#**CLASSIFICATION : TRUE VS FALSE VS OTHER VS MIXTURE (Première version)**

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

On a d'abord classé selon les quatre classes , mais vu que les accuracy et autres mesures étaient trop faibles , on a du entamer une autre technique de classification pour ces quatre classes

```
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 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
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
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
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
from tabulate import tabulate
import numpy as np
import time
from sklearn.metrics._plot.confusion_matrix import
ConfusionMatrixDisplay
```

autorisation

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

```
Mounted at /content/gdrive/
```

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
 bonFakeNEWS4.ipynb
'BON_TRUE FALSE_vs_OTHER_entités_nommées.ipynb'
'Copie de FakeNEWS.ipynb'
'Copie de True_False_Other_Mixture.ipynb'
'Copie de TRUE FALSE_vs_OTHER_entités_nommées.ipynb'
'Copie de  Vrai_Faux._entites_marche_bien.ipynb'
 FakeNewsLastVersion.ipynb
 ml_entiteesNommeesTest.ipynb
'Traitement sémantique'/
 True_False_Other_Mixture_final.ipynb
 Untitled0.ipynb
 version2ml_entiteesNommeesTest.ipynb
'VRAI FAUX OTHER MIXTURE_avec_entites_nommees.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 ..

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

```
#.................................................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]
```

```python
    # 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

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()
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

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
#...................................................................
....................................................................
...........

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

```
dftrain1 =
pd.read_csv("/content/gdrive/MyDrive/projet_ML/newsTrain2.csv",
names=['id','text','title','rating'], header=0,sep=',',
encoding='utf8')
dftrain1.reset_index(drop = True, inplace = True)

dftrain2 = pd.read_csv("/content/gdrive/MyDrive/projet_ML/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([dftrain1, dftrain2], ignore_index=True)


print("Echantillon de mon dataset \n")
print(dftrain.sample(n=10))
print("\n")
print("Quelques informations importantes \n")
dftrain.info()
print("\n")
X_text=dftrain["text"]
X_title=dftrain["title"]


print("le texte est")
display(X_text)
print("\n")
print("le titre est")
display(X_title)
print("\n")
y=dftrain.iloc[0:,-1]
print("voici la dernière case de rating")
display(y)
print("\n")
print("la taille de X_text est",X_text.shape)
```

```python
print("\n")
print("la taille de y_train est " ,y.shape)
print("\n")
y = y.str.lower()
print("Les valeurs de true et false sont:\n", y.value_counts())
```

```
Echantillon de mon dataset

            id                                               text  \
398   5978dc76  Hi, my name is Scott C. Waring and I wrote a f...
487   9ba64668  Murderers of Stephen Lawrence and Garry Newlov...
1132  45d2e875  Tens of thousands of workers will share in a £...
1266  c3dea290  Home Alone 2: Lost in New York is full of viol...
941   9f10a8a9  It was an accurate and judicious answer, so na...
1474  d46e2ede  Share Messenger Tweet Email Whatsapp reddit  A...
649   c4f8a375  I have just been to Buckingham Palace, where H...
442   972048cd  Snowflake students at Oxford University are th...
2445  4ebbbb28  Global warming skeptics sometimes say rising t...
2304  69e7cad4  GEORGIA BECOMES FIRST STATE TO BAN MUSLIM CULT...

                                                  title   rating
398   UFO SIGHTINGS DAILY: Thousands witness UFO Ove...    FALSE
487   Climate Change Wackos Exposed in California Court  mixture
1132  Workers 'to share in £1.8billion pay rise' if ...  mixture
1266  CBC Cuts Donald Trump's 'Home Alone 2' Cameo O...  mixture
941                                A 62% Top Tax Rate?    other
1474  Vladimir Putin’s daughter DIES after second do...    FALSE
649   BRAZIL'S health authority has confirmed a volu...     TRUE
442   A look at Congressman Conor Lamb’s voting reco...    FALSE
2445  Climate scientists drive stake through heart o...  mixture
2304  GEORGIA BECOMES FIRST STATE TO BAN MUSLIM CULT...    FALSE


Quelques informations importantes

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2528 entries, 0 to 2527
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      2528 non-null   object
 1   text    2528 non-null   object
 2   title   2482 non-null   object
 3   rating  2528 non-null   object
dtypes: object(4)
memory usage: 79.1+ KB


le texte est
```

```
0       Distracted driving causes more deaths in Canad...
1       Missouri politicians have made statements afte...
2       Home Alone 2: Lost in New York is full of viol...
3       But things took a turn for the worse when riot...
4       It’s no secret that Epstein and Schiff share a...
                              ...
2523    More than four million calls to the taxman are...
2524    More under-18s are being taken to court for se...
2525    The Government’s much vaunted Help to Buy Isa ...
2526    The late Robin Williams once called cocaine “G...
2527    The late Robin Williams once called cocaine “G...
Name: text, Length: 2528, dtype: object
```

le titre est

```
0       You Can Be Fined $1,500 If Your Passenger Is U...
1           Missouri lawmakers condemn Las Vegas shooting
2       CBC Cuts Donald Trump's 'Home Alone 2' Cameo O...
3       Obama’s Daughters Caught on Camera Burning US ...
4       Leaked Visitor Logs Reveal Schiff’s 78 Visits ...
                              ...
2523    Taxman fails to answer four million calls a ye...
2524    Police catch 11-year-olds being used to sell d...
2525    Help to Buy Isa scandal: 500,000 first-time bu...
2526             A coke-snorting generation of hypocrites
2527             A coke-snorting generation of hypocrites
Name: title, Length: 2528, dtype: object
```

voici la dernière case de rating

```
0         FALSE
1       mixture
2       mixture
3         FALSE
4         FALSE
         ...
2523       TRUE
2524       TRUE
2525      FALSE
2526       TRUE
2527       TRUE
Name: rating, Length: 2528, dtype: object
```

la taille de X_text est (2528,)

```
la taille de y_train est  (2528,)


Les valeurs de true et false sont:
 false      1156
mixture     716
true        422
other       234
Name: rating, dtype: int64
```

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, OTHER et MIXTURE de telle sorte que le nombre de lignes de chacune soit = au nbr de lignes de celle avec le plus petit nbr de lignes. et on mélange le DataFrame.

```
# Compter le nombre d'observations dans chaque catégorie
false_count = dftrain['rating'].value_counts()['FALSE']
mixture_count = dftrain['rating'].value_counts()['mixture']
true_count = dftrain['rating'].value_counts()['TRUE']
other_count = dftrain['rating'].value_counts()['other']

# Trouver le nombre minimum d'observations parmi les catégories
min_count = min(false_count, mixture_count, true_count, other_count)

# Sous-échantillonner les catégories pour équilibrer les quantités
false_sampled = dftrain[dftrain['rating'] ==
'FALSE'].sample(min_count, random_state=42)
mixture_sampled = dftrain[dftrain['rating'] ==
'mixture'].sample(min_count, random_state=42)
true_sampled = dftrain[dftrain['rating'] == 'TRUE'].sample(min_count,
random_state=42)
other_sampled = dftrain[dftrain['rating'] ==
'other'].sample(min_count, random_state=42)
print(false_sampled.shape)
print(true_sampled.shape)

# Concaténer les échantillons pour obtenir un nouveau dataframe
équilibré
dftrain = pd.concat([false_sampled, mixture_sampled, true_sampled,
other_sampled])

# Mélanger aléatoirement les données
dftrain = dftrain.sample(frac=1, random_state=42)
print(dftrain)
X_text=dftrain["text"]
X_title=dftrain["title"]
y=dftrain.iloc[0:,-1]
print("la taille de X_text est",X_text.shape)
print("\n")
```

```python
print("la taille de X_title est",X_title.shape)
print("\n")
print("la taille de y_train est " ,y.shape)
print("\n")
print("les valeurs de TRUE et FALSE maintenant sont
" ,y.value_counts())
```

```
(234, 4)
(234, 4)
            id                                               text  \
2504  c9a710dc  Hillary Clinton's plane passes over Manhattan ...
261   a7b20877  Rashida Tlaib is busy at work during a nationa...
46    fb721890  Natural News The oldest magazine in the United...
1546  ed8a09ac  Ministers are undermining trust in foreign aid...
1781  f454e71d  Today the Education Policy Institute's Indepen...
...        ...                                                ...
1543  3886ead8  The bombshell claim comes from over 20 hours o...
422   da3319cc  This is a rush transcript from Fox News Sunday...
1102  7b9e930d  The use of cocaine in Britain has doubled in s...
2382  48026a71  A ndy Murray served up an ace to John Inverdal...
1325  31d33510  Though the whole world relies on RT-PCR to “di...

                                                  title   rating
2504  Hillary Clinton Boards The Climate Crisis Trai...  mixture
261   Tlaib Files Lawsuit to Ban the American Flag i...    FALSE
46    Still think 5G is harmless? Scientific America...    FALSE
1546  Ministers are undermining trust in foreign aid...     TRUE
1781  Apocalyptic Sea-Level Rise—Just a Thing of the...     TRUE
...                                                 ...      ...
1543  Breaking: Breonna Taylor’s boyfriend says SHE ...    FALSE
422   Pruitt defends decision to withdraw from Paris...  mixture
1102  Britain's cocaine use doubles in last seven ye...    other
2382  Andy Murray aces John Inverdale after BBC pres...  mixture
1325   COVID19 PCR Tests are Scientifically Meaningless    FALSE

[936 rows x 4 columns]
la taille de X_text est (936,)


la taille de X_title est (936,)


la taille de y_train est  (936,)


les valeurs de TRUE et FALSE maintenant sont  mixture    234
FALSE      234
TRUE       234
other      234
Name: rating, dtype: int64
```

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

```
X=dftrain.iloc[0:, 1:3]
print(X)

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size =
0.2,random_state=8)
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  \
2504  Hillary Clinton's plane passes over Manhattan ...
261   Rashida Tlaib is busy at work during a nationa...
46    Natural News The oldest magazine in the United...
1546  Ministers are undermining trust in foreign aid...
1781  Today the Education Policy Institute's Indepen...
...                                                 ...
1543  The bombshell claim comes from over 20 hours o...
422   This is a rush transcript from Fox News Sunday...
1102  The use of cocaine in Britain has doubled in s...
2382  A ndy Murray served up an ace to John Inverdal...
1325  Though the whole world relies on RT-PCR to “di...

                                                  title
2504  Hillary Clinton Boards The Climate Crisis Trai...
261   Tlaib Files Lawsuit to Ban the American Flag i...
46    Still think 5G is harmless? Scientific America...
1546  Ministers are undermining trust in foreign aid...
1781  Apocalyptic Sea-Level Rise—Just a Thing of the...
...                                                 ...
1543  Breaking: Breonna Taylor’s boyfriend says SHE ...
422   Pruitt defends decision to withdraw from Paris...
1102  Britain's cocaine use doubles in last seven ye...
2382  Andy Murray aces John Inverdale after BBC pres...
1325   COVID19 PCR Tests are Scientifically Meaningless

[936 rows x 2 columns]
X_train is                                                    text  \
1727  GETTY - STOCK IMAGE Official figures revealed ...
76    The robots are coming, and they can read.  Art...
335   Florida Rep. Debbie Wasserman Schultz, the Dem...
2060  In the scramble to make sense of the post-inau...
1781  Today the Education Policy Institute's Indepen...
...                                                 ...
359   Latest Breaking News: Martial Law Imminent Gen...
2354  Another earthquake has hit a fracking site in ...
154   Do you support Trump YES NO Created with  The ...
```

```
662   This morning, millions awoke to the news that ...
1292  WHO: You Do NOT Need to Wear a Mask  January 2...

                                                  title
1727  What if We Stopped Pretending the Climate Apoc...
76    Computers are getting better than humans at re...
335       Wasserman Schultz considering 2016 Senate bid
2060  ‘It’s the Way We Were All Born Eating’ - The N...
1781  Apocalyptic Sea-Level Rise—Just a Thing of the...
...                                                 ...
359   Marine Corps. Rebukes Pelosi: “WE DON’T WORK F...
2354  Fracking halted again in Lancashire after 17th...
154   ‘Bionic Man’ Lee Majors Dead At 83;$6 Million ...
662   Re: Trans-Pecos Pipeline, LLC Presidio Crossin...
1292      WHO now saying You do not need to Wear a Mask

[748 rows x 2 columns]
y_train is 1727      other
76        FALSE
335     mixture
2060       TRUE
1781       TRUE
         ...
359       FALSE
2354    mixture
154       FALSE
662     mixture
1292      FALSE
Name: rating, Length: 748, dtype: object
X_test is                                                text  \
2335  Get our daily royal round-up direct to your in...
709   The National Oceanic and Atmospheric Administr...
1171  The oldest and thickest sea ice in the Arctic ...
1182  Huge reductions in meat-eating are essential t...
711   Taco Bell Recall: 2.3 Million Pounds of Beef P...
...                                                 ...
112   Figures released today, the 25th anniversary o...
2032  Bois State University told The Daily Wire Thur...
2089  (CNN) The stakes were high when Bernie Sanders...
1224  Climate change: How 1.5C could change the worl...
629   Three in four labour wards have no consultants...

                                                  title
2335  Princess Eugenie's wedding: Huge bill for taxp...
709   Enemies Of APC Sponsoring Boko Haram Insurgenc...
1171  Arctic’s strongest sea ice breaks up for first...
1182  Huge reduction in meat-eating ‘essential’ to a...
711               Travis Rowley: Race-Baiting Democrats
...                                                 ...
112   “Catastrophic” effect of rail privatisation re...
```

```
2032  Billionaire Jeffrey Epstein arrested and accus...
2089  Julia Roberts: ‘Michelle Obama Isn’t Fit To Cl...
1224  Final call to save the world from 'climate cat...
629   In essence, the larger question being asked is...

[188 rows x 2 columns]
y_test is 2335       TRUE
709       other
1171       TRUE
1182       TRUE
711        TRUE
         ...
112        TRUE
2032      other
2089      other
1224    mixture
629        TRUE
Name: rating, Length: 188, dtype: object
```

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

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.

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


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


# Liste des modèles à tester
models = [
    ('MultinomialNB', MultinomialNB()),
    ('LogisticRegression', LogisticRegression(random_state=42))
]


models.append(('KNN', KNeighborsClassifier()))
```

```
models.append(('CART', DecisionTreeClassifier()))
models.append(('RF', RandomForestClassifier()))
models.append(('SVM', SVC()))


# 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))
    #pipeline.fit(X_train_text,y_train)
all_results=[]
scores=[]
for p in pipelines:
    print(p[1])
    # cross validation en 10 fois
    kfold = KFold(n_splits=10,random_state=seed,shuffle=True)

#     print ("Evaluation de ",p)
    start_time = time.time()
    # application de la classification
    cv_results = cross_val_score(p[1],X_train_text,y_train, cv=kfold,
scoring=score)
    #print("Pour le classifieur",p[0],"on a un score
de",cv_results.mean(),"et un écart type de",cv_results.std())
    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)
    # affichage des résultats
#print ('\nLe meilleur resultat : ',max(results))
```

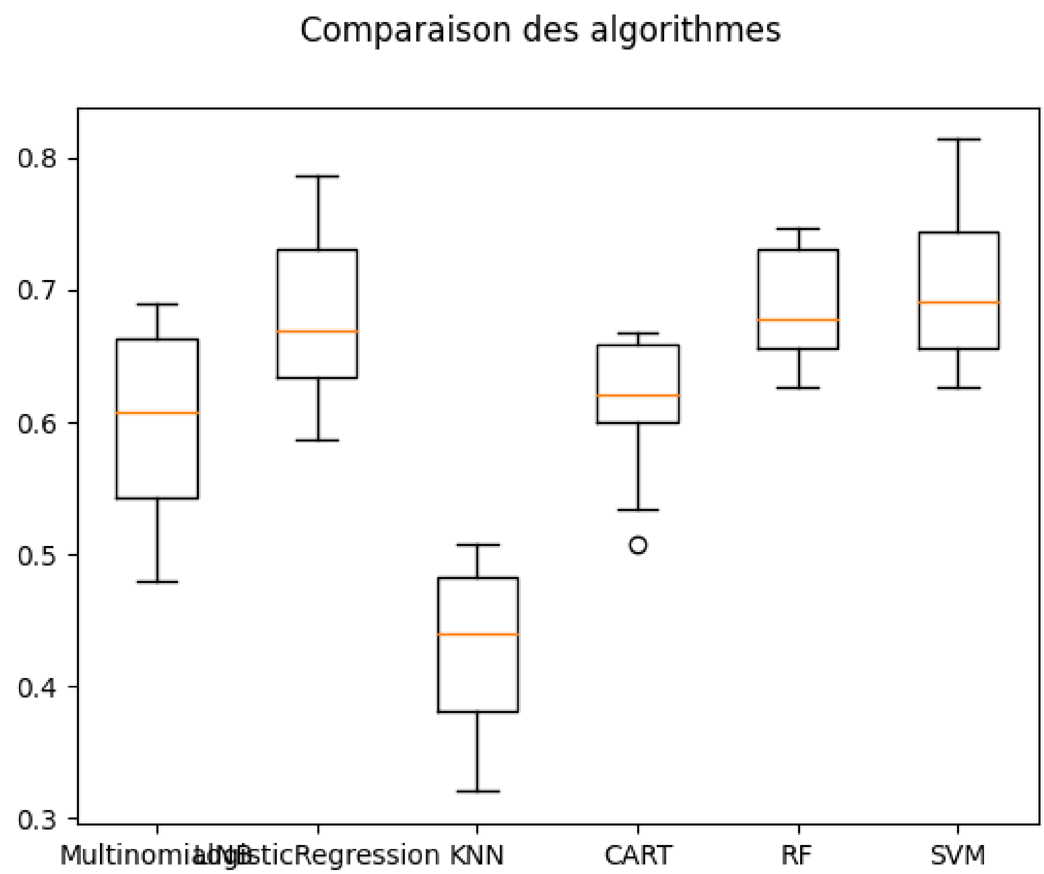
```
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('MultinomialNB', MultinomialNB())])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('LogisticRegression',
LogisticRegression(random_state=42))])
Pipeline(steps=[('normalize',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

```
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('CART', DecisionTreeClassifier())])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('RF', RandomForestClassifier())])
Pipeline(steps=[('normalize', TextNormalizer()), ('tfidf',
TfidfVectorizer()),
                ('SVM', SVC())])
all resultats [('SVM', 0.7018198198198198, 0.05717934768885454),
('RF', 0.6884684684684685, 0.03982870554317204),
('LogisticRegression', 0.6804504504504504, 0.05840602448488134 4),
('CART', 0.6123063063063062, 0.05225399500044966), ('MultinomialNB',
0.6016756756756757, 0.06947473219876249), ('KNN', 0.43052252252252254,
0.05930486044058516 6)]
```

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

```
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 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_SVC = []
X_test_text_SVC = []

X_train_text_RandomForestClassifier = []
X_test_text_RandomForestClassifier = []


for name, pipeline in all_models :

X_train_text_SVC.append(pipeline.fit_transform(X_train['text']).toarra
y())

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['t
ext']).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 = eval('X_train_text_' + model_name)
    X_test_text = eval('X_test_text_' + model_name)
    for i in range (len(X_train_text)):
```

```python
      grid_search = GridSearchCV(model, params[model_name], n_jobs=-1,
verbose=1,scoring=score)
      print("grid search fait")
      print("X_train",X_train_text[i].shape)
      print("y_train",y_train.shape)
      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])
      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
X_train (748, 26615)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.634

meilleur estimateur SVC(kernel='linear', random_state=42)

Accuracy : 0.707
Classification Report
              precision    recall  f1-score   support

       FALSE    0.69444   0.53191   0.60241        47
        TRUE    0.73469   0.66667   0.69903        54
     mixture    0.60465   0.63415   0.61905        41
       other    0.76667   1.00000   0.86792        46

    accuracy                        0.70745       188
   macro avg    0.70011   0.70818   0.69710       188
weighted avg    0.70409   0.70745   0.69876       188

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 'scale'
	kernel: 'linear'
grid search fait
X_train (748, 22447)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.638

meilleur estimateur SVC(kernel='linear', random_state=42)
```

```
Accuracy : 0.649
Classification Report
              precision    recall  f1-score   support

       FALSE    0.61765   0.44681   0.51852        47
        TRUE    0.68000   0.62963   0.65385        54
     mixture    0.53191   0.60976   0.56818        41
       other    0.73684   0.91304   0.81553        46

    accuracy                        0.64894       188
   macro avg    0.64160   0.64981   0.63902       188
weighted avg    0.64602   0.64894   0.64089       188

Ensemble des meilleurs paramètres :
     C: 1.0
     gamma: 'scale'
     kernel: 'linear'
grid search fait
X_train (748, 22307)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.642

meilleur estimateur SVC(kernel='linear', random_state=42)

Accuracy : 0.654
Classification Report
              precision    recall  f1-score   support

       FALSE    0.59091   0.55319   0.57143        47
        TRUE    0.69565   0.59259   0.64000        54
     mixture    0.51111   0.56098   0.53488        41
       other    0.79245   0.91304   0.84848        46

    accuracy                        0.65426       188
   macro avg    0.64753   0.65495   0.64870       188
weighted avg    0.65291   0.65426   0.65095       188

Ensemble des meilleurs paramètres :
     C: 1.0
     gamma: 'scale'
     kernel: 'linear'
grid search fait
X_train (748, 15175)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.643
```

```
meilleur estimateur SVC(C=10, random_state=42)

Accuracy : 0.707
Classification Report
              precision    recall  f1-score   support

       FALSE    0.57778   0.55319   0.56522        47
        TRUE    0.75000   0.61111   0.67347        54
     mixture    0.62222   0.68293   0.65116        41
       other    0.85185   1.00000   0.92000        46

    accuracy                        0.70745       188
   macro avg    0.70046   0.71181   0.70246       188
weighted avg    0.70400   0.70745   0.70186       188

Ensemble des meilleurs paramètres :
     C: 10
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
X_train (748, 22447)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.688

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

Accuracy : 0.723
Classification Report
              precision    recall  f1-score   support

       FALSE    0.62000   0.65957   0.63918        47
        TRUE    0.81081   0.55556   0.65934        54
     mixture    0.62500   0.85366   0.72165        41
       other    0.88889   0.86957   0.87912        46

    accuracy                        0.72340       188
   macro avg    0.73617   0.73459   0.72482       188
weighted avg    0.74169   0.72340   0.72166       188

Ensemble des meilleurs paramètres :
     C: 2
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
X_train (748, 22307)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.697
```

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

Accuracy : 0.734
Classification Report
              precision    recall  f1-score   support

       FALSE    0.69565   0.68085   0.68817        47
        TRUE    0.84211   0.59259   0.69565        54
     mixture    0.54839   0.82927   0.66019        41
       other    0.95238   0.86957   0.90909        46

    accuracy                        0.73404       188
   macro avg    0.75963   0.74307   0.73828       188
weighted avg    0.76842   0.73404   0.73827       188

Ensemble des meilleurs paramètres :
      C: 1
      gamma: 'scale'
      kernel: 'rbf'
grid search fait
X_train (748, 15175)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.686

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

Accuracy : 0.745
Classification Report
              precision    recall  f1-score   support

       FALSE    0.67347   0.70213   0.68750        47
        TRUE    0.84615   0.61111   0.70968        54
     mixture    0.59649   0.82927   0.69388        41
       other    0.93023   0.86957   0.89888        46

    accuracy                        0.74468       188
   macro avg    0.76159   0.75302   0.74748       188
weighted avg    0.76911   0.74468   0.74698       188

Ensemble des meilleurs paramètres :
      C: 2
      gamma: 'scale'
      kernel: 'rbf'
grid search fait
X_train (748, 26615)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

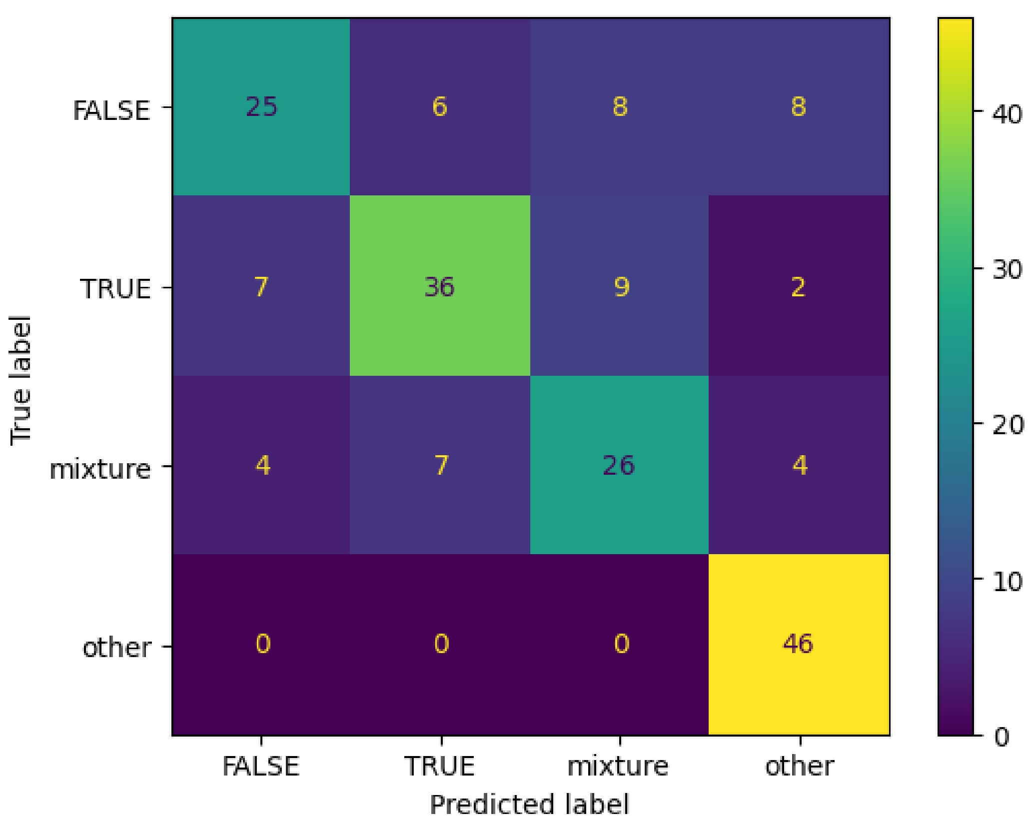
```
meilleur score 0.690

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

Accuracy : 0.745
Classification Report
              precision    recall  f1-score   support

       FALSE    0.62745   0.68085   0.65306        47
        TRUE    0.80000   0.59259   0.68085        54
     mixture    0.65385   0.82927   0.73118        41
       other    0.93333   0.91304   0.92308        46

    accuracy                        0.74468       188
   macro avg    0.75366   0.75394   0.74704       188
weighted avg    0.75761   0.74468   0.74415       188

Ensemble des meilleurs paramètres :
	C: 2
	gamma: 'scale'
	kernel: 'rbf'
grid search fait
X_train (748, 26615)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.667

meilleur estimateur RandomForestClassifier(random_state=42)

Accuracy : 0.723
Classification Report
              precision    recall  f1-score   support

       FALSE    0.58182   0.68085   0.62745        47
        TRUE    0.72000   0.66667   0.69231        54
     mixture    0.71795   0.68293   0.70000        41
       other    0.90909   0.86957   0.88889        46

    accuracy                        0.72340       188
   macro avg    0.73221   0.72500   0.72716       188
weighted avg    0.73127   0.72340   0.72587       188

Ensemble des meilleurs paramètres :
	n_estimators: 100
	max_features: 'sqrt'
grid search fait
X_train (748, 22447)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

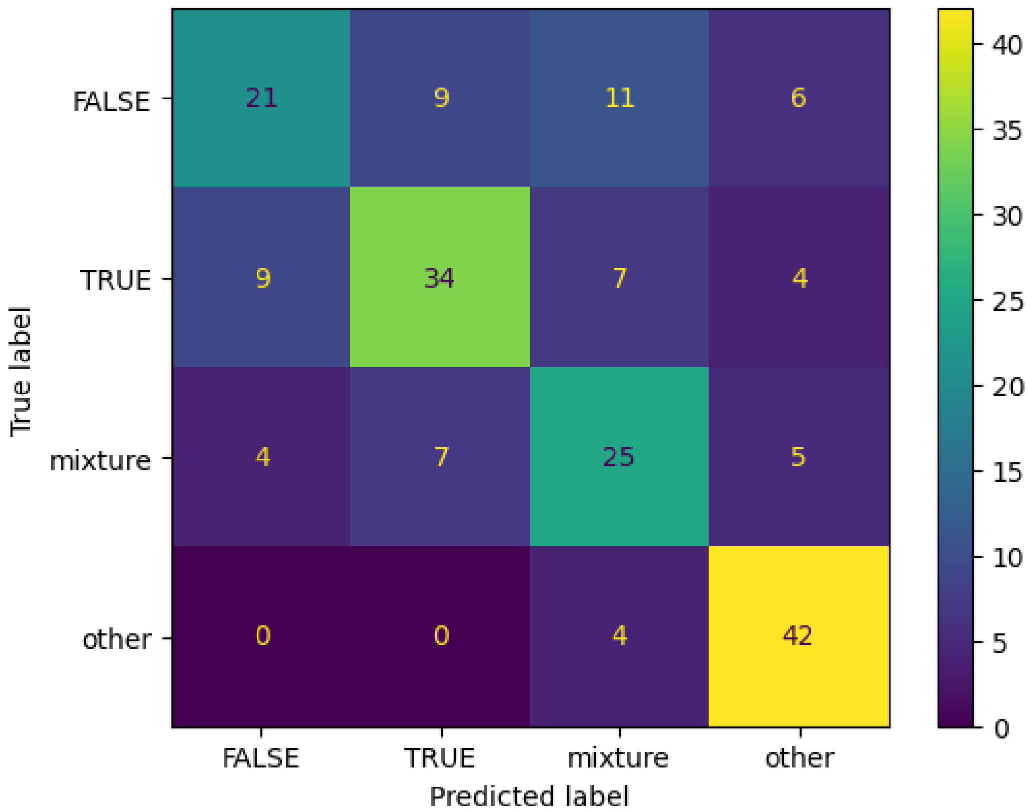
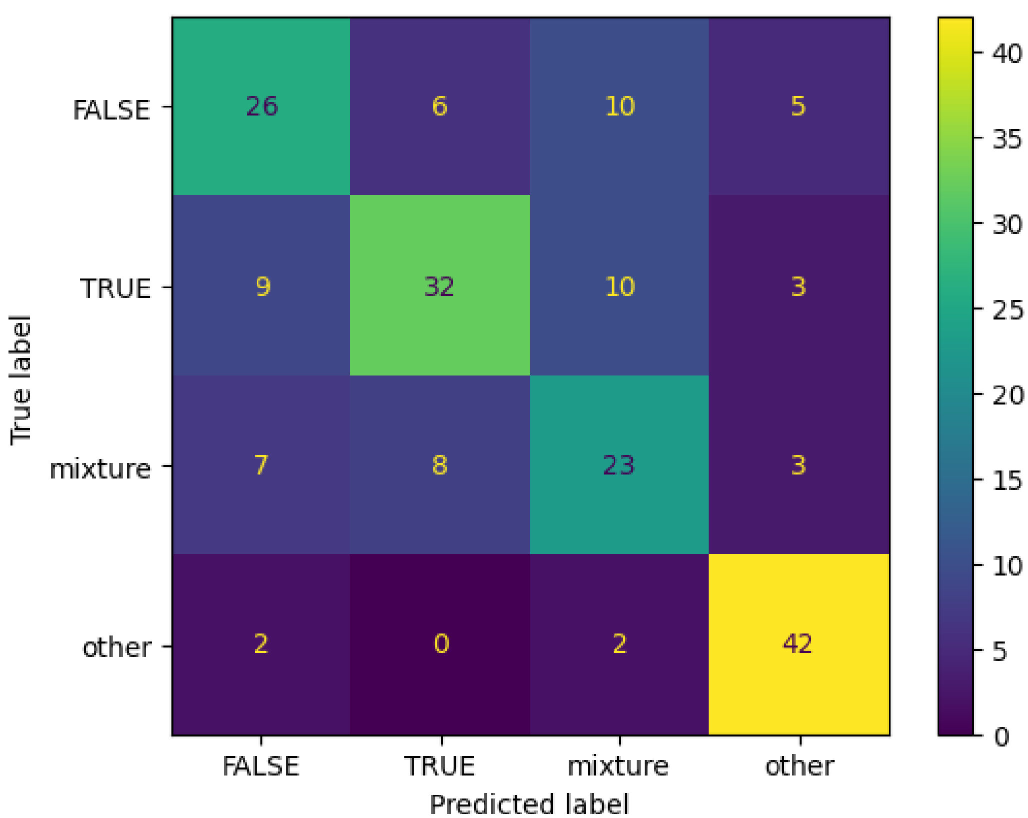
```
meilleur score 0.666

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

Accuracy : 0.676
Classification Report
              precision    recall  f1-score   support

       FALSE    0.51020   0.53191   0.52083        47
        TRUE    0.77500   0.57407   0.65957        54
     mixture    0.57407   0.75610   0.65263        41
       other    0.88889   0.86957   0.87912        46

    accuracy                        0.67553       188
   macro avg    0.68704   0.68291   0.67804       188
weighted avg    0.69285   0.67553   0.67709       188

Ensemble des meilleurs paramètres :
     n_estimators: 100
     max_features: 'log2'
grid search fait
X_train (748, 22307)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.676

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

Accuracy : 0.718
Classification Report
              precision    recall  f1-score   support

       FALSE    0.57692   0.63830   0.60606        47
        TRUE    0.70588   0.66667   0.68571        54
     mixture    0.72500   0.70732   0.71605        41
       other    0.88889   0.86957   0.87912        46

    accuracy                        0.71809       188
   macro avg    0.72417   0.72046   0.72174       188
weighted avg    0.72259   0.71809   0.71974       188

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
grid search fait
X_train (748, 15175)
y_train (748,)
```

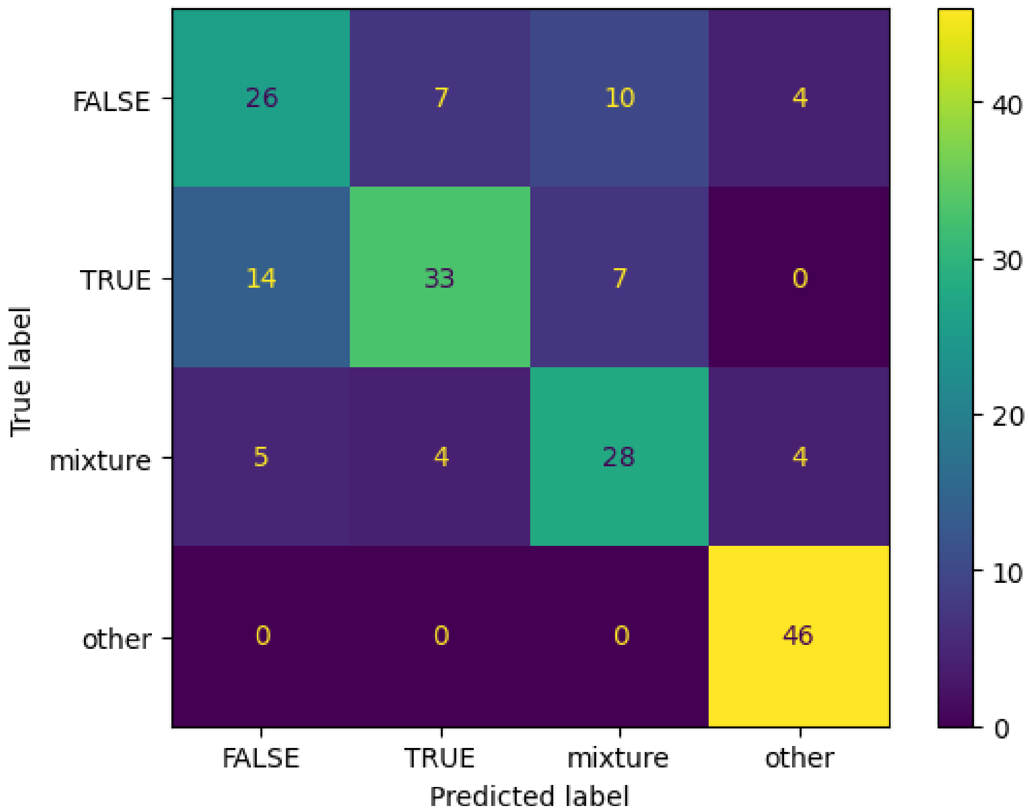
```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.668

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

Accuracy : 0.670
Classification Report
              precision    recall  f1-score   support

       FALSE    0.51923   0.57447   0.54545        47
        TRUE    0.66667   0.59259   0.62745        54
     mixture    0.64286   0.65854   0.65060        41
       other    0.86957   0.86957   0.86957        46

    accuracy                        0.67021       188
   macro avg    0.67458   0.67379   0.67327       188
weighted avg    0.67426   0.67021   0.67124       188

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
grid search fait
X_train (748, 22447)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.687

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

Accuracy : 0.660
Classification Report
              precision    recall  f1-score   support

       FALSE    0.54902   0.59574   0.57143        47
        TRUE    0.68889   0.57407   0.62626        54
     mixture    0.53191   0.60976   0.56818        41
       other    0.88889   0.86957   0.87912        46

    accuracy                        0.65957       188
   macro avg    0.66468   0.66229   0.66125       188
weighted avg    0.66862   0.65957   0.66176       188

Ensemble des meilleurs paramètres :
     n_estimators: 100
     max_features: 'log2'
grid search fait
X_train (748, 22307)
```

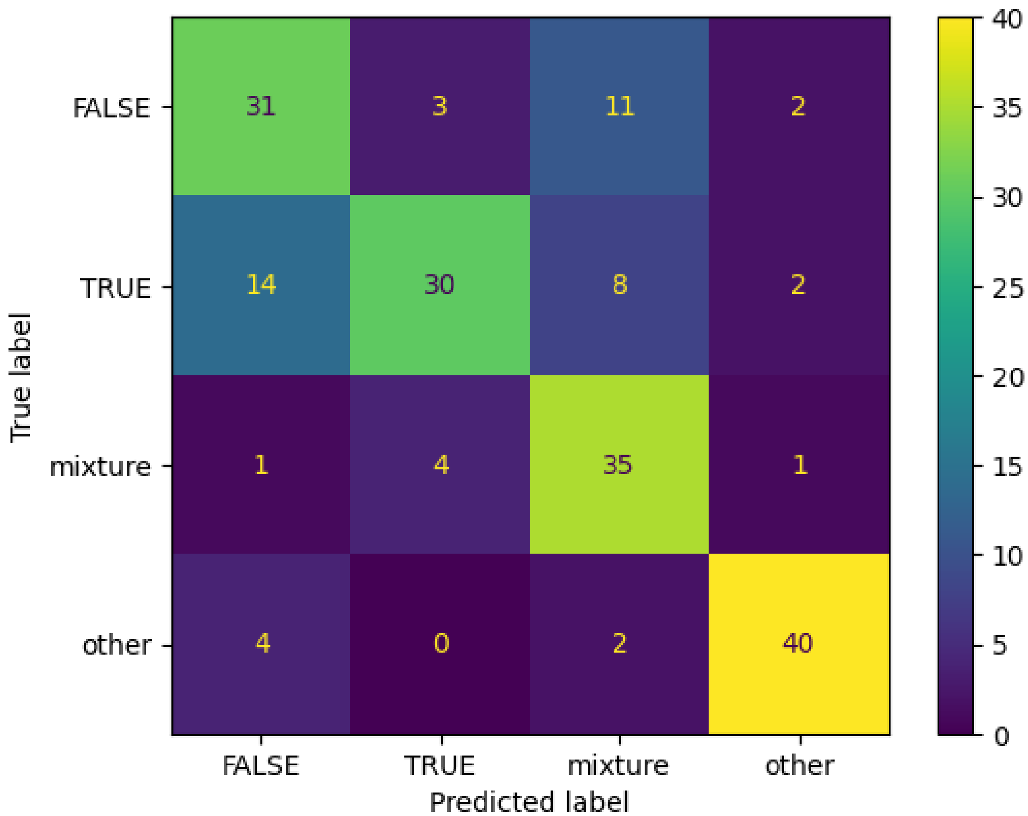
```
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.678

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

Accuracy : 0.702
Classification Report
              precision    recall  f1-score   support

       FALSE    0.53846   0.59574   0.56566        47
        TRUE    0.73333   0.61111   0.66667        54
     mixture    0.64444   0.70732   0.67442        41
       other    0.91304   0.91304   0.91304        46

    accuracy                        0.70213       188
   macro avg    0.70732   0.70680   0.70495       188
weighted avg    0.70920   0.70213   0.70339       188

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
grid search fait
X_train (748, 15175)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.690

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

Accuracy : 0.707
Classification Report
              precision    recall  f1-score   support

       FALSE    0.58824   0.63830   0.61224        47
        TRUE    0.72340   0.62963   0.67327        54
     mixture    0.61702   0.70732   0.65909        41
       other    0.93023   0.86957   0.89888        46

    accuracy                        0.70745       188
   macro avg    0.71472   0.71120   0.71087       188
weighted avg    0.71702   0.70745   0.71012       188

Ensemble des meilleurs paramètres :
     n_estimators: 200
     max_features: 'sqrt'
grid search fait
```

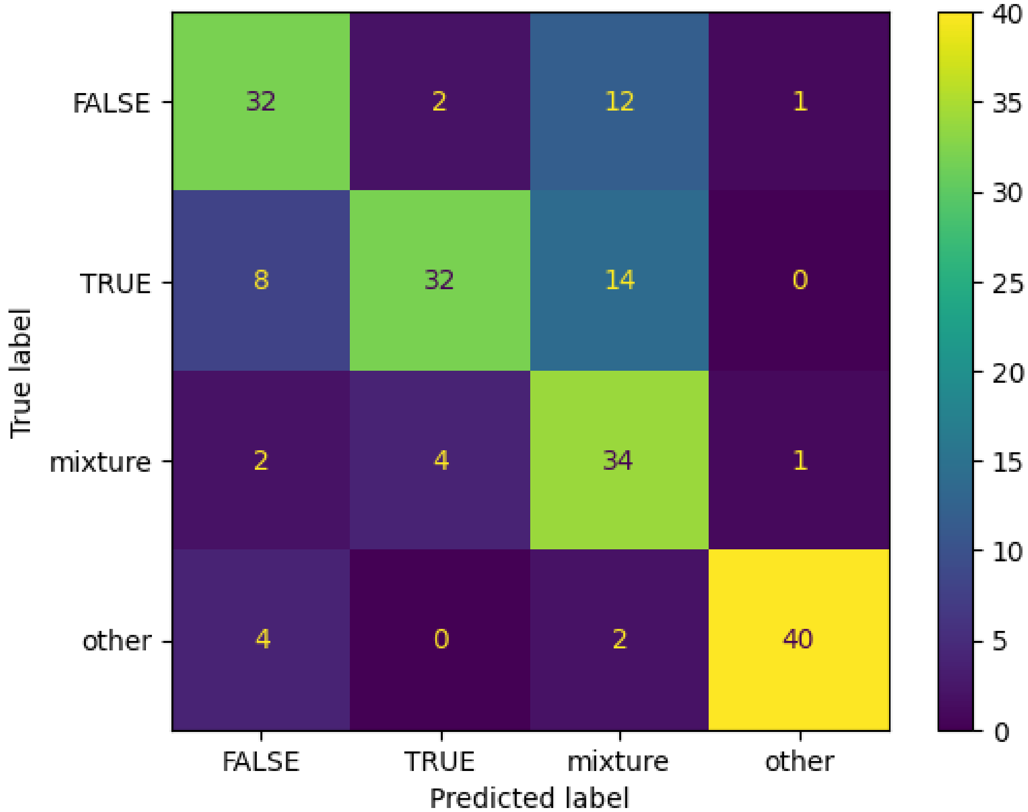
```
X_train (748, 26615)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.678

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

Accuracy : 0.713
Classification Report
              precision    recall  f1-score   support

       FALSE    0.57143   0.76596   0.65455        47
        TRUE    0.70213   0.61111   0.65347        54
     mixture    0.78125   0.60976   0.68493        41
       other    0.86957   0.86957   0.86957        46

    accuracy                        0.71277       188
   macro avg    0.73109   0.71410   0.71563       188
weighted avg    0.72768   0.71277   0.71347       188

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
```

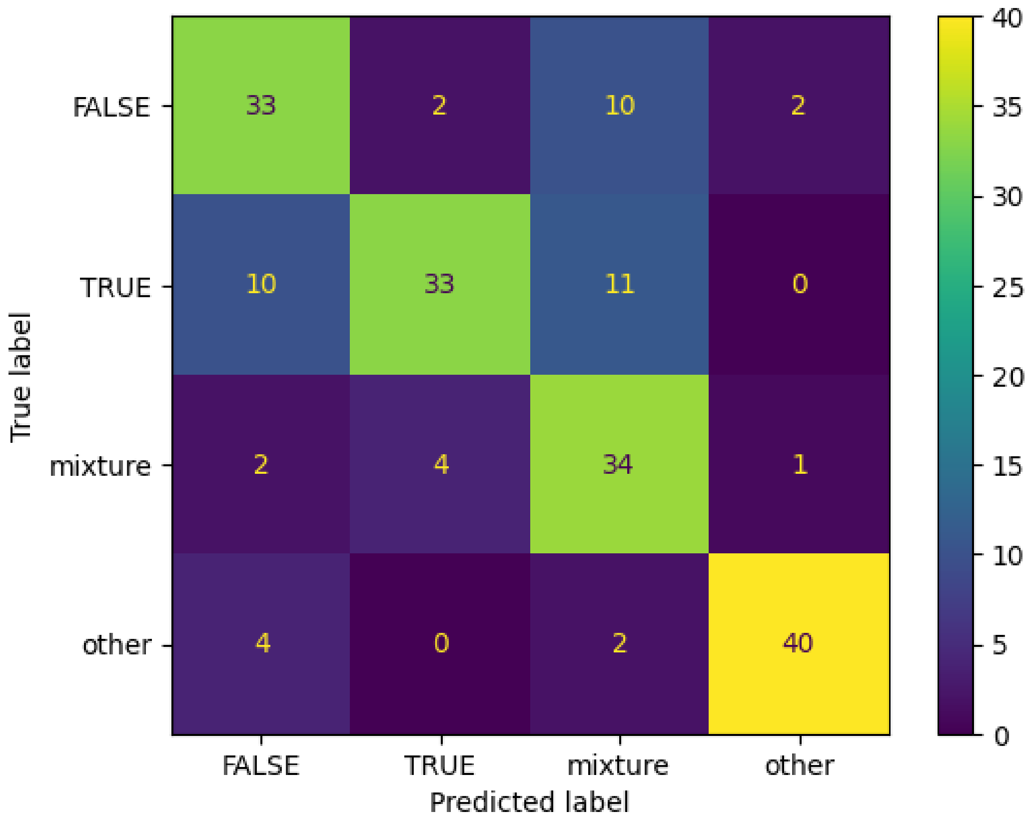


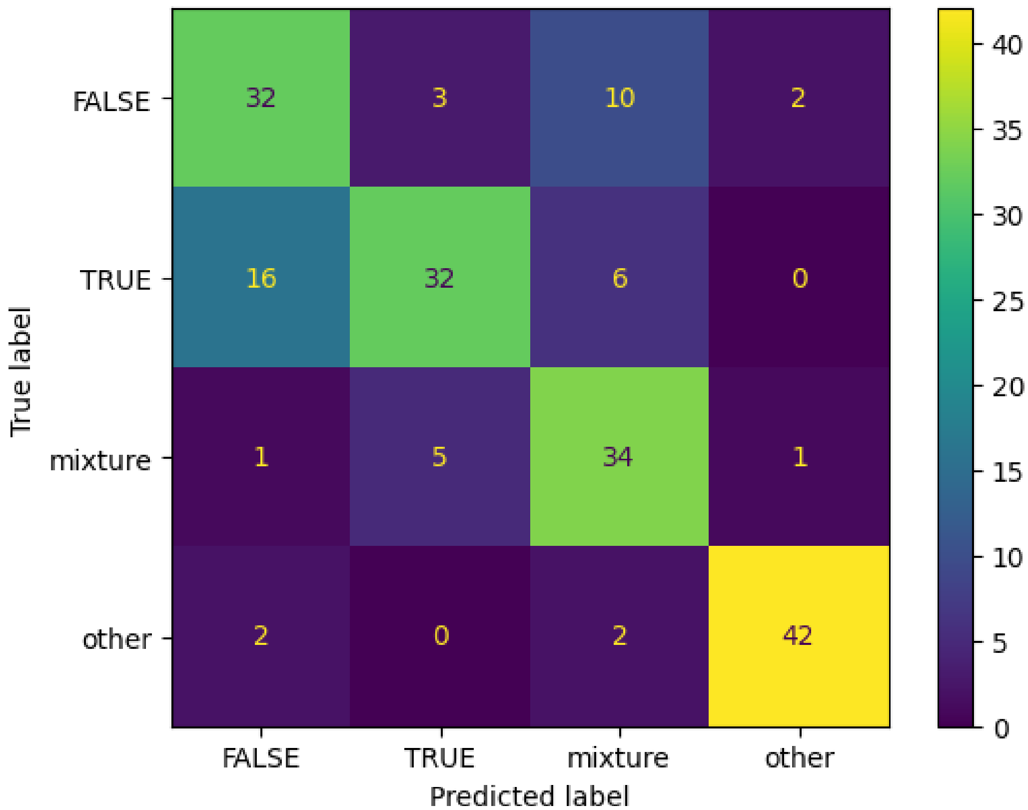

True label
Predicted label
FALSE
TRUE
mixture
other
21 9 11 6
9 34 7 4
4 7 25 5
0 0 4 42
0 5 10 15 20 25 30 35 40

True label
Predicted label
FALSE
TRUE
mixture
other
26 6 10 5
9 32 10 3
7 8 23 3
2 0 2 42
0 5 10 15 20 25 30 35 40


True label
FALSE
TRUE
mixture
other
26 7 10 4
14 33 7 0
5 4 28 4
0 0 0 46
FALSE
TRUE
mixture
other
Predicted label
0
10
20
30
40

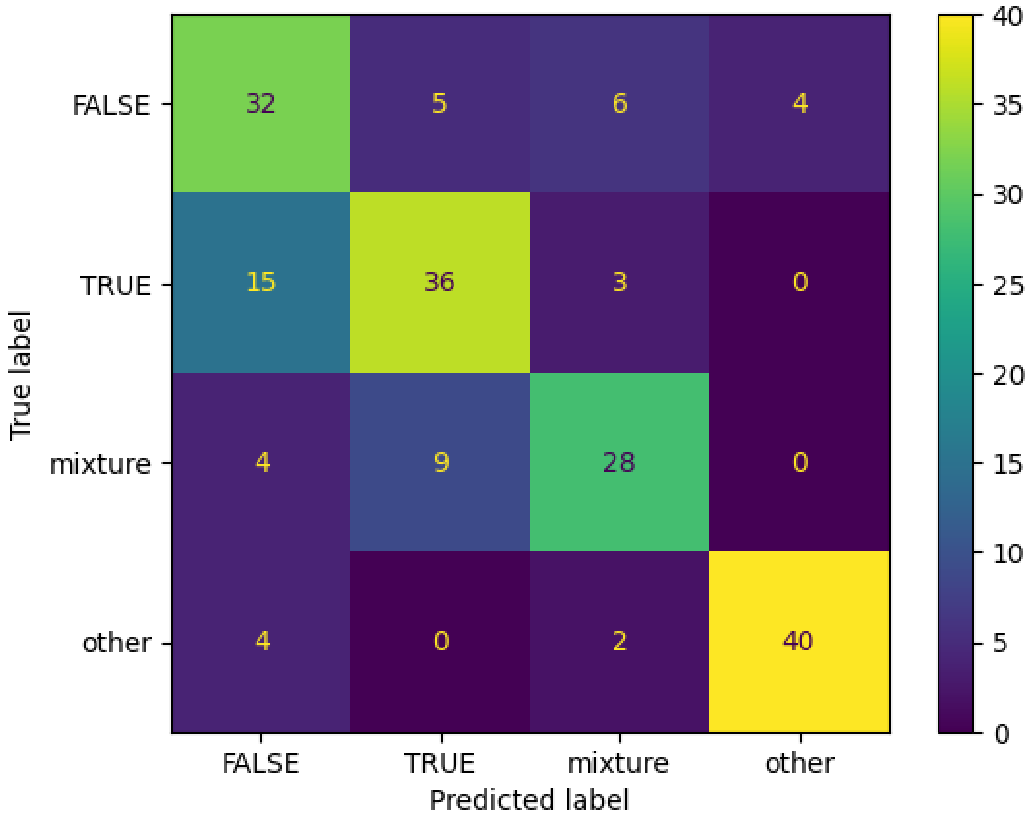

True label
FALSE
TRUE
mixture
other
31 3 11 2
14 30 8 2
1 4 35 1
4 0 2 40
FALSE
TRUE
mixture
other
Predicted label
0
5
10
15
20
25
30
35
40


True label
Predicted label
FALSE
TRUE
mixture
other
32 2 12 1
8 32 14 0
2 4 34 1
4 0 2 40
0
5
10
15
20
25
30
35
40

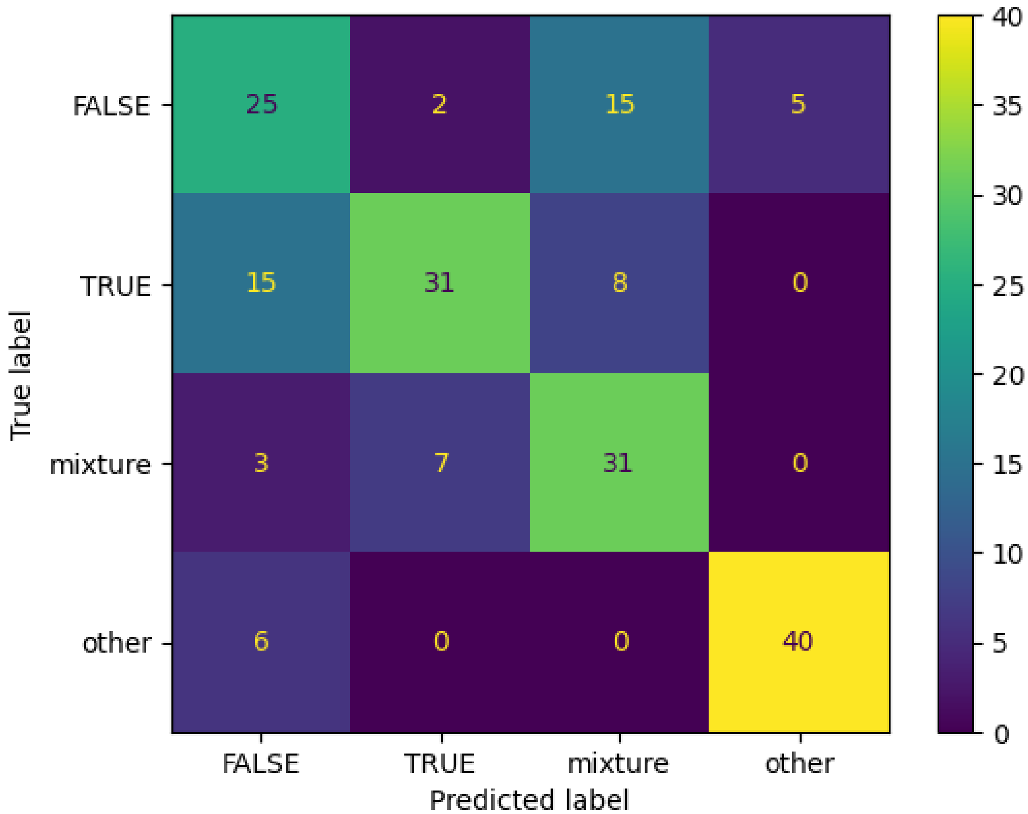

True label
Predicted label
FALSE
TRUE
mixture
other
33 2 10 2
10 33 11 0
2 4 34 1
4 0 2 40
0
5
10
15
20
25
30
35
40

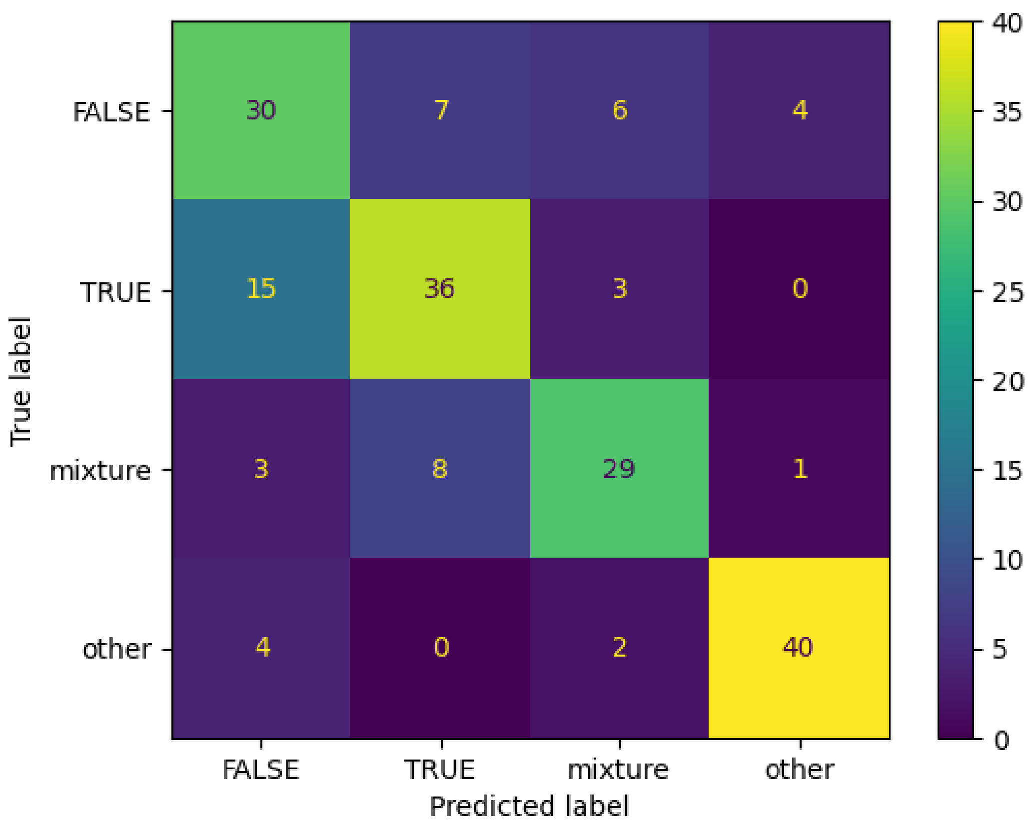


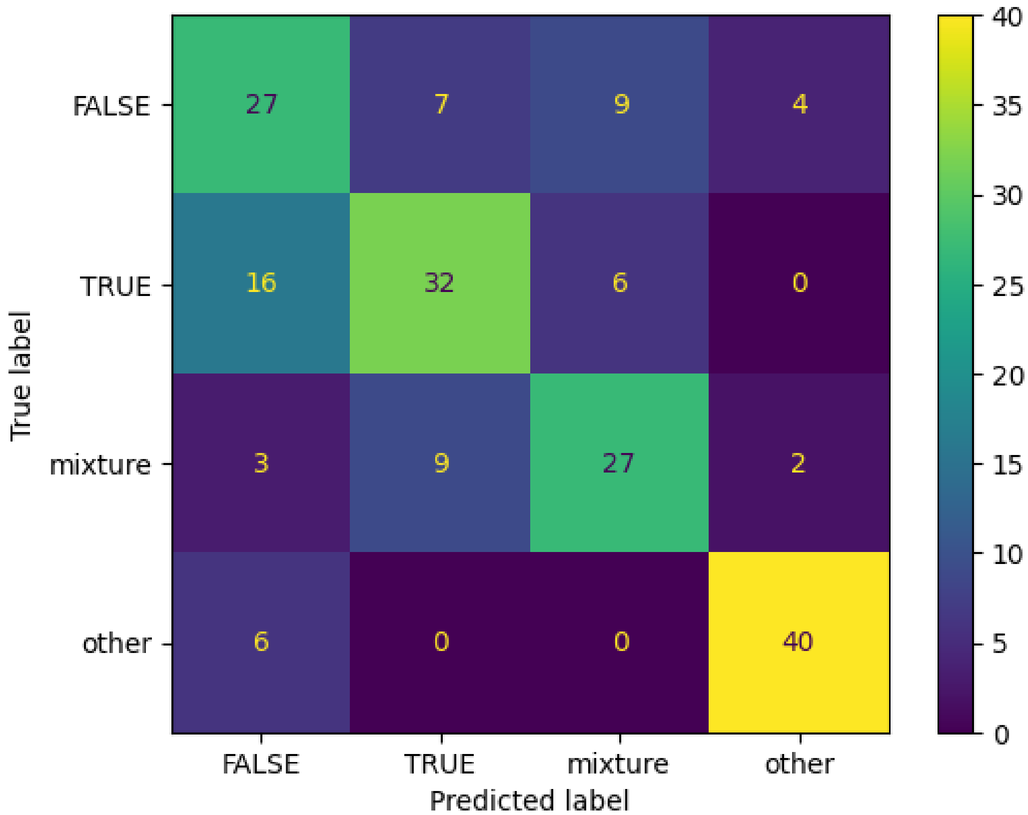
True label
FALSE
TRUE
mixture
other
32
3
10
2
16
32
6
0
1
5
34
1
2
0
2
42
Predicted label
40
35
30
25
20
15
10
5
0



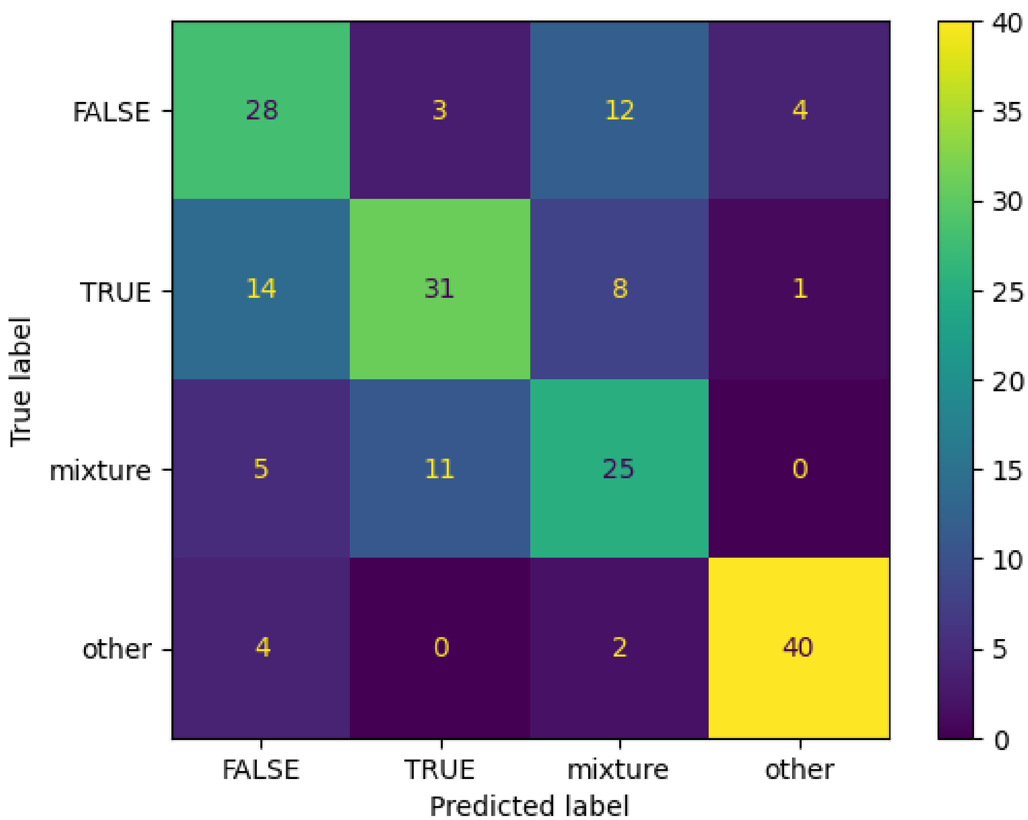
True label
FALSE
TRUE
mixture
other
32
5
6
4
15
36
3
0
4
9
28
0
4
0
2
40
Predicted label
40
35
30
25
20
15
10
5
0


True label
Predicted label
FALSE
TRUE
mixture
other
25
2
15
5
15
31
8
0
3
7
31
0
6
0
0
40
0
5
10
15
20
25
30
35
40

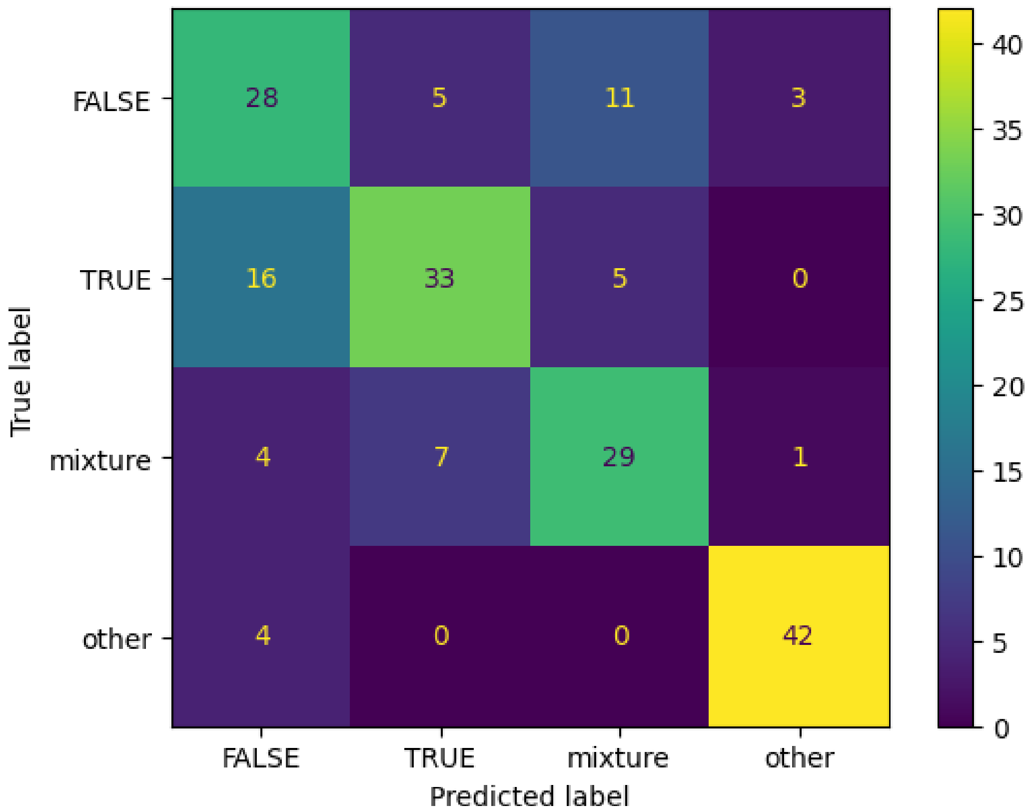

True label
Predicted label
FALSE
TRUE
mixture
other
30
7
6
4
15
36
3
0
3
8
29
1
4
0
2
40
0
5
10
15
20
25
30
35
40

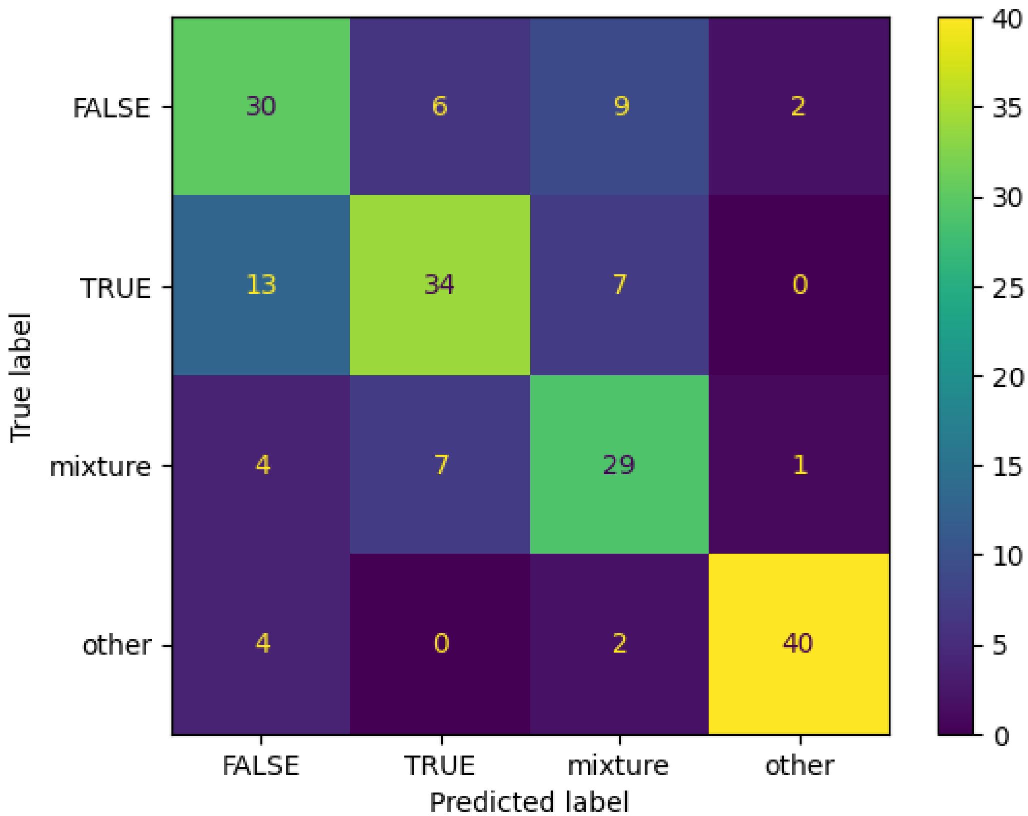


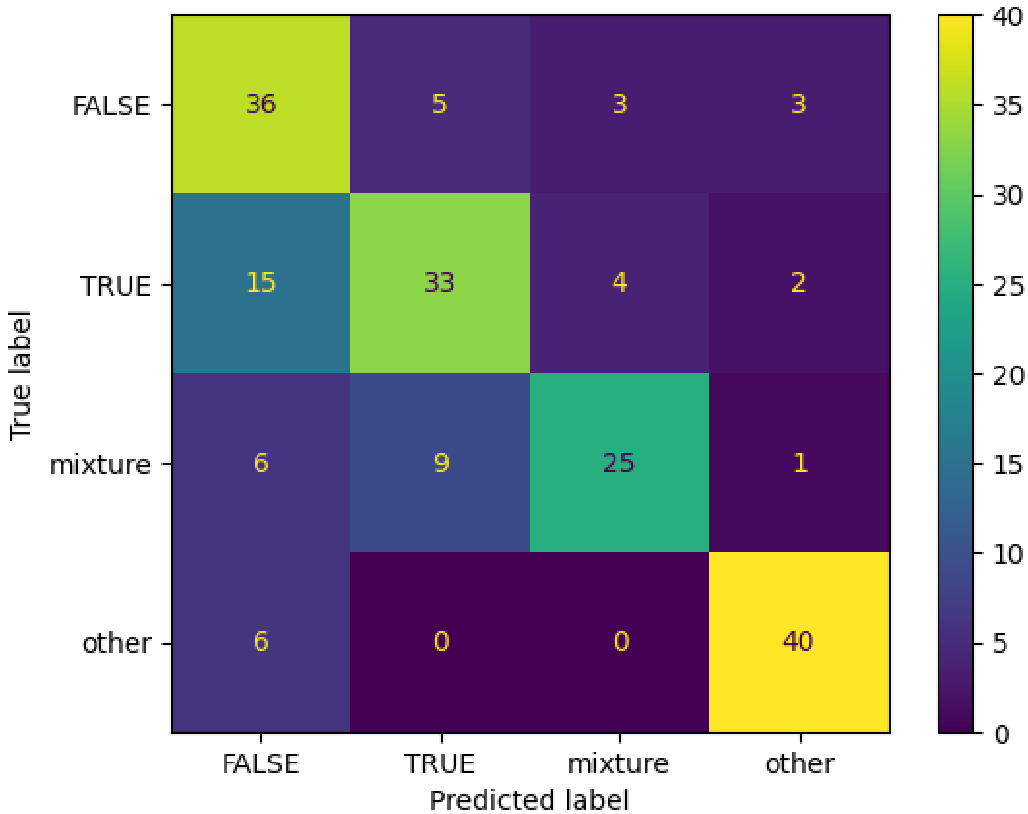
True label
Predicted label
FALSE
TRUE
mixture
other
27
7
9
4
16
32
6
0
3
9
27
2
6
0
0
40



True label
Predicted label
FALSE
TRUE
mixture
other
28
3
12
4
14
31
8
1
5
11
25
0
4
0
2
40


True label
Predicted label
FALSE
TRUE
mixture
other
28
5
11
3
16
33
5
0
4
7
29
1
4
0
0
42
FALSE
TRUE
mixture
other
0
5
10
15
20
25
30
35
40


True label
Predicted label
FALSE
TRUE
mixture
other
30
6
9
2
13
34
7
0
4
7
29
1
4
0
2
40
FALSE
TRUE
mixture
other
0
5
10
15
20
25
30
35
40



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

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.

```
print("X_train", X_train.shape)
print("y_train", y_train.shape)

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


# Liste des modèles à tester
```

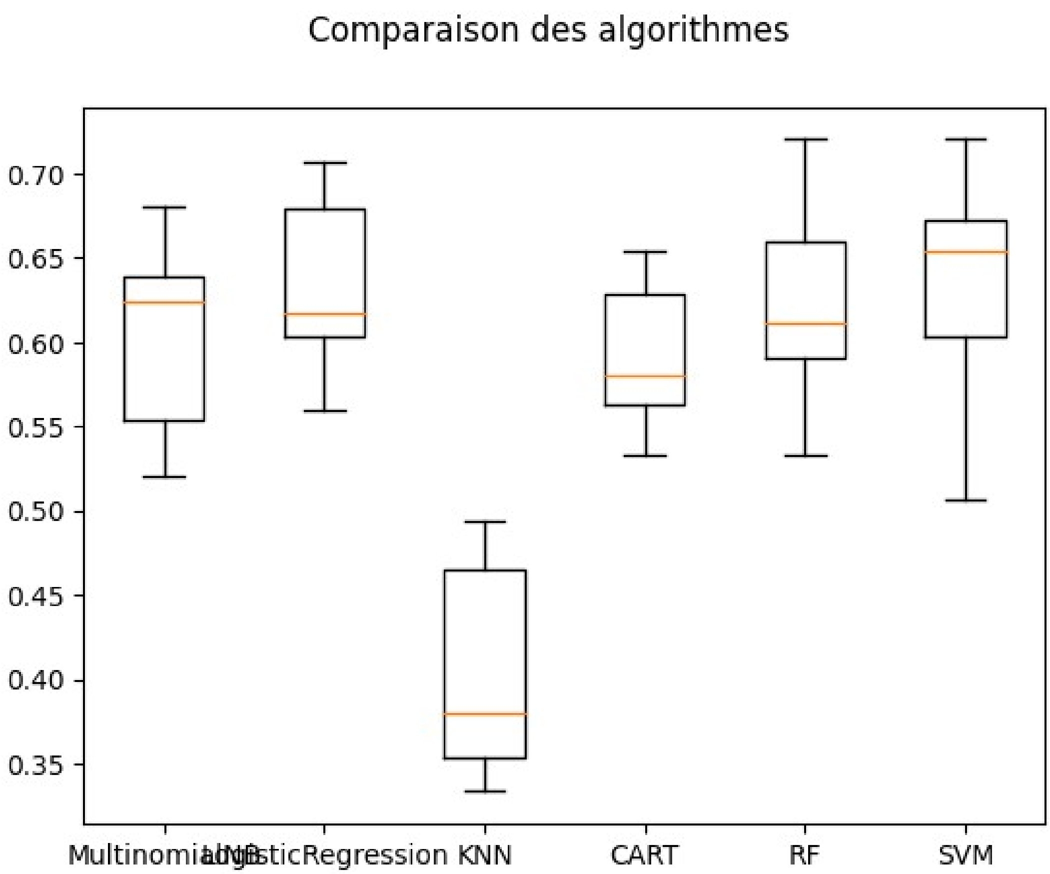
```python
models = [
    ('MultinomialNB', MultinomialNB()),
    ('LogisticRegression', LogisticRegression(random_state=42))
]


models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state=42)))
models.append(('RF', RandomForestClassifier(random_state=42)))
models.append(('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))
    #pipeline.fit(X_train,y_train)
all_results=[]
scores=[]
for p in pipelines:
    print(p[1])
    # cross validation en 10 fois
    kfold = KFold(n_splits=10,random_state=seed,shuffle=True)

#     print ("Evaluation de ",p)
    start_time = time.time()
    # application de la classification
    cv_results = cross_val_score(p[1],X_train['title'],y_train,
cv=kfold, scoring=score)
    #print("Pour le classifieur",p[0],"on a un score
de",cv_results.mean(),"et un écart type de",cv_results.std())
    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)
    # affichage des résultats
#print ('\nLe meilleur resultat : ',max(results))
```

```
X_train (748, 2)
y_train (748,)
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', 0.607009009009009,
0.05808962343049095), ('LogisticRegression', 0.6323963963963964,
0.04747997668516553), ('KNN', 0.4023603603603603,
0.05827331497948403), ('CART', 0.5896576576576578, 0.042260622694191),
('RF', 0.6203963963963963, 0.05357316419402158), ('SVM',
0.6377837837837838, 0.05744883637841297)]
all resultats [('SVM', 0.6377837837837838, 0.05744883637841297),
('LogisticRegression', 0.6323963963963964, 0.04747997668516553),
('RF', 0.6203963963963963, 0.05357316419402158), ('MultinomialNB',
0.607009009009009, 0.05808962343049095), ('CART', 0.5896576576576578,
0.042260622694191), ('KNN', 0.4023603603603603, 0.05827331497948403)]
```

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 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
print("y_train", y_train.shape)
print("y_test", y_test.shape)
#X_test = np.ravel(X_test)
print("X_test", X_test.shape)
# le plus simple est de faire un test sur differents pipelines.


# 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 = [
    ("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']).toar
ray())

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['
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_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")
      print("X_train",X_train_title[i].shape)
      print("y_train",y_train.shape)
      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]))
```

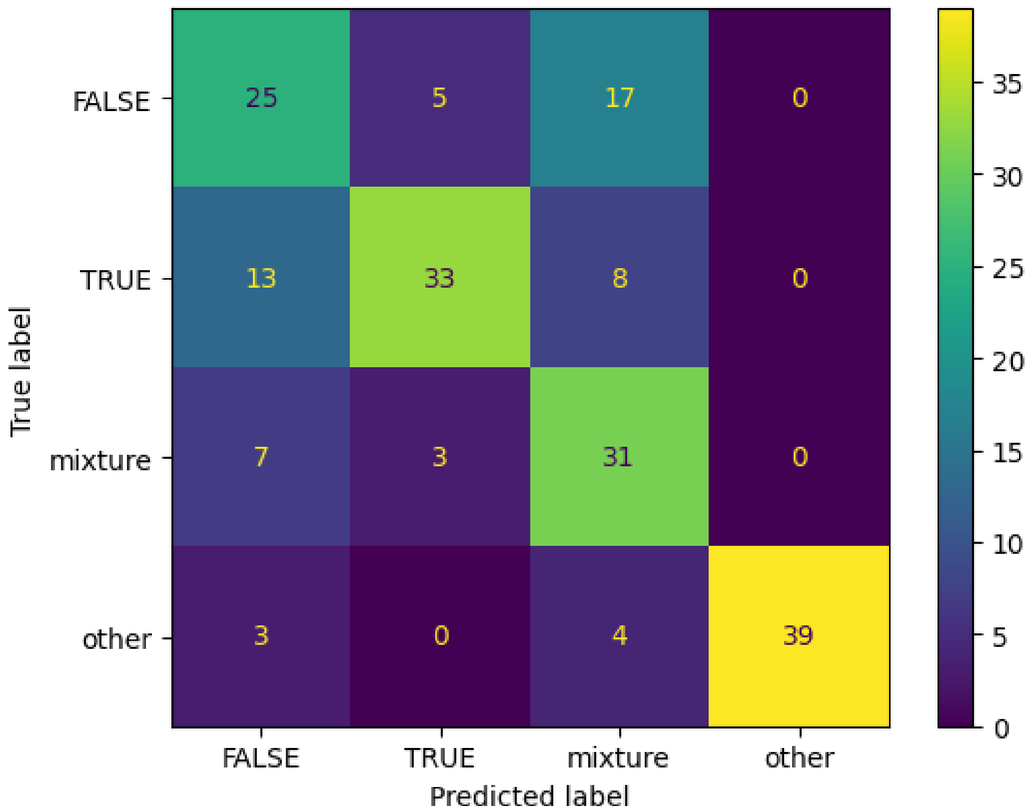
```
y_train (748,)
y_test (188,)
X_test (188, 2)
grid search fait
X_train (748, 5045)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.626
```

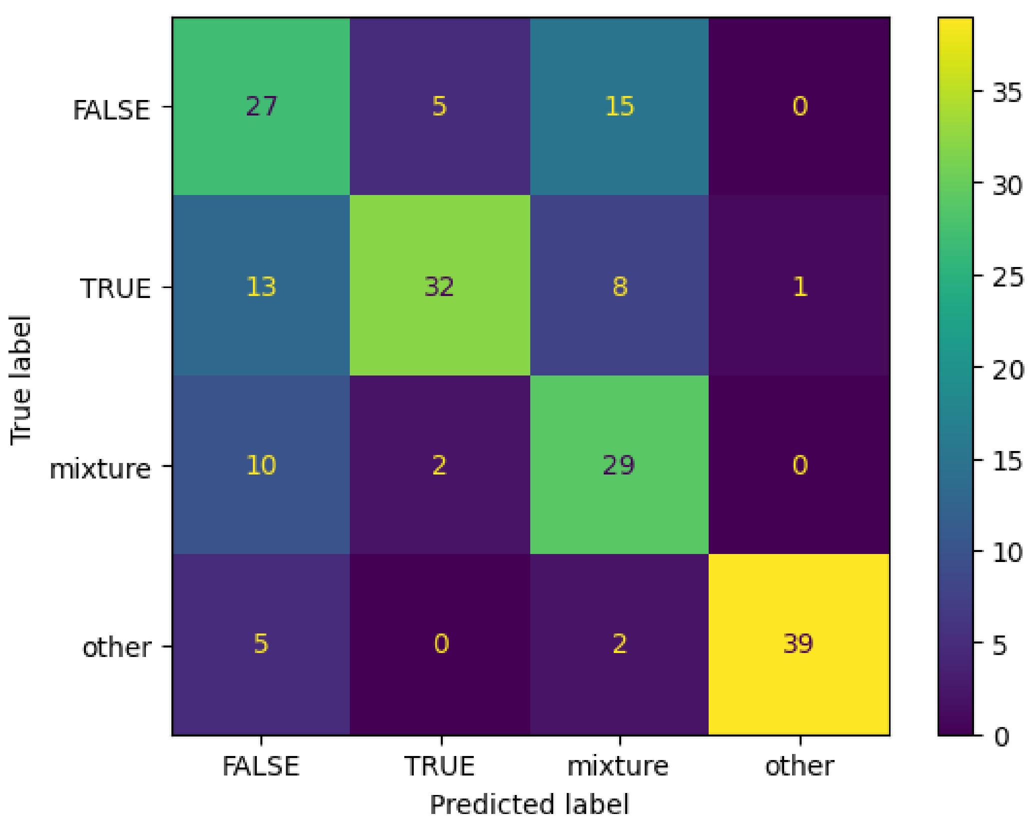
```
meilleur estimateur SVC(C=5, random_state=42)

Accuracy : 0.681
Classification Report
              precision    recall  f1-score   support

       FALSE    0.52083   0.53191   0.52632        47
        TRUE    0.80488   0.61111   0.69474        54
     mixture    0.51667   0.75610   0.61386        41
       other    1.00000   0.84783   0.91765        46

    accuracy                        0.68085       188
   macro avg    0.71059   0.68674   0.68814       188
weighted avg    0.71875   0.68085   0.68954       188

Ensemble des meilleurs paramètres :
     C: 5
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
X_train (748, 4915)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.631

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

Accuracy : 0.676
Classification Report
              precision    recall  f1-score   support

       FALSE    0.49091   0.57447   0.52941        47
        TRUE    0.82051   0.59259   0.68817        54
     mixture    0.53704   0.70732   0.61053        41
       other    0.97500   0.84783   0.90698        46

    accuracy                        0.67553       188
   macro avg    0.70586   0.68055   0.68377       188
weighted avg    0.71409   0.67553   0.68509       188

Ensemble des meilleurs paramètres :
     C: 2
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
X_train (748, 3780)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

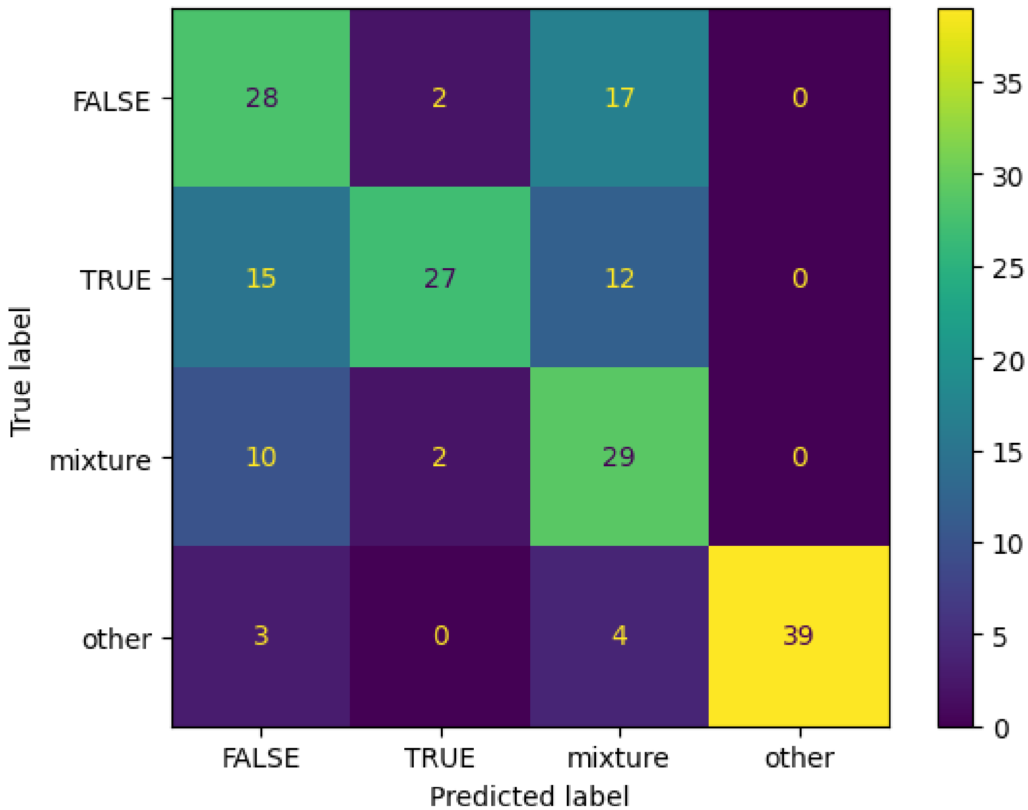
```
meilleur score 0.634

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

Accuracy : 0.654
Classification Report
              precision    recall  f1-score   support

       FALSE    0.50000   0.59574   0.54369        47
        TRUE    0.87097   0.50000   0.63529        54
     mixture    0.46774   0.70732   0.56311        41
       other    1.00000   0.84783   0.91765        46

    accuracy                        0.65426       188
   macro avg    0.70968   0.66272   0.66493       188
weighted avg    0.72186   0.65426   0.66574       188

Ensemble des meilleurs paramètres :
     C: 1
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
X_train (748, 5945)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.650

meilleur estimateur SVC(C=5, random_state=42)

Accuracy : 0.670
Classification Report
              precision    recall  f1-score   support

       FALSE    0.52273   0.48936   0.50549        47
        TRUE    0.82051   0.59259   0.68817        54
     mixture    0.49231   0.78049   0.60377        41
       other    0.97500   0.84783   0.90698        46

    accuracy                        0.67021       188
   macro avg    0.70264   0.67757   0.67610       188
weighted avg    0.71229   0.67021   0.67763       188

Ensemble des meilleurs paramètres :
     C: 5
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
X_train (748, 5045)
y_train (748,)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.632

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

Accuracy : 0.633
Classification Report
              precision    recall  f1-score   support

       FALSE    0.55172   0.34043   0.42105        47
        TRUE    0.79487   0.57407   0.66667        54
     mixture    0.44595   0.80488   0.57391        41
       other    0.84783   0.84783   0.84783        46

    accuracy                        0.63298       188
   macro avg    0.66009   0.64180   0.62736       188
weighted avg    0.67095   0.63298   0.62936       188

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
grid search fait
X_train (748, 4915)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.619

meilleur estimateur RandomForestClassifier(n_estimators=10,
random_state=42)

Accuracy : 0.649
Classification Report
              precision    recall  f1-score   support

       FALSE    0.52083   0.53191   0.52632        47
        TRUE    0.75000   0.55556   0.63830        54
     mixture    0.50877   0.70732   0.59184        41
       other    0.88372   0.82609   0.85393        46

    accuracy                        0.64894       188
   macro avg    0.66583   0.65522   0.65260       188
weighted avg    0.67282   0.64894   0.65293       188

Ensemble des meilleurs paramètres :
     n_estimators: 10
     max_features: 'sqrt'
grid search fait
X_train (748, 3780)
```

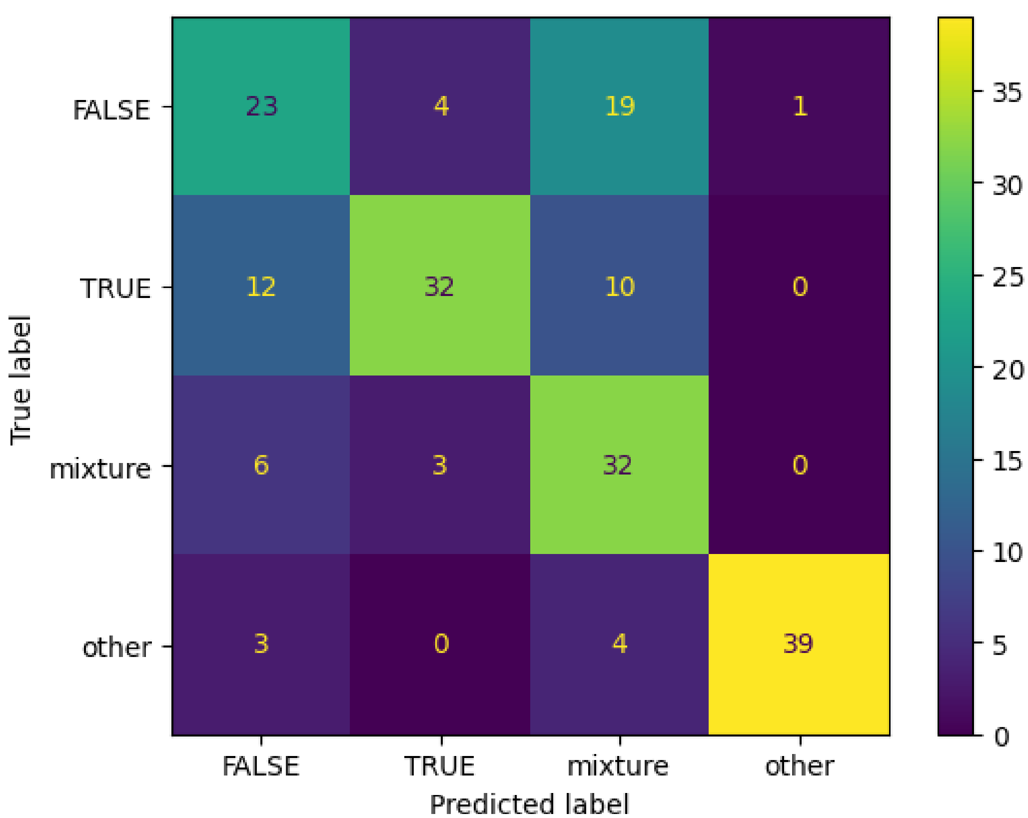
```
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.614

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

Accuracy : 0.638
Classification Report
              precision    recall  f1-score   support

       FALSE    0.53488   0.48936   0.51111        47
        TRUE    0.77143   0.50000   0.60674        54
     mixture    0.48438   0.75610   0.59048        41
       other    0.84783   0.84783   0.84783        46

    accuracy                        0.63830       188
   macro avg    0.65963   0.64832   0.63904       188
weighted avg    0.66838   0.63830   0.63828       188

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
grid search fait
X_train (748, 5945)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.634

meilleur estimateur RandomForestClassifier(random_state=42)

Accuracy : 0.665
Classification Report
              precision    recall  f1-score   support

       FALSE    0.63333   0.40426   0.49351        47
        TRUE    0.76190   0.59259   0.66667        54
     mixture    0.47143   0.80488   0.59459        41
       other    0.89130   0.89130   0.89130        46

    accuracy                        0.66489       188
   macro avg    0.68949   0.67326   0.66152       188
weighted avg    0.69807   0.66489   0.66262       188

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


True label
Predicted label
FALSE
TRUE
mixture
other
25
5
17
0
13
33
8
0
7
3
31
0
3
0
4
39
0
5
10
15
20
25
30
35

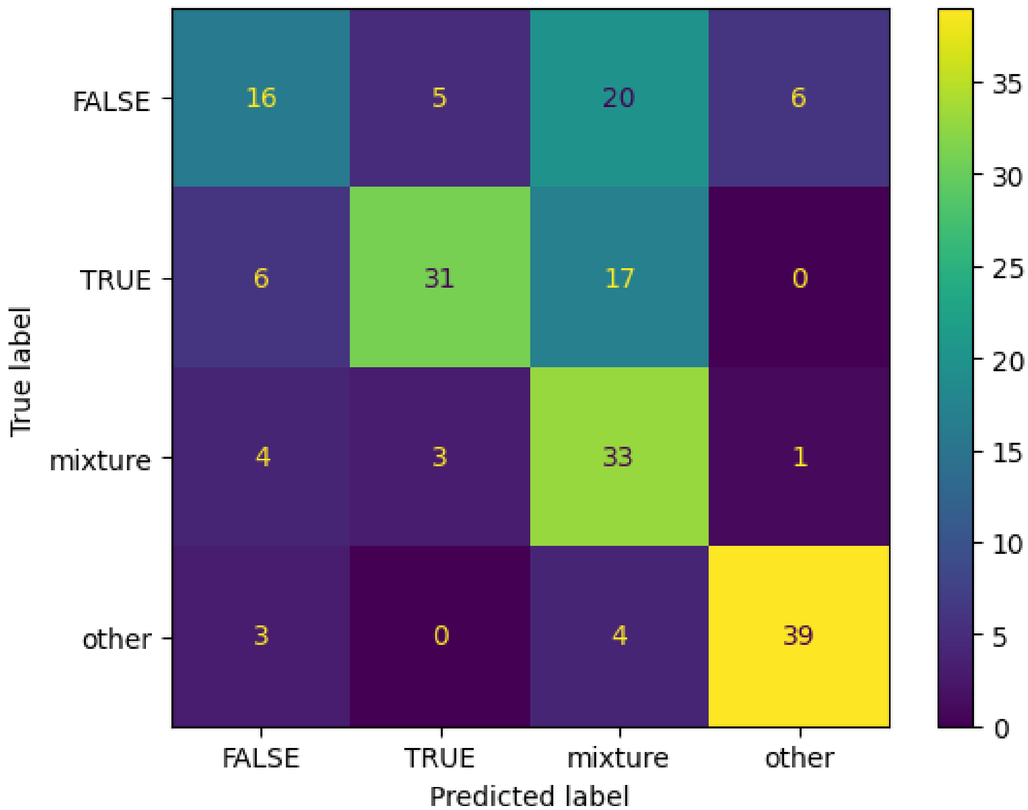

True label
Predicted label
FALSE
TRUE
mixture
other
27
5
15
0
13
32
8
1
10
2
29
0
5
0
2
39
0
5
10
15
20
25
30
35

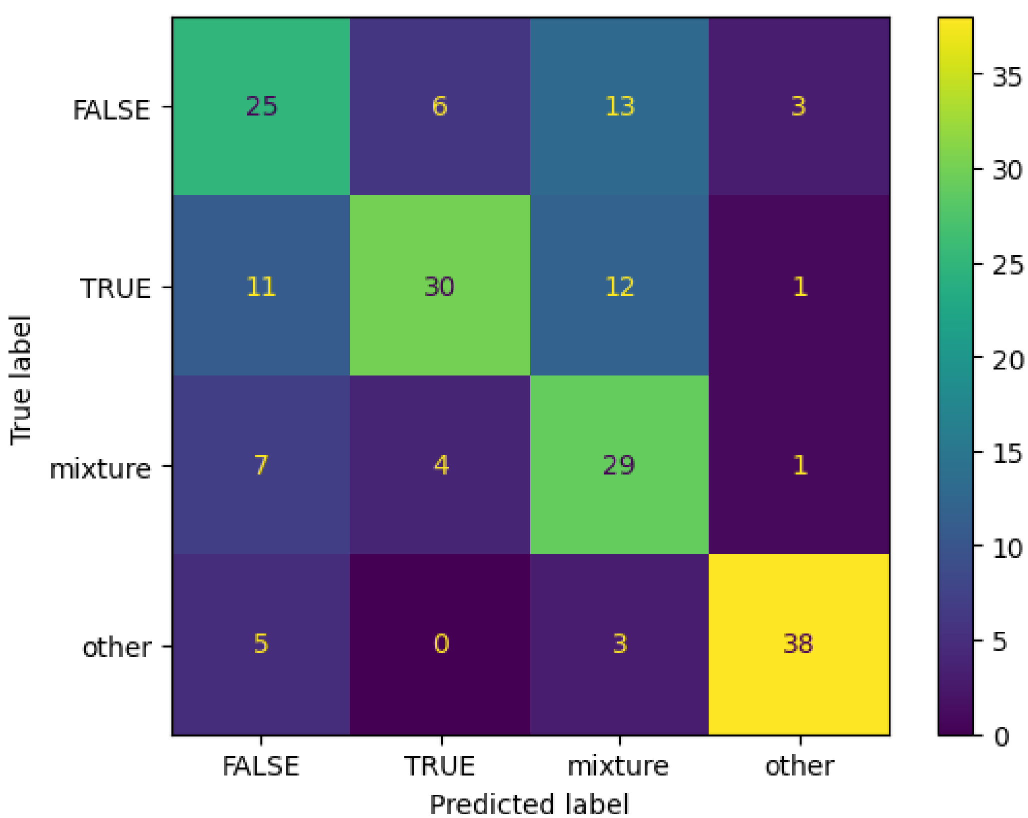


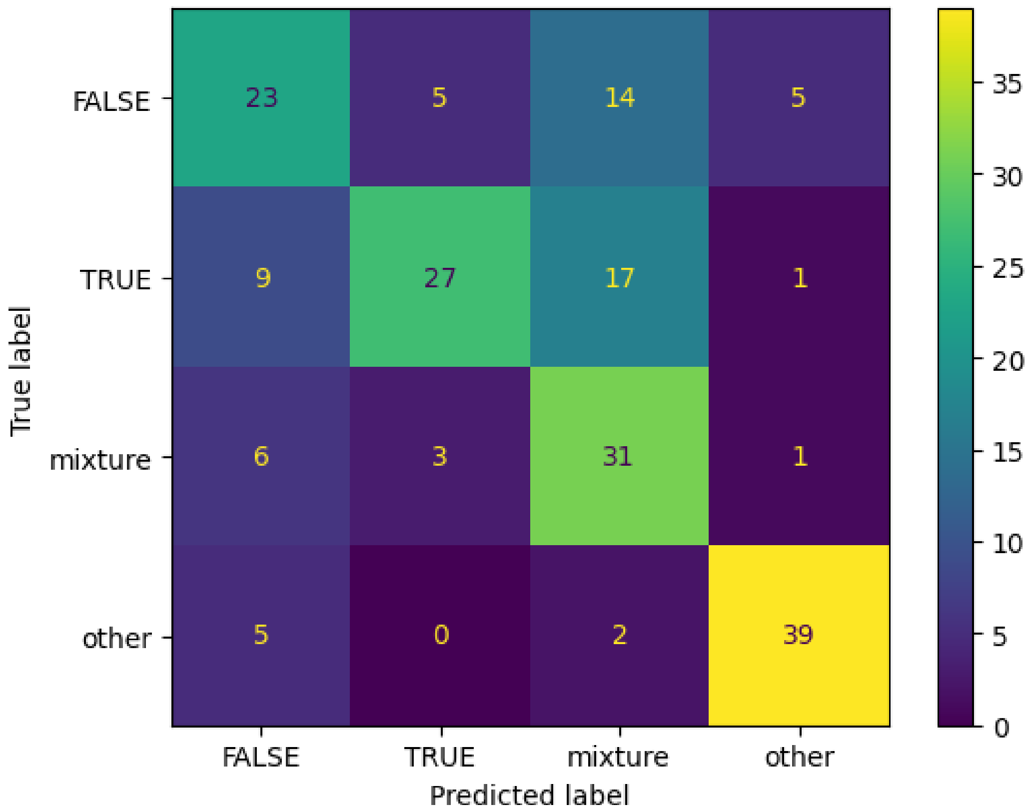
True label
FALSE
TRUE
mixture
other
28
2
17
0
15
27
12
0
10
2
29
0
3
0
4
39
Predicted label
0
5
10
15
20
25
30
35



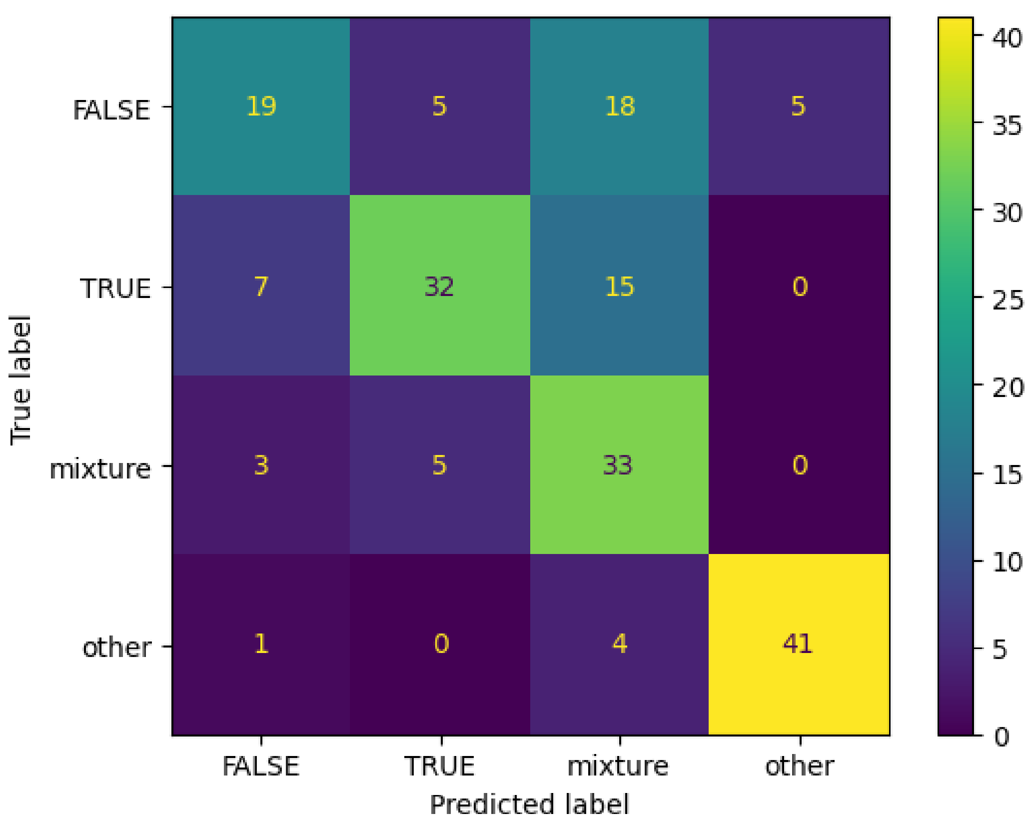
True label
FALSE
TRUE
mixture
other
23
4
19
1
12
32
10
0
6
3
32
0
3
0
4
39
Predicted label
0
5
10
15
20
25
30
35


True label
FALSE
TRUE
mixture
other
16
5
20
6
6
31
17
0
4
3
33
1
3
0
4
39
FALSE
TRUE
mixture
other
Predicted label
35
30
25
20
15
10
5
0


True label
FALSE
TRUE
mixture
other
25
6
13
3
11
30
12
1
7
4
29
1
5
0
3
38
FALSE
TRUE
mixture
other
Predicted label
35
30
25
20
15
10
5
0



FALSE
TRUE
mixture
other
True label
23 5 14 5
9 27 17 1
6 3 31 1
5 0 2 39
FALSE TRUE mixture other
Predicted label
0 5 10 15 20 25 30 35



FALSE
TRUE
mixture
other
True label
19 5 18 5
7 32 15 0
3 5 33 0
1 0 4 41
FALSE TRUE mixture other
Predicted label
0 5 10 15 20 25 30 35 40

##**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.

```
#concaténation
X_train=X_train.apply(lambda row: ' '.join([str(val) for val in row]),
axis=1)
X_test=X_test.apply(lambda row: ' '.join([str(val) for val in row]),
axis=1)
```

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.

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


# Liste des modèles à tester
models = [
    ('MultinomialNB', MultinomialNB()),
    ('LogisticRegression', LogisticRegression(random_state=42))
]


#models.append(('LR', LogisticRegression(solver='lbfgs')))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state=42)))
models.append(('RF', RandomForestClassifier(random_state=42)))
models.append(('SVM', SVC(random_state=42)))

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

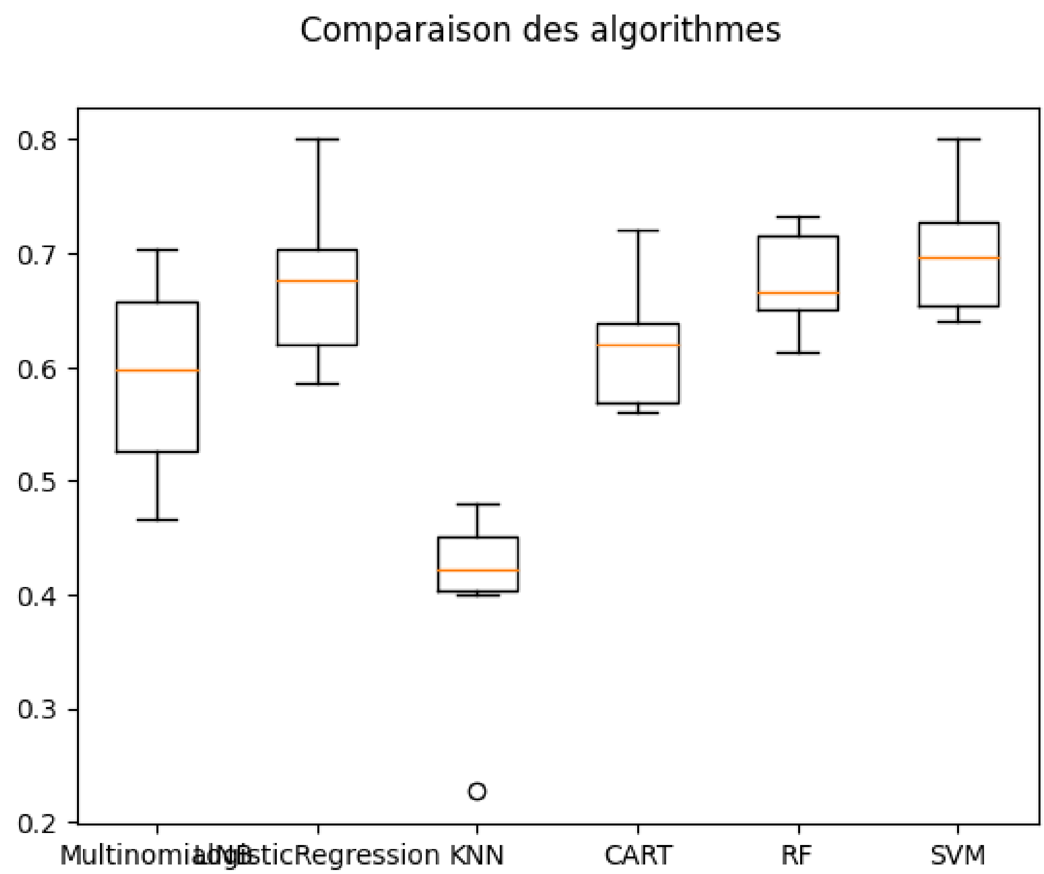
```
        ('tfidf', TfidfVectorizer()),
        (name,model)
    ])
    pipelines.append((name,pipeline))

all_results=[]
scores=[]
for p in pipelines:
    print(p[1])
    # cross validation en 10 fois
    kfold = KFold(n_splits=10,random_state=seed,shuffle=True)

#    print ("Evaluation de ",p)
    start_time = time.time()
    # application de la classification
    cv_results = cross_val_score(p[1],X_train,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)
    # affichage des résultats
#print ('\nLe meilleur resultat : ',max(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', 0.5910450450450451,
```

```
0.07300949147408688), ('LogisticRegression', 0.6724684684684684,
0.059631408484546344), ('KNN', 0.41315315315315315,
0.06735066687527869), ('CART', 0.6202702702702703,
0.05509562560124736), ('RF', 0.6751351351351351, 0.03957480365084483),
('SVM', 0.6965225225225227, 0.047332071300781584)]
all resultats [('SVM', 0.6965225225225227, 0.047332071300781584),
('RF', 0.6751351351351351, 0.03957480365084483),
('LogisticRegression', 0.6724684684684684, 0.059631408484546344),
('CART', 0.6202702702702703, 0.05509562560124736), ('MultinomialNB',
0.5910450450450451, 0.07300949147408688), ('KNN', 0.41315315315315315,
0.06735066687527869)]
```

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 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).

```
# 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_text_SVC = []
X_test_title_text_SVC = []

X_train_title_text_RandomForestClassifier = []
X_test_title_text_RandomForestClassifier = []


for name, pipeline in all_models :

X_train_title_text_SVC.append(pipeline.fit_transform(X_train).toarray(
))
    X_test_title_text_SVC.append(pipeline.transform(X_test).toarray())

X_train_title_text_RandomForestClassifier.append(pipeline.fit_transfor
m(X_train).toarray())

X_test_title_text_RandomForestClassifier.append(pipeline.transform(X_t
est).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_text = eval('X_train_title_text_' + model_name)
    X_test_title_text = eval('X_test_title_text_' + model_name)
    for i in range (len(X_train_title_text)):
      grid_search = GridSearchCV(model, params[model_name], n_jobs=-1,
verbose=1,scoring=score)
      print("grid search fait")
      print("X_train_title_text",X_train_title_text[i].shape)
```

```
      print("y_train",y_train.shape)
      grid_search.fit(X_train_title_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_title_text[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
X_train_title_text (748, 27705)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.651

meilleur estimateur SVC(kernel='linear', random_state=42)

Accuracy : 0.676
Classification Report
              precision    recall  f1-score   support

       FALSE    0.67647   0.48936   0.56790        47
        TRUE    0.71739   0.61111   0.66000        54
     mixture    0.54348   0.60976   0.57471        41
       other    0.74194   1.00000   0.85185        46

    accuracy                        0.67553       188
   macro avg    0.66982   0.67756   0.66362       188
weighted avg    0.67524   0.67553   0.66532       188

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 'scale'
	kernel: 'linear'
grid search fait
X_train_title_text (748, 23058)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.652

meilleur estimateur SVC(kernel='linear', random_state=42)

Accuracy : 0.644
Classification Report
              precision    recall  f1-score   support
```

```
       FALSE    0.66667   0.46809   0.55000        47
        TRUE    0.66000   0.61111   0.63462        54
     mixture    0.48889   0.53659   0.51163        41
       other    0.73333   0.95652   0.83019        46

    accuracy                        0.64362       188
   macro avg    0.63722   0.64308   0.63161       188
weighted avg    0.64229   0.64362   0.63449       188

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 'scale'
	kernel: 'linear'
grid search fait
X_train_title_text (748, 22918)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.639

meilleur estimateur SVC(gamma=0.01, random_state=42)

Accuracy : 0.612
Classification Report
              precision    recall  f1-score   support

       FALSE    0.46667   0.29787   0.36364        47
        TRUE    1.00000   0.42593   0.59740        54
     mixture    0.41758   0.92683   0.57576        41
       other    0.90909   0.86957   0.88889        46

    accuracy                        0.61170       188
   macro avg    0.69833   0.63005   0.60642       188
weighted avg    0.71741   0.61170   0.60556       188

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 0.01
	kernel: 'rbf'
grid search fait
X_train_title_text (748, 15589)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.646

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

Accuracy : 0.670
Classification Report
```

```
              precision    recall  f1-score   support

       FALSE    0.55000   0.46809   0.50575        47
        TRUE    0.68182   0.55556   0.61224        54
     mixture    0.58333   0.68293   0.62921        41
       other    0.82143   1.00000   0.90196        46

    accuracy                        0.67021       188
   macro avg    0.65915   0.67664   0.66229       188
weighted avg    0.66155   0.67021   0.66021       188

Ensemble des meilleurs paramètres :
     C: 10
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
X_train_title_text (748, 23058)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.691

meilleur estimateur SVC(C=5, random_state=42)

Accuracy : 0.739
Classification Report
              precision    recall  f1-score   support

       FALSE    0.64583   0.65957   0.65263        47
        TRUE    0.82051   0.59259   0.68817        54
     mixture    0.61818   0.82927   0.70833        41
       other    0.91304   0.91304   0.91304        46

    accuracy                        0.73936       188
   macro avg    0.74939   0.74862   0.74055       188
weighted avg    0.75536   0.73936   0.73871       188

Ensemble des meilleurs paramètres :
     C: 5
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
X_train_title_text (748, 22918)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.702

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

Accuracy : 0.734
```

```
Classification Report
              precision    recall  f1-score   support

       FALSE    0.69565   0.68085   0.68817        47
        TRUE    0.86486   0.59259   0.70330        54
     mixture    0.54839   0.82927   0.66019        41
       other    0.93023   0.86957   0.89888        46

    accuracy                        0.73404       188
   macro avg    0.75978   0.74307   0.73763       188
weighted avg    0.76954   0.73404   0.73797       188

Ensemble des meilleurs paramètres :
	C: 1
	gamma: 'scale'
	kernel: 'rbf'
grid search fait
X_train_title_text (748, 15589)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.691

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

Accuracy : 0.734
Classification Report
              precision    recall  f1-score   support

       FALSE    0.66667   0.68085   0.67368        47
        TRUE    0.86486   0.59259   0.70330        54
     mixture    0.56667   0.82927   0.67327        41
       other    0.93023   0.86957   0.89888        46

    accuracy                        0.73404       188
   macro avg    0.75711   0.74307   0.73728       188
weighted avg    0.76628   0.73404   0.73720       188

Ensemble des meilleurs paramètres :
	C: 1.0
	gamma: 1
	kernel: 'rbf'
grid search fait
X_train_title_text (748, 27705)
y_train (748,)
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.695

meilleur estimateur SVC(C=7, random_state=42)
```

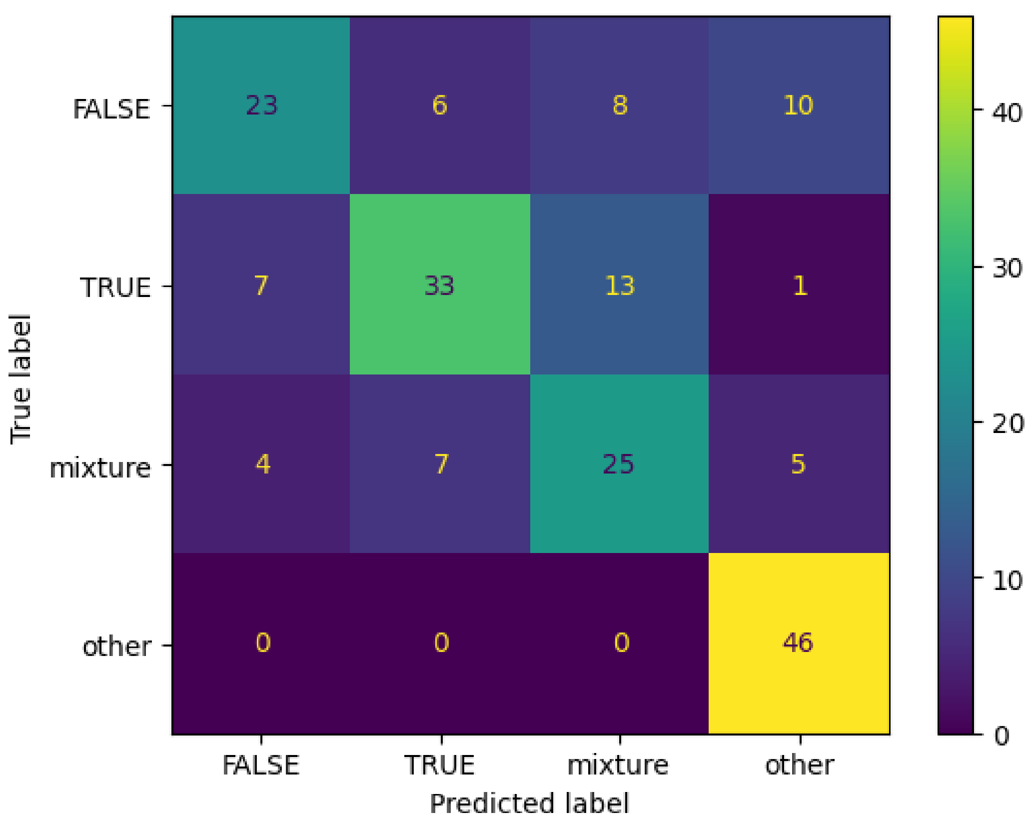
```
Accuracy : 0.745
Classification Report
              precision    recall  f1-score   support

       FALSE    0.68085   0.68085   0.68085        47
        TRUE    0.80000   0.59259   0.68085        54
     mixture    0.60714   0.82927   0.70103        41
       other    0.93333   0.91304   0.92308        46

    accuracy                        0.74468       188
   macro avg    0.75533   0.75394   0.74645       188
weighted avg    0.76078   0.74468   0.74452       188

Ensemble des meilleurs paramètres :
     C: 7
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
X_train_title_text (748, 27705)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.681

meilleur estimateur RandomForestClassifier(random_state=42)

Accuracy : 0.686
Classification Report
              precision    recall  f1-score   support

       FALSE    0.56604   0.63830   0.60000        47
        TRUE    0.66667   0.66667   0.66667        54
     mixture    0.65714   0.56098   0.60526        41
       other    0.86957   0.86957   0.86957        46

    accuracy                        0.68617       188
   macro avg    0.68985   0.68388   0.68537       188
weighted avg    0.68908   0.68617   0.68625       188

Ensemble des meilleurs paramètres :
     n_estimators: 100
     max_features: 'sqrt'
grid search fait
X_train_title_text (748, 23058)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.669

meilleur estimateur RandomForestClassifier(random_state=42)
```

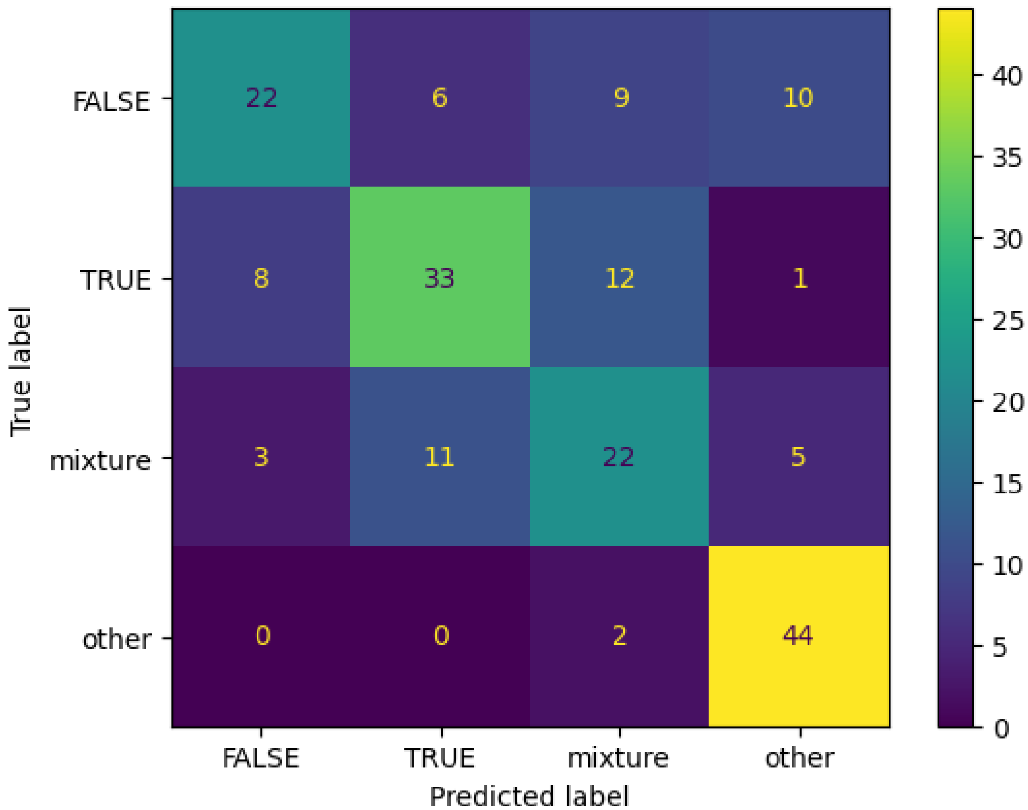
```
Accuracy : 0.707
Classification Report
              precision    recall  f1-score   support

       FALSE    0.53061   0.55319   0.54167        47
        TRUE    0.66102   0.72222   0.69027        54
     mixture    0.80000   0.68293   0.73684        41
       other    0.88889   0.86957   0.87912        46

    accuracy                        0.70745       188
   macro avg    0.72013   0.70698   0.71197       188
weighted avg    0.71448   0.70745   0.70948       188

Ensemble des meilleurs paramètres :
     n_estimators: 100
     max_features: 'sqrt'
grid search fait
X_train_title_text (748, 22918)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.671

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

Accuracy : 0.713
Classification Report
              precision    recall  f1-score   support

       FALSE    0.56364   0.65957   0.60784        47
        TRUE    0.68627   0.64815   0.66667        54
     mixture    0.75676   0.68293   0.71795        41
       other    0.88889   0.86957   0.87912        46

    accuracy                        0.71277       188
   macro avg    0.72389   0.71505   0.71789       188
weighted avg    0.72056   0.71277   0.71513       188

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
grid search fait
X_train_title_text (748, 15589)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.679

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

```
Accuracy : 0.718
Classification Report
              precision    recall  f1-score   support

       FALSE    0.58000   0.61702   0.59794        47
        TRUE    0.68519   0.68519   0.68519        54
     mixture    0.72500   0.70732   0.71605        41
       other    0.90909   0.86957   0.88889        46

    accuracy                        0.71809       188
   macro avg    0.72482   0.71977   0.72202       188
weighted avg    0.72236   0.71809   0.71995       188

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
grid search fait
X_train_title_text (748, 23058)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.675

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

Accuracy : 0.691
Classification Report
              precision    recall  f1-score   support

       FALSE    0.52459   0.68085   0.59259        47
        TRUE    0.77500   0.57407   0.65957        54
     mixture    0.67568   0.60976   0.64103        41
       other    0.84000   0.91304   0.87500        46

    accuracy                        0.69149       188
   macro avg    0.70382   0.69443   0.69205       188
weighted avg    0.70664   0.69149   0.69149       188

Ensemble des meilleurs paramètres :
     n_estimators: 100
     max_features: 'log2'
grid search fait
X_train_title_text (748, 22918)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.675

meilleur estimateur RandomForestClassifier(n_estimators=300,
```

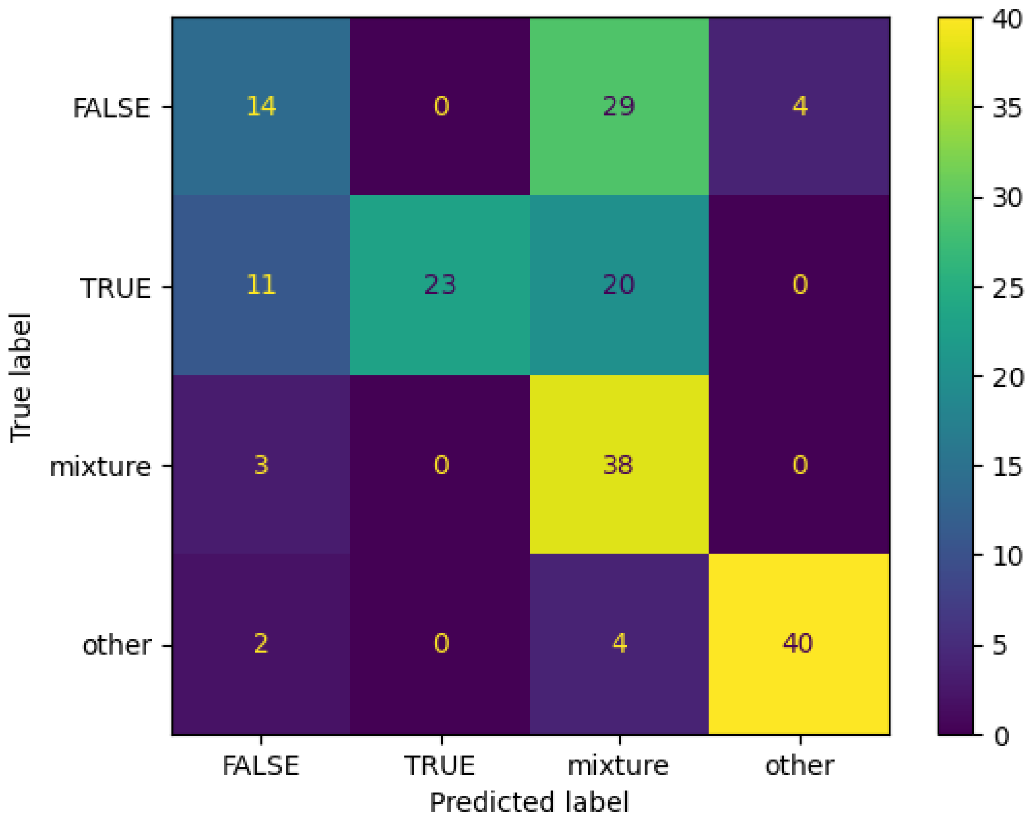
```
random_state=42)

Accuracy : 0.729
Classification Report
              precision    recall  f1-score   support

       FALSE    0.58333   0.74468   0.65421        47
        TRUE    0.70213   0.61111   0.65347        54
     mixture    0.74359   0.70732   0.72500        41
       other    0.95238   0.86957   0.90909        46

    accuracy                        0.72872       188
   macro avg    0.74536   0.73317   0.73544       188
weighted avg    0.74270   0.72872   0.73180       188

Ensemble des meilleurs paramètres :
     n_estimators: 300
     max_features: 'sqrt'
grid search fait
X_train_title_text (748, 15589)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.666

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

Accuracy : 0.739
Classification Report
              precision    recall  f1-score   support

       FALSE    0.63462   0.70213   0.66667        47
        TRUE    0.80000   0.66667   0.72727        54
     mixture    0.63636   0.68293   0.65882        41
       other    0.89362   0.91304   0.90323        46

    accuracy                        0.73936       188
   macro avg    0.74115   0.74119   0.73900       188
weighted avg    0.74587   0.73936   0.74025       188

Ensemble des meilleurs paramètres :
     n_estimators: 100
     max_features: 'log2'
grid search fait
X_train_title_text (748, 27705)
y_train (748,)
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.686
```

```
meilleur estimateur RandomForestClassifier(random_state=42)

Accuracy : 0.686
Classification Report
              precision    recall  f1-score   support

       FALSE    0.50980   0.55319   0.53061        47
        TRUE    0.70833   0.62963   0.66667        54
     mixture    0.64444   0.70732   0.67442        41
       other    0.90909   0.86957   0.88889        46

    accuracy                        0.68617       188
   macro avg    0.69292   0.68993   0.69015       188
weighted avg    0.69389   0.68617   0.68872       188

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

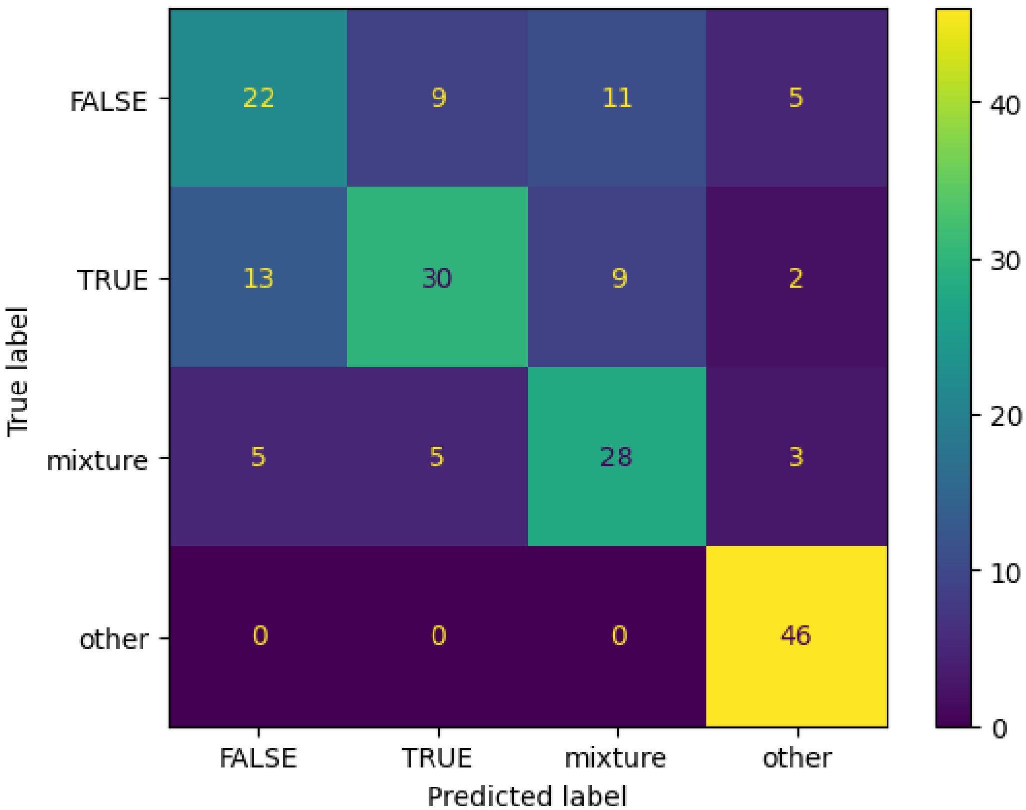


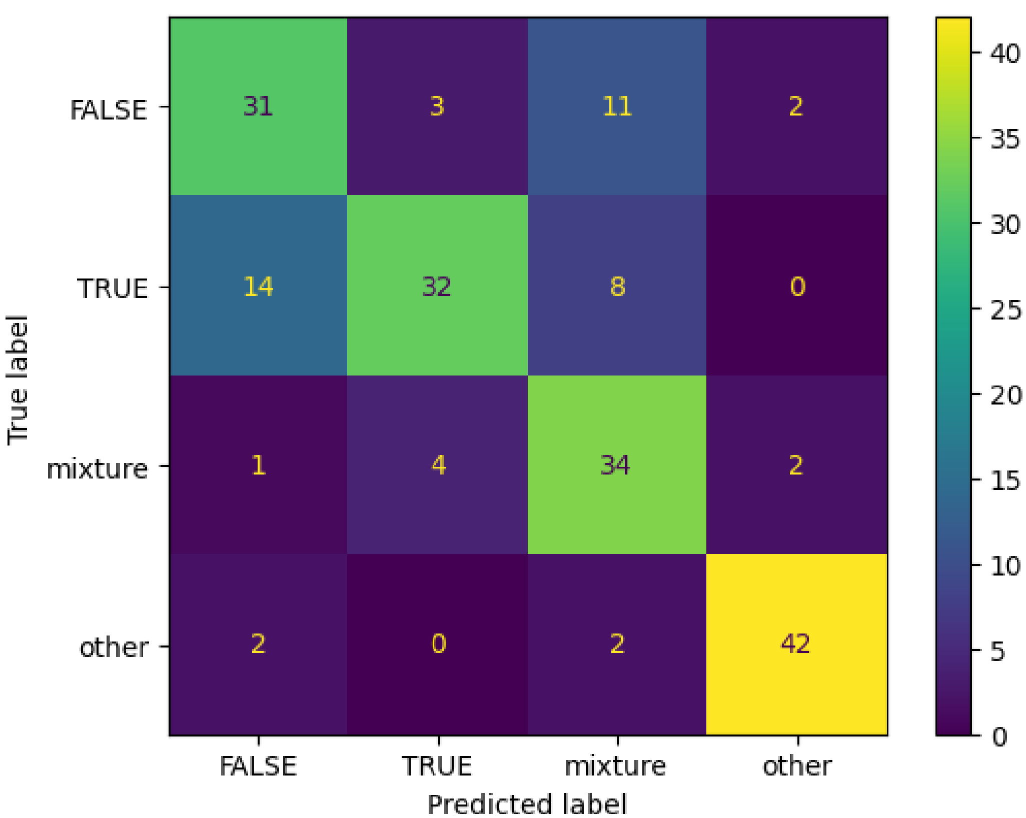


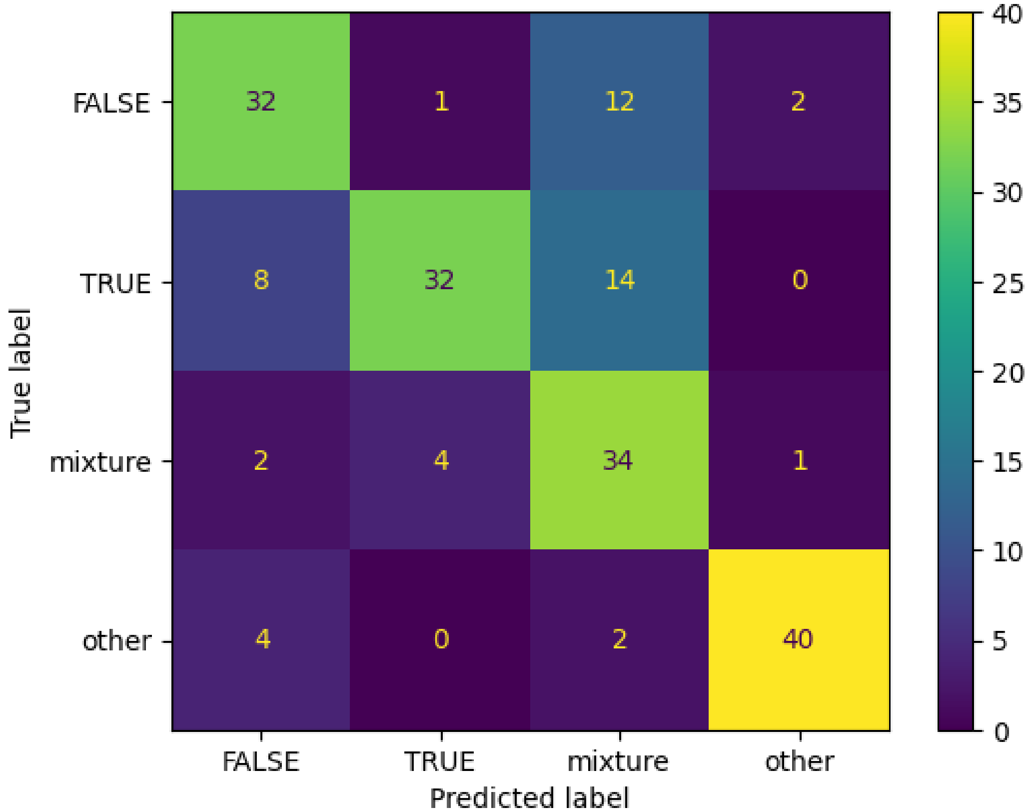
FALSE
TRUE
mixture
other
True label
22
6
9
10
8
33
12
1
3
11
22
5
0
0
2
44
FALSE
TRUE
mixture
other
Predicted label
40
35
30
25
20
15
10
5
0

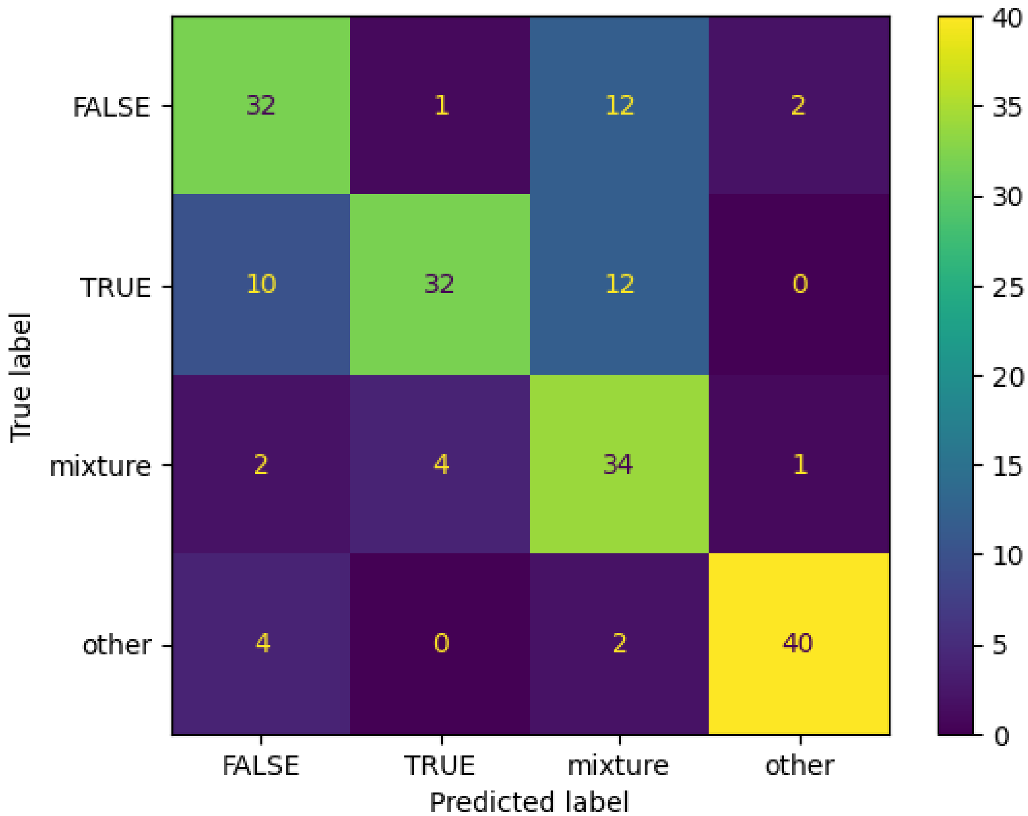


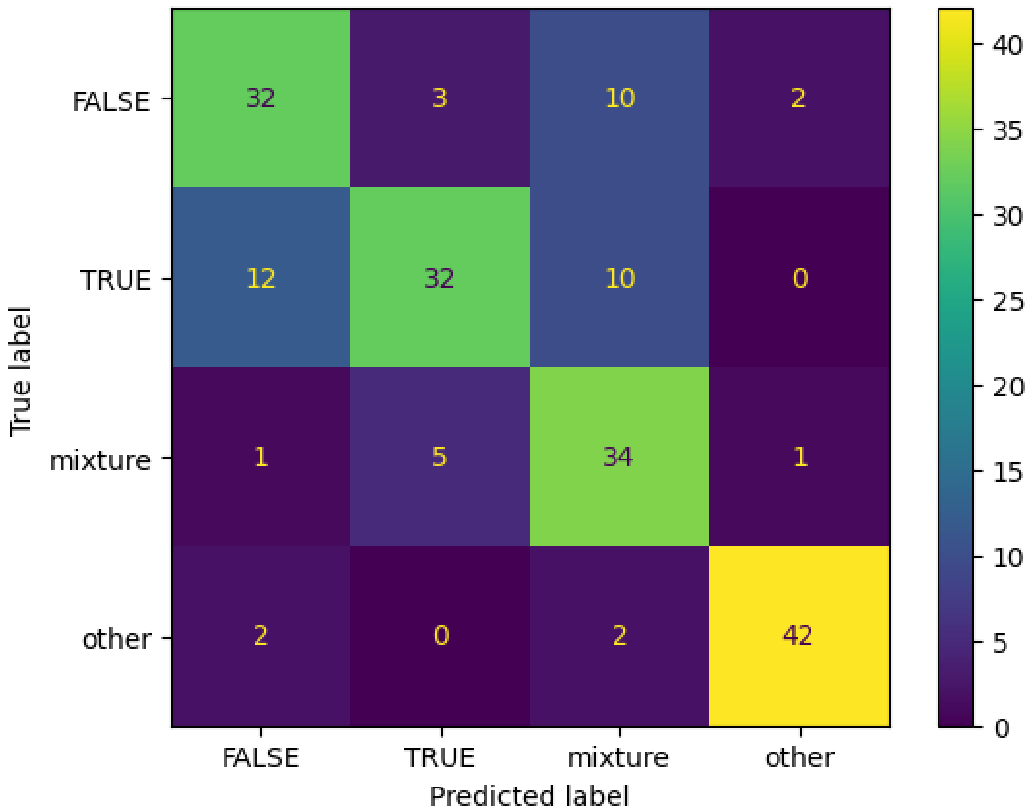
FALSE
TRUE
mixture
other
True label
14
0
29
4
11
23
20
0
3
0
38
0
2
0
4
40
FALSE
TRUE
mixture
other
Predicted label
40
35
30
25
20
15
10
5
0

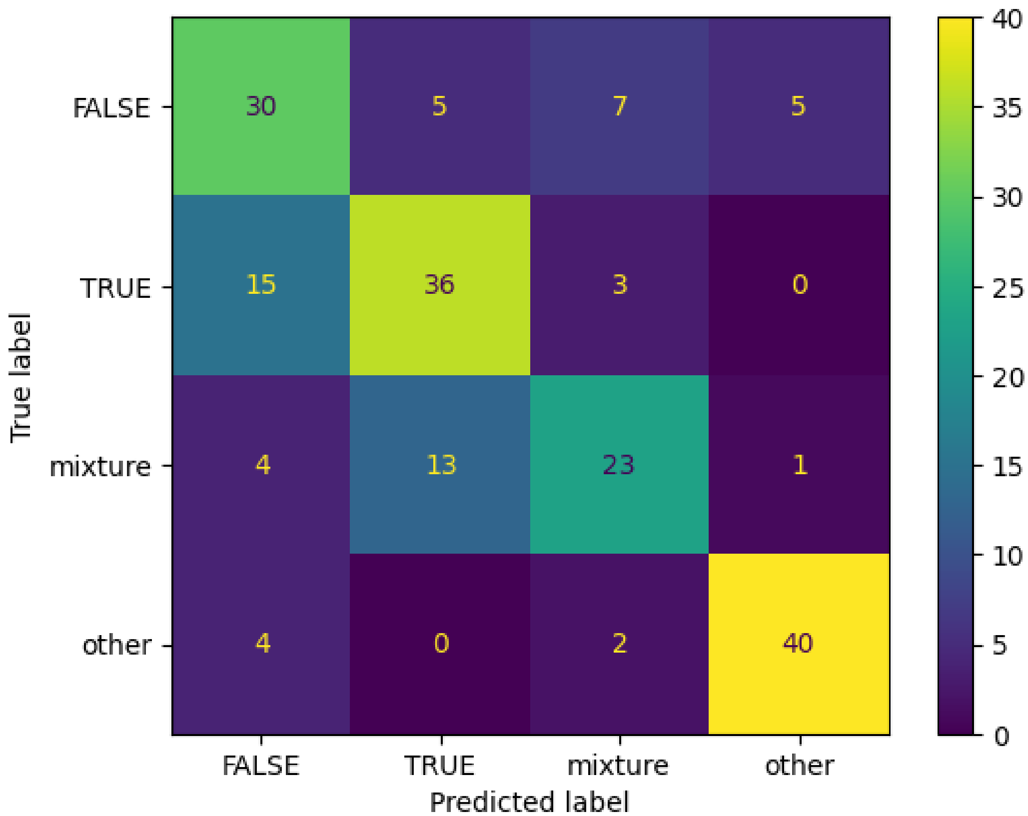

True label
Predicted label
FALSE
TRUE
mixture
other
22
9
11
5
13
30
9
2
5
5
28
3
0
0
0
46
0
10
20
30
40

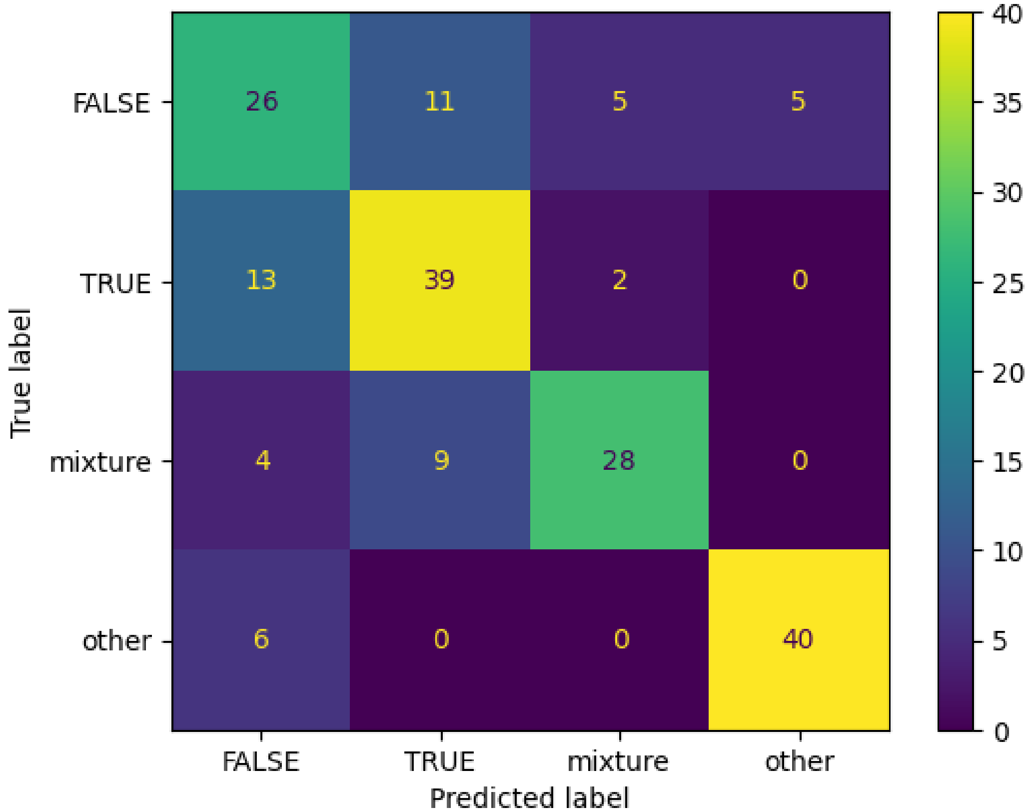

True label
Predicted label
FALSE
TRUE
mixture
other
31
3
11
2
14
32
8
0
1
4
34
2
2
0
2
42
0
5
10
15
20
25
30
35
40

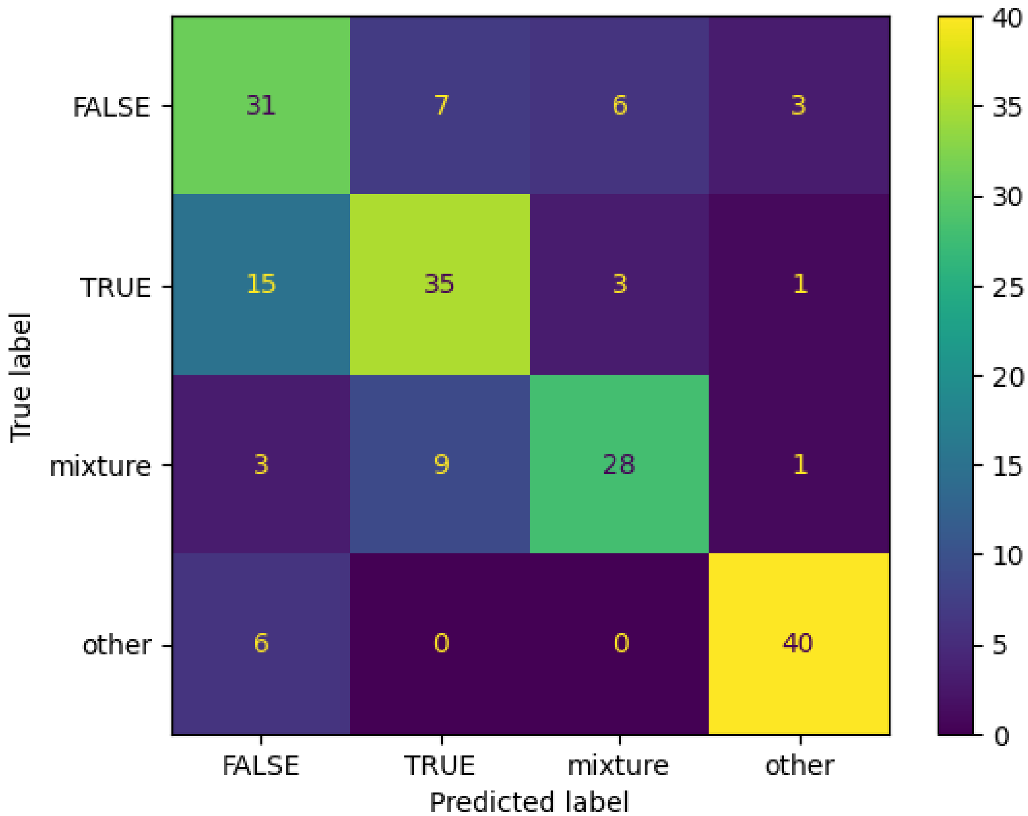

True label
FALSE
TRUE
mixture
other
32
1
12
2
8
32
14
0
2
4
34
1
4
0
2
40
FALSE
TRUE
mixture
other
Predicted label
0
5
10
15
20
25
30
35
40

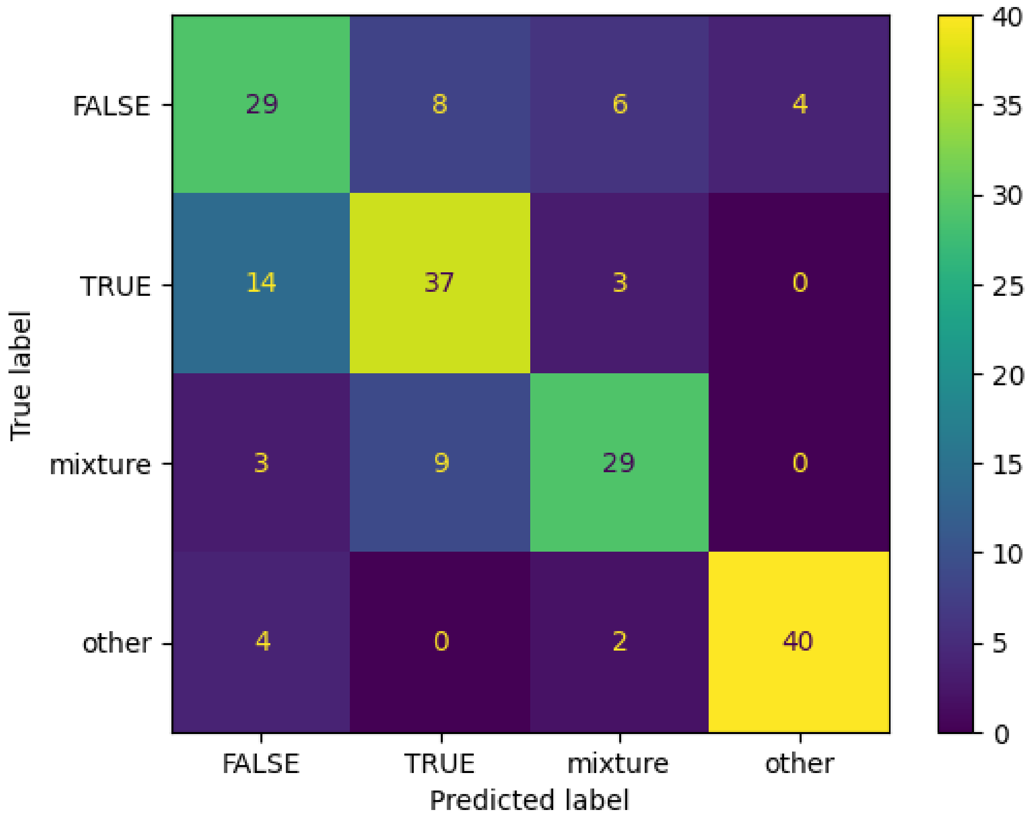

True label
FALSE
TRUE
mixture
other
32
1
12
2
10
32
12
0
2
4
34
1
4
0
2
40
FALSE
TRUE
mixture
other
Predicted label
0
5
10
15
20
25
30
35
40

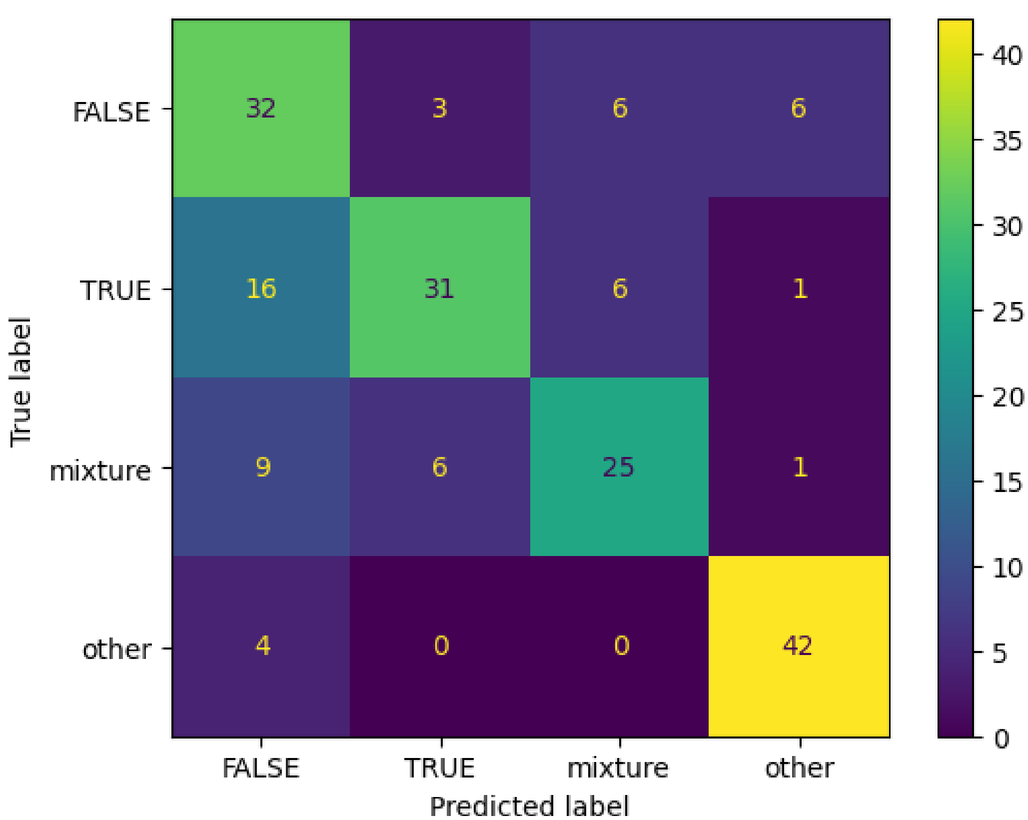

True label
Predicted label
FALSE
TRUE
mixture
other
32 3 10 2
12 32 10 0
1 5 34 1
2 0 2 42
0
5
10
15
20
25
30
35
40

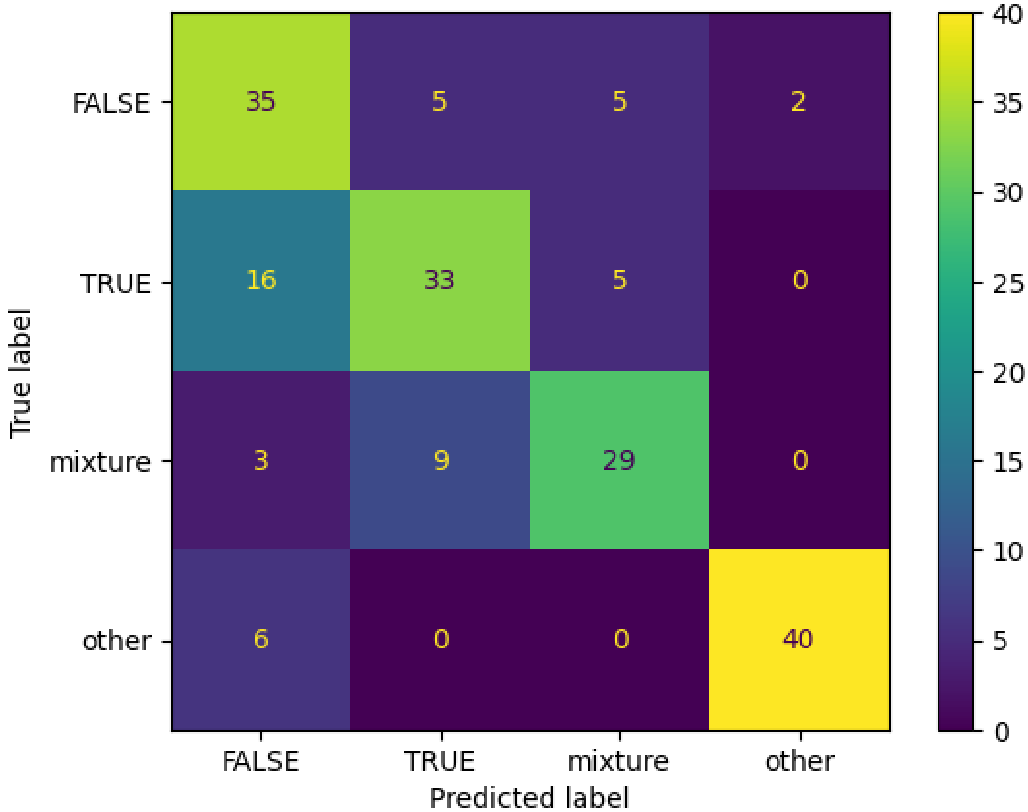

True label
Predicted label
FALSE
TRUE
mixture
other
30 5 7 5
15 36 3 0
4 13 23 1
4 0 2 40
0
5
10
15
20
25
30
35
40

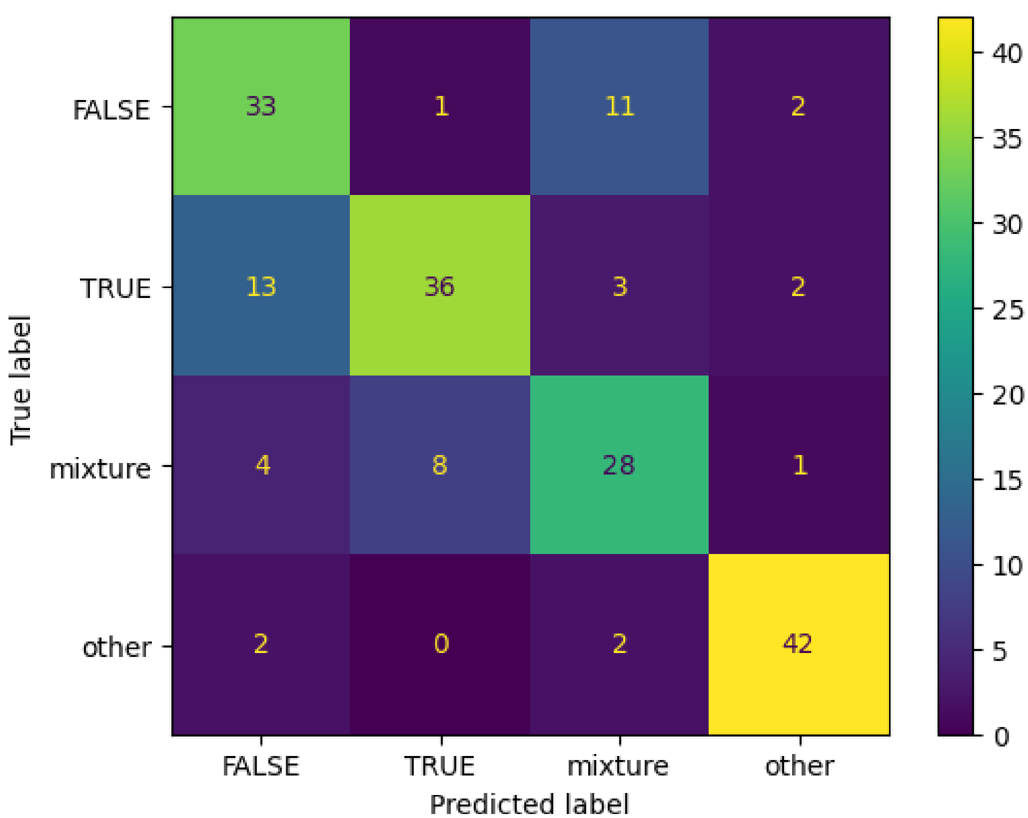

True label
Predicted label
FALSE
TRUE
mixture
other
26 11 5 5
13 39 2 0
4 9 28 0
6 0 0 40
0
5
10
15
20
25
30
35
40

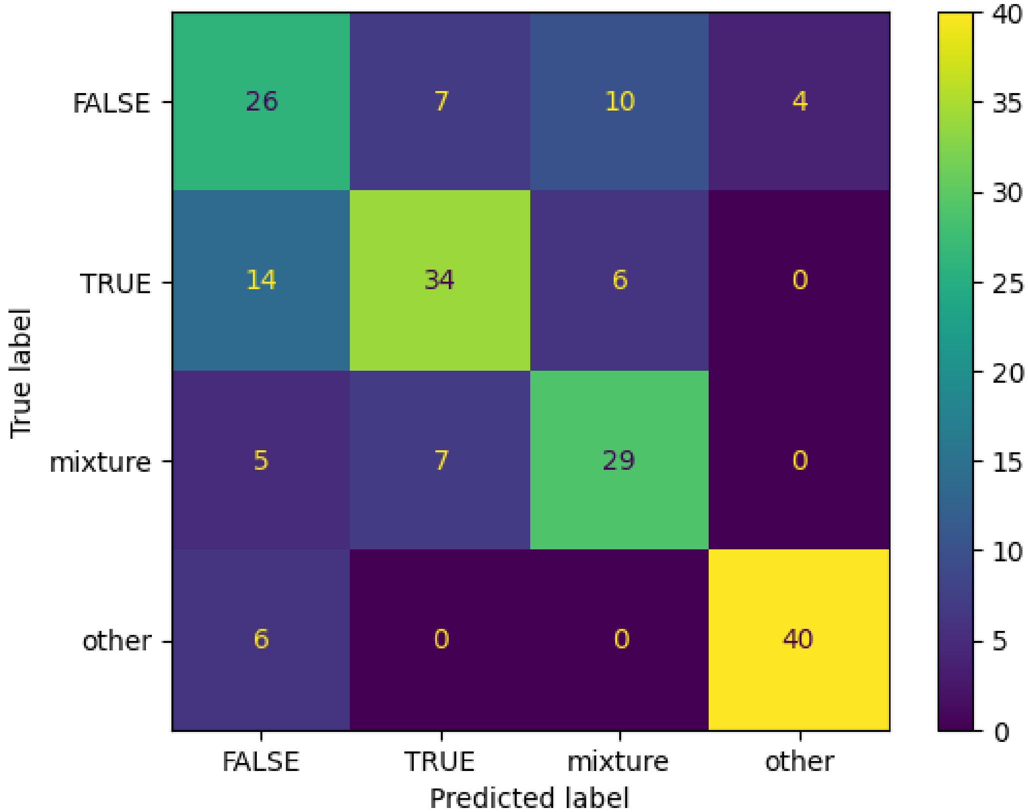

True label
Predicted label
FALSE
TRUE
mixture
other
31 7 6 3
15 35 3 1
3 9 28 1
6 0 0 40
0
5
10
15
20
25
30
35
40


True label
FALSE
TRUE
mixture
other
29
8
6
4
14
37
3
0
3
9
29
0
4
0
2
40
Predicted label
0
5
10
15
20
25
30
35
40


True label
FALSE
TRUE
mixture
other
32
3
6
6
16
31
6
1
9
6
25
1
4
0
0
42
Predicted label
0
5
10
15
20
25
30
35
40


True label
Predicted label
FALSE
TRUE
mixture
other
35 5 5 2
16 33 5 0
3 9 29 0
6 0 0 40
0
5
10
15
20
25
30
35
40


True label
Predicted label
FALSE
TRUE
mixture
other
33 1 11 2
13 36 3 2
4 8 28 1
2 0 2 42
0
5
10
15
20
25
30
35
40



True label
Predicted label
FALSE
TRUE
mixture
other
26
7
10
4
14
34
6
0
5
7
29
0
6
0
0
40
40
35
30
25
20
15
10
5
0