    0.96898   0.96618   0.96746        94
weighted avg    0.96821   0.96809   0.96804        94

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 0.5
	kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.901

meilleur estimateur SVC(gamma=0.5, random_state=42)

Accuracy : 0.968
Classification Report
              precision    recall  f1-score   support

       OTHER    0.97500   0.95122   0.96296        41
  TRUE/FALSE    0.96296   0.98113   0.97196        53

    accuracy                        0.96809        94
   macro avg    0.96898   0.96618   0.96746        94
weighted avg    0.96821   0.96809   0.96804        94

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 0.5
	kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.880
```

```
meilleur estimateur SVC(C=2, random_state=42)

Accuracy : 0.947
Classification Report
              precision    recall  f1-score   support

       OTHER    0.92857   0.95122   0.93976        41
  TRUE/FALSE    0.96154   0.94340   0.95238        53

    accuracy                        0.94681        94
   macro avg    0.94505   0.94731   0.94607        94
weighted avg    0.94716   0.94681   0.94688        94

Ensemble des meilleurs paramètres :
     C: 2
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.882

meilleur estimateur SVC(C=1, random_state=42)

Accuracy : 0.947
Classification Report
              precision    recall  f1-score   support

       OTHER    0.92857   0.95122   0.93976        41
  TRUE/FALSE    0.96154   0.94340   0.95238        53

    accuracy                        0.94681        94
   macro avg    0.94505   0.94731   0.94607        94
weighted avg    0.94716   0.94681   0.94688        94

Ensemble des meilleurs paramètres :
     C: 1
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.880

meilleur estimateur SVC(C=2, random_state=42)

Accuracy : 0.947
Classification Report
              precision    recall  f1-score   support
```

```
       OTHER    0.92857   0.95122   0.93976        41
  TRUE/FALSE    0.96154   0.94340   0.95238        53

    accuracy                        0.94681        94
   macro avg    0.94505   0.94731   0.94607        94
weighted avg    0.94716   0.94681   0.94688        94

Ensemble des meilleurs paramètres :
      C: 2
      gamma: 'scale'
      kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.880

meilleur estimateur SVC(C=1, random_state=42)

Accuracy : 0.947
Classification Report
              precision    recall  f1-score   support

       OTHER    0.92857   0.95122   0.93976        41
  TRUE/FALSE    0.96154   0.94340   0.95238        53

    accuracy                        0.94681        94
   macro avg    0.94505   0.94731   0.94607        94
weighted avg    0.94716   0.94681   0.94688        94

Ensemble des meilleurs paramètres :
      C: 1
      gamma: 'scale'
      kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 20 candidates, totalling 100 fits

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/
_validation.py:378: FitFailedWarning:
10 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
----------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
```

```
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

----------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
elasticnet penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_searc
h.py:952: UserWarning: One or more of the test scores are non-finite:
[       nan 0.82623423        nan 0.7914955  0.7514955  0.80486486
 0.82086486 0.82623423 0.8129009  0.8049009  0.82623423 0.83686486
 0.82623423 0.82623423 0.82353153 0.78079279 0.7621982  0.82623423
 0.82623423 0.82623423]
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(

meilleur score 0.837

meilleur estimateur LogisticRegression(fit_intercept=False,
random_state=42)

Accuracy : 0.851
Classification Report
              precision    recall  f1-score   support

       OTHER    0.76471   0.95122   0.84783        41
  TRUE/FALSE    0.95349   0.77358   0.85417        53

    accuracy                        0.85106        94
   macro avg    0.85910   0.86240   0.85100        94
weighted avg    0.87115   0.85106   0.85140        94

Ensemble des meilleurs paramètres :
     penalty: 'l2'
     C: 1.0
     fit_intercept: False
     solver: 'lbfgs'
     max_iter: 100
grid search fait
Fitting 5 folds for each of 20 candidates, totalling 100 fits

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/
_validation.py:378: FitFailedWarning:
10 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
---------------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
```

```
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

----------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
elasticnet penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_searc
h.py:952: UserWarning: One or more of the test scores are non-finite:
[       nan 0.80738739        nan 0.81834234 0.74882883 0.80479279
 0.81279279 0.80738739 0.79142342 0.79409009 0.80738739 0.8181982
 0.80738739 0.80738739 0.80209009 0.7649009  0.73016216 0.80738739
 0.80738739 0.80738739]
  warnings.warn(

meilleur score 0.818

meilleur estimateur LogisticRegression(penalty='none',
random_state=42)

Accuracy : 0.862
Classification Report
              precision    recall  f1-score   support

       OTHER    0.78000   0.95122   0.85714        41
  TRUE/FALSE    0.95455   0.79245   0.86598        53

    accuracy                        0.86170        94
   macro avg    0.86727   0.87184   0.86156        94
weighted avg    0.87841   0.86170   0.86213        94
```

```
Ensemble des meilleurs paramètres :
     penalty: 'none'
     C: 1.0
     fit_intercept: True
     solver: 'lbfgs'
     max_iter: 100
grid search fait
Fitting 5 folds for each of 20 candidates, totalling 100 fits

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/
_validation.py:378: FitFailedWarning:
10 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
----------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

----------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
```

```
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
elasticnet penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_searc
h.py:952: UserWarning: One or more of the test scores are non-finite:
[       nan 0.81816216        nan 0.8074955  0.75135135 0.81545946
 0.8181982  0.81816216 0.8154955  0.81545946 0.81816216 0.81812613
 0.81816216 0.81816216 0.81823423 0.8181982  0.8074955  0.81816216
 0.81816216 0.81816216]
  warnings.warn(

meilleur score 0.818

meilleur estimateur LogisticRegression(random_state=42,
solver='liblinear')

Accuracy : 0.851
Classification Report
              precision    recall  f1-score   support

       OTHER    0.76471   0.95122   0.84783        41
  TRUE/FALSE    0.95349   0.77358   0.85417        53

    accuracy                        0.85106        94
   macro avg    0.85910   0.86240   0.85100        94
weighted avg    0.87115   0.85106   0.85140        94

Ensemble des meilleurs paramètres :
     penalty: 'l2'
     C: 1.0
     fit_intercept: True
     solver: 'liblinear'
     max_iter: 100
grid search fait
Fitting 5 folds for each of 20 candidates, totalling 100 fits

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/
_validation.py:378: FitFailedWarning:
10 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
----------------------------------------------------------------------
```

```
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

----------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
elasticnet penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_searc
h.py:952: UserWarning: One or more of the test scores are non-finite:
[       nan 0.81005405        nan 0.81272072 0.77005405 0.80482883
 0.82075676 0.81005405 0.80479279 0.79679279 0.81005405 0.8396036
 0.81005405 0.81005405 0.81279279 0.82612613 0.81275676 0.81005405
 0.81005405 0.81005405]
  warnings.warn(

meilleur score 0.840

meilleur estimateur LogisticRegression(fit_intercept=False,
random_state=42)
```

```
Accuracy : 0.819
Classification Report
              precision    recall  f1-score   support

       OTHER    0.72222   0.95122   0.82105        41
  TRUE/FALSE    0.95000   0.71698   0.81720        53

    accuracy                        0.81915        94
   macro avg    0.83611   0.83410   0.81913        94
weighted avg    0.85065   0.81915   0.81888        94

Ensemble des meilleurs paramètres :
     penalty: 'l2'
     C: 1.0
     fit_intercept: False
     solver: 'lbfgs'
     max_iter: 100
grid search fait
Fitting 5 folds for each of 20 candidates, totalling 100 fits

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/
_validation.py:378: FitFailedWarning:
10 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
----------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

----------------------------------------------------------------------
```

```
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
elasticnet penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_searc
h.py:952: UserWarning: One or more of the test scores are non-finite:
[       nan 0.8289009         nan 0.84493694 0.51603604 0.51603604
 0.58821622 0.8289009  0.8396036  0.84493694 0.8289009  0.8261982
 0.8289009  0.8289009  0.8289009  0.8289009  0.8289009  0.8289009
 0.8289009  0.8289009 ]
  warnings.warn(

meilleur score 0.845

meilleur estimateur LogisticRegression(C=100, random_state=42)

Accuracy : 0.894
Classification Report
              precision    recall  f1-score   support

       OTHER    0.82979   0.95122   0.88636        41
  TRUE/FALSE    0.95745   0.84906   0.90000        53

    accuracy                        0.89362        94
   macro avg    0.89362   0.90014   0.89318        94
weighted avg    0.90177   0.89362   0.89405        94

Ensemble des meilleurs paramètres :
      penalty: 'l2'
      C: 100
      fit_intercept: True
      solver: 'lbfgs'
      max_iter: 100
grid search fait
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/
_validation.py:378: FitFailedWarning:
10 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

--------------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
elasticnet penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_searc
h.py:952: UserWarning: One or more of the test scores are non-finite:
```

```
[       nan 0.84227027        nan 0.85827027 0.51603604 0.51603604
 0.60691892 0.84227027 0.86627027 0.8636036  0.84227027 0.81012613
 0.84227027 0.84227027 0.83956757 0.84227027 0.84227027 0.84227027
 0.84227027 0.84227027]
  warnings.warn(

meilleur score 0.866

meilleur estimateur LogisticRegression(C=10, random_state=42)

Accuracy : 0.894
Classification Report
              precision    recall  f1-score   support

       OTHER    0.82979   0.95122   0.88636        41
  TRUE/FALSE    0.95745   0.84906   0.90000        53

    accuracy                        0.89362        94
   macro avg    0.89362   0.90014   0.89318        94
weighted avg    0.90177   0.89362   0.89405        94

Ensemble des meilleurs paramètres :
     penalty: 'l2'
     C: 10
     fit_intercept: True
     solver: 'lbfgs'
     max_iter: 100
grid search fait
Fitting 5 folds for each of 20 candidates, totalling 100 fits

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/
_validation.py:378: FitFailedWarning:
10 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
----------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
```

```
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

----------------------------------------------------------------------
----------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
elasticnet penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_searc
h.py:952: UserWarning: One or more of the test scores are non-finite:
[       nan 0.85027027        nan 0.86356757 0.51603604 0.51603604
 0.62032432 0.85027027 0.87700901 0.88234234 0.85027027 0.81538739
 0.85027027 0.85027027 0.85027027 0.85027027 0.85027027 0.85027027
 0.85027027 0.85027027]
  warnings.warn(

meilleur score 0.882

meilleur estimateur LogisticRegression(C=100, random_state=42)

Accuracy : 0.862
Classification Report
              precision    recall  f1-score   support

       OTHER    0.78000   0.95122   0.85714        41
  TRUE/FALSE    0.95455   0.79245   0.86598        53

    accuracy                        0.86170        94
   macro avg    0.86727   0.87184   0.86156        94
weighted avg    0.87841   0.86170   0.86213        94
```

```
Ensemble des meilleurs paramètres :
      penalty: 'l2'
      C: 100
      fit_intercept: True
      solver: 'lbfgs'
      max_iter: 100
grid search fait
Fitting 5 folds for each of 20 candidates, totalling 100 fits

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/
_validation.py:378: FitFailedWarning:
10 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
```

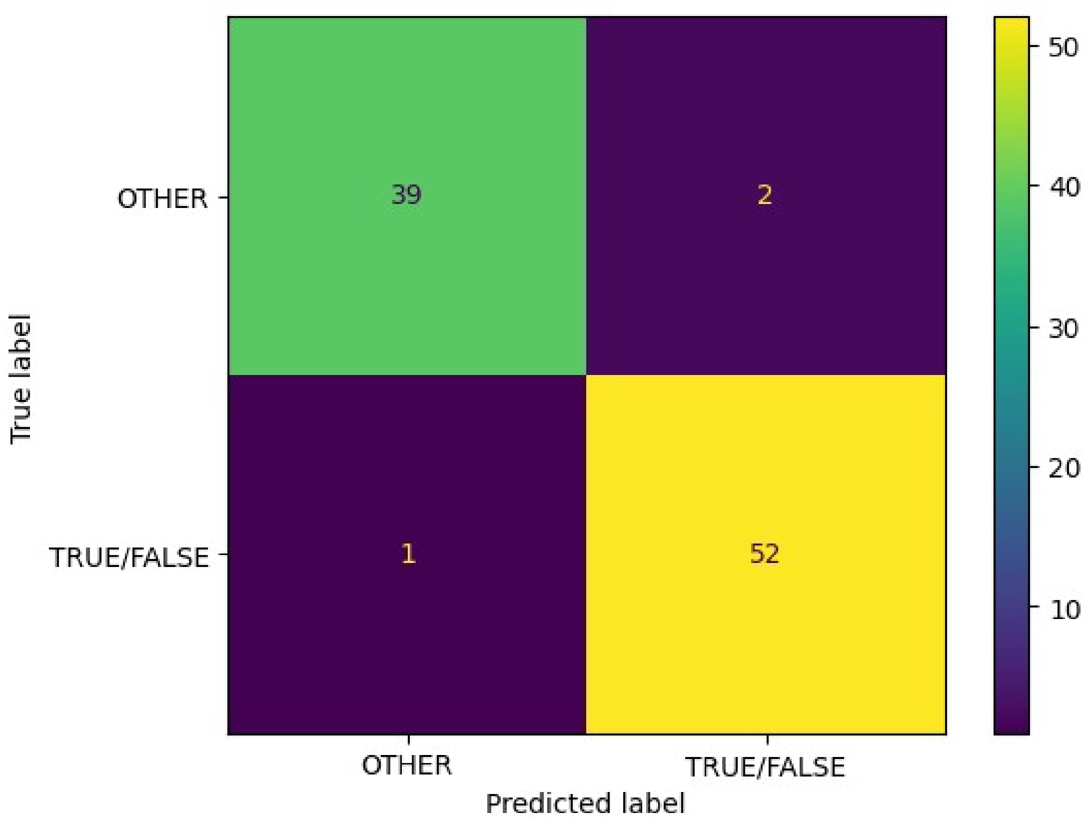
```
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
elasticnet penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_searc
h.py:952: UserWarning: One or more of the test scores are non-finite:
[       nan 0.8341982         nan 0.86371171 0.51603604 0.51603604
 0.58014414 0.8341982  0.86097297 0.86097297 0.8341982  0.8341982
 0.8341982  0.8341982  0.8341982  0.8341982  0.8341982  0.8341982
 0.8341982  0.8341982 ]
  warnings.warn(

meilleur score 0.864

meilleur estimateur LogisticRegression(penalty='none',
random_state=42)

Accuracy : 0.894
Classification Report
              precision    recall  f1-score   support

       OTHER    0.82979   0.95122   0.88636        41
  TRUE/FALSE    0.95745   0.84906   0.90000        53

    accuracy                        0.89362        94
   macro avg    0.89362   0.90014   0.89318        94
weighted avg    0.90177   0.89362   0.89405        94

Ensemble des meilleurs paramètres :
     penalty: 'none'
     C: 1.0
     fit_intercept: True
     solver: 'lbfgs'
     max_iter: 100
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.864

meilleur estimateur RandomForestClassifier(max_features='log2',
random_state=42)

Accuracy : 0.947
Classification Report
              precision    recall  f1-score   support
```

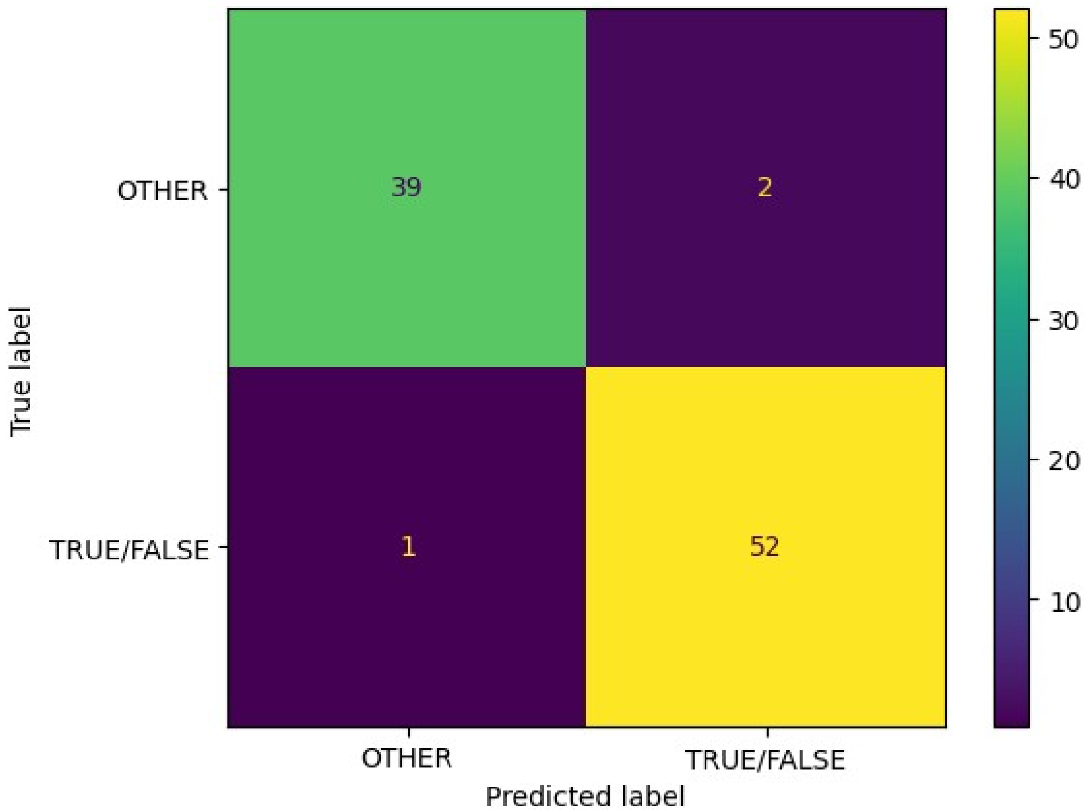
```
       OTHER    0.92857   0.95122   0.93976        41
  TRUE/FALSE    0.96154   0.94340   0.95238        53

    accuracy                        0.94681        94
   macro avg    0.94505   0.94731   0.94607        94
weighted avg    0.94716   0.94681   0.94688        94

Ensemble des meilleurs paramètres :
     n_estimators: 100
     max_features: 'log2'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.864

meilleur estimateur RandomForestClassifier(n_estimators=300,
random_state=42)

Accuracy : 0.926
Classification Report
              precision    recall  f1-score   support

       OTHER    0.88636   0.95122   0.91765        41
  TRUE/FALSE    0.96000   0.90566   0.93204        53

    accuracy                        0.92553        94
   macro avg    0.92318   0.92844   0.92484        94
weighted avg    0.92788   0.92553   0.92576        94

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.824

meilleur estimateur RandomForestClassifier(n_estimators=300,
random_state=42)

Accuracy : 0.936
Classification Report
              precision    recall  f1-score   support

       OTHER    0.87234   1.00000   0.93182        41
  TRUE/FALSE    1.00000   0.88679   0.94000        53

    accuracy                        0.93617        94
   macro avg    0.93617   0.94340   0.93591        94
weighted avg    0.94432   0.93617   0.93643        94
```

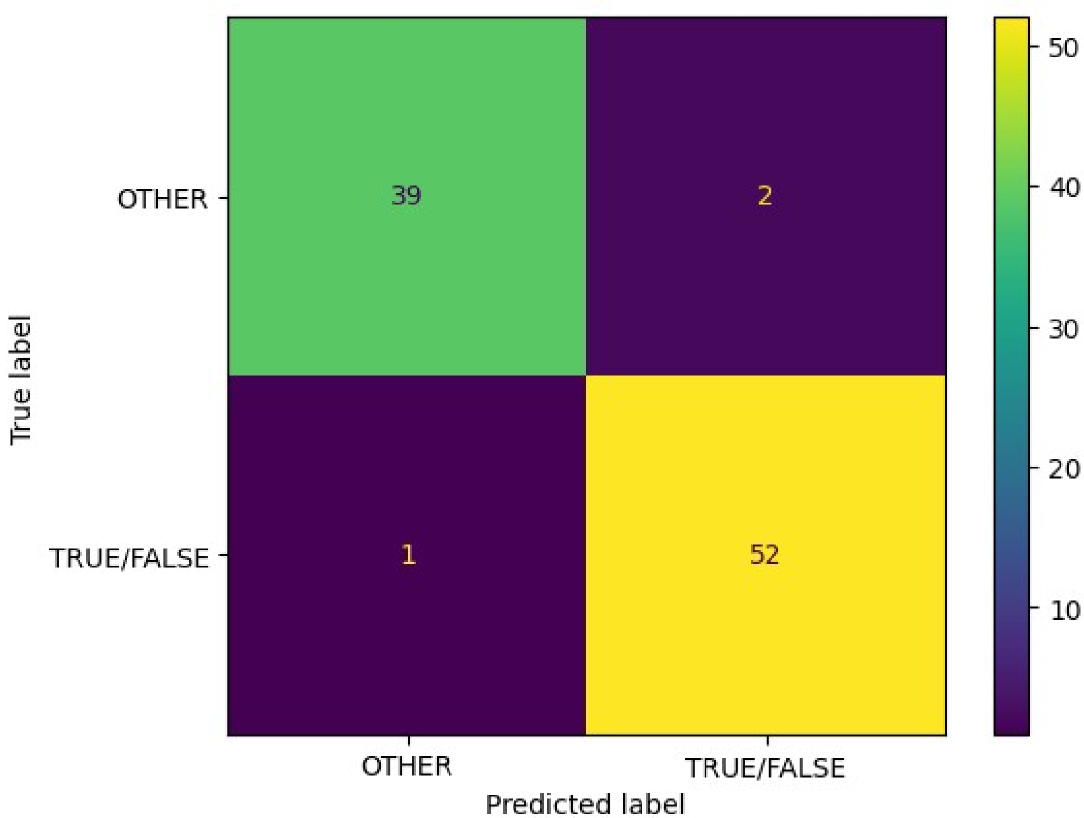
```
Ensemble des meilleurs paramètres :
      n_estimators: 300
      max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.829

meilleur estimateur RandomForestClassifier(max_features='log2',
random_state=42)

Accuracy : 0.915
Classification Report
              precision    recall  f1-score   support

       OTHER    0.86667   0.95122   0.90698        41
  TRUE/FALSE    0.95918   0.88679   0.92157        53

    accuracy                        0.91489        94
   macro avg    0.91293   0.91901   0.91427        94
weighted avg    0.91883   0.91489   0.91520        94

Ensemble des meilleurs paramètres :
      n_estimators: 100
      max_features: 'log2'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.864

meilleur estimateur RandomForestClassifier(max_features='log2',
random_state=42)

Accuracy : 0.957
Classification Report
              precision    recall  f1-score   support

       OTHER    0.95122   0.95122   0.95122        41
  TRUE/FALSE    0.96226   0.96226   0.96226        53

    accuracy                        0.95745        94
   macro avg    0.95674   0.95674   0.95674        94
weighted avg    0.95745   0.95745   0.95745        94

Ensemble des meilleurs paramètres :
      n_estimators: 100
      max_features: 'log2'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

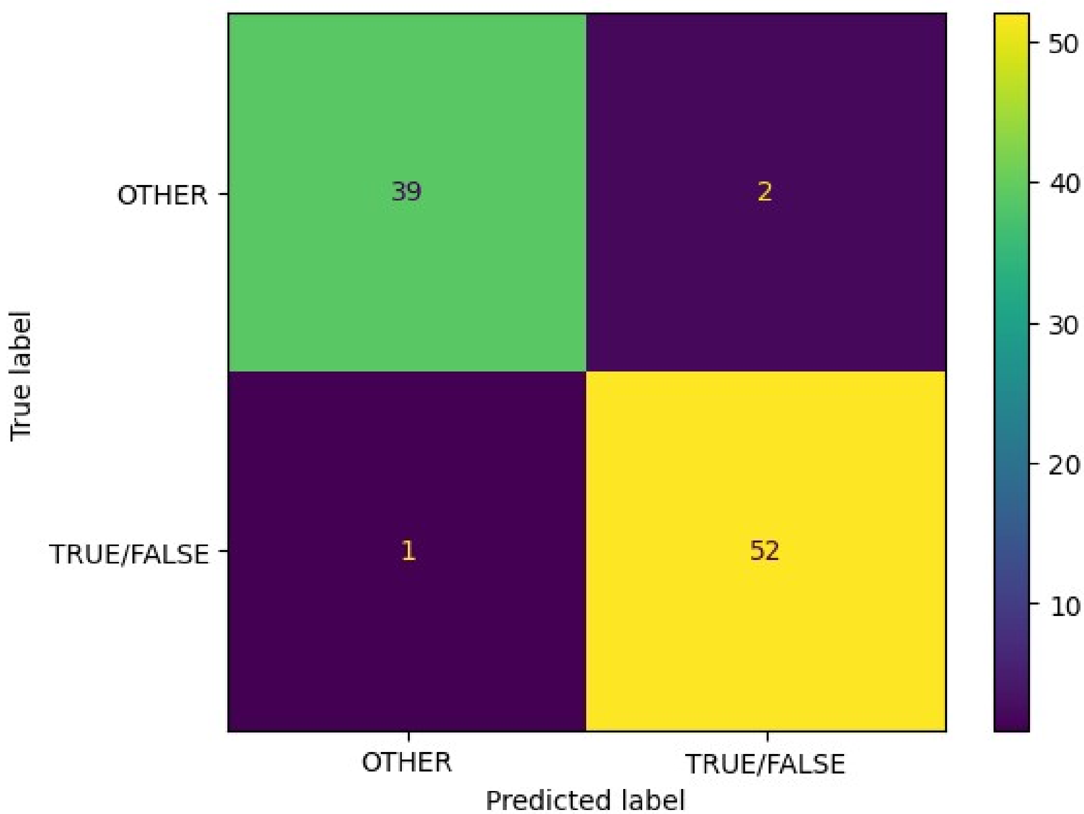
```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_plot/
confusion_matrix.py:136: RuntimeWarning: More than 20 figures have
been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and
may consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`). Consider using
`matplotlib.pyplot.close()`.
  fig, ax = plt.subplots()

meilleur score 0.853

meilleur estimateur RandomForestClassifier(n_estimators=200,
random_state=42)

Accuracy : 0.894
Classification Report
              precision    recall  f1-score   support

       OTHER    0.82979   0.95122   0.88636        41
  TRUE/FALSE    0.95745   0.84906   0.90000        53

    accuracy                        0.89362        94
   macro avg    0.89362   0.90014   0.89318        94
weighted avg    0.90177   0.89362   0.89405        94

Ensemble des meilleurs paramètres :
     n_estimators: 200
     max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.853

meilleur estimateur RandomForestClassifier(max_features='log2',
random_state=42)

Accuracy : 0.894
Classification Report
              precision    recall  f1-score   support

       OTHER    0.82979   0.95122   0.88636        41
  TRUE/FALSE    0.95745   0.84906   0.90000        53

    accuracy                        0.89362        94
   macro avg    0.89362   0.90014   0.89318        94
weighted avg    0.90177   0.89362   0.89405        94

Ensemble des meilleurs paramètres :
     n_estimators: 100
     max_features: 'log2'
```

```
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.853

meilleur estimateur RandomForestClassifier(n_estimators=200,
random_state=42)

Accuracy : 0.947
Classification Report
              precision    recall  f1-score   support

       OTHER    0.89130   1.00000   0.94253        41
  TRUE/FALSE    1.00000   0.90566   0.95050        53

    accuracy                        0.94681        94
   macro avg    0.94565   0.95283   0.94651        94
weighted avg    0.95259   0.94681   0.94702        94

Ensemble des meilleurs paramètres :
     n_estimators: 200
     max_features: 'sqrt'
```

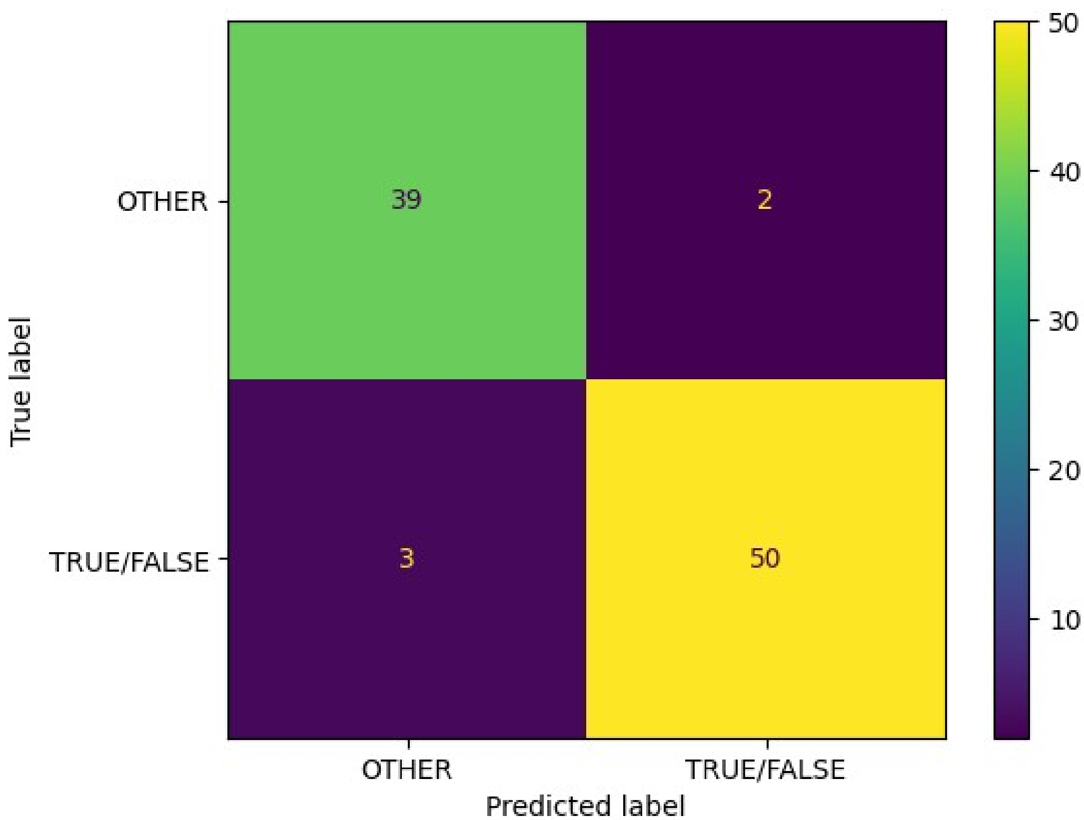



OTHER
TRUE/FALSE
True label
39
2
1
52
OTHER
TRUE/FALSE
Predicted label
50
40
30
20
10

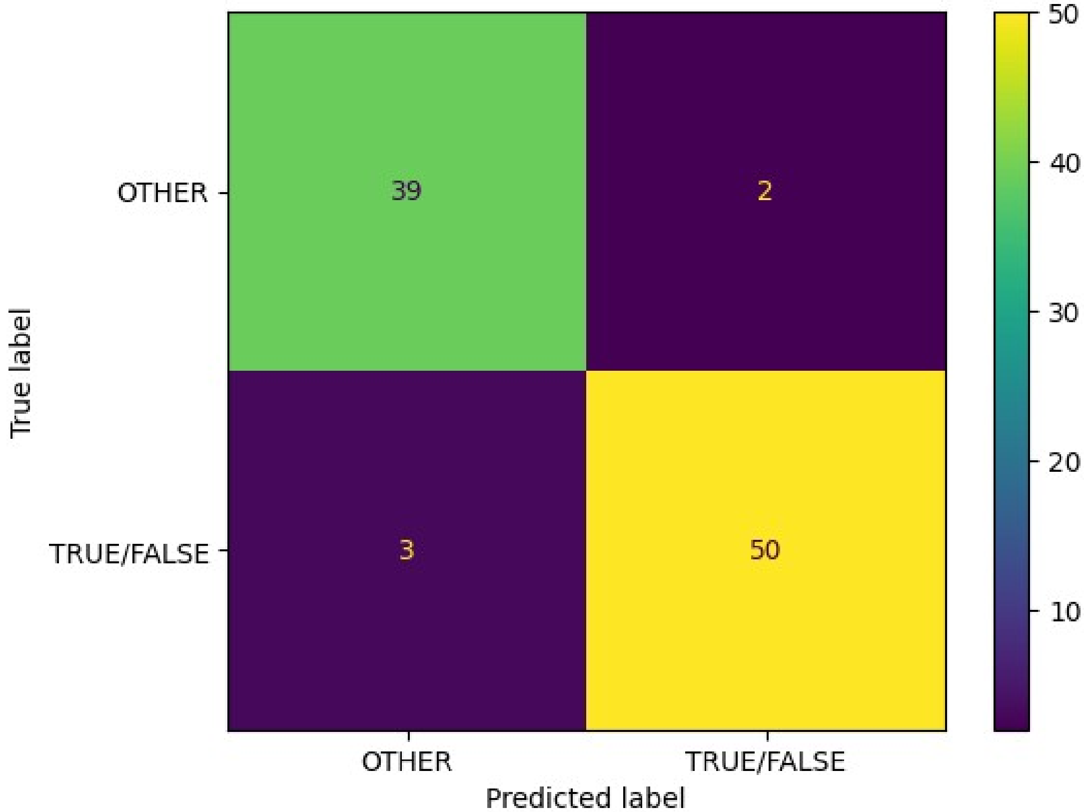

OTHER
TRUE/FALSE
True label
39
2
1
52
OTHER
TRUE/FALSE
Predicted label
50
40
30
20
10

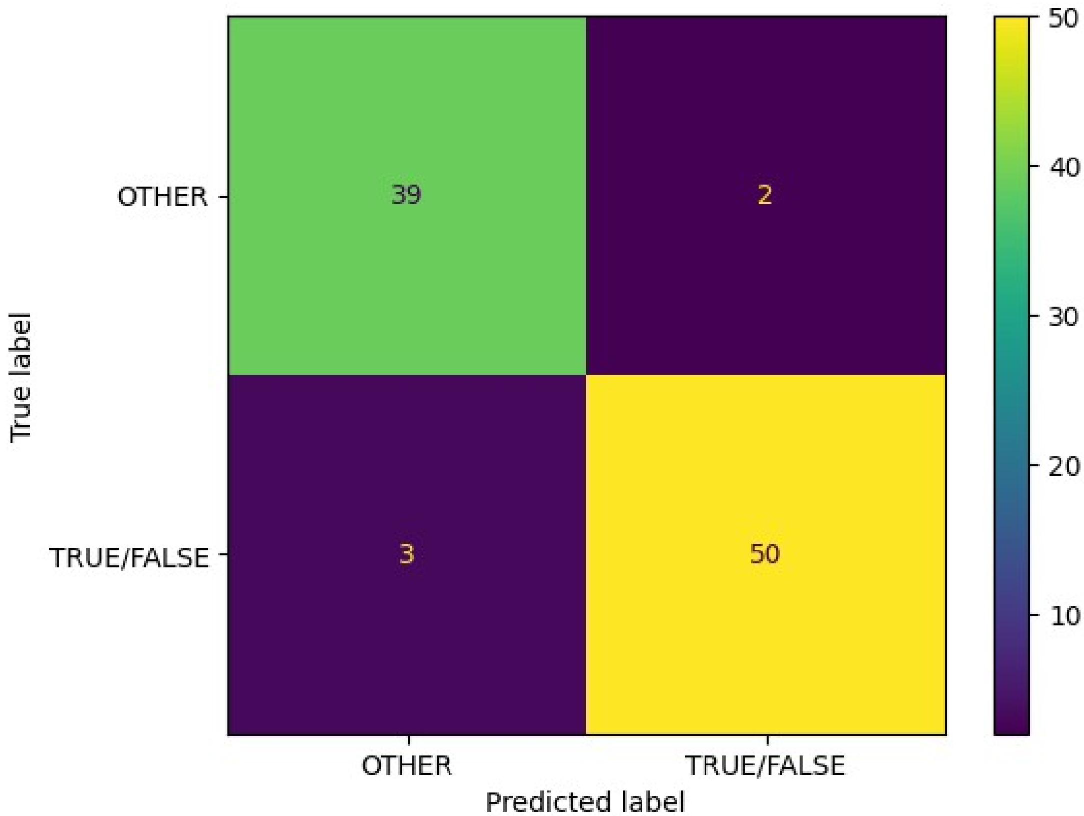


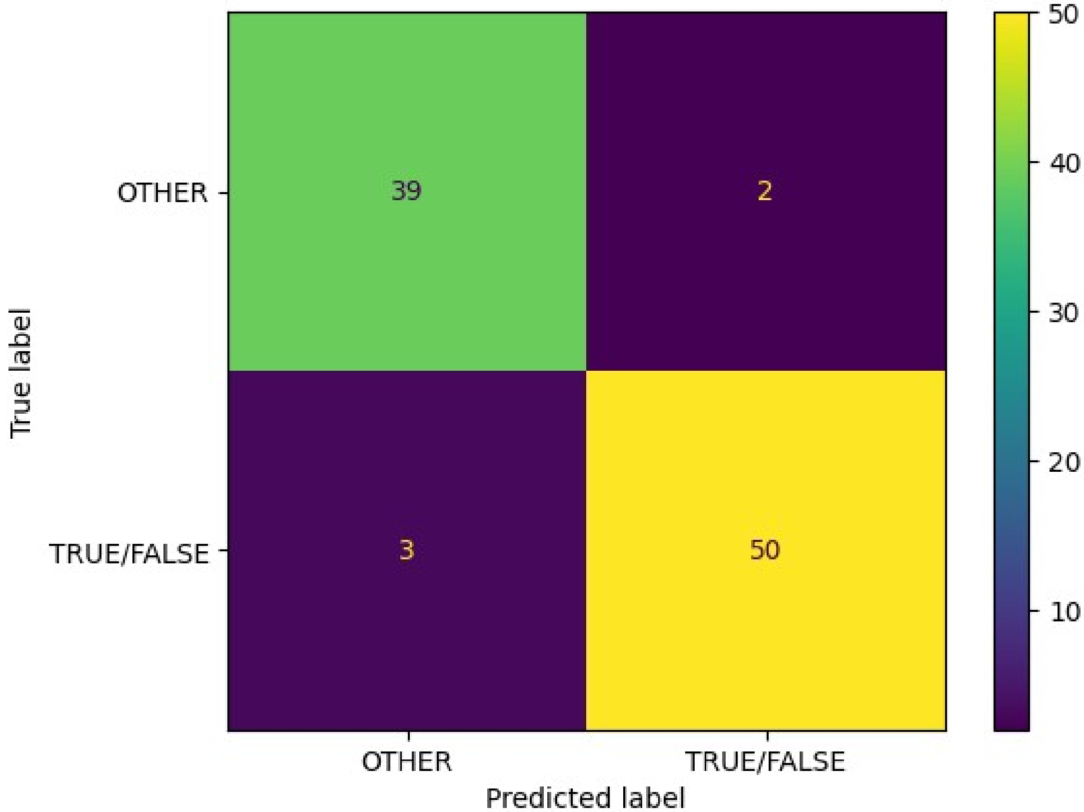
OTHER
TRUE/FALSE
True label
39
2
1
52
OTHER
TRUE/FALSE
Predicted label
50
40
30
20
10



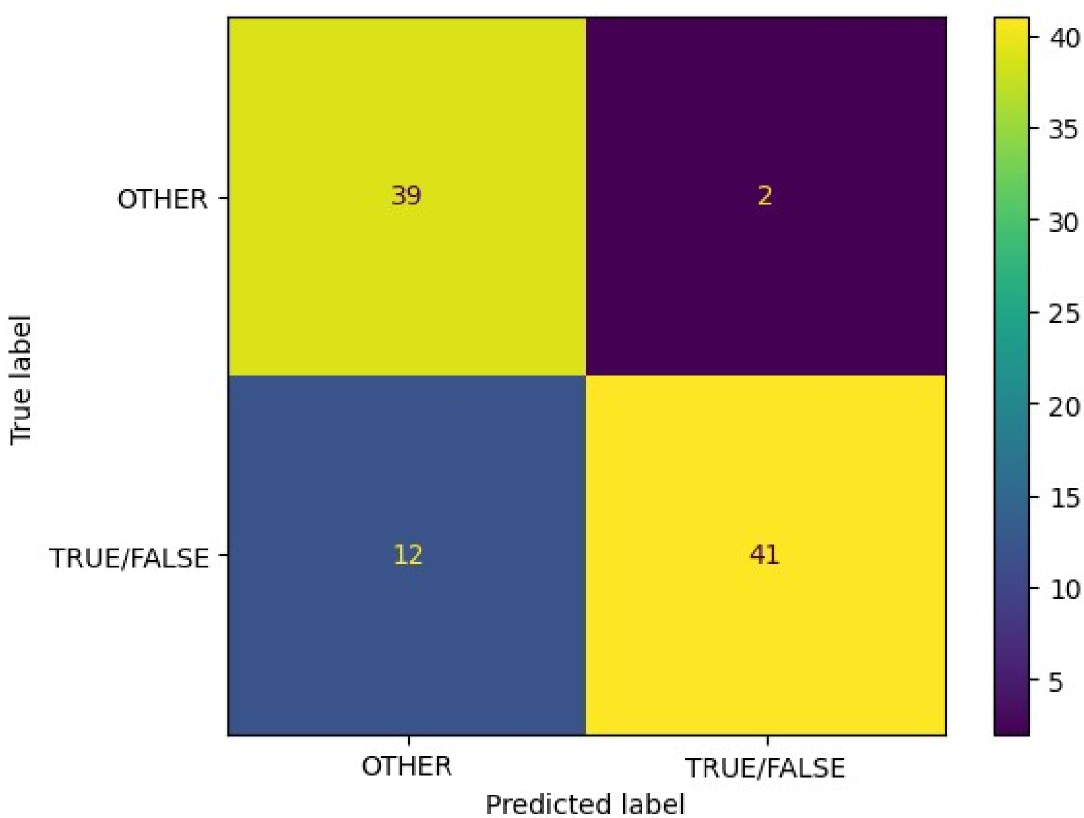
OTHER
TRUE/FALSE
True label
39
2
3
50
OTHER
TRUE/FALSE
Predicted label
50
40
30
20
10


OTHER
TRUE/FALSE
True label
Predicted label
39
2
3
50
50
40
30
20
10

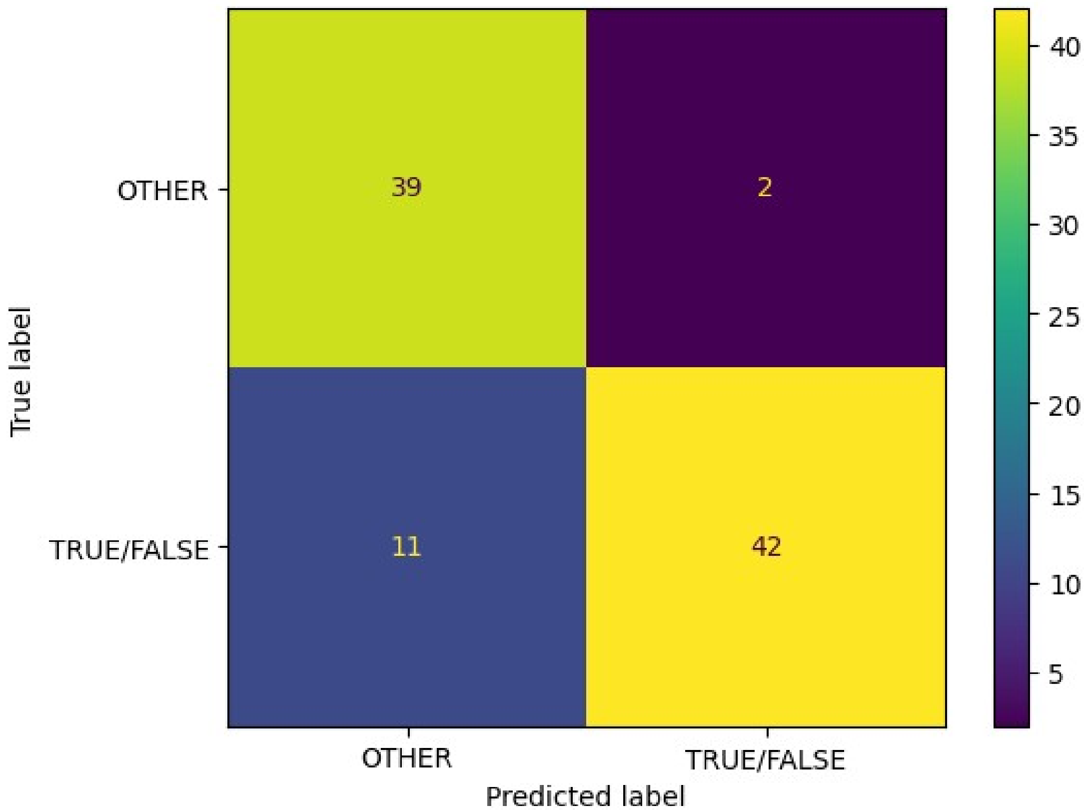

OTHER
TRUE/FALSE
True label
Predicted label
39
2
3
50
50
40
30
20
10



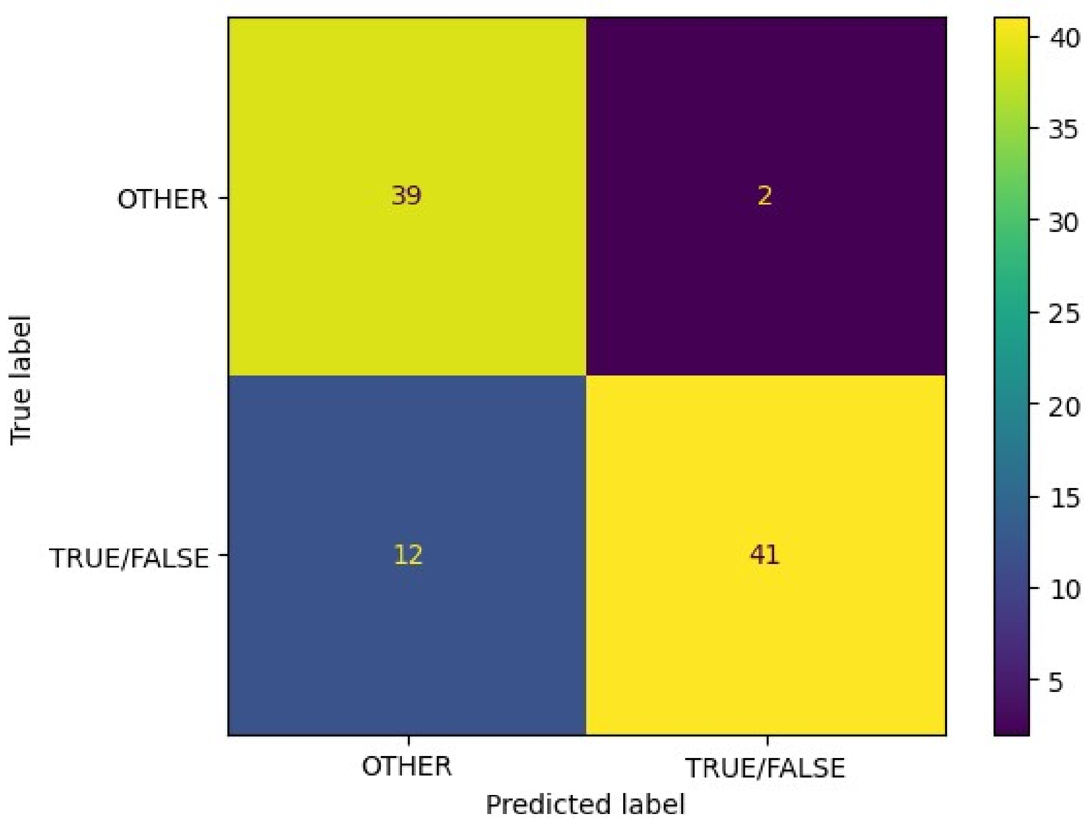
OTHER
TRUE/FALSE
39
2
3
50
OTHER
TRUE/FALSE
Predicted label
True label
50
40
30
20
10



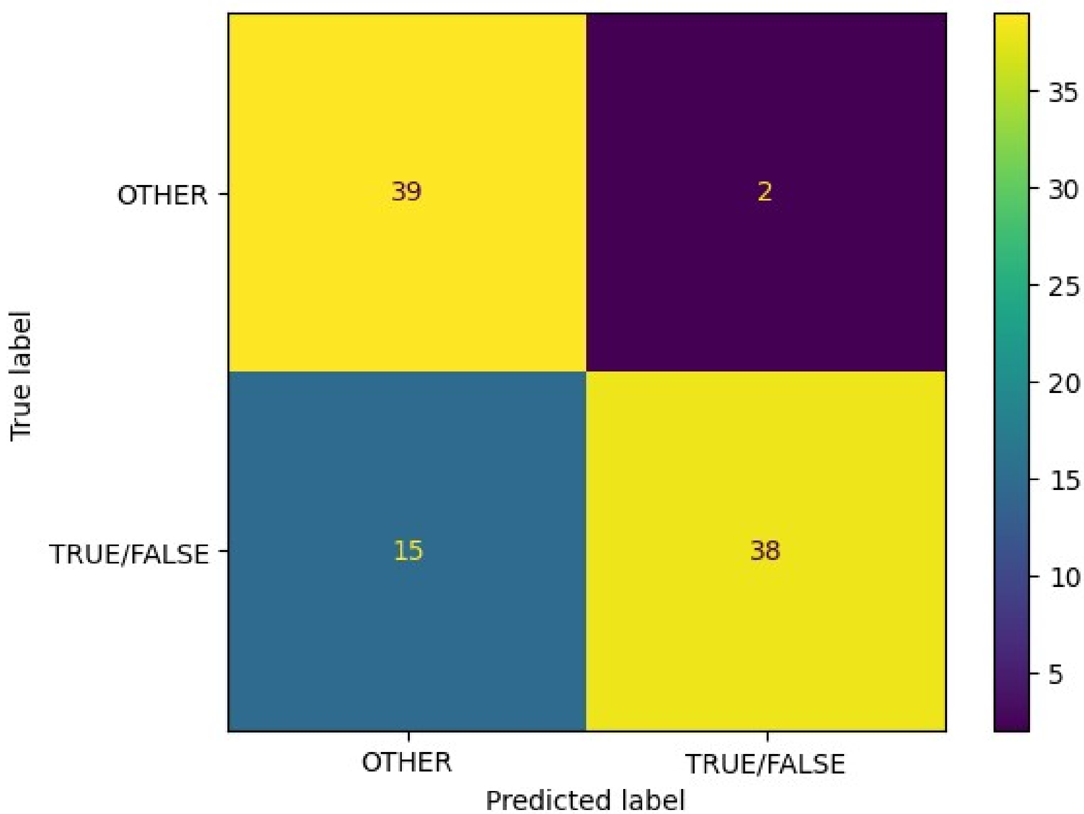
OTHER
TRUE/FALSE
39
2
12
41
OTHER
TRUE/FALSE
Predicted label
True label
40
35
30
25
20
15
10
5

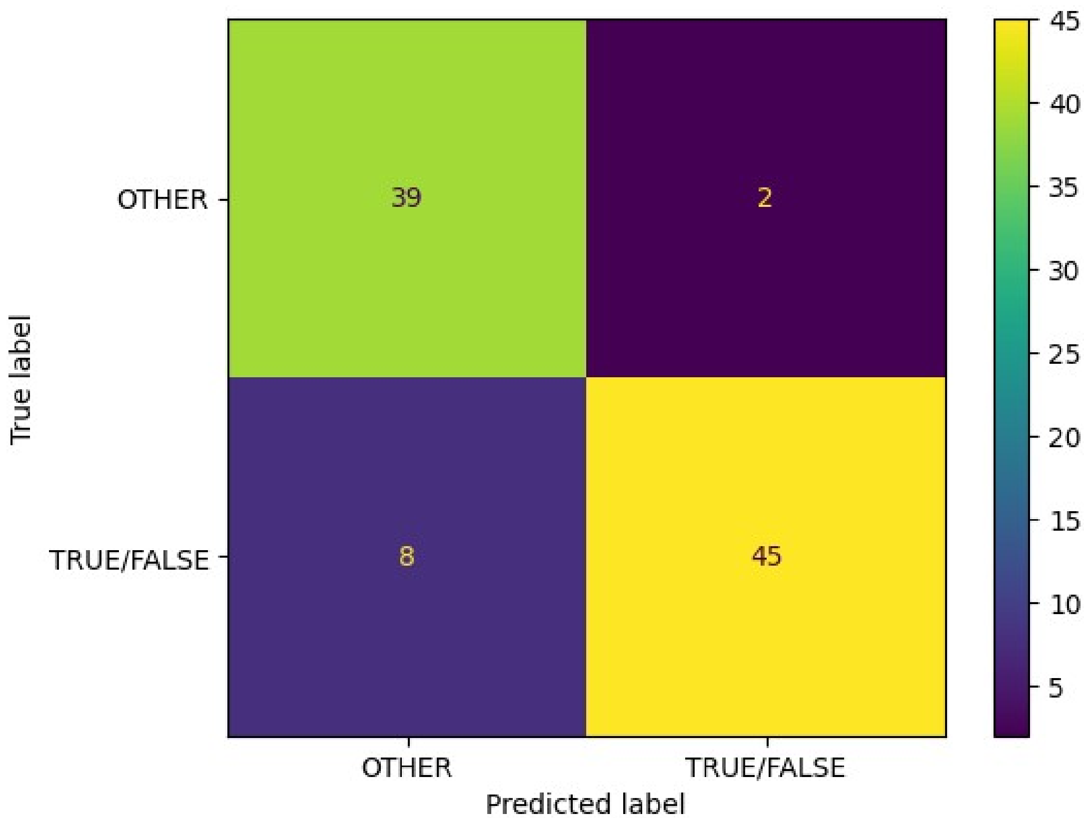


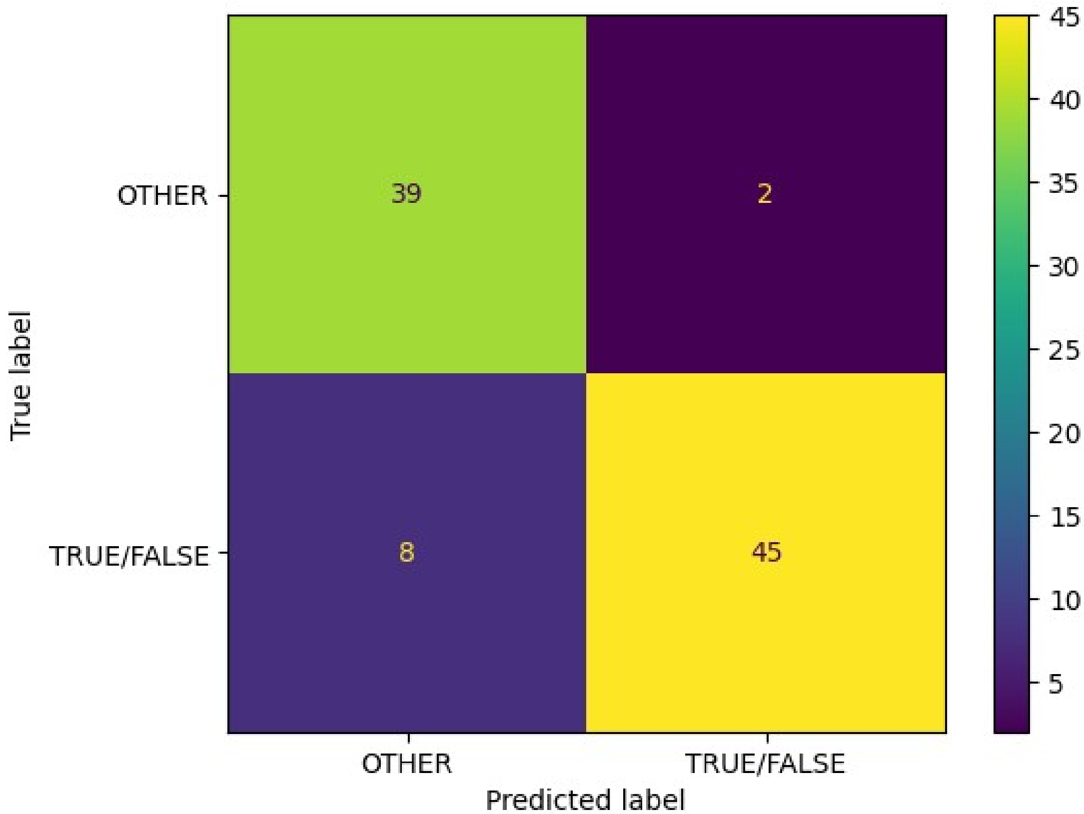
OTHER
TRUE/FALSE
True label
39
2
11
42
OTHER
TRUE/FALSE
Predicted label
5
10
15
20
25
30
35
40

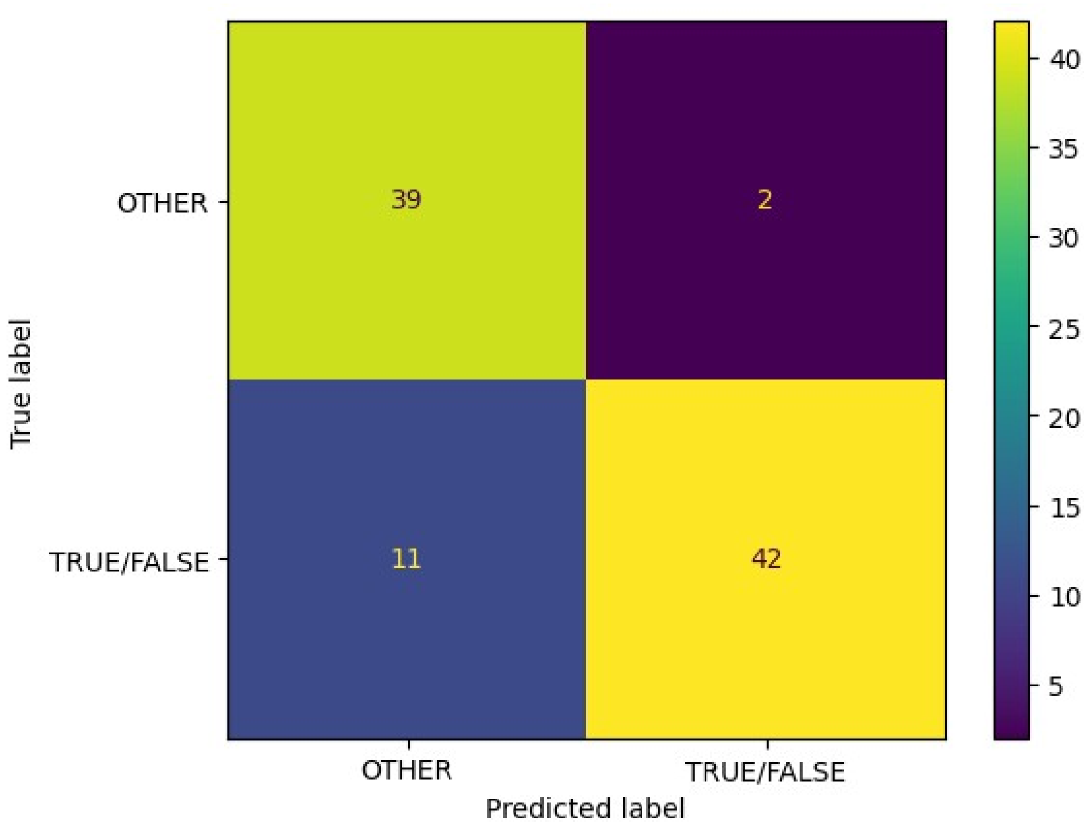


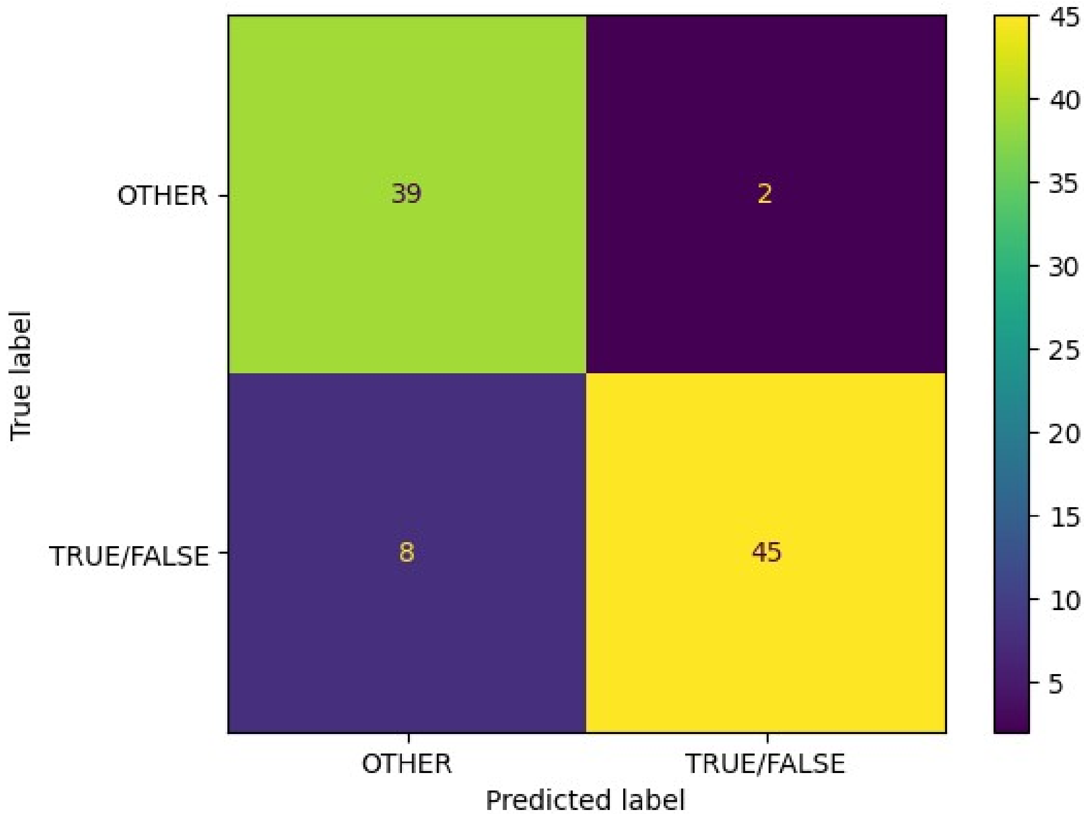
OTHER
TRUE/FALSE
True label
39
2
12
41
OTHER
TRUE/FALSE
Predicted label
5
10
15
20
25
30
35
40

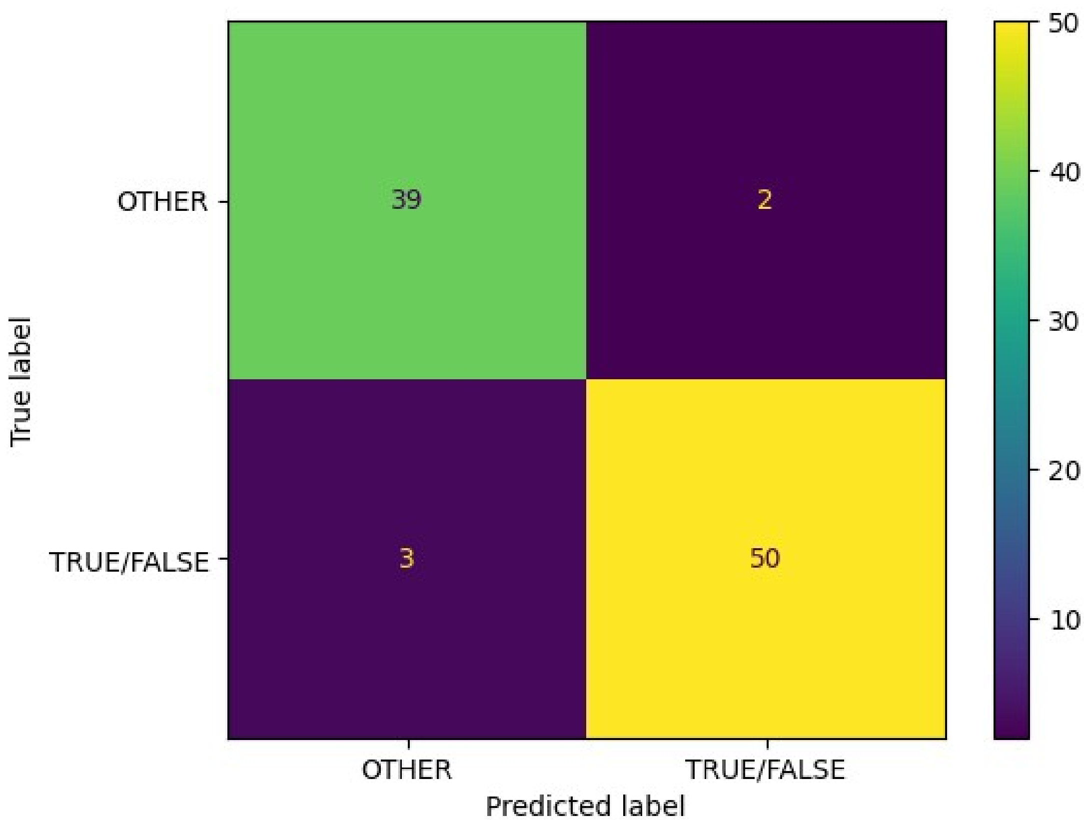

39
2
15
38
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
35
30
25
20
15
10
5

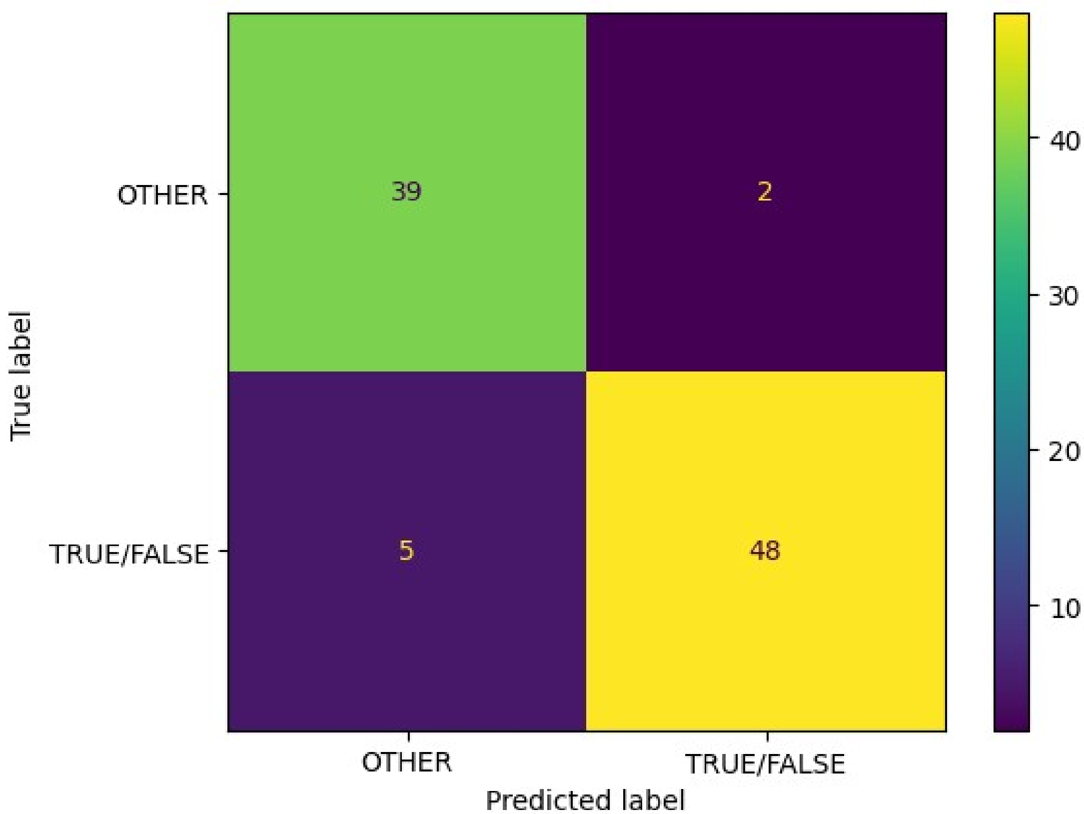

39
2
8
45
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
45
40
35
30
25
20
15
10
5


39
2
8
45
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
45
40
35
30
25
20
15
10
5

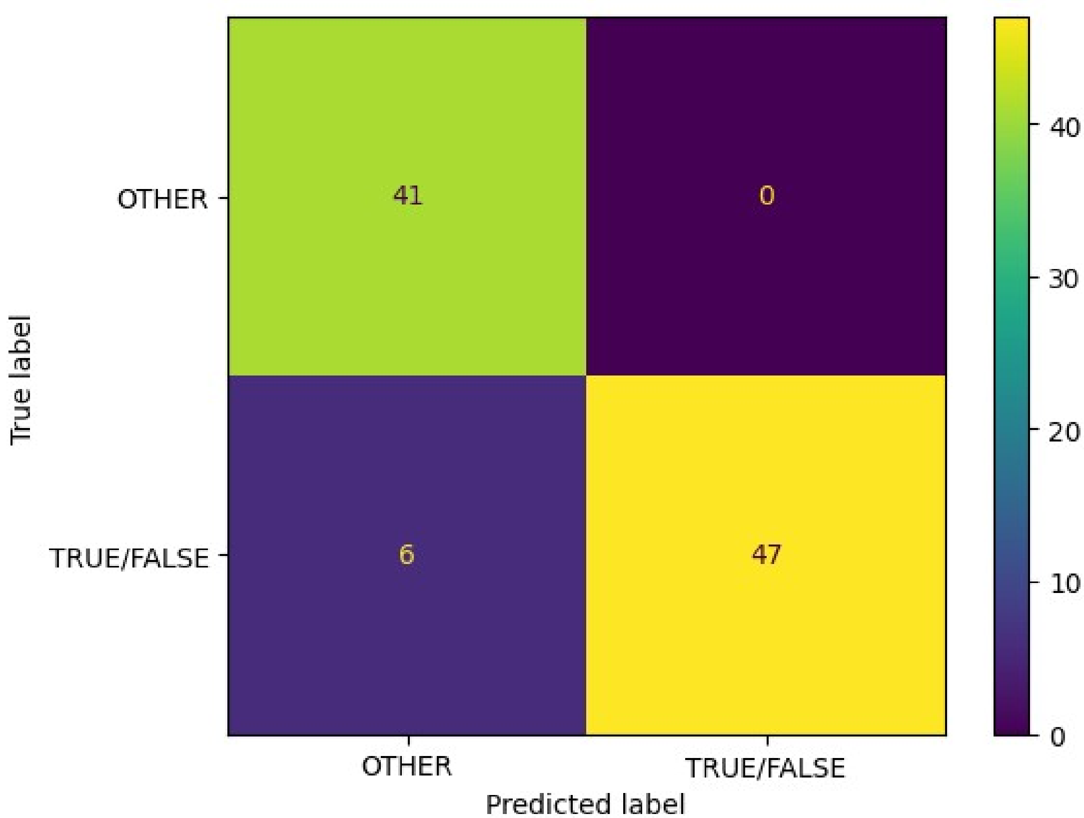

39
2
11
42
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
40
35
30
25
20
15
10
5


45
40
35
30
25
20
15
10
5
OTHER
TRUE/FALSE
True label
39
2
8
45
OTHER
TRUE/FALSE
Predicted label

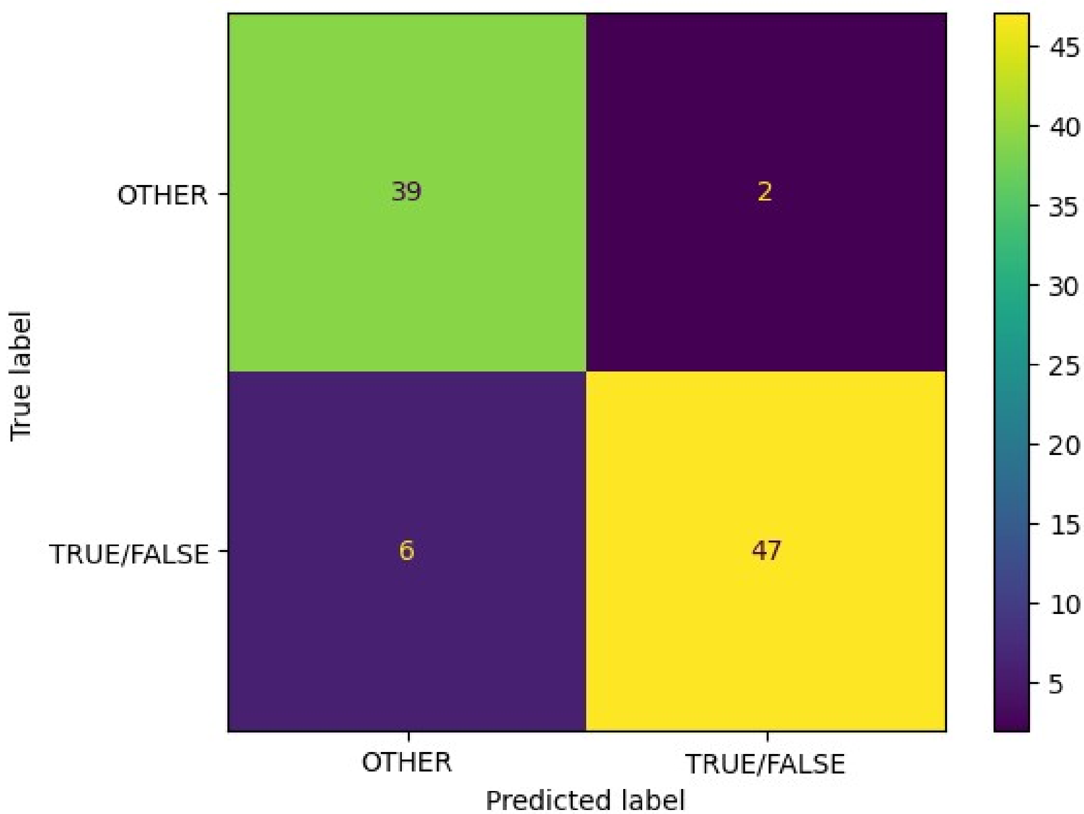

50
40
30
20
10
OTHER
TRUE/FALSE
True label
39
2
3
50
OTHER
TRUE/FALSE
Predicted label



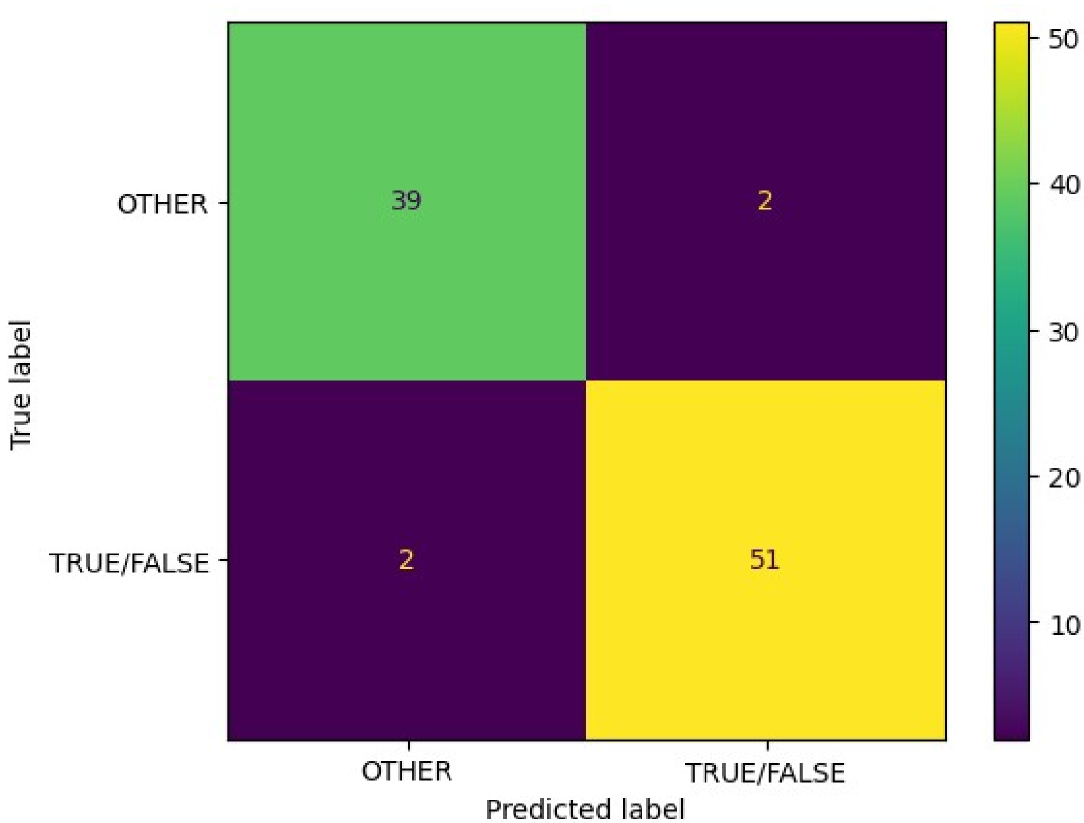
OTHER
TRUE/FALSE
True label
39
2
5
48
OTHER
TRUE/FALSE
Predicted label
40
30
20
10



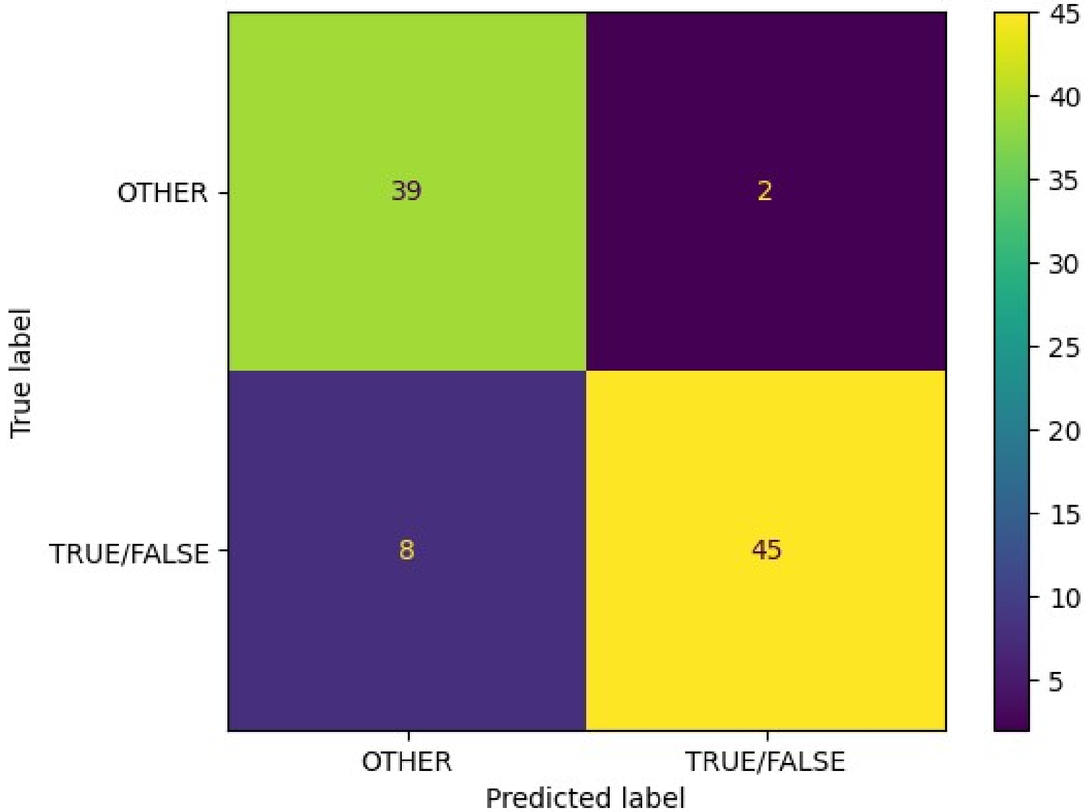
OTHER
TRUE/FALSE
True label
41
0
6
47
OTHER
TRUE/FALSE
Predicted label
40
30
20
10
0



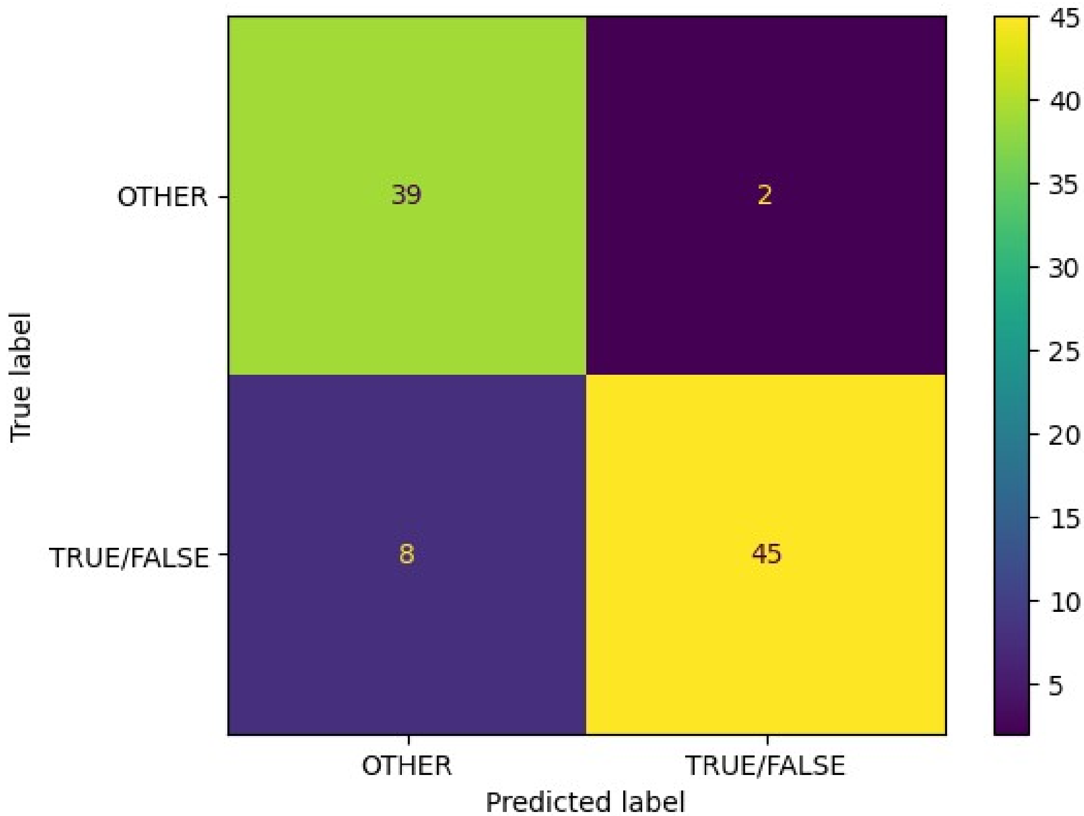
True label
OTHER
TRUE/FALSE
39
2
6
47
Predicted label
45
40
35
30
25
20
15
10
5



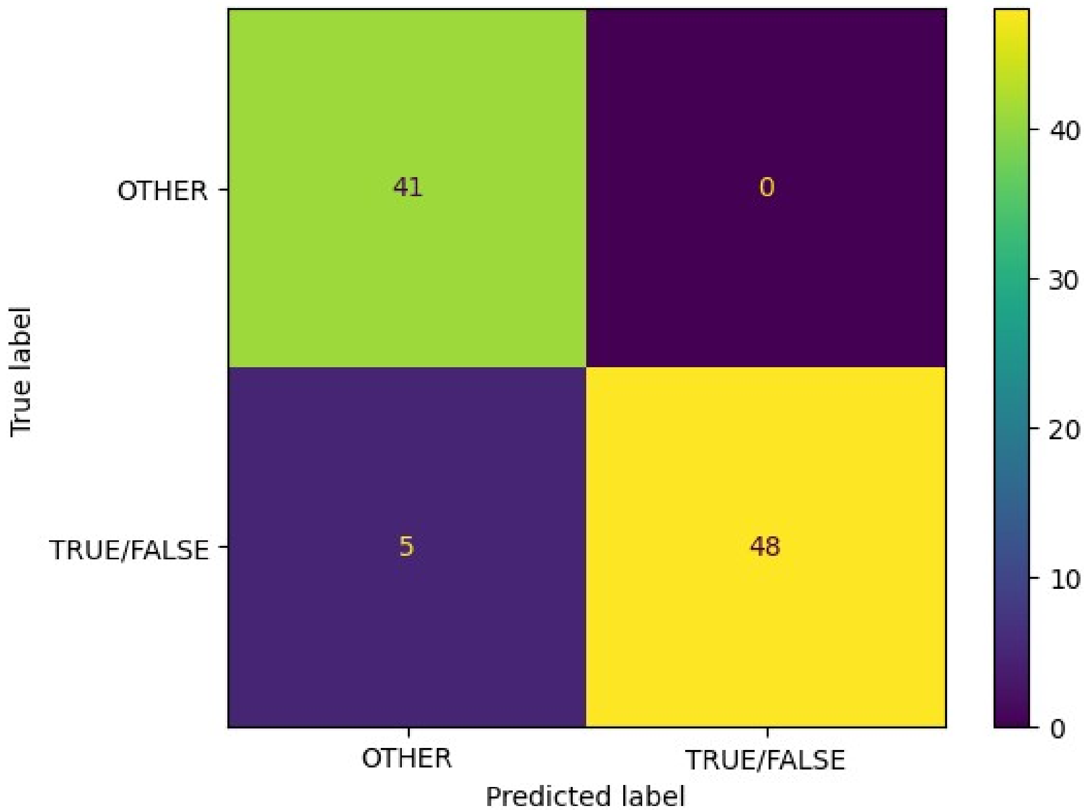
True label
OTHER
TRUE/FALSE
39
2
2
51
Predicted label
50
40
30
20
10



OTHER
TRUE/FALSE
True label
39
2
8
45
OTHER
TRUE/FALSE
Predicted label
45
40
35
30
25
20
15
10
5



OTHER
TRUE/FALSE
True label
39
2
8
45
OTHER
TRUE/FALSE
Predicted label
45
40
35
30
25
20
15
10
5



##**Etape 3 : Classification selon la colonne TITRE :**

Vu qu'on va travailler sur la colonne titre, on va séléctionner cette dernière depuis le X_train et X_test pour apprendre et tester après.

```
X_train_title = X_train['title']
X_train_title.reset_index(drop = True, inplace = True)
X_test_title = X_test['title']
X_test_title.reset_index(drop = True, inplace = True)
```

Ici, c'est une étape importante, on va tester différents classifieurs, pour chacun des classifieurs, on va appliquer le prétraitement + Vectorisation TfIdf, et on applique une cross_val_score avec un Kfold de 10 fois, par la suite on stocke dans une liste all_results la moyenne des accuracy + l'écart type et on la trie par ordre décroissant de moyenne d'accuracy et d'écart type. on remarque que les 2 meilleurs sont SVM et RF qu'on va séléctionner pour leur appliquer le GridSearch sur les paramètres des prétraitements + leurs hyperparamètres pour pouvoir choisir le meilleur.

```
np.random.seed(42)  # Set the random seed for NumPy

score = 'accuracy'
seed = 7
allresults = []
results = []
names = []
```

```python
# Liste des modèles à tester
models = [
    ('MultinomialNB', MultinomialNB()),
    ('LogisticRegression', LogisticRegression(random_state=42)),
    ('KNN', KNeighborsClassifier()),
    ('CART', DecisionTreeClassifier(random_state=42)),
    ('RF', RandomForestClassifier(random_state=42)),
    ('SVM', SVC(random_state=42))
]


# Création d'un pipeline pour chaque modèle
pipelines = []
for name,model in models:
    pipeline = Pipeline([
        ('normalize', TextNormalizer()),
        ('tfidf', TfidfVectorizer()),
        (name,model)
    ])
    pipelines.append((name,pipeline))

all_results=[]
scores=[]
names=[]
for p in pipelines:
    print(p[1])
    # cross validation en 10 fois
    kfold = KFold(n_splits=10,random_state=seed,shuffle=True)
    start_time = time.time()
    # application de la classification
    cv_results = cross_val_score(p[1],X_train_title,y_train, cv=kfold, 
scoring=score)
    scores.append(cv_results)
    names.append(p[0])
    all_results.append((p[0],cv_results.mean(),cv_results.std()))
    end_time = time.time()


all_results = sorted(all_results, key=lambda x: (-x[1], -x[2]))
print("all resultats", all_results)
```

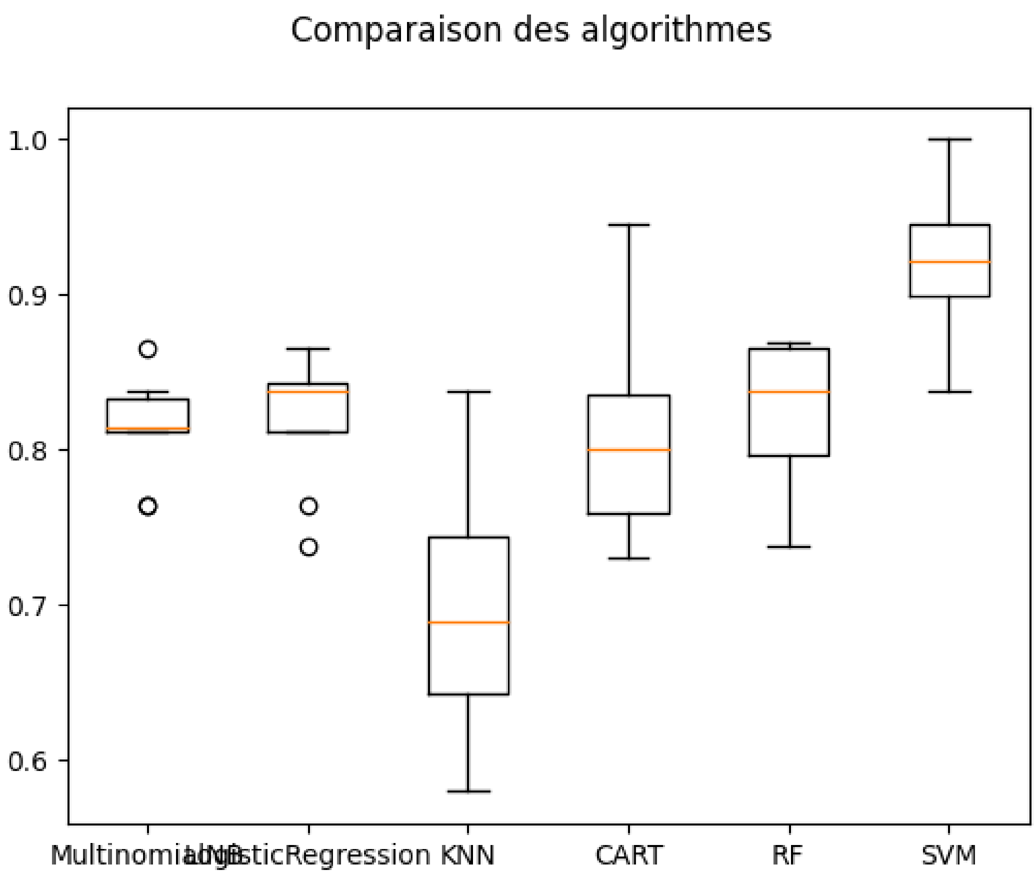
```
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf', 
TfidfVectorizer()),
                ('MultinomialNB', MultinomialNB())])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf', 
TfidfVectorizer()),
                ('LogisticRegression', 
LogisticRegression(random_state=42))])
```

```
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf', 
TfidfVectorizer()),
                ('KNN', KNeighborsClassifier())])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf', 
TfidfVectorizer()),
                ('CART', DecisionTreeClassifier(random_state=42))])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf', 
TfidfVectorizer()),
                ('RF', RandomForestClassifier(random_state=42))])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf', 
TfidfVectorizer()),
                ('SVM', SVC(random_state=42))])
all resultats [('SVM', 0.9172119487908962, 0.0469354278350516 34), 
('RF', 0.82375533428165, 0.04554453420366589), ('LogisticRegression', 
0.8211237553342817, 0.03992729512239782), ('MultinomialNB', 
0.8130867709815078, 0.029869945063979022), ('CART', 
0.8074679943100996, 0.06125324646965 2356), ('KNN', 0.6956614509246088, 
0.07157923146711843)]
```

On affiche les accuracy de chaque classifieur, on remarque la médiane (en rouge) de chaque et l'écart type aussi.

```
fig = plt.figure()
fig.suptitle('Comparaison des algorithmes')
ax = fig.add_subplot(111)
plt.boxplot(scores)
ax.set_xticklabels(names)
```

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



Choisir les meilleurs paramètres pour SVM et RF :

On a un pipeline pour chaque prétraitement différent, on essaye pas mal (miniscule, lemmatisation, miniscule + lemmatisation..) et on stocke le fit_transorm de nos X_train, X_test sur les pipelines dans des listes qui vont contenir tous les fit_transform des pipelines pour chaque classifieur, par la suite on parcourt ces listes là, on itère dessus, et chaque élement de la liste (train) va passer par le GridSearch et puis on predict sur son corresapondant dans liste (test).

```
np.random.seed(42)  # Set the random seed for NumPy

# le plus simple est de faire un test sur differents pipelines.
# pipeline de l'utilisation de CountVectorizer sur le texte avec differents pre-traitements
CV_brut = Pipeline([('cleaner', TextNormalizer()),
                    ('count_vectorizer', 
CountVectorizer(lowercase=False))])
CV_lowcase = Pipeline([('cleaner', 
TextNormalizer(removestopwords=False,lowercase=True,

getstemmer=False,removedigit=False)),
                    ('count_vectorizer', 
CountVectorizer(lowercase=False))])
```

```
CV_lowStop = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=False,removedigit=False)),
                    ('count_vectorizer',
CountVectorizer(lowercase=False))])

CV_lowStopstem = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=True,removedigit=False)),
                    ('count_vectorizer',
CountVectorizer(lowercase=False))])

# pipeline de l'utilisation de TfidfVectorizer avec differents pre-
traitements
TFIDF_brut = Pipeline ([('cleaner', TextNormalizer()),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])

TFIDF_lowcase = Pipeline([('cleaner',
TextNormalizer(removestopwords=False,lowercase=True,

getstemmer=False,removedigit=False)),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])
TFIDF_lowStop = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=False,removedigit=False)),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])

TFIDF_lowStopstem = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=True,removedigit=False)),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])


# Liste de tous les modeles à tester
all_models = [
    ("CV_brut", CV_brut),
    ("CV_lowcase", CV_lowcase),
    ("CV_lowStop", CV_lowStop),
    ("CV_lowStopstem",CV_lowStopstem),
    ("TFIDF_lowcase", TFIDF_lowcase),
    ("TFIDF_lowStop", TFIDF_lowStop),
```

```
    ("TFIDF_lowStopstem",TFIDF_lowStopstem),
    ("TFIDF_brut", TFIDF_brut)
]

X_train_title_SVC = []
X_test_title_SVC = []

X_train_title_RandomForestClassifier = []
X_test_title_RandomForestClassifier = []


for name, pipeline in all_models :

X_train_title_SVC.append(pipeline.fit_transform(X_train_title).toarray
())

X_test_title_SVC.append(pipeline.transform(X_test_title).toarray())

X_train_title_RandomForestClassifier.append(pipeline.fit_transform(X_t
rain_title).toarray())

X_test_title_RandomForestClassifier.append(pipeline.transform(X_test_t
itle).toarray())


models = {
    'SVC': SVC(random_state=42),
    'RandomForestClassifier': RandomForestClassifier(random_state=42)
}


params = {'SVC': [{'C': [0.001, 0.01, 0.1, 1,2,5,7,10]},
             {'gamma': [0.001, 0.01, 0.1,0.2,0.3,0.5,0.7,1]},
             {'kernel': ['linear', 'rbf']}],
    'RandomForestClassifier': [{'n_estimators': [10, 50, 100, 200,
300]},
                              {'max_features': ['auto', 'sqrt',
'log2']}]
}

for model_name, model in models.items():
    score='accuracy'
    X_train_title = eval('X_train_title_' + model_name)
    X_test_title = eval('X_test_title_' + model_name)
    for i in range (len(X_train_title)):
      grid_search = GridSearchCV(model, params[model_name], n_jobs=-1,
verbose=1,scoring=score)
      print("grid search fait")
      grid_search.fit(X_train_title[i],y_train)
```

```
    print ('meilleur score %0.3f'%(grid_search.best_score_),'\n')
    print ('meilleur estimateur',grid_search.best_estimator_,'\n')
    y_pred = grid_search.predict(X_test_title[i])
    MyshowAllScores(y_test,y_pred)

    print("Ensemble des meilleurs paramètres :")
    best_parameters = grid_search.best_estimator_.get_params()
    for param_dict in params[model_name]:
      for param_name, param_value in param_dict.items():
          print("\t%s: %r" % (param_name,
best_parameters[param_name]))
```

```
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.906

meilleur estimateur SVC(gamma=0.3, random_state=42)

Accuracy : 0.894
Classification Report
              precision    recall  f1-score   support

       OTHER    0.82979   0.95122   0.88636        41
  TRUE/FALSE    0.95745   0.84906   0.90000        53

    accuracy                        0.89362        94
   macro avg    0.89362   0.90014   0.89318        94
weighted avg    0.90177   0.89362   0.89405        94

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 0.3
	kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.901

meilleur estimateur SVC(gamma=0.3, random_state=42)

Accuracy : 0.904
Classification Report
              precision    recall  f1-score   support

       OTHER    0.84783   0.95122   0.89655        41
  TRUE/FALSE    0.95833   0.86792   0.91089        53

    accuracy                        0.90426        94
   macro avg    0.90308   0.90957   0.90372        94
weighted avg    0.91013   0.90426   0.90464        94
```

```
Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 0.3
	kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.898

meilleur estimateur SVC(gamma=0.7, random_state=42)

Accuracy : 0.947
Classification Report
              precision    recall  f1-score   support

       OTHER    0.92857   0.95122   0.93976        41
  TRUE/FALSE    0.96154   0.94340   0.95238        53

    accuracy                        0.94681        94
   macro avg    0.94505   0.94731   0.94607        94
weighted avg    0.94716   0.94681   0.94688        94

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 0.7
	kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.898

meilleur estimateur SVC(gamma=0.5, random_state=42)

Accuracy : 0.936
Classification Report
              precision    recall  f1-score   support

       OTHER    0.90698   0.95122   0.92857        41
  TRUE/FALSE    0.96078   0.92453   0.94231        53

    accuracy                        0.93617        94
   macro avg    0.93388   0.93787   0.93544        94
weighted avg    0.93732   0.93617   0.93632        94

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 0.5
	kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

```
meilleur score 0.904

meilleur estimateur SVC(C=1, random_state=42)

Accuracy : 0.968
Classification Report
              precision    recall  f1-score   support

       OTHER    0.97500   0.95122   0.96296        41
  TRUE/FALSE    0.96296   0.98113   0.97196        53

    accuracy                        0.96809        94
   macro avg    0.96898   0.96618   0.96746        94
weighted avg    0.96821   0.96809   0.96804        94

Ensemble des meilleurs paramètres :
     C: 1
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.901

meilleur estimateur SVC(C=2, random_state=42)

Accuracy : 0.957
Classification Report
              precision    recall  f1-score   support

       OTHER    0.95122   0.95122   0.95122        41
  TRUE/FALSE    0.96226   0.96226   0.96226        53

    accuracy                        0.95745        94
   macro avg    0.95674   0.95674   0.95674        94
weighted avg    0.95745   0.95745   0.95745        94

Ensemble des meilleurs paramètres :
     C: 2
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.898

meilleur estimateur SVC(C=1, random_state=42)

Accuracy : 0.968
Classification Report
              precision    recall  f1-score   support
```

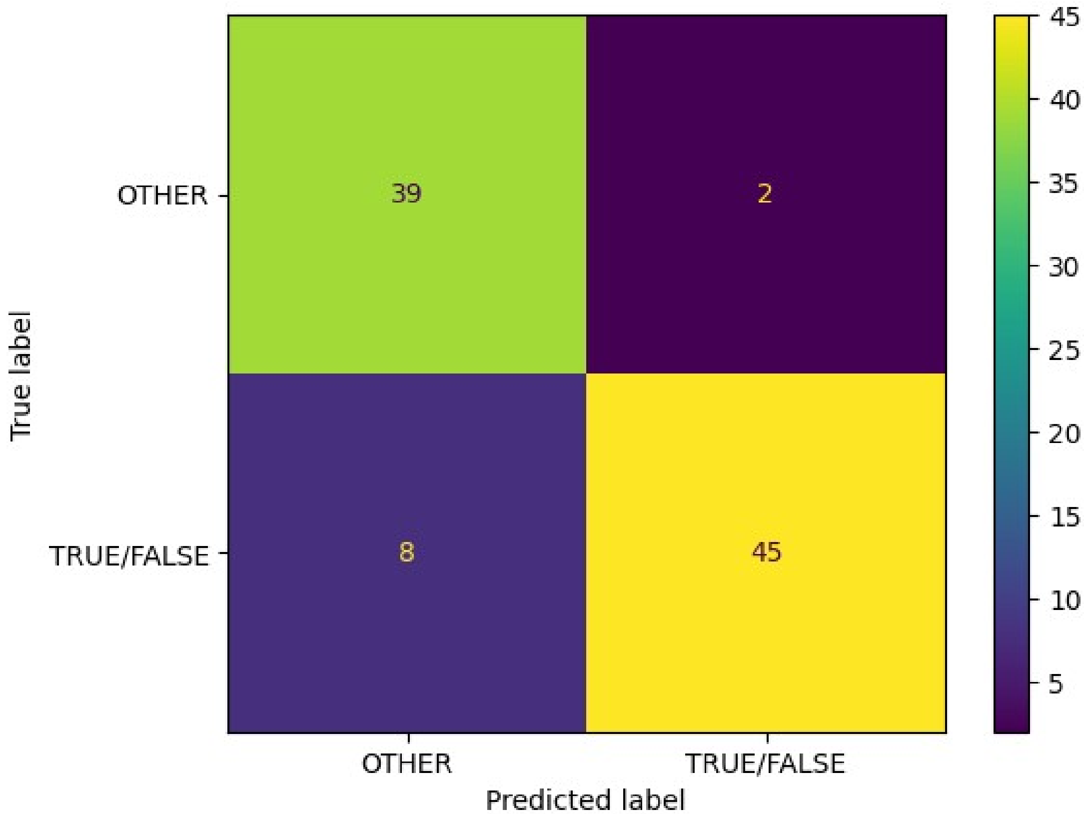
```
       OTHER    0.97500   0.95122   0.96296        41
  TRUE/FALSE    0.96296   0.98113   0.97196        53

    accuracy                        0.96809        94
   macro avg    0.96898   0.96618   0.96746        94
weighted avg    0.96821   0.96809   0.96804        94

Ensemble des meilleurs paramètres :
      C: 1
      gamma: 'scale'
      kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.904

meilleur estimateur SVC(C=1, random_state=42)

Accuracy : 0.968
Classification Report
              precision    recall  f1-score   support

       OTHER    0.97500   0.95122   0.96296        41
  TRUE/FALSE    0.96296   0.98113   0.97196        53

    accuracy                        0.96809        94
   macro avg    0.96898   0.96618   0.96746        94
weighted avg    0.96821   0.96809   0.96804        94

Ensemble des meilleurs paramètres :
      C: 1
      gamma: 'scale'
      kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.797

meilleur estimateur RandomForestClassifier(n_estimators=200,
random_state=42)

Accuracy : 0.830
Classification Report
              precision    recall  f1-score   support

       OTHER    0.73585   0.95122   0.82979        41
  TRUE/FALSE    0.95122   0.73585   0.82979        53

    accuracy                        0.82979        94
   macro avg    0.84353   0.84353   0.82979        94
```

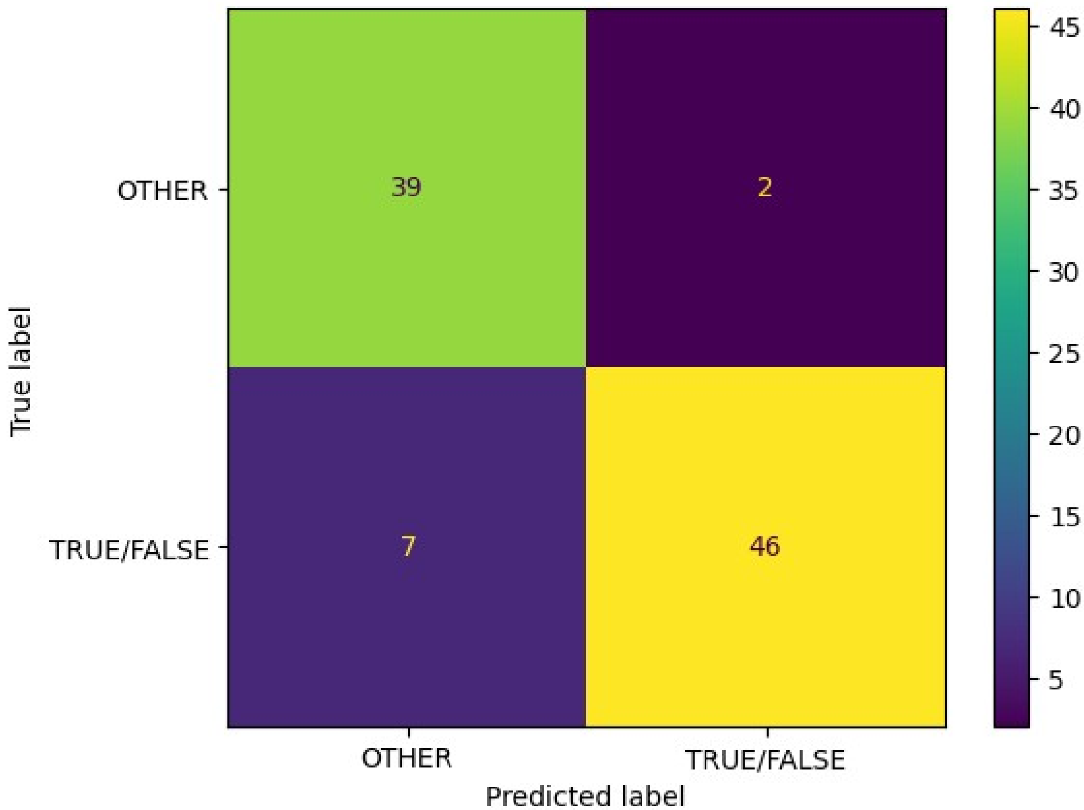
```
weighted avg    0.85728   0.82979   0.82979        94

Ensemble des meilleurs paramètres :
	n_estimators: 200
	max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.818

meilleur estimateur RandomForestClassifier(n_estimators=50,
random_state=42)

Accuracy : 0.851
Classification Report
              precision    recall  f1-score   support

       OTHER    0.74545   1.00000   0.85417        41
  TRUE/FALSE    1.00000   0.73585   0.84783        53

    accuracy                        0.85106        94
   macro avg    0.87273   0.86792   0.85100        94
weighted avg    0.88897   0.85106   0.85059        94

Ensemble des meilleurs paramètres :
	n_estimators: 50
	max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.842

meilleur estimateur RandomForestClassifier(n_estimators=50,
random_state=42)

Accuracy : 0.883
Classification Report
              precision    recall  f1-score   support

       OTHER    0.81250   0.95122   0.87640        41
  TRUE/FALSE    0.95652   0.83019   0.88889        53

    accuracy                        0.88298        94
   macro avg    0.88451   0.89070   0.88265        94
weighted avg    0.89370   0.88298   0.88344        94

Ensemble des meilleurs paramètres :
	n_estimators: 50
	max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

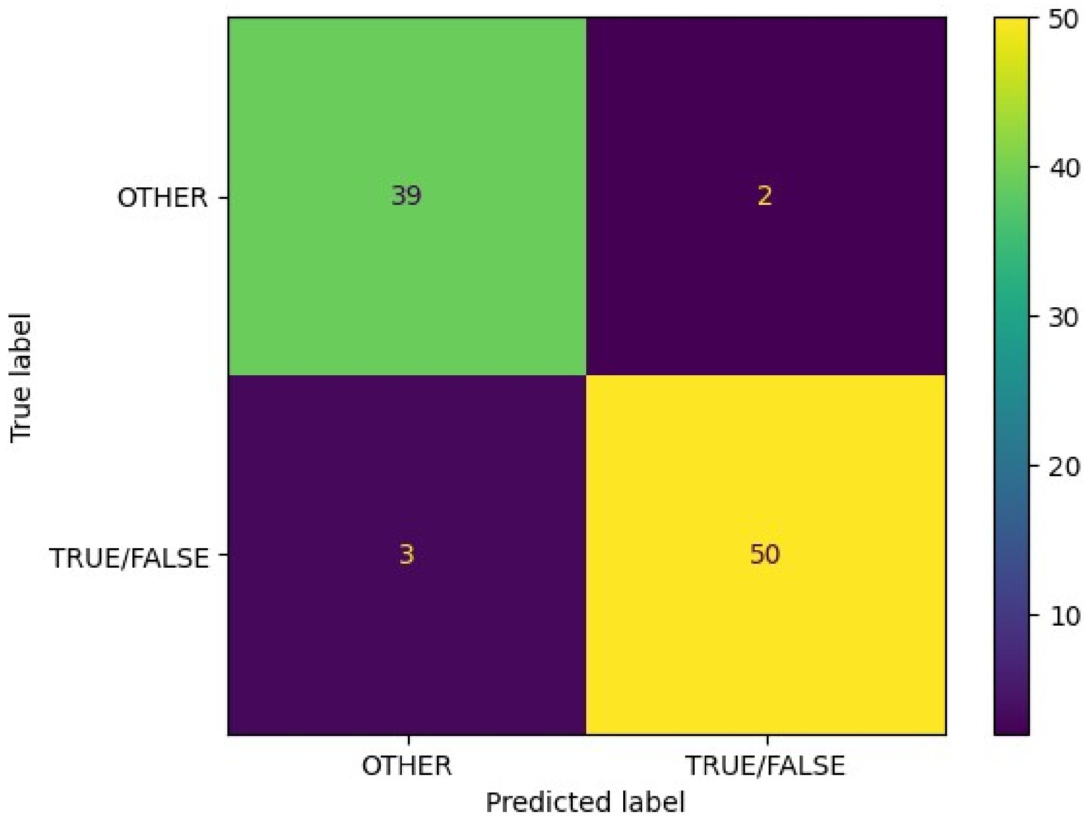
```
meilleur score 0.794

meilleur estimateur RandomForestClassifier(n_estimators=200,
random_state=42)

Accuracy : 0.872
Classification Report
              precision    recall  f1-score   support

       OTHER    0.79592   0.95122   0.86667        41
  TRUE/FALSE    0.95556   0.81132   0.87755        53

    accuracy                        0.87234        94
   macro avg    0.87574   0.88127   0.87211        94
weighted avg    0.88593   0.87234   0.87280        94

Ensemble des meilleurs paramètres :
     n_estimators: 200
     max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.818

meilleur estimateur RandomForestClassifier(n_estimators=200,
random_state=42)

Accuracy : 0.819
Classification Report
              precision    recall  f1-score   support

       OTHER    0.70690   1.00000   0.82828        41
  TRUE/FALSE    1.00000   0.67925   0.80899        53

    accuracy                        0.81915        94
   macro avg    0.85345   0.83962   0.81864        94
weighted avg    0.87216   0.81915   0.81740        94

Ensemble des meilleurs paramètres :
     n_estimators: 200
     max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.842

meilleur estimateur RandomForestClassifier(n_estimators=200,
random_state=42)

Accuracy : 0.915
Classification Report
```

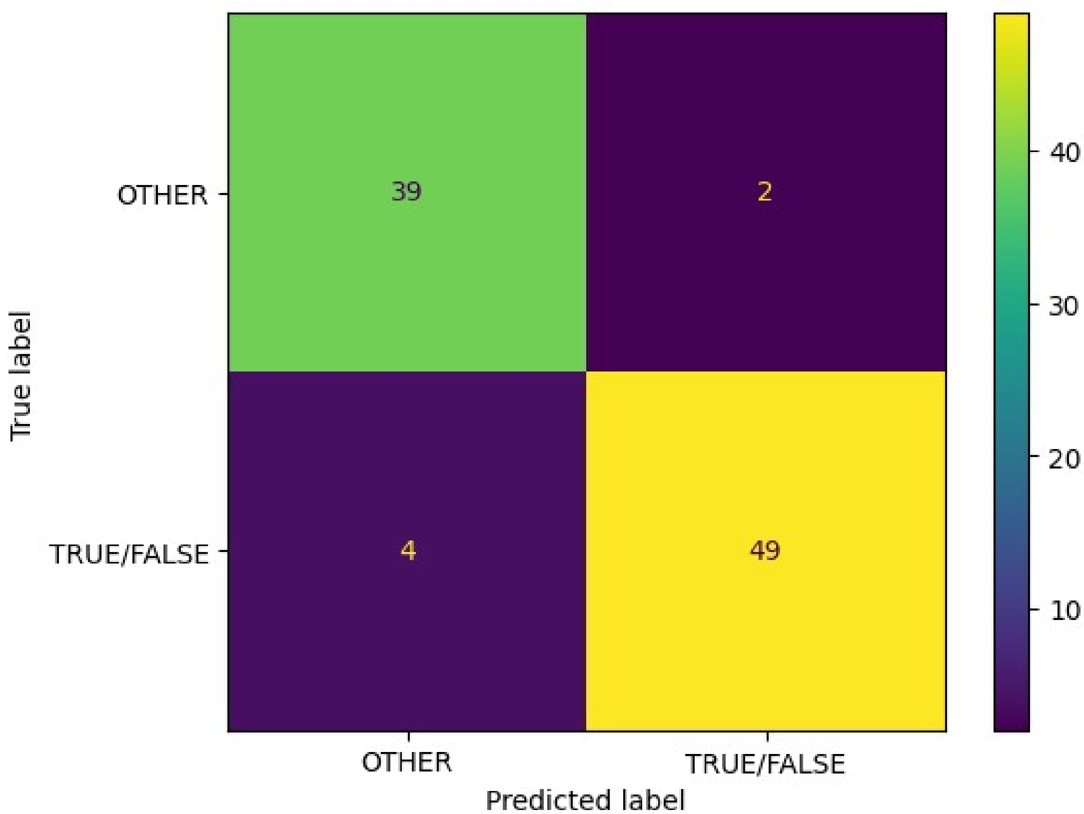
```
              precision    recall  f1-score   support

       OTHER    0.86667   0.95122   0.90698        41
  TRUE/FALSE    0.95918   0.88679   0.92157        53

    accuracy                        0.91489        94
   macro avg    0.91293   0.91901   0.91427        94
weighted avg    0.91883   0.91489   0.91520        94

Ensemble des meilleurs paramètres :
      n_estimators: 200
      max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.808

meilleur estimateur RandomForestClassifier(n_estimators=200,
random_state=42)

Accuracy : 0.915
Classification Report
              precision    recall  f1-score   support

       OTHER    0.86667   0.95122   0.90698        41
  TRUE/FALSE    0.95918   0.88679   0.92157        53

    accuracy                        0.91489        94
   macro avg    0.91293   0.91901   0.91427        94
weighted avg    0.91883   0.91489   0.91520        94

Ensemble des meilleurs paramètres :
      n_estimators: 200
      max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.856

meilleur estimateur RandomForestClassifier(n_estimators=50,
random_state=42)

Accuracy : 0.851
Classification Report
              precision    recall  f1-score   support

       OTHER    0.74545   1.00000   0.85417        41
  TRUE/FALSE    1.00000   0.73585   0.84783        53

    accuracy                        0.85106        94
   macro avg    0.87273   0.86792   0.85100        94
```

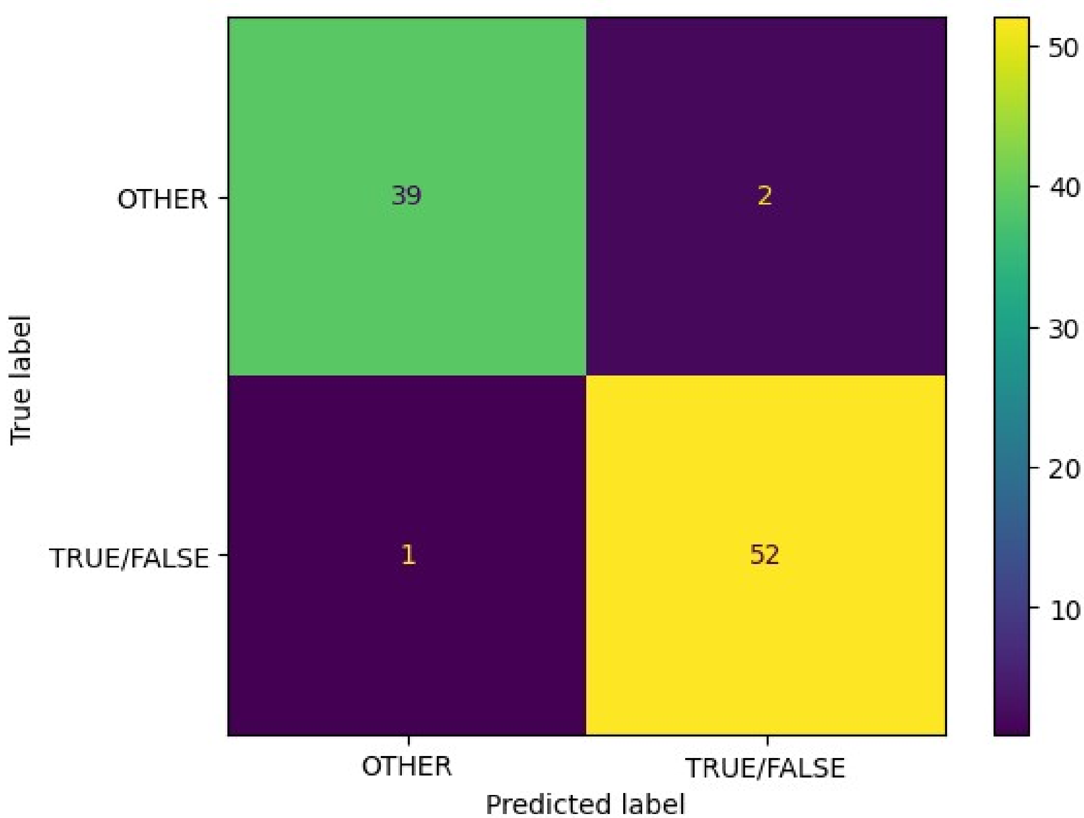
```
weighted avg    0.88897   0.85106   0.85059        94

Ensemble des meilleurs paramètres :
      n_estimators: 50
      max_features: 'sqrt'
```

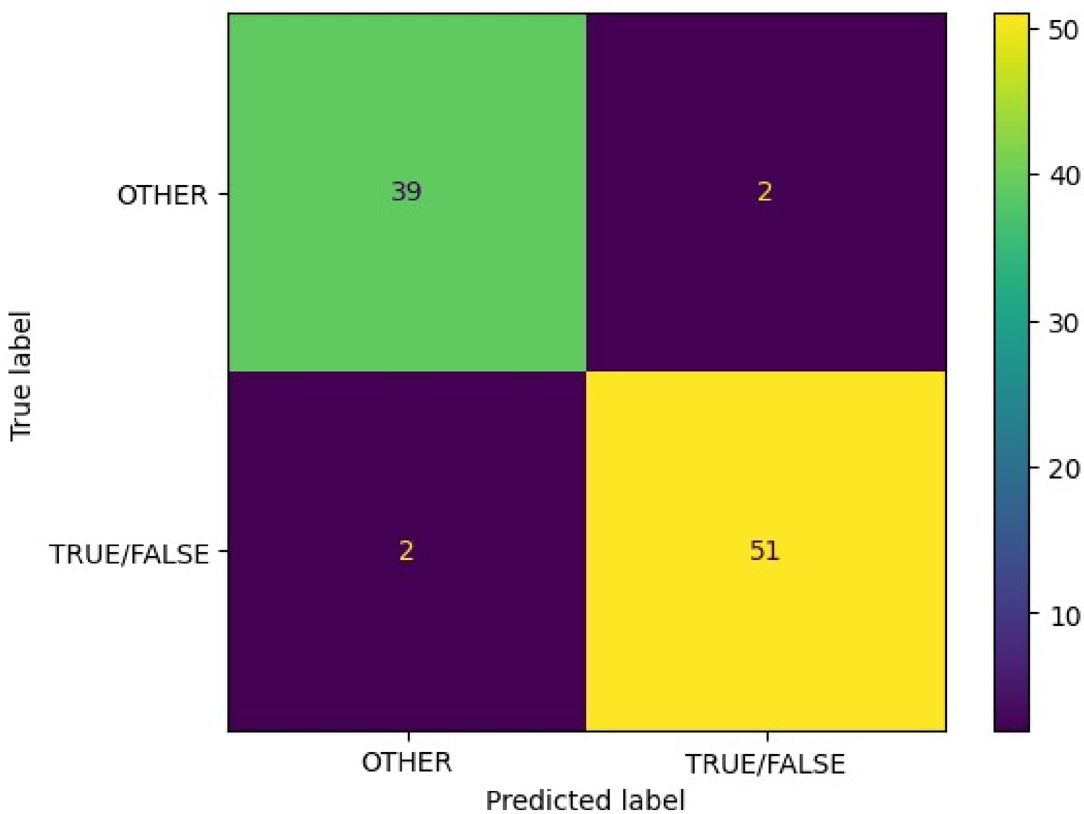


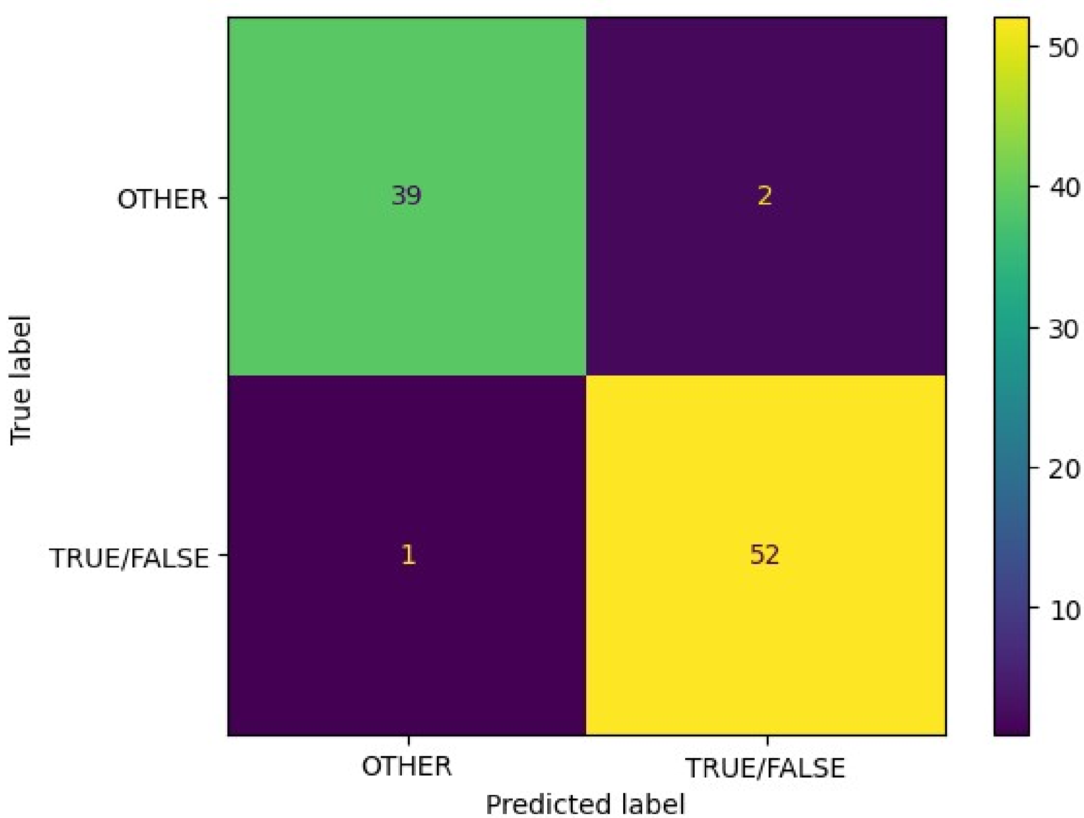

39
2
7
46
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
45
40
35
30
25
20
15
10
5

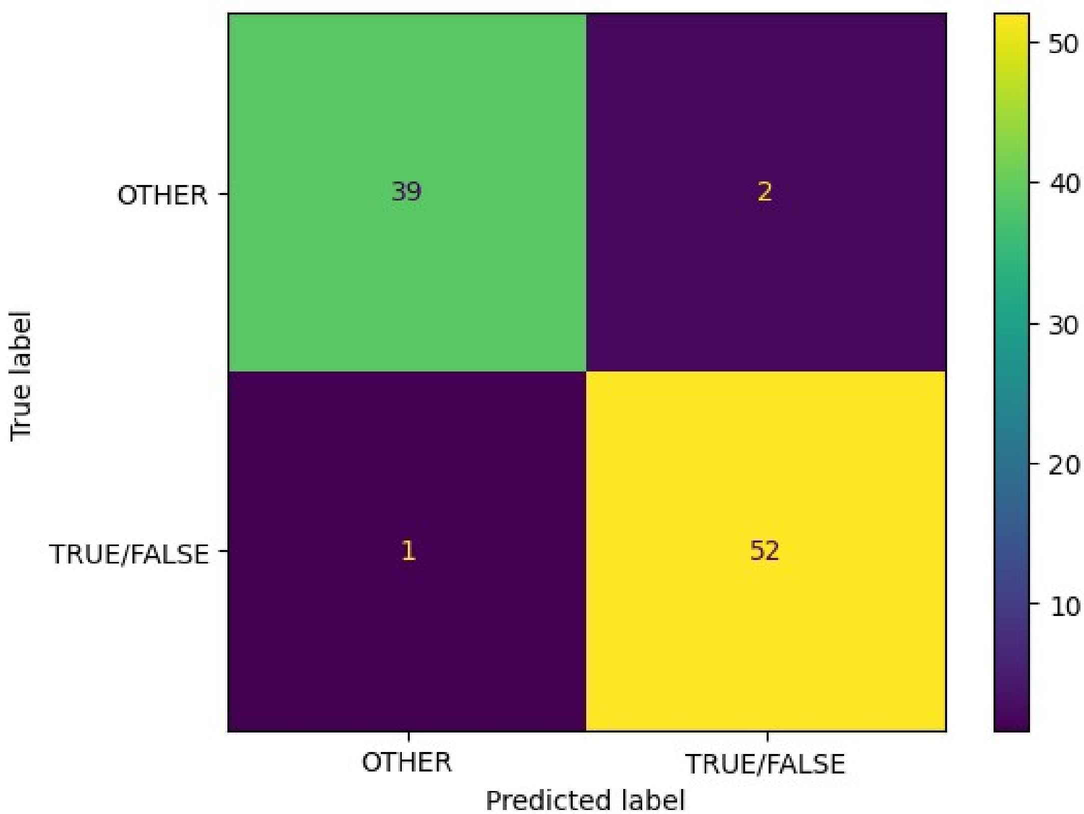

39
2
3
50
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
50
40
30
20
10


OTHER
TRUE/FALSE
True label
39
2
4
49
OTHER
TRUE/FALSE
Predicted label
10
20
30
40

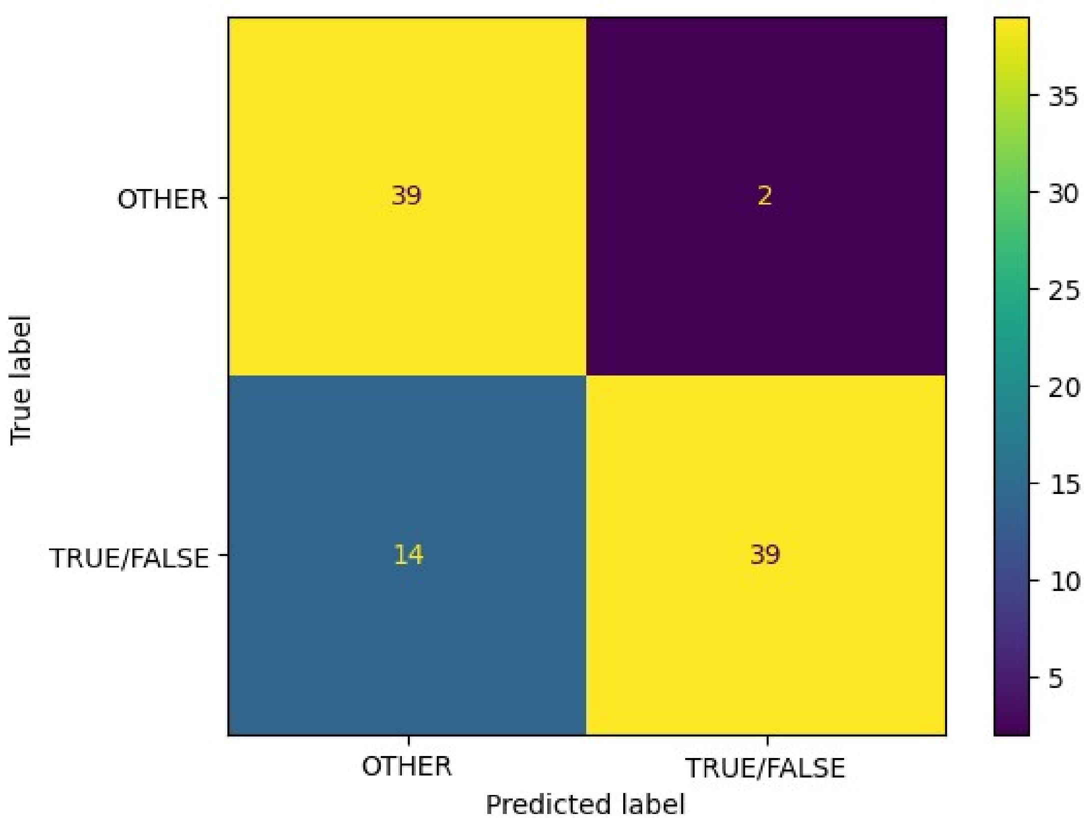

OTHER
TRUE/FALSE
True label
39
2
1
52
OTHER
TRUE/FALSE
Predicted label
10
20
30
40
50


True label
OTHER
TRUE/FALSE
Predicted label
OTHER
TRUE/FALSE
39
2
2
51
50
40
30
20
10

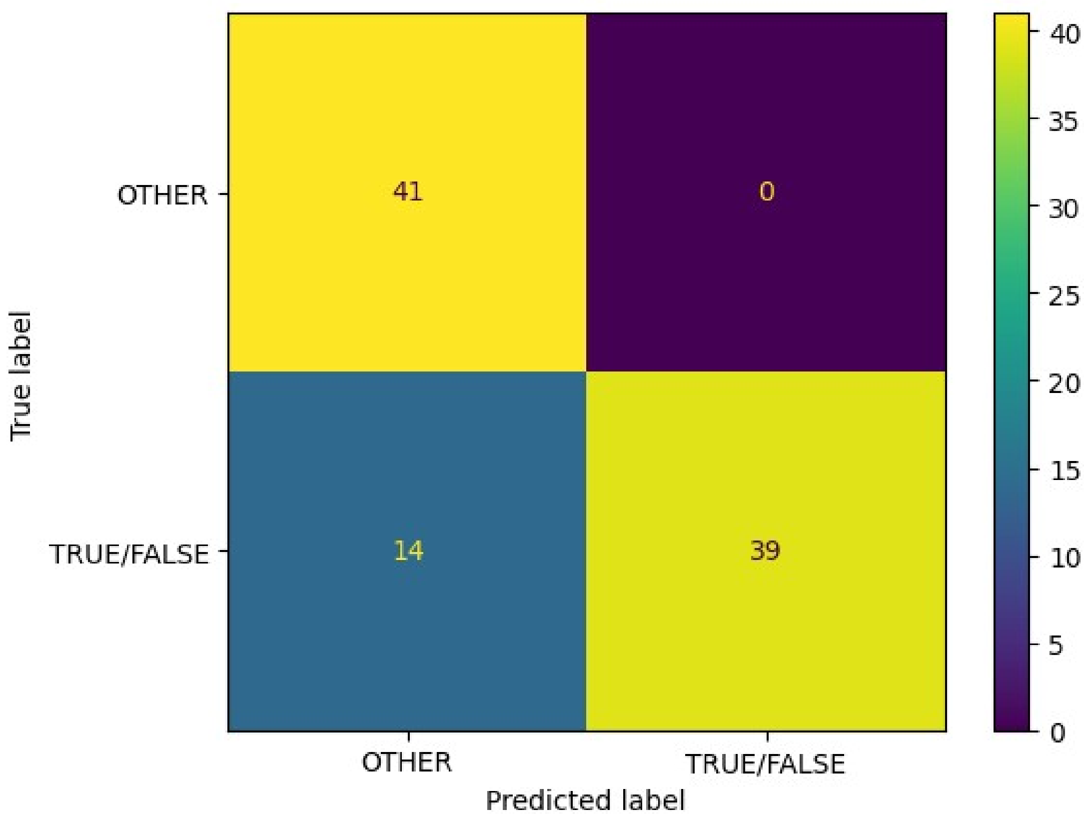

True label
OTHER
TRUE/FALSE
Predicted label
OTHER
TRUE/FALSE
39
2
1
52
50
40
30
20
10

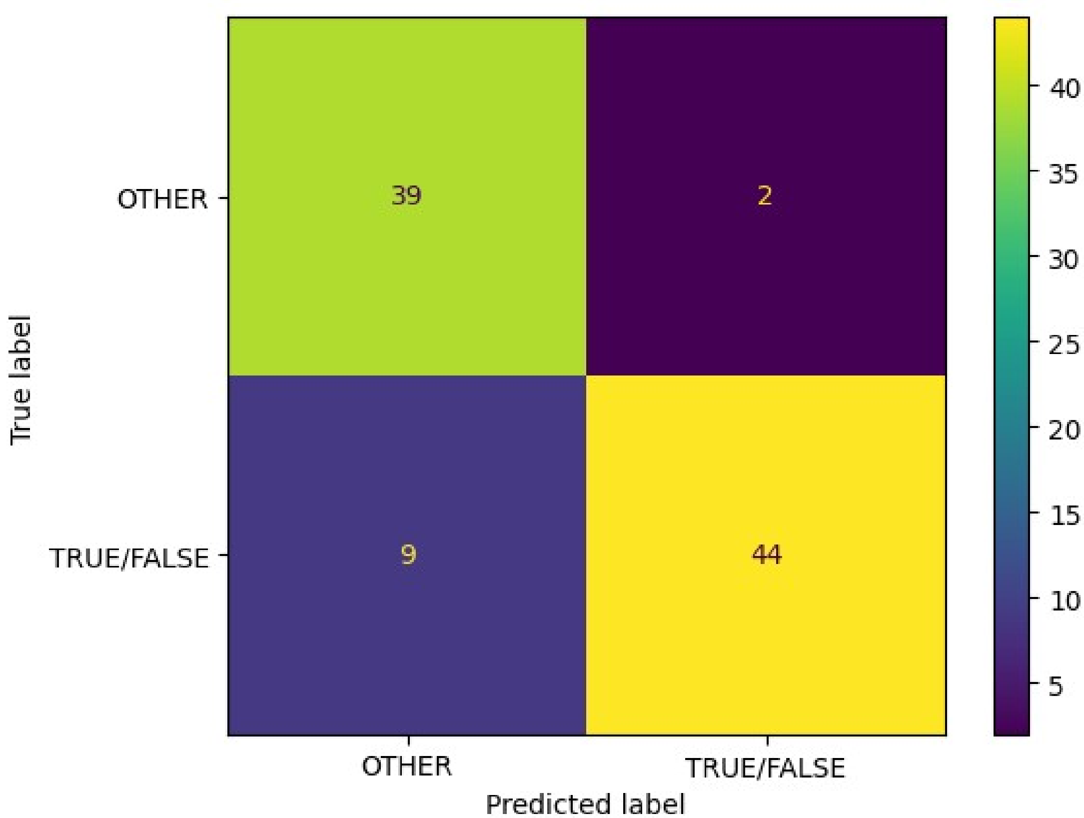


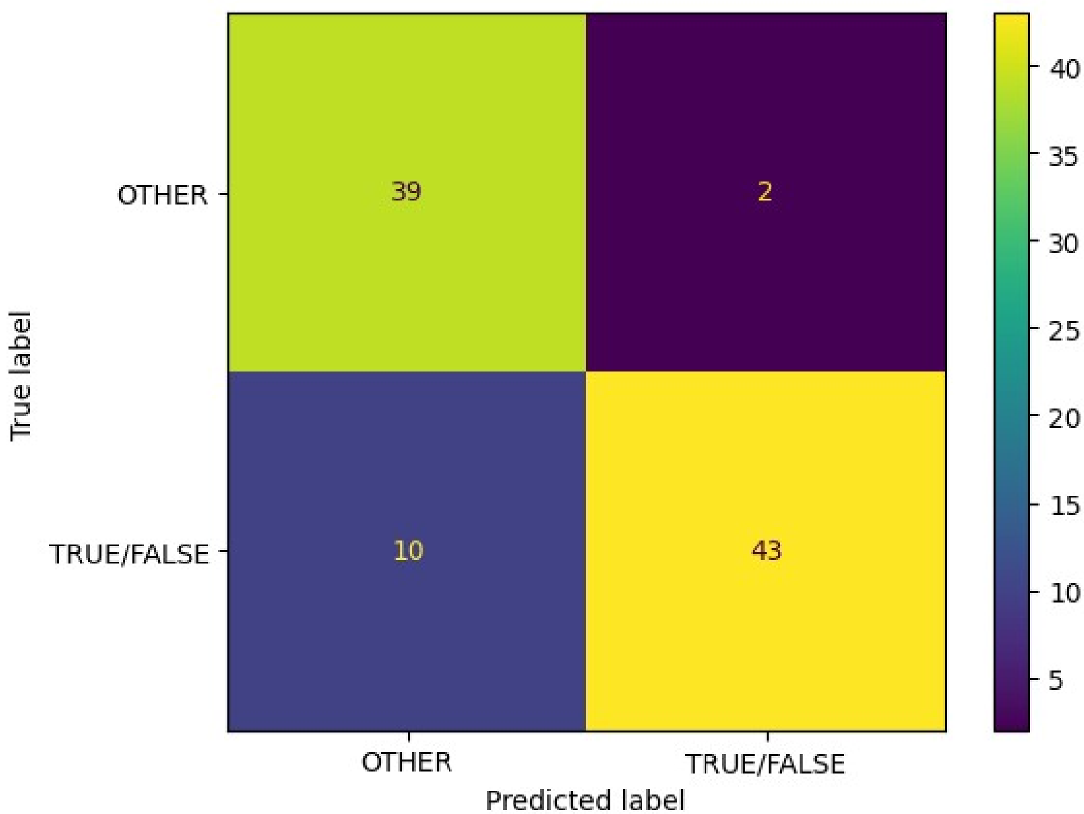
OTHER
TRUE/FALSE
True label
39
2
1
52
50
40
30
20
10
OTHER
TRUE/FALSE
Predicted label



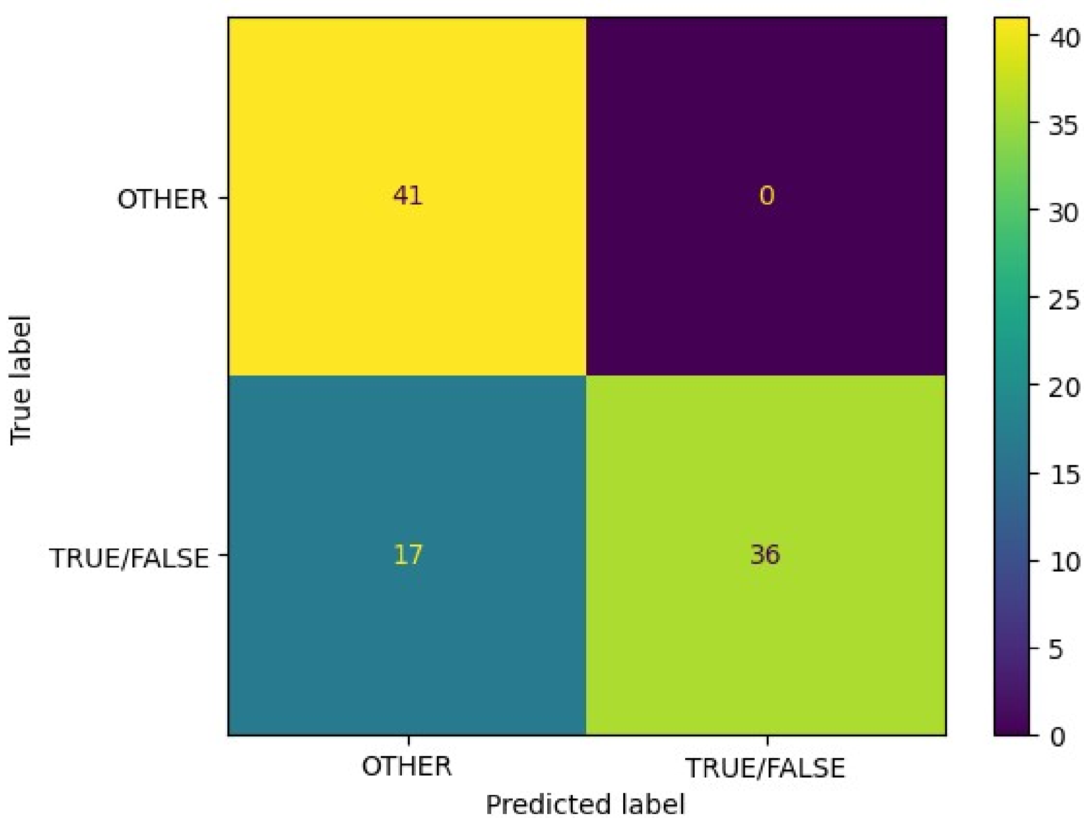
OTHER
TRUE/FALSE
True label
39
2
14
39
35
30
25
20
15
10
5
OTHER
TRUE/FALSE
Predicted label


OTHER
TRUE/FALSE
41
0
14
39
True label
Predicted label
OTHER
TRUE/FALSE
0
5
10
15
20
25
30
35
40

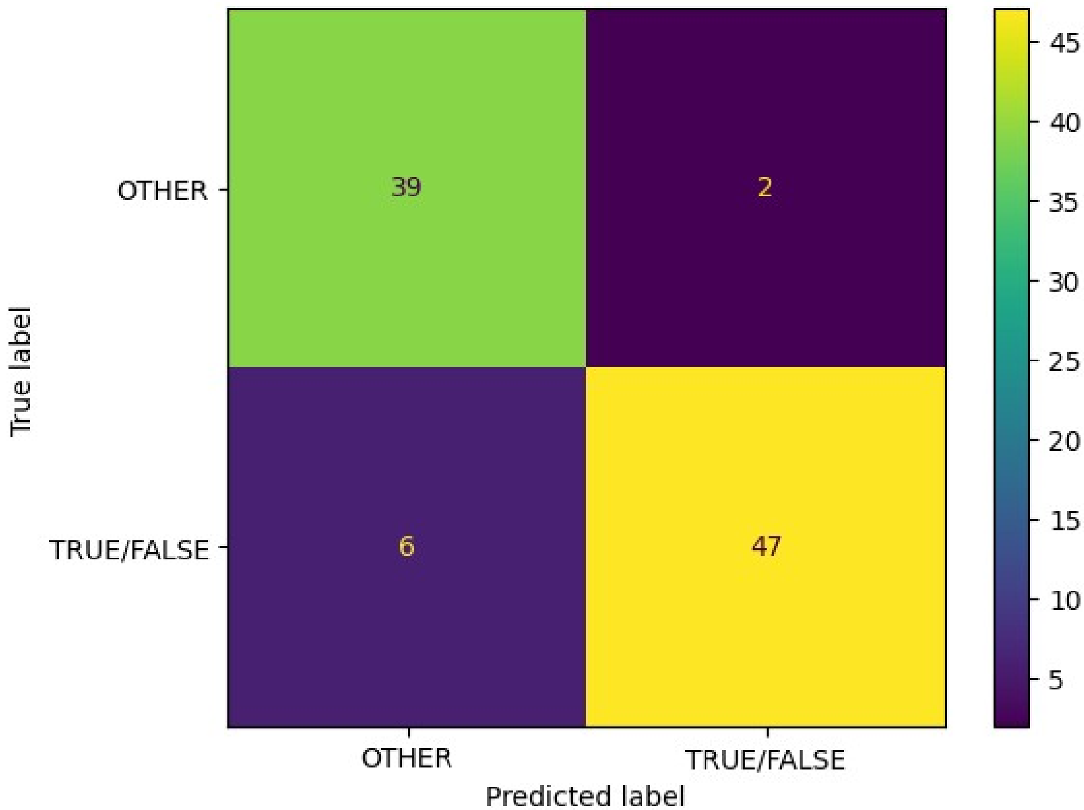

OTHER
TRUE/FALSE
39
2
9
44
True label
Predicted label
OTHER
TRUE/FALSE
5
10
15
20
25
30
35
40

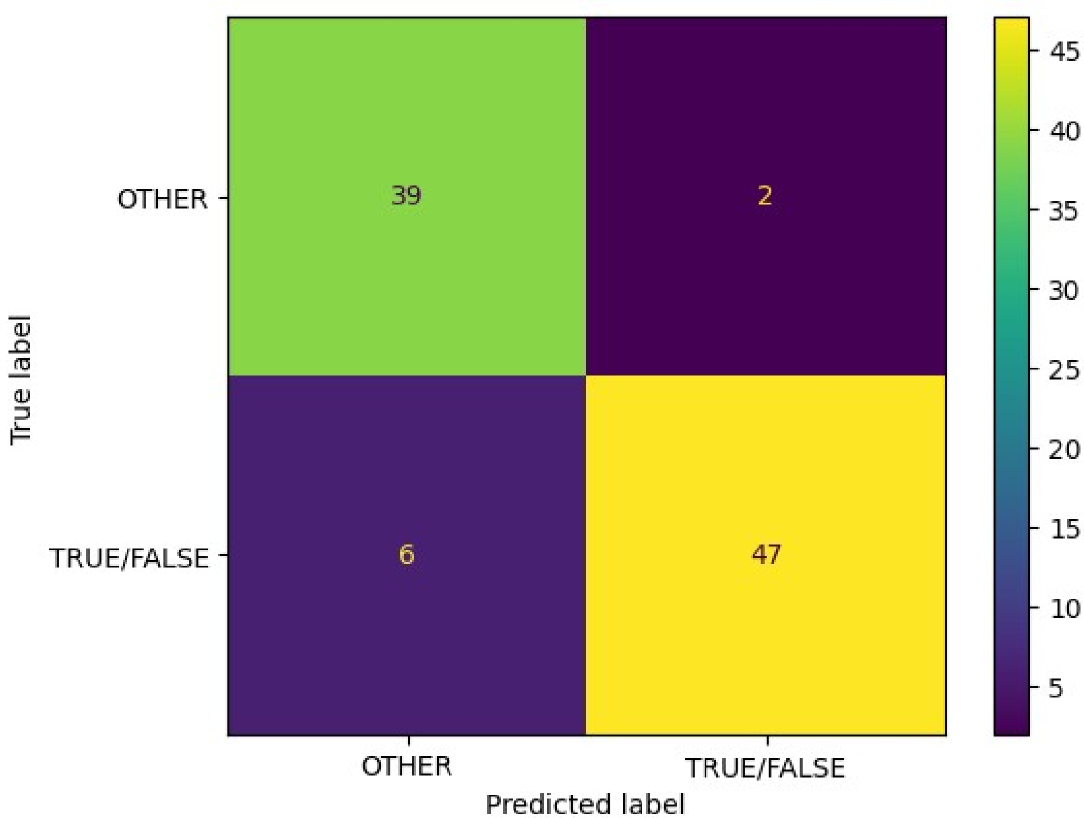


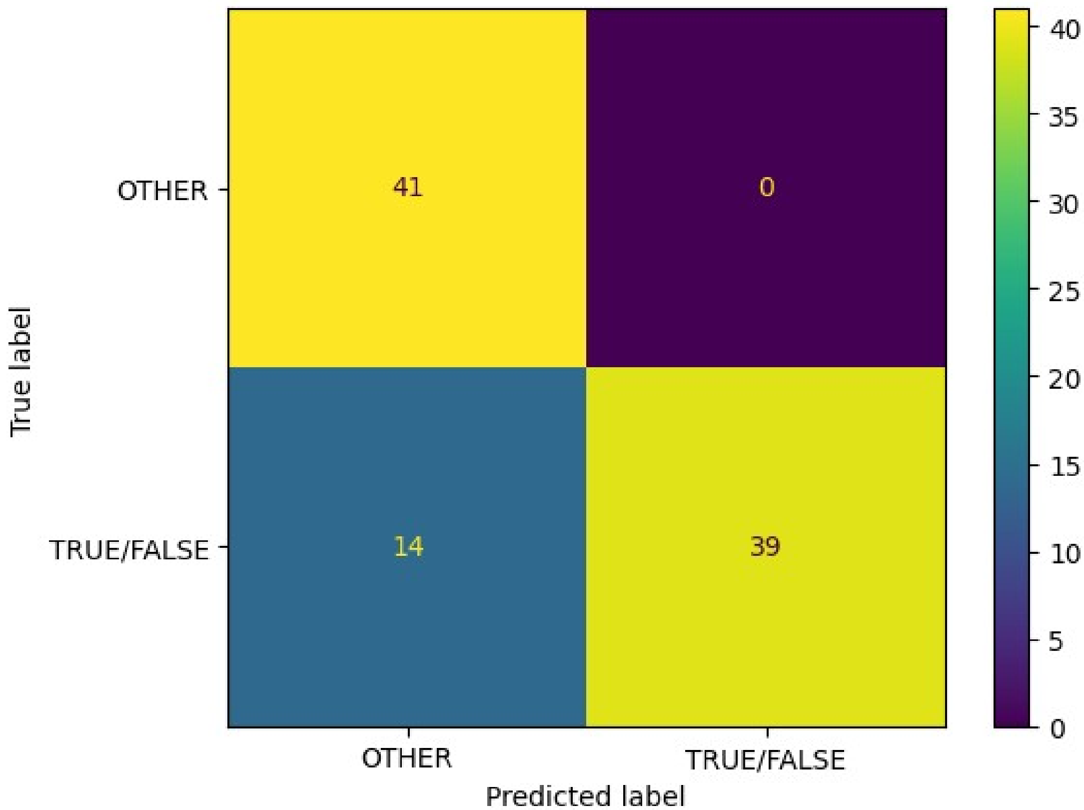
OTHER
TRUE/FALSE
39
2
10
43
True label
Predicted label
40
35
30
25
20
15
10
5



OTHER
TRUE/FALSE
41
0
17
36
True label
Predicted label
40
35
30
25
20
15
10
5
0


OTHER
TRUE/FALSE
39
2
6
47
True label
OTHER
TRUE/FALSE
Predicted label
45
40
35
30
25
20
15
10
5


OTHER
TRUE/FALSE
39
2
6
47
True label
OTHER
TRUE/FALSE
Predicted label
45
40
35
30
25
20
15
10
5



##**Etape 4 : Classification selon le TITRE ET TEXT ENSEMBLE (Concaténés):**

- On va à partir de X_train concaténer les 2 colonnes TEXT et TITLE en mettant un espace entre les deux
- Vu qu'on va travailler sur la colonne text_titre qu'on vient de créer, on va séléctionner cette dernière depuis le X_train et X_test pour apprendre et tester après.

```
train_text_title = X_train.apply(lambda x : '{}
{}'.format(x['text'],x['title']),axis=1)
test_text_title = X_test.apply(lambda x : '{}
{}'.format(x['text'],x['title']),axis=1)

X_train['text_title'] = train_text_title
X_train_text_title = X_train['text_title']
X_train_text_title.reset_index(drop = True, inplace = True)

X_test['text_title'] = test_text_title
X_test_text_title = X_test['text_title']
X_test_text_title.reset_index(drop = True, inplace = True)

print("le texte et titre du train sont")
display(X_train_text_title)
```

```
print("le texte et titre du test sont")
display(X_test_text_title)
```

```
le texte et titre du train sont

0      Historians may look to 2015 as the year when s...
1      Coronavirus may be sexually transmitted and ca...
2      Contractors bidding for work with the governme...
3      More CO2 would actually help the planet , says...
4      To say out-loud that you find the results of t...
                             ...
369    Last week, in the days leading up to Sanders' ...
370    This is one in a series of articles taken from...
371    Food parcels arriving at the Community Service...
372    On Tuesday, radio show host John Fredricks sta...
373    A South African pastor, Alfred Ndlovu has died...
Name: text_title, Length: 374, dtype: object

le texte et titre du test sont

0     Thank you to Universities UK UUK for hosting u...
1     Can any government statistics on COVID-19 deat...
2     MOSCOW — Russian President Vladimir Putin over...
3     Please enable cookies on your web browser in o...
4     PUPILS aged just five have been accused of sex...
                            ...
89    With a smile on her face, City Clerk Susana Me...
90    It was an accurate and judicious answer, so na...
91    Barack Obama, a former President of the US, wa...
92    Pennsylvania rejects 372,000 mail-in ballots, ...
93    Rises in National Insurance Contributions NICS...
Name: text_title, Length: 94, dtype: object
```

Ici, c'est une étape importante, on va tester différents classifieurs, pour chacun des classifieurs, on va appliquer le prétraitement + Vectorisation TfIdf, et on applique une cross_val_score avec un Kfold de 10 fois, par la suite on stocke dans une liste all_results la moyenne des accuracy + l'écart type et on la trie par ordre décroissant de moyenne d'accuracy et d'écart type. on remarque que les 2 meilleurs sont SVM et RF qu'on va séléctionner pour leur appliquer le GridSearch sur les paramètres des prétraitements + leurs hyperparamètres pour pouvoir choisir le meilleur.

```
np.random.seed(42)  # Set the random seed for NumPy

score = 'accuracy'
seed = 7
allresults = []
results = []
names = []


# Liste des modèles à tester
```

```python
models = [
    ('MultinomialNB', MultinomialNB()),
    ('LogisticRegression', LogisticRegression(random_state=42)),
    ('KNN', KNeighborsClassifier()),
    ('CART', DecisionTreeClassifier(random_state=42)),
    ('RF', RandomForestClassifier(random_state=42)),
    ('SVM', SVC(random_state=42))
]


# Création d'un pipeline pour chaque modèle
pipelines = []
for name,model in models:
    pipeline = Pipeline([
        ('normalize', TextNormalizer()),
        ('tfidf', TfidfVectorizer()),
        (name,model)
    ])
    pipelines.append((name,pipeline))

all_results=[]
scores=[]
names=[]
for p in pipelines:
    print(p[1])
    # cross validation en 10 fois
    kfold = KFold(n_splits=10,random_state=seed,shuffle=True)
    start_time = time.time()
    # application de la classification
    cv_results = cross_val_score(p[1],X_train_text_title,y_train,
cv=kfold, scoring=score)
    scores.append(cv_results)
    names.append(p[0])
    all_results.append((p[0],cv_results.mean(),cv_results.std()))
    end_time = time.time()


all_results = sorted(all_results, key=lambda x: (-x[1], -x[2]))
print("all resultats", all_results)
```

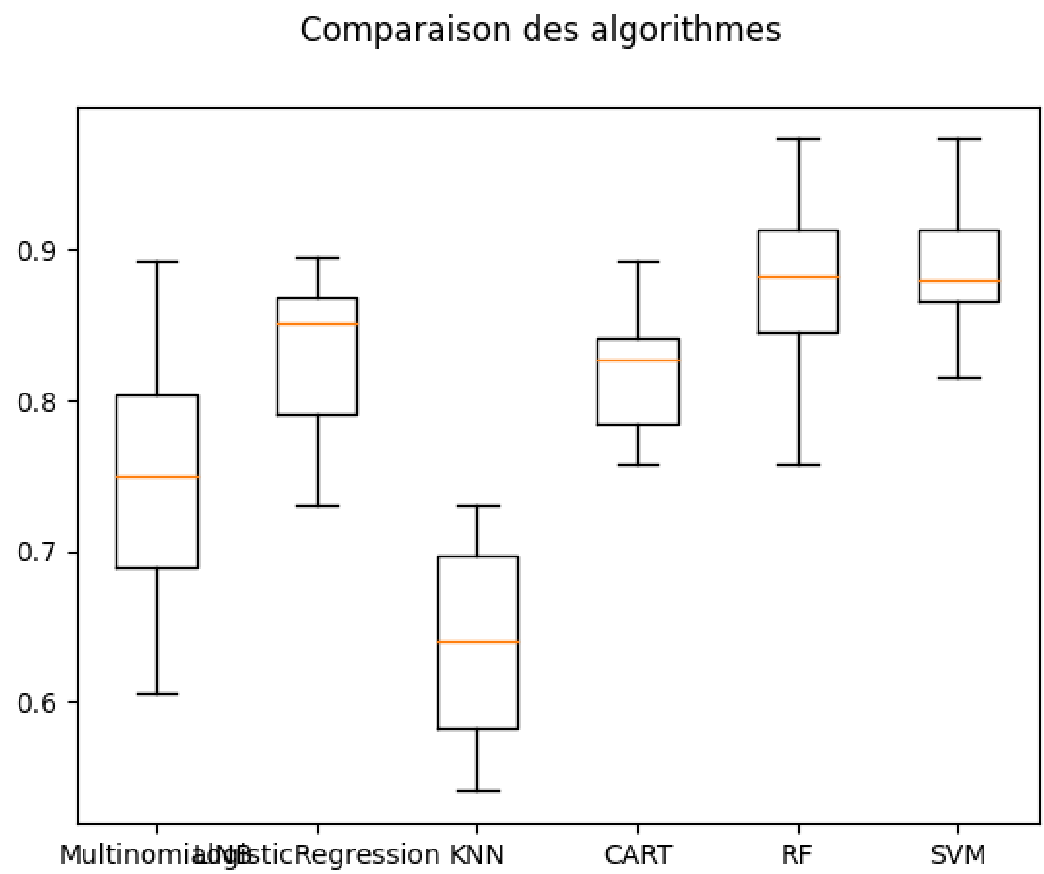
```
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('MultinomialNB', MultinomialNB())])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('LogisticRegression',
LogisticRegression(random_state=42))])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('KNN', KNeighborsClassifier())])
```

```
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('CART', DecisionTreeClassifier(random_state=42))])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('RF', RandomForestClassifier(random_state=42))])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('SVM', SVC(random_state=42))])
all resultats [('SVM', 0.8876955903271693, 0.045964273722778934),
('RF', 0.8743954480796585, 0.05765956829705948),
('LogisticRegression', 0.8287339971550498, 0.0578545300951412),
('CART', 0.8155049786628734, 0.04056074962345757), ('MultinomialNB',
0.7514935988620198, 0.08135587683040285), ('KNN', 0.6389046941678521,
0.06555206867584254)]
```

On affiche les accuracy de chaque classifieur, on remarque la médiane (en rouge) de chaque et l'écart type aussi.

```
fig = plt.figure()
fig.suptitle('Comparaison des algorithmes')
ax = fig.add_subplot(111)
plt.boxplot(scores)
ax.set_xticklabels(names)
```

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



Choisir les meilleurs paramètres pour SVM et RF :

On a un pipeline pour chaque prétraitement différent, on essaye pas mal (miniscule, lemmatisation, miniscule + lemmatisation..) et on stocke le fit_transorm de nos X_train, X_test sur les pipelines dans des listes qui vont contenir tous les fit_transform des pipelines pour chaque classifieur, par la suite on parcourt ces listes là, on itère dessus, et chaque élement de la liste (train) va passer par le GridSearch et puis on predict sur son corresapondant dans liste (test).

```
np.random.seed(42)  # Set the random seed for NumPy

# le plus simple est de faire un test sur differents pipelines.
# pipeline de l'utilisation de CountVectorizer sur le texte avec
differents pre-traitements
CV_brut = Pipeline([('cleaner', TextNormalizer()),
                    ('count_vectorizer',
CountVectorizer(lowercase=False))])
CV_lowcase = Pipeline([('cleaner',
TextNormalizer(removestopwords=False,lowercase=True,

getstemmer=False,removedigit=False)),
                    ('count_vectorizer',
CountVectorizer(lowercase=False))])
```

```
CV_lowStop = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=False,removedigit=False)),
                    ('count_vectorizer',
CountVectorizer(lowercase=False))])

CV_lowStopstem = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=True,removedigit=False)),
                    ('count_vectorizer',
CountVectorizer(lowercase=False))])

# pipeline de l'utilisation de TfidfVectorizer avec differents pre-
traitements
TFIDF_brut = Pipeline ([('cleaner', TextNormalizer()),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])

TFIDF_lowcase = Pipeline([('cleaner',
TextNormalizer(removestopwords=False,lowercase=True,

getstemmer=False,removedigit=False)),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])
TFIDF_lowStop = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=False,removedigit=False)),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])

TFIDF_lowStopstem = Pipeline([('cleaner',
TextNormalizer(removestopwords=True,lowercase=True,

getstemmer=True,removedigit=False)),
                    ('tfidf_vectorizer',
TfidfVectorizer(lowercase=False))])


# Liste de tous les modeles à tester
all_models = [
    ("CV_brut", CV_brut),
    ("CV_lowcase", CV_lowcase),
    ("CV_lowStop", CV_lowStop),
    ("CV_lowStopstem",CV_lowStopstem),
    ("TFIDF_lowcase", TFIDF_lowcase),
    ("TFIDF_lowStop", TFIDF_lowStop),
```

```
    ("TFIDF_lowStopstem",TFIDF_lowStopstem),
    ("TFIDF_brut", TFIDF_brut)
]

X_train_text_title_SVC = []
X_test_text_title_SVC = []

X_train_text_title_RandomForestClassifier = []
X_test_text_title_RandomForestClassifier = []


for name, pipeline in all_models :

X_train_text_title_SVC.append(pipeline.fit_transform(X_train_text_titl
e).toarray())

X_test_text_title_SVC.append(pipeline.transform(X_test_text_title).toa
rray())

X_train_text_title_RandomForestClassifier.append(pipeline.fit_transfor
m(X_train_text_title).toarray())

X_test_text_title_RandomForestClassifier.append(pipeline.transform(X_t
est_text_title).toarray())


models = {
    'SVC': SVC(random_state=42),
    'RandomForestClassifier': RandomForestClassifier(random_state=42)
}


params = {'SVC': [{'C': [0.001, 0.01, 0.1, 1,2,5,7,10]},
             {'gamma': [0.001, 0.01, 0.1,0.2,0.3,0.5,0.7,1]},
             {'kernel': ['linear', 'rbf']}],
    'RandomForestClassifier': [{'n_estimators': [10, 50, 100, 200,
300]},
                              {'max_features': ['auto', 'sqrt',
'log2']}]
}

for model_name, model in models.items():
    score='accuracy'
    X_train_text_title = eval('X_train_text_title_' + model_name)
    X_test_text_title = eval('X_test_text_title_' + model_name)
    for i in range (len(X_train_text_title)):
      grid_search = GridSearchCV(model, params[model_name], n_jobs=-1,
verbose=1,scoring=score)
      print("grid search fait")
```

```
      grid_search.fit(X_train_text_title[i],y_train)
      print ('meilleur score %0.3f'%(grid_search.best_score_),'\n')
      print ('meilleur estimateur',grid_search.best_estimator_,'\n')
      y_pred = grid_search.predict(X_test_text_title[i])
      MyshowAllScores(y_test,y_pred)

      print("Ensemble des meilleurs paramètres :")
      best_parameters = grid_search.best_estimator_.get_params()
      for param_dict in params[model_name]:
        for param_name, param_value in param_dict.items():
            print("\t%s: %r" % (param_name,
best_parameters[param_name]))
```

```
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.901

meilleur estimateur SVC(gamma=0.2, random_state=42)

Accuracy : 0.979
Classification Report
              precision    recall  f1-score   support

       OTHER    1.00000   0.95122   0.97500        41
  TRUE/FALSE    0.96364   1.00000   0.98148        53

    accuracy                        0.97872        94
   macro avg    0.98182   0.97561   0.97824        94
weighted avg    0.97950   0.97872   0.97865        94

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 0.2
	kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.901

meilleur estimateur SVC(gamma=0.2, random_state=42)

Accuracy : 0.979
Classification Report
              precision    recall  f1-score   support

       OTHER    1.00000   0.95122   0.97500        41
  TRUE/FALSE    0.96364   1.00000   0.98148        53

    accuracy                        0.97872        94
   macro avg    0.98182   0.97561   0.97824        94
```

```
weighted avg    0.97950   0.97872   0.97865        94

Ensemble des meilleurs paramètres :
     C: 1.0
     gamma: 0.2
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.901

meilleur estimateur SVC(gamma=0.1, random_state=42)

Accuracy : 0.968
Classification Report
              precision    recall  f1-score   support

       OTHER    0.97500   0.95122   0.96296        41
  TRUE/FALSE    0.96296   0.98113   0.97196        53

    accuracy                        0.96809        94
   macro avg    0.96898   0.96618   0.96746        94
weighted avg    0.96821   0.96809   0.96804        94

Ensemble des meilleurs paramètres :
     C: 1.0
     gamma: 0.1
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.901

meilleur estimateur SVC(gamma=0.1, random_state=42)

Accuracy : 0.968
Classification Report
              precision    recall  f1-score   support

       OTHER    0.97500   0.95122   0.96296        41
  TRUE/FALSE    0.96296   0.98113   0.97196        53

    accuracy                        0.96809        94
   macro avg    0.96898   0.96618   0.96746        94
weighted avg    0.96821   0.96809   0.96804        94

Ensemble des meilleurs paramètres :
     C: 1.0
     gamma: 0.1
     kernel: 'rbf'
grid search fait
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.874

meilleur estimateur SVC(C=2, random_state=42)

Accuracy : 0.947
Classification Report
              precision    recall  f1-score   support

       OTHER    0.92857   0.95122   0.93976        41
  TRUE/FALSE    0.96154   0.94340   0.95238        53

    accuracy                        0.94681        94
   macro avg    0.94505   0.94731   0.94607        94
weighted avg    0.94716   0.94681   0.94688        94

Ensemble des meilleurs paramètres :
     C: 2
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.869

meilleur estimateur SVC(C=1, random_state=42)

Accuracy : 0.947
Classification Report
              precision    recall  f1-score   support

       OTHER    0.92857   0.95122   0.93976        41
  TRUE/FALSE    0.96154   0.94340   0.95238        53

    accuracy                        0.94681        94
   macro avg    0.94505   0.94731   0.94607        94
weighted avg    0.94716   0.94681   0.94688        94

Ensemble des meilleurs paramètres :
     C: 1
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.869

meilleur estimateur SVC(gamma=0.5, random_state=42)

Accuracy : 0.894
Classification Report
```

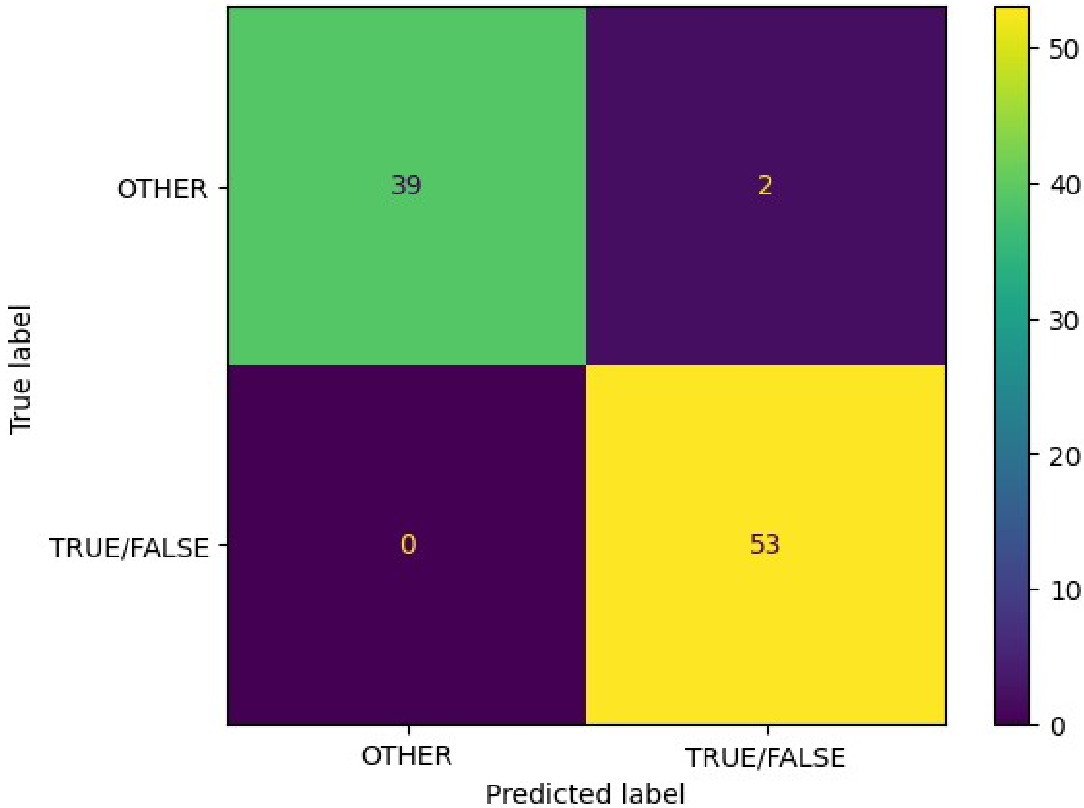
```
              precision    recall  f1-score   support

       OTHER    0.84444   0.92683   0.88372        41
  TRUE/FALSE    0.93878   0.86792   0.90196        53

    accuracy                        0.89362        94
   macro avg    0.89161   0.89738   0.89284        94
weighted avg    0.89763   0.89362   0.89401        94

Ensemble des meilleurs paramètres :
      C: 1.0
      gamma: 0.5
      kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.877

meilleur estimateur SVC(C=1, random_state=42)

Accuracy : 0.947
Classification Report
              precision    recall  f1-score   support

       OTHER    0.92857   0.95122   0.93976        41
  TRUE/FALSE    0.96154   0.94340   0.95238        53

    accuracy                        0.94681        94
   macro avg    0.94505   0.94731   0.94607        94
weighted avg    0.94716   0.94681   0.94688        94

Ensemble des meilleurs paramètres :
      C: 1
      gamma: 'scale'
      kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.840

meilleur estimateur RandomForestClassifier(random_state=42)

Accuracy : 0.936
Classification Report
              precision    recall  f1-score   support

       OTHER    0.90698   0.95122   0.92857        41
  TRUE/FALSE    0.96078   0.92453   0.94231        53

    accuracy                        0.93617        94
   macro avg    0.93388   0.93787   0.93544        94
```

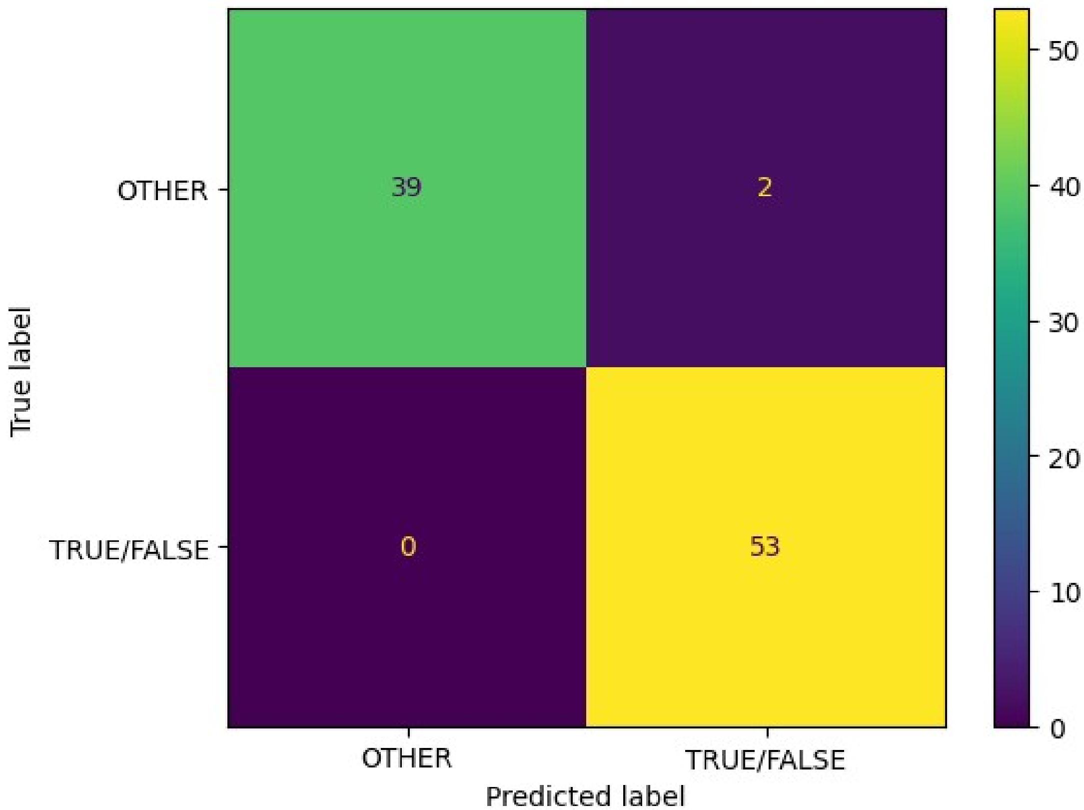
```
weighted avg    0.93732   0.93617   0.93632        94

Ensemble des meilleurs paramètres :
	n_estimators: 100
	max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.864

meilleur estimateur RandomForestClassifier(n_estimators=200,
random_state=42)

Accuracy : 0.926
Classification Report
              precision    recall  f1-score   support

       OTHER    0.88636   0.95122   0.91765        41
  TRUE/FALSE    0.96000   0.90566   0.93204        53

    accuracy                        0.92553        94
   macro avg    0.92318   0.92844   0.92484        94
weighted avg    0.92788   0.92553   0.92576        94

Ensemble des meilleurs paramètres :
	n_estimators: 200
	max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.815

meilleur estimateur RandomForestClassifier(random_state=42)

Accuracy : 0.904
Classification Report
              precision    recall  f1-score   support

       OTHER    0.84783   0.95122   0.89655        41
  TRUE/FALSE    0.95833   0.86792   0.91089        53

    accuracy                        0.90426        94
   macro avg    0.90308   0.90957   0.90372        94
weighted avg    0.91013   0.90426   0.90464        94

Ensemble des meilleurs paramètres :
	n_estimators: 100
	max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.832
```

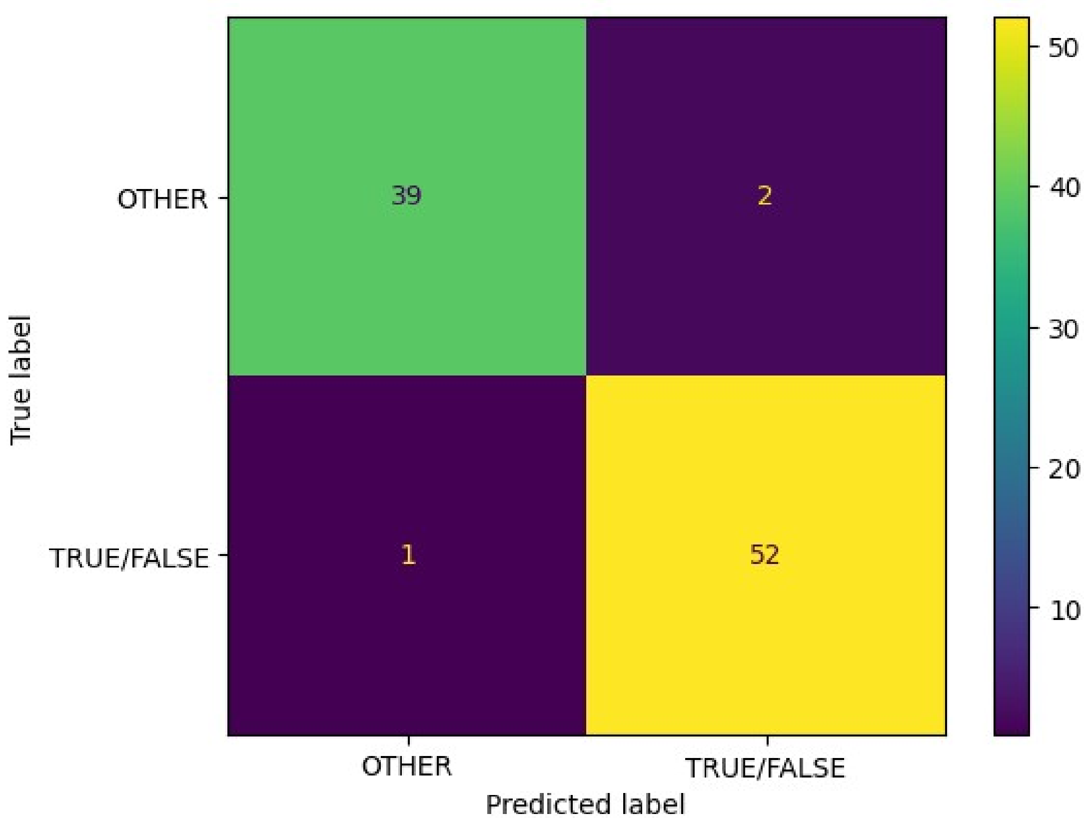
```
meilleur estimateur RandomForestClassifier(n_estimators=200,
random_state=42)

Accuracy : 0.883
Classification Report
              precision    recall  f1-score   support

       OTHER    0.81250   0.95122   0.87640        41
  TRUE/FALSE    0.95652   0.83019   0.88889        53

    accuracy                        0.88298        94
   macro avg    0.88451   0.89070   0.88265        94
weighted avg    0.89370   0.88298   0.88344        94

Ensemble des meilleurs paramètres :
     n_estimators: 200
     max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.869

meilleur estimateur RandomForestClassifier(n_estimators=200,
random_state=42)

Accuracy : 0.926
Classification Report
              precision    recall  f1-score   support

       OTHER    0.88636   0.95122   0.91765        41
  TRUE/FALSE    0.96000   0.90566   0.93204        53

    accuracy                        0.92553        94
   macro avg    0.92318   0.92844   0.92484        94
weighted avg    0.92788   0.92553   0.92576        94

Ensemble des meilleurs paramètres :
     n_estimators: 200
     max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.856

meilleur estimateur RandomForestClassifier(n_estimators=300,
random_state=42)

Accuracy : 0.894
Classification Report
              precision    recall  f1-score   support
```

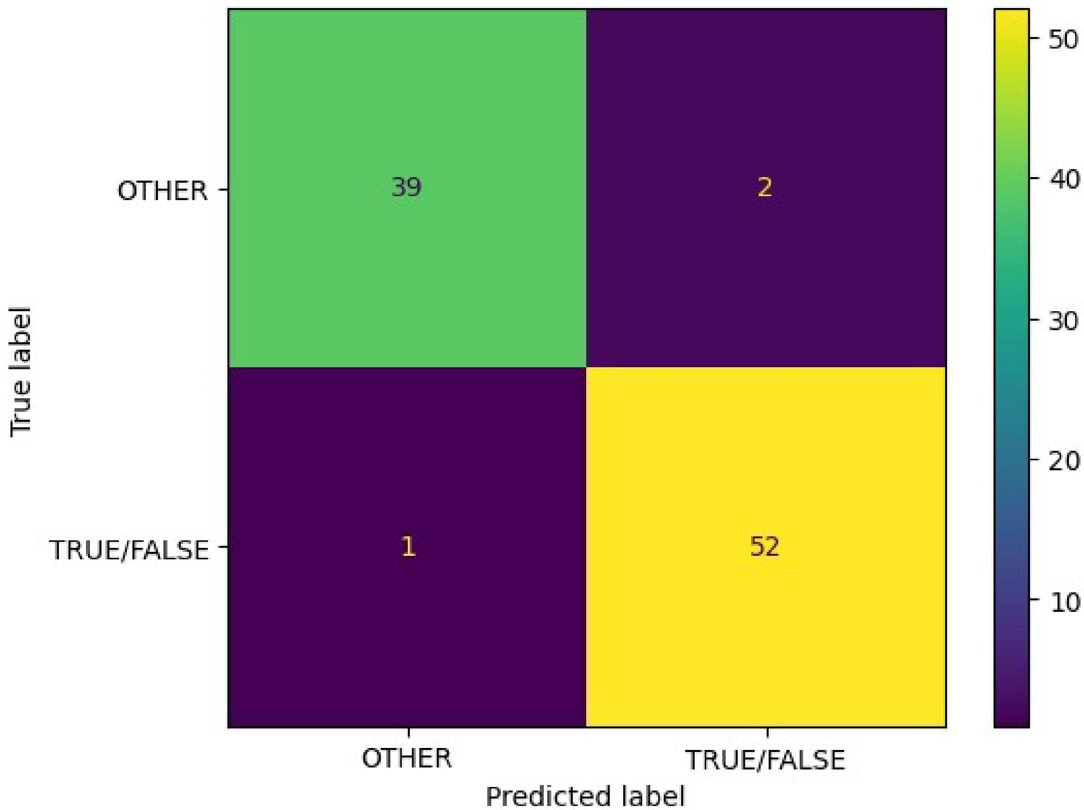
```
       OTHER    0.82979   0.95122   0.88636        41
  TRUE/FALSE    0.95745   0.84906   0.90000        53

    accuracy                        0.89362        94
   macro avg    0.89362   0.90014   0.89318        94
weighted avg    0.90177   0.89362   0.89405        94

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.837

meilleur estimateur RandomForestClassifier(n_estimators=300,
random_state=42)

Accuracy : 0.915
Classification Report
              precision    recall  f1-score   support

       OTHER    0.86667   0.95122   0.90698        41
  TRUE/FALSE    0.95918   0.88679   0.92157        53

    accuracy                        0.91489        94
   macro avg    0.91293   0.91901   0.91427        94
weighted avg    0.91883   0.91489   0.91520        94

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.845

meilleur estimateur RandomForestClassifier(n_estimators=50,
random_state=42)

Accuracy : 0.883
Classification Report
              precision    recall  f1-score   support

       OTHER    0.81250   0.95122   0.87640        41
  TRUE/FALSE    0.95652   0.83019   0.88889        53

    accuracy                        0.88298        94
   macro avg    0.88451   0.89070   0.88265        94
weighted avg    0.89370   0.88298   0.88344        94
```

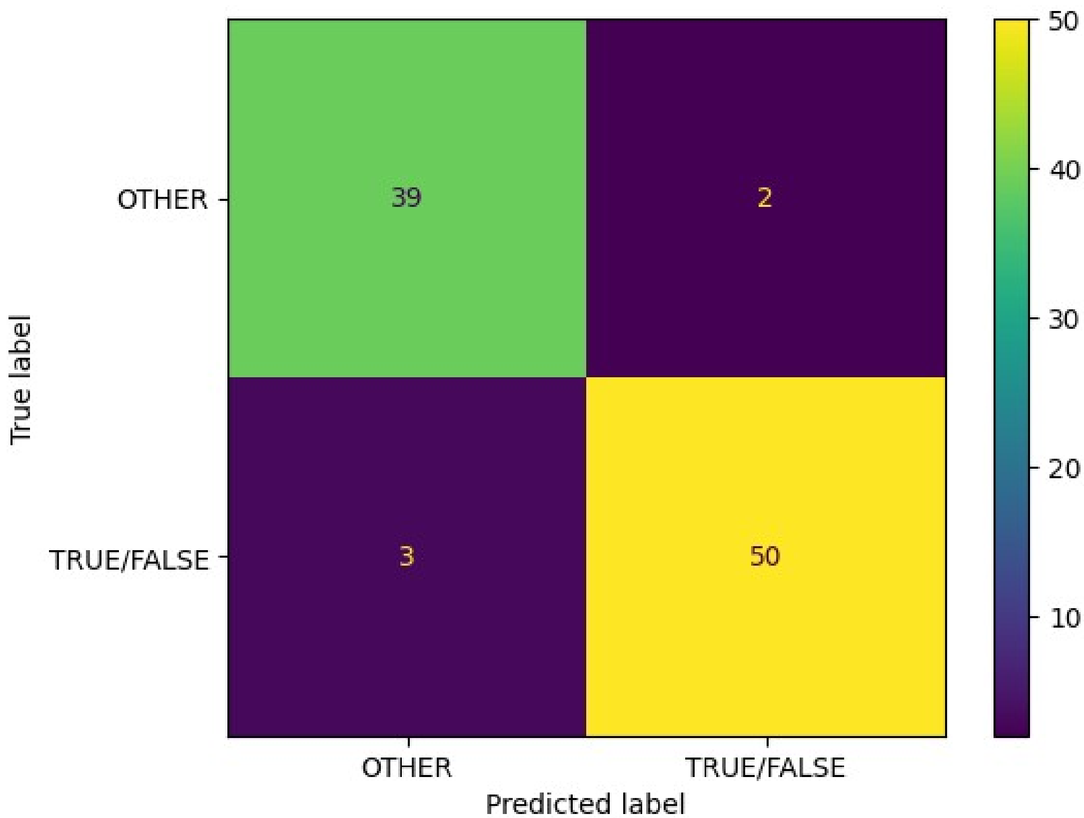
```
Ensemble des meilleurs paramètres :
      n_estimators: 50
      max_features: 'sqrt'
```

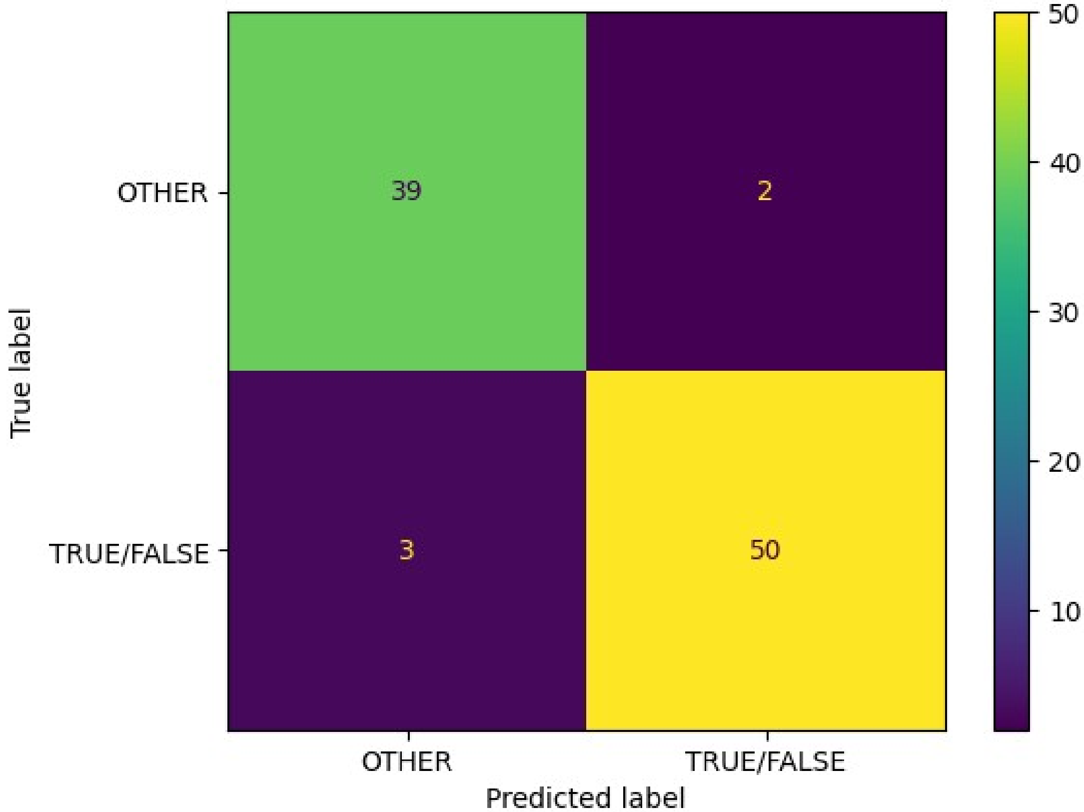


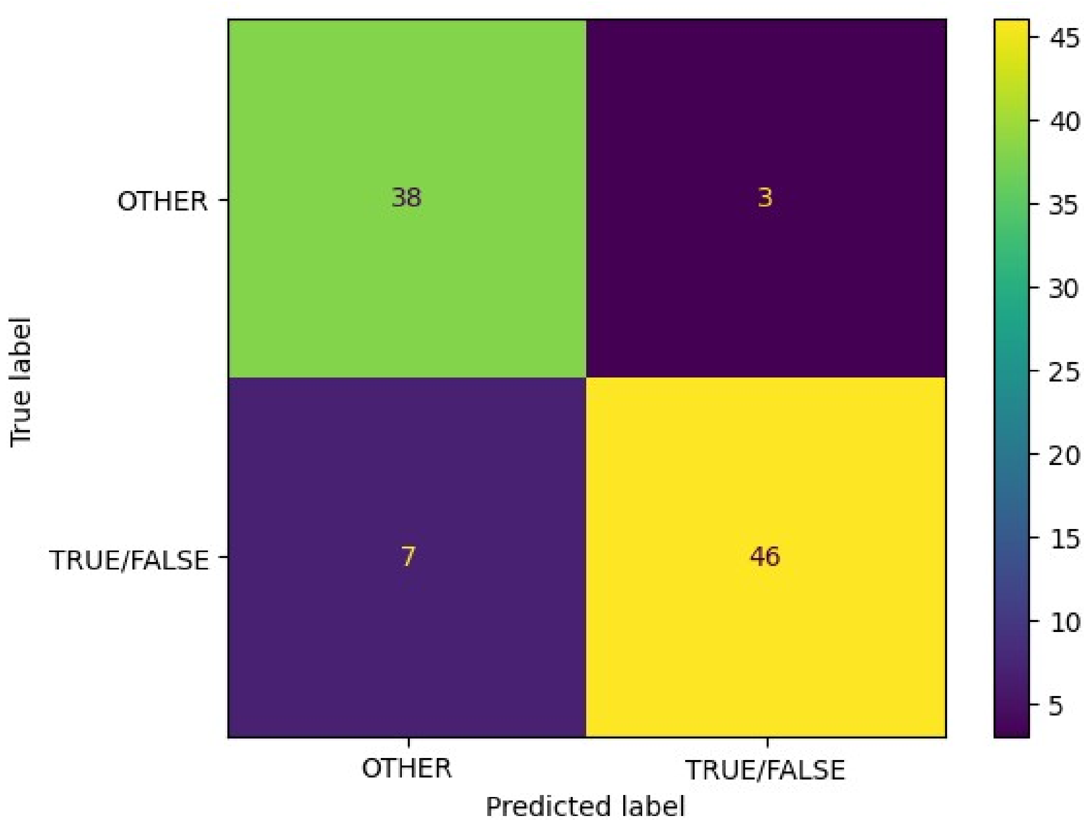

39
2
0
53
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
50
40
30
20
10
0

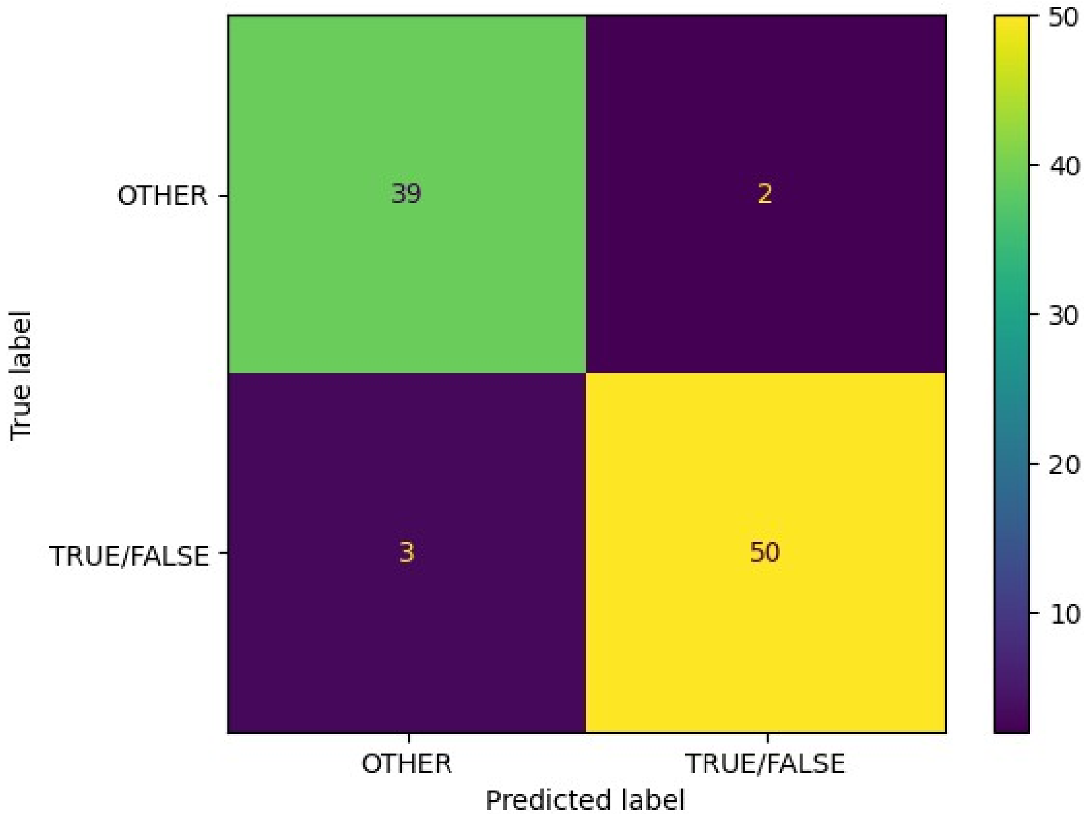

39
2
1
52
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
50
40
30
20
10

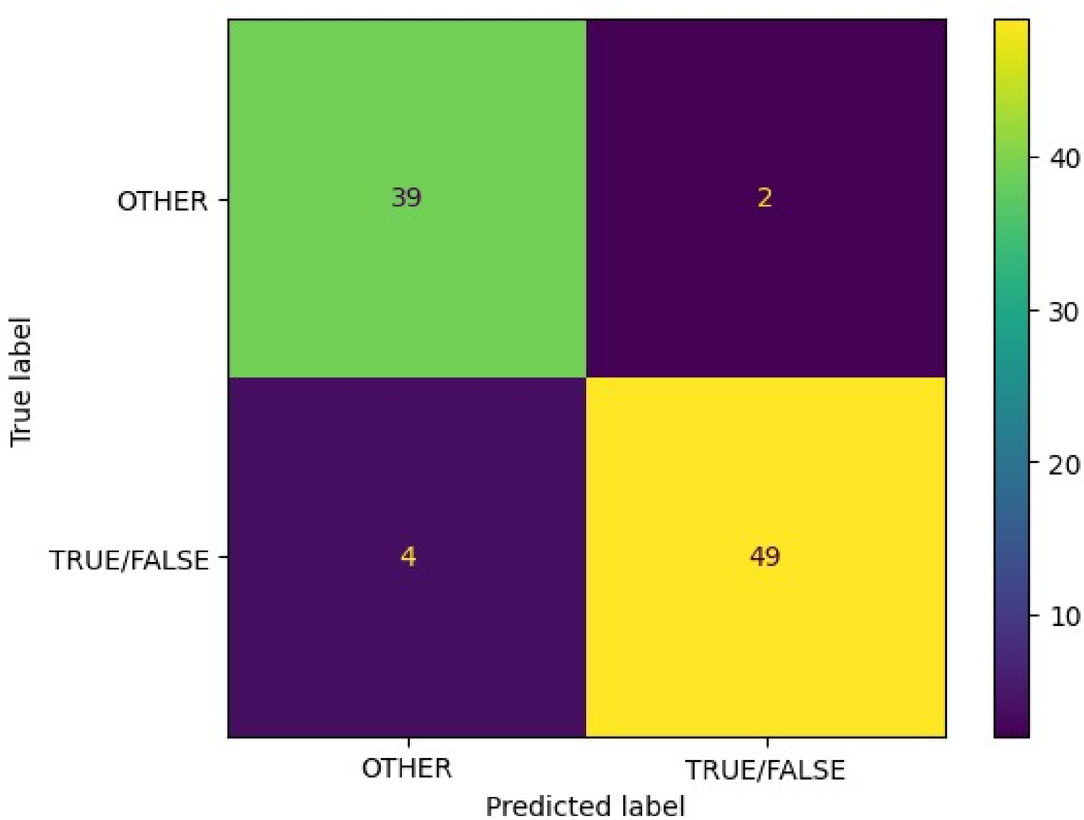

OTHER
TRUE/FALSE
True label
Predicted label
39
2
1
52
50
40
30
20
10

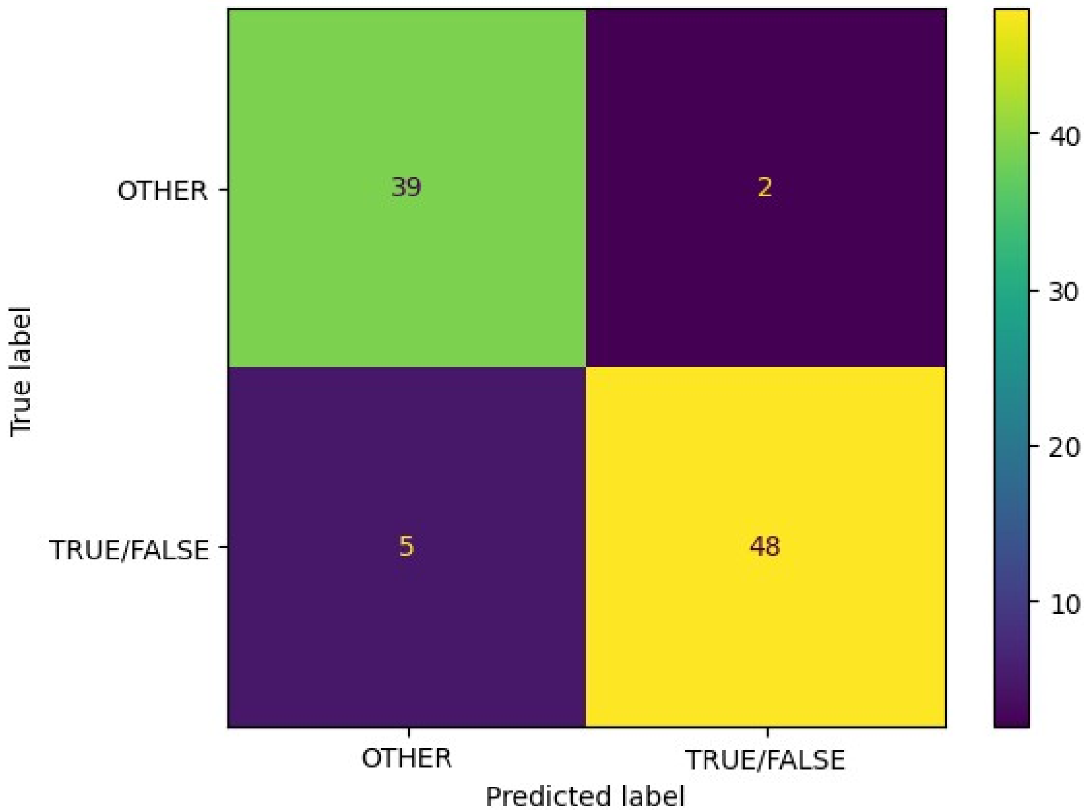

OTHER
TRUE/FALSE
True label
Predicted label
39
2
3
50
50
40
30
20
10

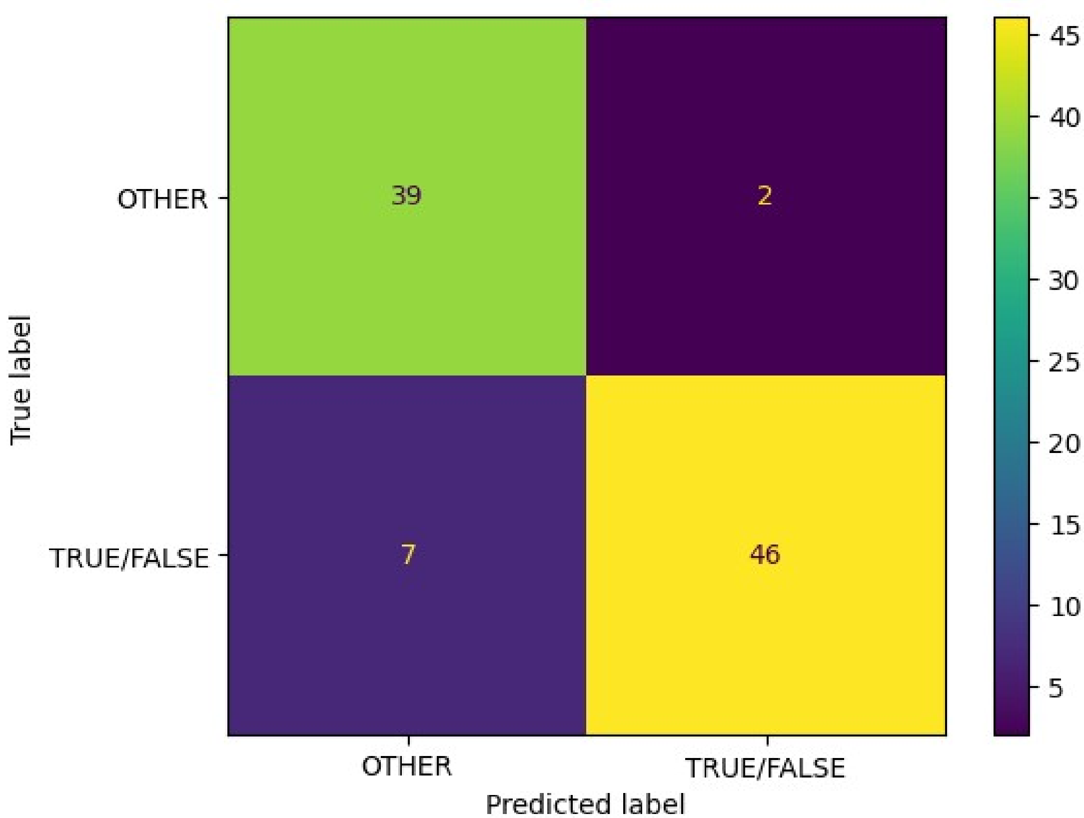

OTHER
TRUE/FALSE
True label
Predicted label
39
2
3
50
10
20
30
40
50

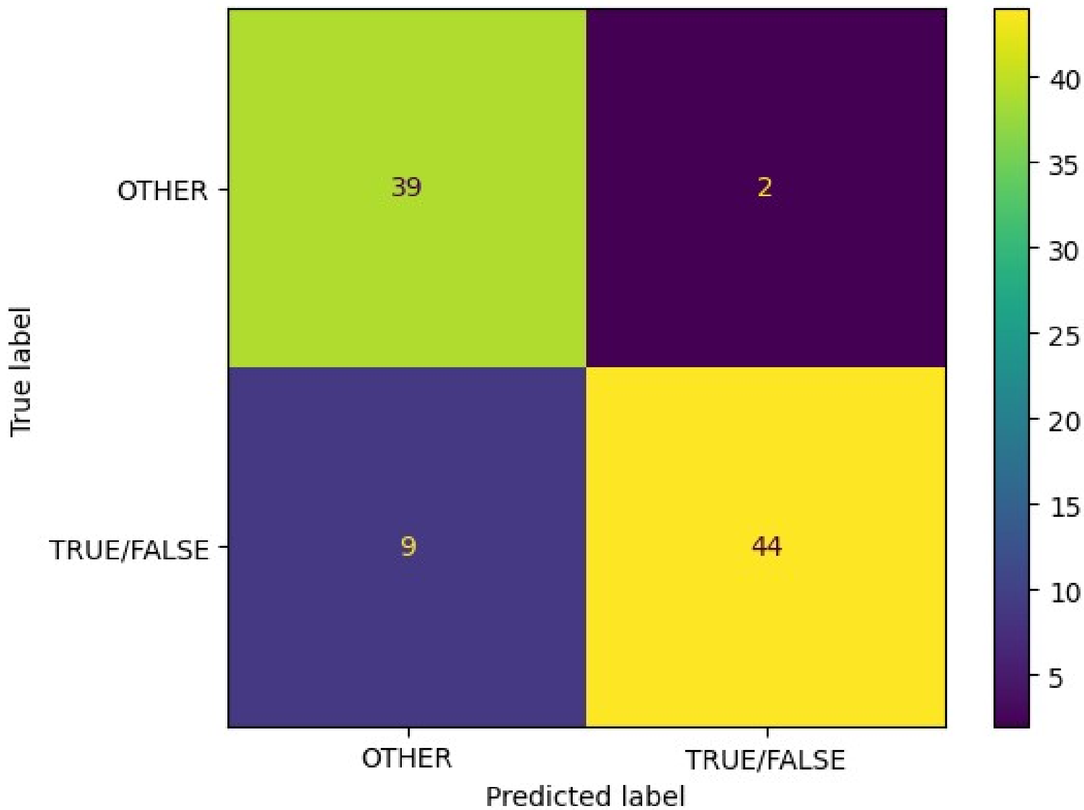

OTHER
TRUE/FALSE
True label
Predicted label
38
3
7
46
5
10
15
20
25
30
35
40
45

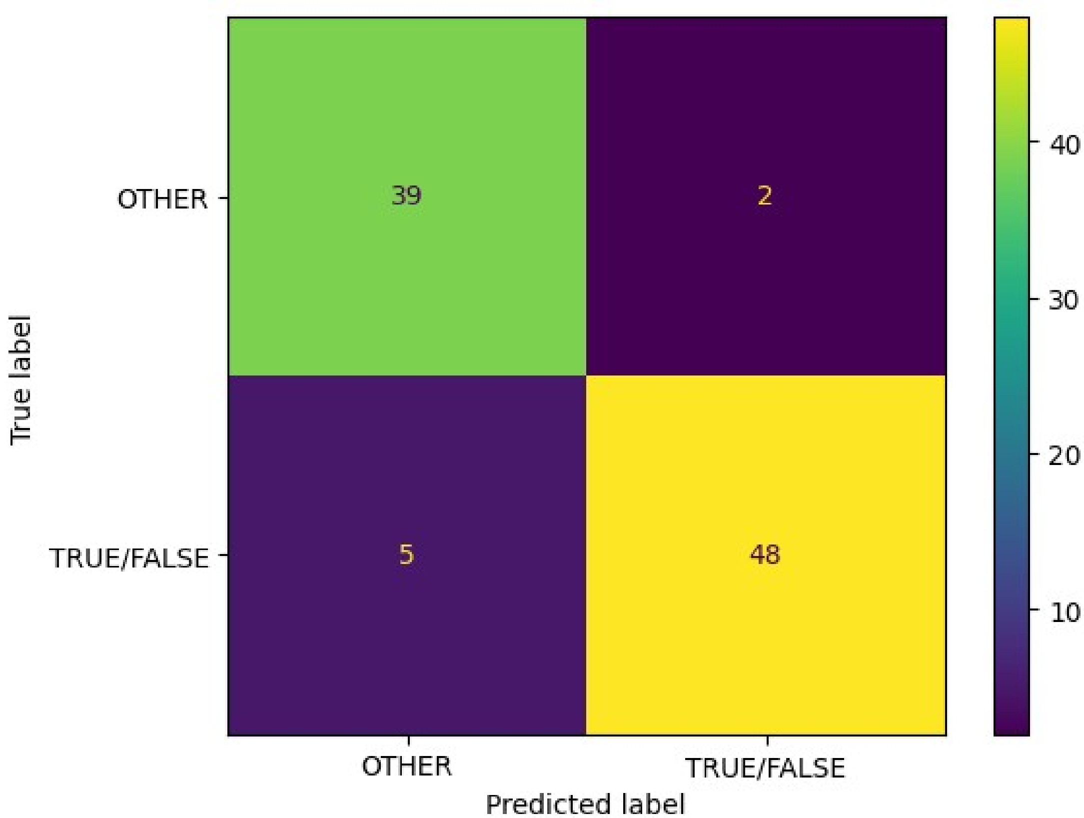

OTHER
TRUE/FALSE
39
2
3
50
True label
Predicted label
OTHER
TRUE/FALSE
50
40
30
20
10

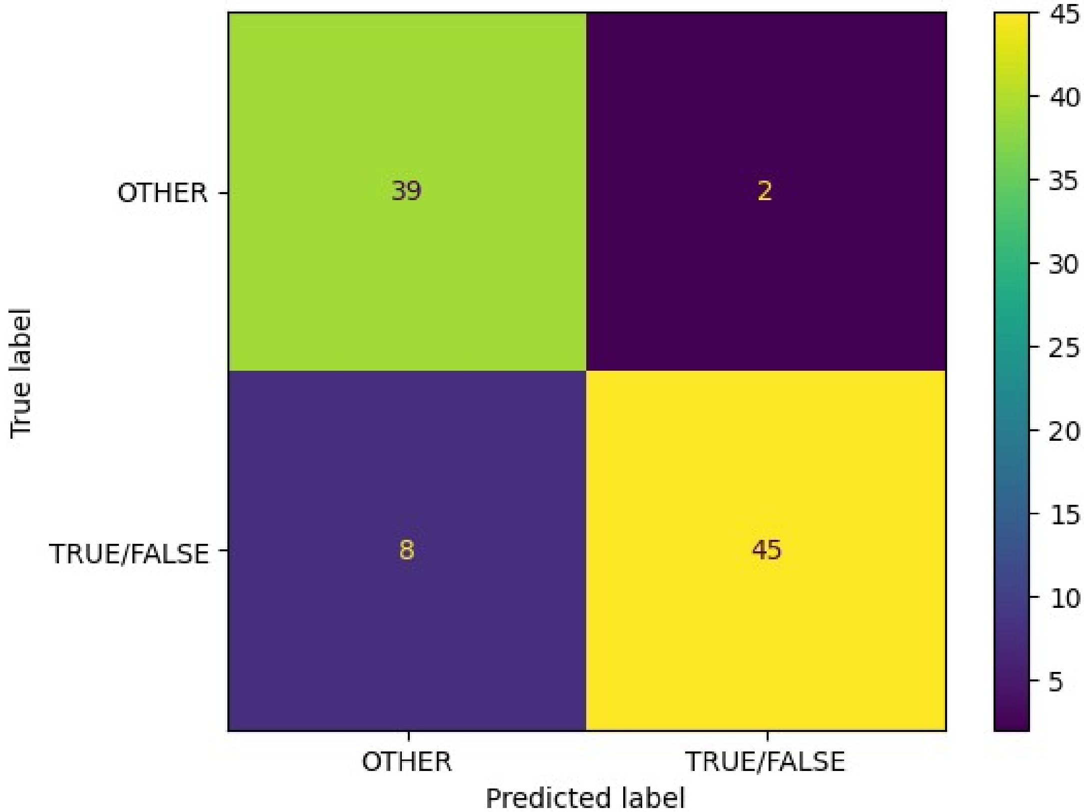

OTHER
TRUE/FALSE
39
2
4
49
True label
Predicted label
OTHER
TRUE/FALSE
40
30
20
10

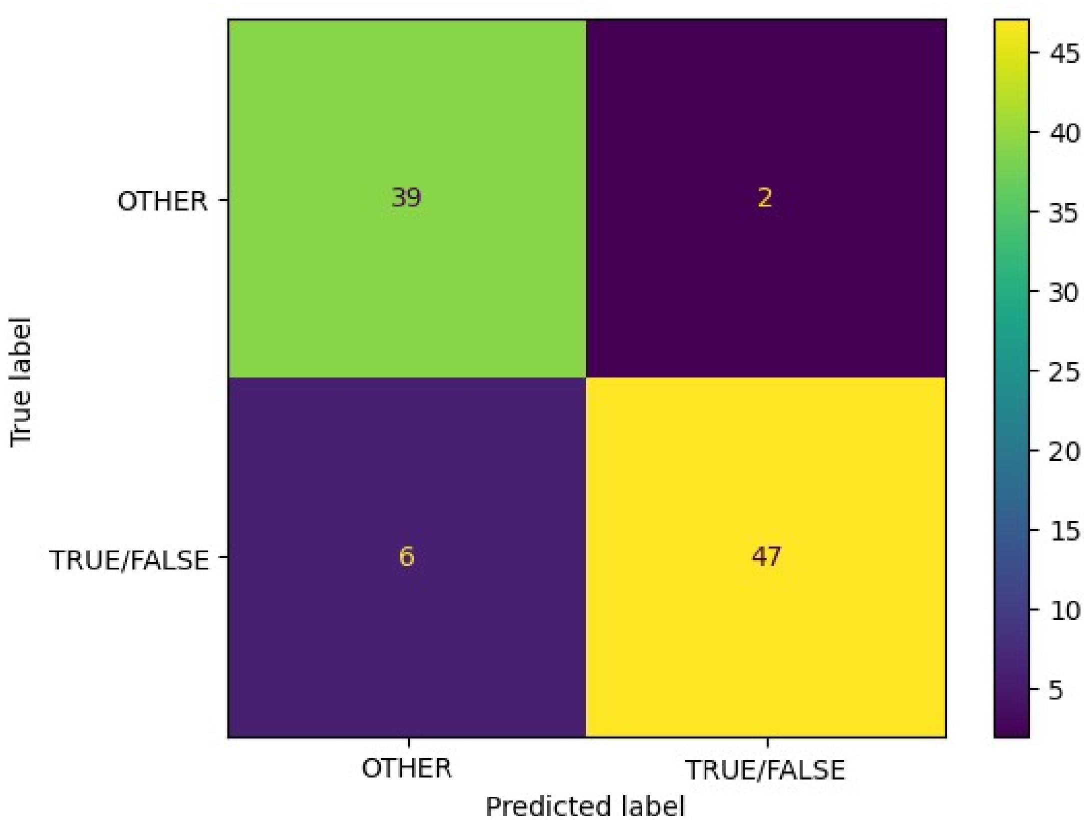

39
2
5
48
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
40
30
20
10

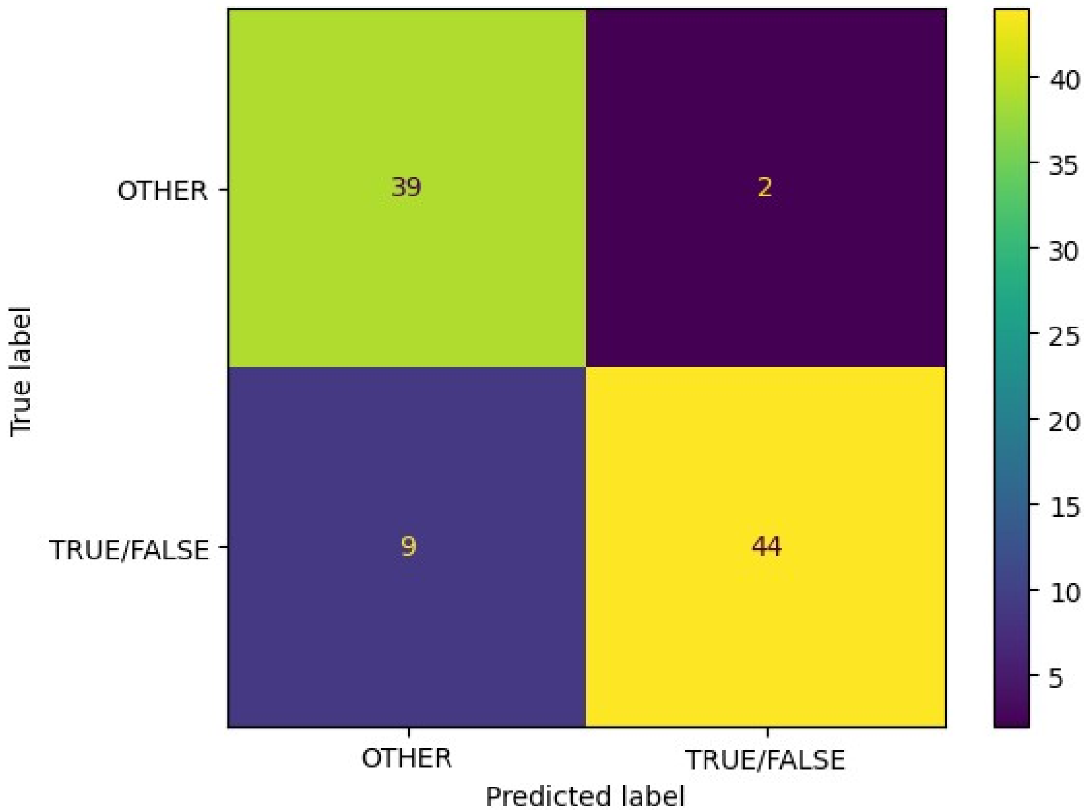

39
2
7
46
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
45
40
35
30
25
20
15
10
5


39
2
9
44
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
40
35
30
25
20
15
10
5


39
2
5
48
OTHER
TRUE/FALSE
True label
OTHER
TRUE/FALSE
Predicted label
40
30
20
10


OTHER
TRUE/FALSE
True label
39
2
8
45
OTHER
TRUE/FALSE
Predicted label
45
40
35
30
25
20
15
10
5


OTHER
TRUE/FALSE
True label
39
2
6
47
OTHER
TRUE/FALSE
Predicted label
45
40
35
30
25
20
15
10
5



#Visualisation :

```
!pip install umap-learn[plot]
!pip install holoviews
!pip install -U ipykernel

from sklearn.decomposition import TruncatedSVD
import plotly.express as px
from sklearn.manifold import TSNE
# Umap
import umap.plot
from umap import UMAP

X_test_copy = X_test.copy()

tfidf=TfidfVectorizer()
vector_tfidf=tfidf.fit_transform(X_test_copy['text'])

# 2D
umap = UMAP(n_components=2, init='random', random_state=0)
projection = umap.fit_transform(vector_tfidf)

fig = px.scatter(
    projection, x=0, y=1,
    color=y_test, labels={'color': 'RATING'}
```

```
)

fig.show()
```