#CLASSIFICATION : TOPIC MODELING -TRUE vs FALSE :

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

# librairie spacy
import spacy

# librairies de gensim
import gensim
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
from gensim.models import Phrases
```

```
from gensim.models.phrases import Phraser
from gensim import corpora
from gensim import models

nltk.download('wordnet')
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

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  import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.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
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

#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
# fonction qui affiche le classification report et la matrice de
confusion
from sklearn import metrics
from sklearn.metrics import confusion_matrix , ConfusionMatrixDisplay
from sklearn.metrics import classification_report



import re
import spacy
import gensim
import string
import nltk
from nltk.corpus import stopwords
from nltk.corpus import wordnet
import gensim
from gensim.utils import simple_preprocess
from gensim.models import Phrases
from gensim.models.phrases import Phraser
from gensim import corpora
from gensim import models
nltk.download('wordnet')
nltk.download('stopwords')
import gensim
from gensim import corpora
import gensim
from gensim.models import Phrases
from gensim.models.phrases import Phraser
stop_words = set(stopwords.words('english'))

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


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
import numpy as np
```

```
[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 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!
```

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
 avecscaler.pkl
 Classification_de_données_textuelles2023.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_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_TOPIC_MODELLING.ipynb'
'TRUE FALSE_vs_OTHER.ipynb'
'TRUE FALSE_vs_OTHER_TOPIC_MODELLING.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 ..

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]

    # 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]   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]   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
- Sélectionner que les lignes où on a True ou False

```
#Ici je cherche à séléctionner que les labels TRUE et FALSE, donc les LIGNES qui contiennent au rating TRUE et FALSE 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'])]
print("Echantillon de mon dataset \n")
print(dftrain.sample(n=10))
print("\n")
print("Quelques informations importantes \n")
dftrain.info()

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

print("le type de X_test est" ,X_text.columns)
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("voici la dernière case")
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 et FALSE sont " ,y.value_counts())
```

```
Echantillon de mon dataset
```

```
            id                                                text  \
671   7957e58e  “There is a plot to discredit the President an...
512   0c21be0d  Audit of signed envelopes won’t change outcome...
2332  14010ae5  Today, the Court of Appeal has ruled that the ...
1350  4684f622  Introduction  Many conservatives have been con...
2222  c57f8234  People with lung cancer are dying after being ...
1590  d382b7f2  America is a Christian nation and the most ess...
2052  08294d0a  Nearly 40,000 Wisconsinites would lose benefit...
98    8b626796  The moving vans are scheduled for Wednesday.  ...
2126  476c6264  An offer of foreign tax allegations against Cl...
1655  ad45e0f7  WASHINGTON, DC — The Pentagon has issued an in...

                                                  title rating
671   MH17 case: Kremlin spokesman declines to comme...  FALSE
512    How the Stimulus Fell Short - The New York Times   TRUE
2332  Court of appeal rules that ‘bedroom tax’ is un...  FALSE
1350  Is It True that Chief Justice John Roberts Vis...  FALSE
2222  A team of scientists from Canada have identifi...  FALSE
1590  Trump Issues Order Deeming Church An Essential...  FALSE
2052  "Anyone raising concern about the safety of Co...   TRUE
98    BREAKING: Disneyland is abandoning California,...  FALSE
2126  Scott Walker plans to give schools $100 millio...   TRUE
1655  Pentagon Confirms Coronavirus Accidently Got I...  FALSE


Quelques informations importantes

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1578 entries, 0 to 2527
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      1578 non-null   object
 1   text    1578 non-null   object
 2   title   1554 non-null   object
 3   rating  1578 non-null   object
dtypes: object(4)
memory usage: 61.6+ KB
le type de X_test est Index(['text'], dtype='object')
le texte est

                                                   text
0     Distracted driving causes more deaths in Canad...
3     But things took a turn for the worse when riot...
4     It’s no secret that Epstein and Schiff share a...
6     November 23, 2019  The U.S. Food and Drug Admi...
7     Trump confirms this was a bombing, not an acci...
...                                                 ...
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...

[1578 rows x 1 columns]

le titre est

                                                  title
0     You Can Be Fined $1,500 If Your Passenger Is U...
3     Obama’s Daughters Caught on Camera Burning US ...
4     Leaked Visitor Logs Reveal Schiff’s 78 Visits ...
6     FDA Shocking Study: Cells Used In Vaccines Con...
7     Israel Hits Beirut with Nuclear Missile, Trump...
...                                                 ...
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

[1578 rows x 1 columns]

voici la dernière case

0       FALSE
3       FALSE
4       FALSE
6       FALSE
7       FALSE
        ...
2523     TRUE
2524     TRUE
2525    FALSE
2526     TRUE
2527     TRUE
Name: rating, Length: 1578, dtype: object

la taille de X_text est (1578, 1)
la taille de y_train est  (1578,)
les valeurs de TRUE et FALSE sont  FALSE    1156
TRUE      422
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 FALSE de telle sorte que le nombre de lignes de FALSE soit = au nbr de lignes de TRUE. 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 = dftrain[dftrain['rating'] == 'FALSE']
df_true = dftrain [dftrain['rating'] == 'TRUE']

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

# Concaténer les deux dataframes
dftrain = pd.concat([df_false_subsampled, df_true])

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



text_title = dftrain.apply(lambda x : '{}
{}'.format(x['text'],x['title']),axis=1)

dftrain['text_title'] = text_title

#print("le titre est")
#display(X_title)
#X_test=dftest.iloc[1:, :4]
y=dftrain.iloc[0:,3:4]
print("le y est")
display(y.shape)
#print("la taille de X_text est",X_text.shape)
#print("la taille de y_train est " ,y.shape)
#print("les valeurs de TRUE et FALSE maintenant sont
" ,y.value_counts())
```

```
le y est
```

```
(844, 1)
```

Installation des librairies qu'on utilise pour le topic modeling

```
!pip install pyLDAvis
!pip install -U gensim
!pip install --upgrade numpy
!pip uninstall numpy
!pip install numpy
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting pyLDAvis
  Downloading pyLDAvis-3.4.1-py3-none-any.whl (2.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.6/2.6 MB 42.7 MB/s eta
0:00:00
```

```
ent already satisfied: numexpr in /usr/local/lib/python3.10/dist-
packages (from pyLDAvis) (2.8.4)
Requirement already satisfied: gensim in
/usr/local/lib/python3.10/dist-packages (from pyLDAvis) (4.3.1)
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (from pyLDAvis) (1.10.1)
Collecting pandas>=2.0.0
  Downloading pandas-2.0.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.3/12.3 MB 86.7 MB/s eta
0:00:00
ent already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from pyLDAvis) (1.2.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from pyLDAvis) (67.7.2)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from pyLDAvis) (3.1.2)
Collecting funcy
  Downloading funcy-2.0-py2.py3-none-any.whl (30 kB)
Requirement already satisfied: scikit-learn>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from pyLDAvis) (1.2.2)
Collecting numpy>=1.24.2
  Downloading numpy-1.24.3-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 77.6 MB/s eta
0:00:00
ent already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=2.0.0->pyLDAvis)
(2023.3)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=2.0.0->pyLDAvis)
(2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=2.0.0->pyLDAvis)
(2.8.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0-
>pyLDAvis) (3.1.0)
Requirement already satisfied: smart-open>=1.8.1 in
/usr/local/lib/python3.10/dist-packages (from gensim->pyLDAvis)
(6.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->pyLDAvis)
(2.1.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas>=2.0.0->pyLDAvis) (1.16.0)
Installing collected packages: funcy, numpy, pandas, pyLDAvis
  Attempting uninstall: numpy
    Found existing installation: numpy 1.22.4
```

```
    Uninstalling numpy-1.22.4:
      Successfully uninstalled numpy-1.22.4
  Attempting uninstall: pandas
    Found existing installation: pandas 1.5.3
    Uninstalling pandas-1.5.3:
      Successfully uninstalled pandas-1.5.3
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
tensorflow 2.12.0 requires numpy<1.24,>=1.22, but you have numpy
1.24.3 which is incompatible.
numba 0.56.4 requires numpy<1.24,>=1.18, but you have numpy 1.24.3
which is incompatible.
google-colab 1.0.0 requires pandas~=1.5.3, but you have pandas 2.0.1
which is incompatible.
Successfully installed funcy-2.0 numpy-1.24.3 pandas-2.0.1 pyLDAvis-
3.4.1
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gensim in
/usr/local/lib/python3.10/dist-packages (4.3.1)
Requirement already satisfied: numpy>=1.18.5 in
/usr/local/lib/python3.10/dist-packages (from gensim) (1.24.3)
Requirement already satisfied: smart-open>=1.8.1 in
/usr/local/lib/python3.10/dist-packages (from gensim) (6.3.0)
Requirement already satisfied: scipy>=1.7.0 in
/usr/local/lib/python3.10/dist-packages (from gensim) (1.10.1)
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (1.24.3)
Found existing installation: numpy 1.24.3
Uninstalling numpy-1.24.3:
  Would remove:
    /usr/local/bin/f2py
    /usr/local/bin/f2py3
    /usr/local/bin/f2py3.10
    /usr/local/lib/python3.10/dist-packages/numpy-1.24.3.dist-info/*
    /usr/local/lib/python3.10/dist-packages/numpy.libs/libgfortran-
040039e1.so.5.0.0

/usr/local/lib/python3.10/dist-packages/numpy.libs/libopenblas64_p-r0-
15028c96.3.21.so
    /usr/local/lib/python3.10/dist-packages/numpy.libs/libquadmath-
96973f99.so.0.0.0
    /usr/local/lib/python3.10/dist-packages/numpy/*
Proceed (Y/n)? y
  Successfully uninstalled numpy-1.24.3
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
```

```
Collecting numpy
  Using cached numpy-1.24.3-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3 MB)
Installing collected packages: numpy
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
tensorflow 2.12.0 requires numpy<1.24,>=1.22, but you have numpy
1.24.3 which is incompatible.
numba 0.56.4 requires numpy<1.24,>=1.18, but you have numpy 1.24.3
which is incompatible.
google-colab 1.0.0 requires pandas~=1.5.3, but you have pandas 2.0.1
which is incompatible.
Successfully installed numpy-1.24.3
```

Dans cette cellule on trouve les définitions de toutes les fonctions qu'on utilise pour le topic modeling:

- MyCleanTextsforLDA pour le nettoyage du text, elle renvoie les bigrammes, corpus bow, corpus tfIdf, le dictionnaire des mots
- dominant_topic qui extrait le topic dominant du corpus
- format_topics_sentence elle renvoie un DataFrame qui contient le topic dominant de chaque document du corpus, ses keywords, et le pourcentage de sa contribution dans le document
- compute_coherences_values pour calculer la cohérence
- MyGridSearchLda elle applique différentes valeurs pour num_topics, eta et alpha et renvoie un DataFrame trié par ordre décroissant de cohérence
- get_best_coherence_values pour tester différents nombre de topics et choisir le meilleur compromis

```
nlp = spacy.load("en_core_web_sm", disable=['parser', 'ner'])
#nlp = spacy.load('en', disable=['parser', 'ner'])

def MyCleanTextsforLDA(texts,
                       min_count=1, # nombre d'apparitions minimale
pour un bigram
                       threshold=2,
                       no_below=1, # nombre minimum d'apparitions pour
être dans le dictionnaire
                       no_above=0.5, # pourcentage maximal (sur la
taille totale du corpus) pour filtrer
                       stop_words=stop_words
                       ):

    allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']
    sentences=texts.copy()

    # suppression des caractères spéciaux
    sentences = [re.sub(r'[^\w\s]', ' ', str(sentence)) for sentence
```

```
in sentences]
    # suppression de tous les caractères uniques
    sentences = [re.sub(r'\s+[a-zA-Z]\s+', ' ', str(sentence)) for
sentence in sentences]
    # substitution des espaces multiples par un seul espace
    sentences = [re.sub(r'\s+', ' ', str(sentence), flags=re.I) for
sentence in sentences]

    # conversion en minuscule et split des mots dans les textes
    sentences = [sentence.lower().split() for sentence in sentences]

    # utilisation de spacy pour ne retenir que les allowed_postags
    texts_out = []
    for sent in sentences:
        if len(sent) < (nlp.max_length): # si le texte est trop grand
            doc = nlp(" ".join(sent))
            texts_out.append(" ".join([token.lemma_ for token in doc
if token.pos_ in allowed_postags]))
        else:
            texts_out.append(sent)
    sentences=texts_out

    # suppression des stopwords
    words =[[word for word in simple_preprocess(str(doc)) if word not
in stop_words] for doc in sentences]


    # recherche des bigrammes
    bigram = Phrases(words, min_count, threshold,delimiter=' ')
    bigram_phraser = Phraser(bigram)


    # sauvergarde des tokens et des bigrammes
    bigram_token = []
    for sent in words:
        bigram_token.append(bigram_phraser[sent])


    # creation du vocabulaire
    dictionary = gensim.corpora.Dictionary(bigram_token)


    # il est possible de filtrer des mots en fonction de leur
occurrence d'apparitions
    #dictionary.filter_extremes(no_below, no_above)
    # et de compacter le dictionnaire
    # dictionary.compactify()
    corpus = [dictionary.doc2bow(text) for text in bigram_token]

    # recuperaction du tfidf plutôt que uniquement le bag of words
```

```
    tfidf = models.TfidfModel(corpus)
    corpus_tfidf = tfidf[corpus]

    return corpus, corpus_tfidf, dictionary, bigram_token




def dominant_topic(model,corpus,num_topics):
    # recuperation du vecteur associé
    # creation d'un dictionnaire pour stocker les résultats
    topic_dictionary = {i: [] for i in range(num_topics)}
    topic_probability_scores =
model.get_document_topics(corpus,minimum_probability=0.000)
    if len(topic_probability_scores) ==1 : # il y a plusieurs
predictions on recupere la premiere
        row=topic_probability_scores[0]
    else: # on concatene les predictions
        tab=[]
        for j in range (len(topic_probability_scores)):
            tab.append(topic_probability_scores[j])
        row=tab
    # parcours des différents topics
    for (topic_num, prop_topic) in row:
            topic_dictionary[topic_num].append(prop_topic)
    # tri pour avoir le plus grand en premier
    list_proba=topic_dictionary
    topic_dictionary=sorted(topic_dictionary,
key=topic_dictionary.get,reverse = True)
    return topic_dictionary, list_proba


def format_topics_sentences(ldamodel, corpus, texts):
    # Initialisation du dataframe de sortie
    sent_topics_df = pd.DataFrame()

    # Recherche le topic dominant pour chaque document
    for i, row_list in enumerate(ldamodel[corpus]):
        row = row_list[0] if ldamodel.per_word_topics else row_list

        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        # Donne le topic dominant, le pourcentage de contribution
        # et les mots clés pour chaque document
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # => topic dominant
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in
wp])
```

```
                sent_topics_df =
sent_topics_df.append(pd.Series([int(topic_num), round(prop_topic,4),
topic_keywords]), ignore_index=True)
            else:
                break
    sent_topics_df.columns = ['topic_dominant', 'pourcentage_contrib',
'topic_keywords']

    # Ajout du texte original à la fin de la sortie
    contents = pd.Series(texts)
    sent_topics_df = pd.concat([sent_topics_df, contents], axis=1)
    return(sent_topics_df)



# ce code est inspiré de
# https://towardsdatascience.com/evaluate-topic-model-in-python-
latent-dirichlet-allocation-lda-7d57484bb5d0
def compute_coherence_values(corpus, dictionary, listtokens, k, alpha,
eta):

    lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                           id2word=dictionary,
                                           num_topics=k,
                                           random_state=100,
                                           chunksize=100,
                                           passes=10,
                                           alpha=alpha,
                                           eta=eta,
                                           per_word_topics=True)

    coherence_model_lda = CoherenceModel(model=lda_model,
texts=listtokens, dictionary=dictionary, coherence='c_v')

    return coherence_model_lda.get_coherence()

def MyGridSearchLda
(corpus,listtokens,dictionnary,nb_topics,alpha,eta,verbose=1):

    grid = {}
    model_results = {'topics': [],
                     'alpha': [],
                     'eta': [],
                     'coherence': []
                    }
    # iteration sur le nombre de topics
    for k in nb_topics:
        # iteration sur les valeurs d'alpha
        for a in alpha:
            # iteration sur les valeurs de eta
```

```
            for e in eta:
                # calcul du score de coherence
                cv = compute_coherence_values(corpus=corpus,
                                              dictionary=dictionary,
                                              listtokens=listtokens,
                                              k=k, alpha=a, eta=e)
                if verbose==1:
                    print ('topics:', k, ' alpha: %0.3f  eta: %0.3f
coherence: %0.3f'%(a,e,cv))

                # sauvegarde des résultats
                model_results['topics'].append(k)
                model_results['alpha'].append(a)
                model_results['eta'].append(e)
                model_results['coherence'].append(cv)

    df_result=pd.DataFrame(model_results)
    df_result = df_result.sort_values('coherence',ascending=False)
    df_result.reset_index(drop=True, inplace=True)
    return df_result



def get_best_coherence_values(corpus, dictionary, listtokens, start=5,
stop=15, step=2):
    coherence_values = []
    model_list = []
    for num_topics in range(start, stop, step):
        lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                               id2word=dictionary,
                                               num_topics=num_topics,
                                               random_state=100,
                                               chunksize=100,
                                               passes=10,
                                               per_word_topics=True)

        coherence_model_lda = CoherenceModel(model=lda_model,
texts=listtokens, dictionary=dictionary, coherence='c_v')
        model_list.append(lda_model)
        coherence_values.append(coherence_model_lda.get_coherence())
    return model_list, coherence_values
```

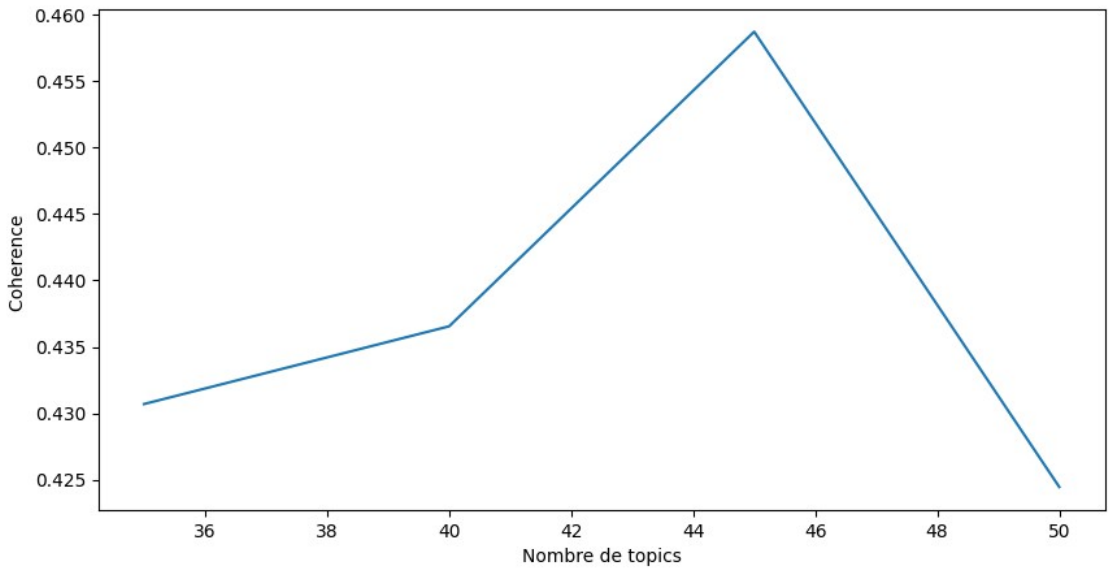
On commence par applique la fonction MyCleantextsforLDA sur la colonne text_title (combinaison des 2 colonnes) et puis on teste différentes valeurs pour pouvoir trouver le bon nombre de topics

```
import pyLDAvis
import pyLDAvis.gensim_models as gensimvis

dftrain.reset_index(drop = True, inplace = True)
```

```
display(dftrain)
dftrain_txt_ttl = dftrain.text_title
stop = stopwords.words('english')

# enrichissement des stopwords
stop.extend(['always','try','go','get','make','would','really',

'like','came','got','article','creativecommons','license','http'])
corpus, corpus_tfidf, dictionary,
bigram_token=MyCleanTextsforLDA(dftrain_txt_ttl)



# test sur un intervalle de 6 à 15 en utilisant le corpus Bow
start=35
stop=55
step=5
model_list, coherence_values =
get_best_coherence_values(dictionary=dictionary,
                                                    corpus=corpus,


listtokens=bigram_token,
                                                    start=start,
stop=stop, step=step)


# affichage du graphe associé à la recherche du nombre de topics
plt.figure(figsize=(10,5))
x = range(start, stop, step)
plt.plot(x, coherence_values)
plt.xlabel("Nombre de topics")
plt.ylabel("Coherence ")
#plt.legend(("Valeurs de cohérencescoherence_values"), loc='best')
plt.show()
```

```
          id                                                text  \
0    f85ea242  It’s been a long time coming, but finally we h...
1    684c9ba4  Constitutional Attorney Matthew DePerno is an ...
2    6c88493a  The United States is witnessing a massive, dan...
3    2aac10a5  After three decades on the bench, Sarah Parker...
4    1e83af88  Based on actual results and accounting for sta...
..        ...                                                ...
839  0de76e26  5 Million Muslim Children In Yemen Died due to...
840  3886ead8  The bombshell claim comes from over 20 hours o...
841  8e197ce3  BILL GATES EXPLAINS THAT THE COVID VACCINE WIL...
842  01ed1b22  Let our journalists help you make sense of the...
843  31d33510  Though the whole world relies on RT-PCR to “di...
```

```
                                                  title rating  \
0    JK Rowling Confirms Stance Against Transgender...   TRUE
1    MI Sec of State Official Caught On Video Telli...  FALSE
2    What science can tell us about the links betwe...   TRUE
3          Sarah Parker leaves legacy on Supreme Court   TRUE
4    Current Actual Election Result Update: Preside...  FALSE
..                                                 ...    ...
839  Re: Meeting the need for isolation space for h...  FALSE
840  Breaking: Breonna Taylor’s boyfriend says SHE ...  FALSE
841  A quote from Politifact: Gates never said that...  FALSE
842  Before This Election, Newt Gingrich Believed t...   TRUE
843   COVID19 PCR Tests are Scientifically Meaningless  FALSE

                                            text_title
0    It’s been a long time coming, but finally we h...
1    Constitutional Attorney Matthew DePerno is an ...
2    The United States is witnessing a massive, dan...
3    After three decades on the bench, Sarah Parker...
4    Based on actual results and accounting for sta...
..                                                 ...
839  5 Million Muslim Children In Yemen Died due to...
840  The bombshell claim comes from over 20 hours o...
841  BILL GATES EXPLAINS THAT THE COVID VACCINE WIL...
842  Let our journalists help you make sense of the...
843  Though the whole world relies on RT-PCR to “di...

[844 rows x 5 columns]
```



45 semble une bonne valeur pour le nombre de topics

- On entraîne notre modèle LDA avec ce nombre de topics, et puis on affiche les topcis avec les mots associés + leurs poids dans chaque topic

- On affiche par la suite la cohérence et la perplexité (qui est censée être petite)
- On applique la méthode format_topics_sentences sur le modèle lda entraîné et notre colonne de text+titre pour avoir un tableau de topic dominant + son pourcentage de contribution et les mots-clés de chaque topic

```
num_words=20 # nombre de mots par topics


num_topic_best=45

lda_model = gensim.models.ldamulticore.LdaMulticore(
                           corpus=corpus,
                           num_topics=num_topic_best,
                           id2word=dictionary,
                           chunksize=100,
                           workers=7,
                           passes=10,
                           random_state=100,
                           eval_every = 1,
                           per_word_topics=True)

print ("Affichage des ",num_topic_best," différents topics pour le
corpus TF-IDF :")

for idx, topic in lda_model.print_topics(-1,num_words):
    print('Topic : {} Words : {}'.format(idx, topic))


coherence_model_lda = CoherenceModel(model=lda_model,
texts=bigram_token, dictionary=dictionary,
                                     coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('Cohérence  : ', coherence_lda)

print('Perplexité : ', lda_model.log_perplexity(corpus))


df_topic_sents_keywords = format_topics_sentences(ldamodel=lda_model,
corpus=corpus, texts=dftrain.text_title)


display(df_topic_sents_keywords)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Affichage des 45 différents topics pour le corpus TF-IDF :
Topic : 0 Words : 0.009*"year" + 0.008*"work" + 0.007*"say" +
0.005*"come" + 0.005*"cent" + 0.005*"open border" + 0.004*"last year"
+ 0.004*"public" + 0.004*"image" + 0.004*"people" + 0.003*"national" +
0.003*"number foreign" + 0.003*"policy see" + 0.003*"worker double" +
0.003*"getty stock" + 0.003*"terrorist attack" + 0.003*"law" +
0.002*"labour" + 0.002*"rise" + 0.002*"long term"
Topic : 1 Words : 0.006*"say" + 0.005*"climate change" + 0.004*"still"
+ 0.004*"also" + 0.004*"year" + 0.004*"maybe" + 0.004*"need" +
0.004*"rise" + 0.003*"world" + 0.003*"planet" + 0.003*"new" +
0.003*"far" + 0.003*"global" + 0.003*"hope" + 0.003*"scientist" +
0.003*"keep" + 0.003*"scenario" + 0.003*"denial" + 0.003*"even" +
0.003*"temperature"
Topic : 2 Words : 0.011*"child" + 0.011*"say" + 0.008*"school" +
0.006*"gender neutral" + 0.005*"asuu" + 0.004*"learn new" +
0.004*"student world" + 0.004*"student" + 0.004*"time" +
0.003*"caption photo" + 0.003*"get" + 0.003*"see" + 0.003*"new way" +
0.003*"want" + 0.003*"hide caption" + 0.003*"girl boy" + 0.003*"world
learn" + 0.003*"photo student" + 0.003*"use" + 0.002*"trade learn"
Topic : 3 Words : 0.013*"pay" + 0.012*"say" + 0.011*"low pay" +
0.011*"work" + 0.008*"reform" + 0.008*"government" + 0.007*"worker" +
0.006*"people" + 0.005*"business" + 0.004*"right" + 0.004*"call" +
0.004*"plan" + 0.004*"hour contract" + 0.003*"year" + 0.003*"well" +
0.003*"report" + 0.003*"labour" + 0.003*"high" + 0.003*"make" +
0.003*"help"
Topic : 4 Words : 0.014*"say" + 0.004*"officer" + 0.003*"people" +
0.003*"year" + 0.003*"shoot" + 0.003*"year old" + 0.003*"see" +
0.002*"time" + 0.002*"drug" + 0.002*"lung" + 0.002*"law enforcement" +
0.002*"people cease" + 0.002*"black lung" + 0.002*"knock" +
0.002*"cop" + 0.002*"tell" + 0.002*"grand jury" + 0.002*"black
economy" + 0.002*"police" + 0.002*"state"
Topic : 5 Words : 0.008*"lung cancer" + 0.007*"say" + 0.005*"percent
increase" + 0.005*"take" + 0.005*"use" + 0.005*"work" + 0.004*"study"
+ 0.003*"wear mask" + 0.003*"people" + 0.003*"mask respirator" +
0.003*"make" + 0.003*"risk death" + 0.003*"year" + 0.003*"strain" +
0.003*"covid" + 0.003*"drug" + 0.002*"risk serious" + 0.002*"time" +
0.002*"laboratory confirm" + 0.002*"mask"
Topic : 6 Words : 0.007*"conservative beaver" + 0.006*"sea level" +
0.006*"church" + 0.005*"say" + 0.005*"trump" + 0.004*"pope first" +
0.004*"vatican well" + 0.004*"executive order" + 0.003*"agent first" +
0.003*"time" + 0.003*"essential business" + 0.003*"see" +
0.003*"police" + 0.003*"global sea" + 0.003*"need" + 0.003*"life" +
0.002*"charge" + 0.002*"level rise" + 0.002*"inch year" + 0.002*"call"
Topic : 7 Words : 0.007*"school" + 0.006*"say" + 0.006*"child" +
0.005*"need" + 0.004*"know" + 0.004*"include" + 0.004*"free school" +
0.003*"year" + 0.003*"also" + 0.003*"people" + 0.003*"use" +
0.003*"call" + 0.003*"address" + 0.003*"well" + 0.003*"get" +
0.003*"report" + 0.003*"make" + 0.002*"add" + 0.002*"patient" +
0.002*"number"
Topic : 8 Words : 0.018*"say" + 0.004*"see" + 0.004*"year" +

0.004*"change" + 0.004*"also" + 0.004*"people" + 0.003*"make" +
0.003*"climate change" + 0.003*"state" + 0.003*"ice" + 0.003*"result"
+ 0.003*"quality control" + 0.002*"work" + 0.002*"get" + 0.002*"small
business" + 0.002*"know" + 0.002*"well" + 0.002*"use" + 0.002*"report"
+ 0.002*"trump"
Topic : 9 Words : 0.005*"year" + 0.003*"people" + 0.003*"departure
average" + 0.003*"century average" + 0.003*"message verifyerror" +
0.003*"metropolitan police" + 0.003*"wave crash" + 0.003*"covid
restriction" + 0.003*"nhs worker" + 0.003*"getty image" + 0.003*"first
time" + 0.002*"th century" + 0.002*"far right" + 0.002*"fourth high" +
0.002*"year record" + 0.002*"email address" + 0.002*"record year" +
0.002*"temperature" + 0.002*"average sea" + 0.002*"temperature th"
Topic : 10 Words : 0.011*"say" + 0.008*"ice shelf" + 0.007*"ice sheet"
+ 0.005*"warm water" + 0.005*"year old" + 0.004*"also" +
0.003*"scientist" + 0.003*"year" + 0.003*"global warming" +
0.003*"shelf" + 0.003*"mile ice" + 0.003*"cape shelf" + 0.002*"go" +
0.002*"take" + 0.002*"part" + 0.002*"warm ocean" + 0.002*"school" +
0.002*"state" + 0.002*"ice" + 0.002*"see"
Topic : 11 Words : 0.007*"also" + 0.006*"state" + 0.006*"government" +
0.005*"confirm case" + 0.004*"people" + 0.004*"health care" +
0.004*"health" + 0.004*"presidential task" + 0.004*"day" + 0.004*"come
week" + 0.003*"report" + 0.003*"say" + 0.003*"federal government" +
0.003*"country" + 0.003*"ask" + 0.002*"many" + 0.002*"covid" +
0.002*"work" + 0.002*"increase number" + 0.002*"capital territory"
Topic : 12 Words : 0.009*"court" + 0.007*"human right" + 0.007*"case"
+ 0.004*"government" + 0.004*"image getty" + 0.004*"trump" +
0.003*"today" + 0.003*"even" + 0.003*"get" + 0.003*"law" +
0.003*"move" + 0.003*"strong" + 0.002*"politician" + 0.002*"police
officer" + 0.002*"mask compliance" + 0.002*"act" + 0.002*"play soon" +
0.002*"video auto" + 0.002*"pelosi" + 0.002*"first"
Topic : 13 Words : 0.008*"vote" + 0.005*"already vote" + 0.004*"say" +
0.004*"tell" + 0.003*"kind problem" + 0.003*"clear vote" +
0.003*"election official" + 0.003*"use" + 0.002*"show" +
0.002*"problem" + 0.002*"people" + 0.002*"also" + 0.002*"get count" +
0.002*"election security" + 0.002*"homeless" + 0.002*"record" +
0.002*"include" + 0.002*"many people" + 0.002*"get" + 0.002*"end"
Topic : 14 Words : 0.005*"world" + 0.004*"mean believe" +
0.004*"state" + 0.003*"also" + 0.003*"see" + 0.003*"people" +
0.003*"average" + 0.003*"take" + 0.003*"vaccine develop" + 0.003*"drug
produce" + 0.003*"change" + 0.003*"lead" + 0.003*"say" + 0.003*"help
people" + 0.003*"new law" + 0.002*"study" + 0.002*"build well" +
0.002*"go far" + 0.002*"follow footstep" + 0.002*"know full"
Topic : 15 Words : 0.009*"say" + 0.008*"muslim woman" + 0.005*"year" +
0.005*"report" + 0.004*"economically inactive" + 0.003*"make" +
0.003*"people" + 0.003*"woman" + 0.002*"year old" + 0.002*"get" +
0.002*"work" + 0.002*"high school" + 0.002*"see" + 0.002*"likely" +
0.002*"high" + 0.002*"come" + 0.002*"school" + 0.002*"pan ideology" +
0.002*"report say" + 0.002*"death"
Topic : 16 Words : 0.016*"say" + 0.009*"year" + 0.005*"change" +
0.003*"make" + 0.003*"get" + 0.002*"mental health" + 0.002*"campaign"

+ 0.002*"spectrum health" + 0.002*"temperature" + 0.002*"country" +
0.002*"public charge" + 0.002*"care" + 0.002*"plead guilty" +
0.002*"need" + 0.002*"go" + 0.002*"place" + 0.002*"point" +
0.002*"know" + 0.002*"well" + 0.002*"far"
Topic : 17 Words : 0.004*"country" + 0.004*"use" + 0.004*"show" +
0.003*"say" + 0.003*"mental health" + 0.003*"year" + 0.003*"general
national" + 0.003*"low mortality" + 0.002*"talk american" +
0.002*"service" + 0.002*"report development" + 0.002*"particular
concern" + 0.002*"arrest charge" + 0.002*"former president" +
0.002*"first" + 0.002*"let know" + 0.002*"arrest espionage" +
0.002*"ocean" + 0.002*"people" + 0.002*"take"
Topic : 18 Words : 0.008*"say" + 0.007*"change" + 0.007*"carbon
dioxide" + 0.004*"year" + 0.003*"work" + 0.003*"turn" + 0.003*"much" +
0.003*"temperature rise" + 0.003*"climate change" + 0.002*"state" +
0.002*"even" + 0.002*"lead" + 0.002*"research center" + 0.002*"high
sea" + 0.002*"assistant leave" + 0.002*"wood research" + 0.002*"amount
carbon" + 0.002*"researcher fairbank" + 0.002*"know much" +
0.002*"core lab"
Topic : 19 Words : 0.005*"also" + 0.003*"covid false" +
0.003*"produce" + 0.003*"pcr test" + 0.003*"comment" + 0.003*"positive
reporter" + 0.003*"biden admit" + 0.003*"high cycle" + 0.003*"memo
chairman" + 0.003*"right cue" + 0.002*"national identity" +
0.002*"show" + 0.002*"know" + 0.002*"say" + 0.002*"religious people" +
0.002*"flag religious" + 0.002*"national flag" + 0.002*"national
civic" + 0.002*"divide national" + 0.002*"coast coast"
Topic : 20 Words : 0.011*"say" + 0.004*"get" + 0.003*"people" +
0.003*"night" + 0.003*"take" + 0.003*"case" + 0.003*"royal british" +
0.002*"come" + 0.002*"care" + 0.002*"happen" + 0.002*"time" +
0.002*"tell" + 0.002*"cost" + 0.002*"vaccine" + 0.002*"deny
allegation" + 0.002*"sexual harassment" + 0.002*"year jail" +
0.002*"lead" + 0.002*"last" + 0.002*"picture"
Topic : 21 Words : 0.010*"say" + 0.004*"junior doctor" + 0.003*"work"
+ 0.003*"doctor" + 0.003*"also" + 0.002*"make" + 0.002*"increase" +
0.002*"government" + 0.002*"year" + 0.002*"mask" + 0.002*"high" +
0.002*"plan" + 0.002*"global warming" + 0.002*"number" + 0.002*"even"
+ 0.002*"prison" + 0.002*"last month" + 0.001*"headline" + 0.001*"also
danger" + 0.001*"go away"
Topic : 22 Words : 0.021*"say" + 0.008*"year" + 0.007*"child" +
0.004*"romney" + 0.004*"also" + 0.003*"use path" + 0.003*"money" +
0.003*"school" + 0.003*"week" + 0.003*"presidential campaign" +
0.003*"early" + 0.002*"time" + 0.002*"form" + 0.002*"child die" +
0.002*"take" + 0.002*"country" + 0.002*"need" + 0.002*"campaign" +
0.002*"state convention" + 0.002*"win percent"
Topic : 23 Words : 0.019*"say" + 0.006*"year" + 0.006*"people" +
0.004*"trump" + 0.003*"vaccine" + 0.003*"tell" + 0.003*"day" +
0.003*"real estate" + 0.003*"lose" + 0.003*"make" + 0.003*"work" +
0.003*"get" + 0.003*"cut" + 0.003*"legislature" + 0.003*"pay" +
0.003*"way" + 0.003*"elisa" + 0.003*"democracy campaign" + 0.003*"part
time" + 0.002*"number"
Topic : 24 Words : 0.008*"test" + 0.008*"say" + 0.007*"even" +

0.006*"virus" + 0.005*"year" + 0.005*"pcr test" + 0.005*"also" +
0.005*"change" + 0.004*"study" + 0.004*"sar cov" + 0.004*"people" +
0.004*"paper" + 0.004*"result" + 0.003*"come" + 0.003*"temperature" +
0.003*"world" + 0.003*"global warming" + 0.003*"level" + 0.003*"show"
+ 0.003*"climate change"
Topic : 25 Words : 0.010*"absentee ballot" + 0.009*"say" +
0.009*"rejection rate" + 0.004*"reject ballot" + 0.004*"high energy" +
0.004*"parallel universe" + 0.004*"also" + 0.003*"trump" + 0.003*"year
ago" + 0.003*"change" + 0.003*"poll worker" + 0.003*"increase" +
0.003*"presidential election" + 0.003*"state law" + 0.003*"ballot
reject" + 0.003*"ballot rejection" + 0.003*"correctly fill" +
0.003*"earth" + 0.002*"work experiment" + 0.002*"time"
Topic : 26 Words : 0.004*"state" + 0.003*"government" + 0.002*"people"
+ 0.002*"propose rule" + 0.002*"vote" + 0.002*"work" + 0.002*"come" +
0.002*"country" + 0.002*"year" + 0.002*"take" + 0.002*"city" +
0.002*"make" + 0.002*"food stamp" + 0.002*"see" + 0.001*"also" +
0.001*"death" + 0.001*"know" + 0.001*"accord" + 0.001*"public" +
0.001*"year old"
Topic : 27 Words : 0.008*"say" + 0.008*"case" + 0.006*"people" +
0.006*"mask" + 0.006*"time" + 0.005*"go" + 0.005*"navy" + 0.005*"make"
+ 0.004*"year" + 0.004*"way" + 0.004*"take" + 0.004*"day" +
0.003*"many" + 0.003*"leave" + 0.003*"see" + 0.003*"political" +
0.003*"good" + 0.003*"government" + 0.003*"sailor" + 0.003*"also"
Topic : 28 Words : 0.008*"say" + 0.008*"virus" + 0.005*"vitamin" +
0.005*"people" + 0.005*"health visitor" + 0.004*"public health" +
0.004*"coronavirus" + 0.004*"show" + 0.003*"also" + 0.003*"level" +
0.003*"report" + 0.003*"state" + 0.003*"year" + 0.003*"risk" +
0.003*"make" + 0.002*"fungus" + 0.002*"case" + 0.002*"work" +
0.002*"find" + 0.002*"child"
Topic : 29 Words : 0.007*"year" + 0.004*"also" + 0.004*"say" +
0.003*"great barrier" + 0.003*"face mask" + 0.003*"work" +
0.003*"property tax" + 0.002*"time" + 0.002*"school" + 0.002*"number"
+ 0.002*"affordable housing" + 0.002*"government" + 0.002*"impact
statement" + 0.002*"ever" + 0.002*"go" + 0.002*"people" +
0.002*"child" + 0.002*"public policy" + 0.001*"see" + 0.001*"show"
Topic : 30 Words : 0.006*"recruit" + 0.005*"study" + 0.005*"case" +
0.004*"mask work" + 0.004*"wear" + 0.004*"fail" + 0.004*"even" +
0.004*"take" + 0.004*"work" + 0.004*"mask" + 0.004*"group" +
0.004*"place" + 0.004*"month" + 0.003*"eat" + 0.003*"salisbury" +
0.003*"skripal" + 0.003*"russian" + 0.003*"last week" +
0.003*"authority" + 0.003*"virus"
Topic : 31 Words : 0.004*"patient die" + 0.003*"take" +
0.002*"insurrection act" + 0.002*"governor" + 0.002*"exchange" +
0.002*"statement" + 0.002*"moderator" + 0.002*"deep state" +
0.002*"oxygen" + 0.002*"exposure" + 0.002*"federal government" +
0.002*"cause coronavirus" + 0.002*"hemoglobin" + 0.002*"deprivation" +
0.002*"structure function" + 0.002*"walker" + 0.002*"decision" +
0.002*"alter" + 0.002*"punt" + 0.002*"erpenbach"
Topic : 32 Words : 0.011*"say" + 0.009*"people" + 0.005*"report" +
0.005*"number" + 0.004*"year" + 0.003*"work" + 0.003*"government" +

0.003*"credit score" + 0.003*"social credit" + 0.003*"measle vaccine" + 0.003*"country" + 0.003*"get" + 0.003*"system" + 0.003*"think" + 0.003*"life" + 0.003*"see" + 0.003*"long term" + 0.002*"coronavirus pandemic" + 0.002*"coronavirus" + 0.002*"also"
Topic : 33 Words : 0.010*"guillotine" + 0.005*"people" + 0.005*"say" + 0.005*"government" + 0.005*"company" + 0.004*"use" + 0.004*"bid" + 0.003*"go" + 0.003*"cut" + 0.003*"device" + 0.003*"canadian government" + 0.003*"detain" + 0.003*"many people" + 0.003*"time" + 0.003*"question" + 0.003*"position" + 0.003*"execute" + 0.003*"year" + 0.003*"island" + 0.003*"canadian"
Topic : 34 Words : 0.013*"say" + 0.006*"day" + 0.003*"dad" + 0.003*"also" + 0.003*"number" + 0.003*"child" + 0.003*"climate change" + 0.002*"mental health" + 0.002*"man" + 0.002*"state" + 0.002*"image" + 0.002*"know experience" + 0.002*"process meat" + 0.002*"challenge" + 0.002*"tell" + 0.002*"know" + 0.002*"problem" + 0.002*"get" + 0.002*"campaign" + 0.002*"survey"
Topic : 35 Words : 0.008*"heatwave picture" + 0.008*"scorch saharan" + 0.008*"seek relief" + 0.006*"saharan heatwave" + 0.006*"relief scorch" + 0.006*"say" + 0.005*"ballot" + 0.005*"native american" + 0.005*"see" + 0.003*"action" + 0.003*"student" + 0.003*"build wall" + 0.003*"climate change" + 0.003*"audit" + 0.003*"medium" + 0.002*"also" + 0.002*"statement" + 0.002*"people" + 0.002*"tell" + 0.002*"temperature"
Topic : 36 Words : 0.007*"earth" + 0.007*"show" + 0.006*"change" + 0.006*"climate change" + 0.005*"state" + 0.005*"year" + 0.004*"also" + 0.004*"even" + 0.003*"human" + 0.003*"depend" + 0.003*"first" + 0.003*"global warming" + 0.003*"planet" + 0.002*"say" + 0.002*"american worker" + 0.002*"labor day" + 0.002*"big american" + 0.002*"national" + 0.002*"time great" + 0.002*"fact"
Topic : 37 Words : 0.011*"say" + 0.007*"voter fraud" + 0.006*"state" + 0.006*"ex cnrp" + 0.005*"number" + 0.005*"presidential election" + 0.005*"vote" + 0.005*"new driver" + 0.005*"election day" + 0.004*"also" + 0.003*"go" + 0.003*"ice shelf" + 0.003*"obtain new" + 0.003*"day registration" + 0.003*"enough swing" + 0.003*"fraudulent vote" + 0.003*"driver license" + 0.003*"registrant register" + 0.003*"college vote" + 0.003*"fide resident"
Topic : 38 Words : 0.010*"high education" + 0.004*"excellence framework" + 0.004*"say" + 0.004*"work" + 0.003*"year" + 0.003*"student" + 0.003*"getty image" + 0.003*"last year" + 0.003*"ensure student" + 0.003*"graduate earning" + 0.003*"drive value" + 0.003*"green paper" + 0.003*"teaching excellence" + 0.003*"money student" + 0.002*"want" + 0.002*"world" + 0.002*"get" + 0.002*"young people" + 0.002*"number" + 0.002*"well"
Topic : 39 Words : 0.010*"say" + 0.007*"year" + 0.006*"food waste" + 0.005*"prison system" + 0.004*"first century" + 0.003*"country" + 0.003*"directive" + 0.003*"tackle food" + 0.003*"people" + 0.003*"also" + 0.003*"choice tool" + 0.002*"time" + 0.002*"last year" + 0.002*"problem" + 0.002*"twentieth century" + 0.002*"use" + 0.002*"sea level" + 0.002*"cause death" + 0.002*"sally coate" + 0.002*"teach first"

```
Topic : 40 Words : 0.010*"people" + 0.006*"virus" + 0.006*"say" +
0.005*"get" + 0.005*"year" + 0.004*"vaccine" + 0.004*"disease" +
0.003*"report" + 0.003*"use" + 0.003*"day nhs" + 0.003*"also" +
0.002*"study" + 0.002*"see" + 0.002*"even" + 0.002*"coronavirus" +
0.002*"go" + 0.002*"time" + 0.002*"cancer" + 0.002*"know" +
0.002*"day"
Topic : 41 Words : 0.010*"state" + 0.006*"year ago" + 0.006*"say" +
0.005*"vote" + 0.005*"mean" + 0.004*"motorcycle manufacture" +
0.004*"go" + 0.004*"election" + 0.003*"elect president" +
0.003*"popular vote" + 0.003*"current system" + 0.003*"national
popular" + 0.003*"day" + 0.003*"child" + 0.003*"child benefit" +
0.003*"speed mile" + 0.003*"fine pay" + 0.003*"mile hour" +
0.003*"well" + 0.003*"call"
Topic : 42 Words : 0.020*"say" + 0.010*"bill" + 0.005*"state" +
0.004*"year" + 0.004*"wetland" + 0.003*"also" + 0.003*"discount" +
0.003*"include" + 0.003*"job" + 0.003*"make" + 0.003*"legislation" +
0.002*"get" + 0.002*"pay" + 0.002*"pier" + 0.002*"power" +
0.002*"allow" + 0.002*"business" + 0.002*"group" + 0.002*"mercury" +
0.002*"change"
Topic : 43 Words : 0.003*"use" + 0.003*"time" + 0.003*"year old" +
0.003*"gugino attempt" + 0.003*"court hearing" + 0.003*"buffalo
police" + 0.003*"professional agitator" + 0.002*"state" + 0.002*"year"
+ 0.002*"case" + 0.002*"show" + 0.002*"back" + 0.002*"buffalo cop" +
0.002*"twitter com" + 0.002*"emerge show" + 0.002*"second degree" +
0.002*"year prison" + 0.002*"leave" + 0.002*"public opinion" +
0.002*"peaceful protester"
Topic : 44 Words : 0.011*"vaccine" + 0.010*"say" + 0.007*"poverty" +
0.006*"year" + 0.005*"time" + 0.004*"work" + 0.004*"take" +
0.004*"first" + 0.003*"asian american" + 0.003*"use" + 0.003*"covid
vaccine" + 0.003*"new" + 0.003*"school" + 0.003*"likely" +
0.003*"government" + 0.003*"resource center" + 0.003*"record" +
0.002*"sea ice" + 0.002*"lead cause" + 0.002*"see"
Cohérence  :  0.3499304230649635
Perplexité :  -11.057605969633673

     topic_dominant  pourcentage_contrib  \
0                24               0.9948
1                35               0.6276
2                 9               0.8125
3                22               0.9954
4                14               0.9935
..              ...                  ...
839              22               0.9916
840               4               0.9964
841              44               0.9976
842              41               0.9937
843              24               0.7681

                                        topic_keywords  \
0    test, say, even, virus, year, pcr test, also, ...
```

```
1    heatwave picture, scorch saharan, seek relief,...
2    year, people, departure average, century avera...
3    say, year, child, romney, also, use path, mone...
4    world, mean believe, state, also, see, people,...
..                                                 ...
839  say, year, child, romney, also, use path, mone...
840  say, officer, people, year, shoot, year old, s...
841  vaccine, say, poverty, year, time, work, take,...
842  state, year ago, say, vote, mean, motorcycle m...
843  test, say, even, virus, year, pcr test, also, ...

                                            text_title
0    It’s been a long time coming, but finally we h...
1    Constitutional Attorney Matthew DePerno is an ...
2    The United States is witnessing a massive, dan...
3    After three decades on the bench, Sarah Parker...
4    Based on actual results and accounting for sta...
..                                                 ...
839  5 Million Muslim Children In Yemen Died due to...
840  The bombshell claim comes from over 20 hours o...
841  BILL GATES EXPLAINS THAT THE COVID VACCINE WIL...
842  Let our journalists help you make sense of the...
843  Though the whole world relies on RT-PCR to “di...

[844 rows x 4 columns]
```

On rajoute les mots-clés à notre DataFrame de départ pour pouvoir faire la classification

- On a essayé la classification sur les keywords uniquement mais l'accuracy était très basse donc on va essayer de rajouter les mots-clés à notre text, titre, text+titre respectivement
- On split notre jeu de données en jeu d'apprentissage et de test

```
# modification du dataframe pour intégrer les mots associés au topic
dominant à chaque document
dftrain['keywords']=df_topic_sents_keywords['topic_keywords']
display(dftrain)
```

```
# selection des données
X=pd.concat([dftrain.iloc[:,1:3], dftrain.iloc[:,4:6]],
axis=1).reset_index(drop=True)
#X = dftrain.keywords
y=dftrain.rating

# Création d'un jeu d'apprentissage et de test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2,random_state=8)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

	id	text	\
0	f85ea242	It’s been a long time coming, but finally we h...	
1	684c9ba4	Constitutional Attorney Matthew DePerno is an ...	
2	6c88493a	The United States is witnessing a massive, dan...	
3	2aac10a5	After three decades on the bench, Sarah Parker...	
4	1e83af88	Based on actual results and accounting for sta...	
..	...	...	
839	0de76e26	5 Million Muslim Children In Yemen Died due to...	
840	3886ead8	The bombshell claim comes from over 20 hours o...	
841	8e197ce3	BILL GATES EXPLAINS THAT THE COVID VACCINE WIL...	
842	01ed1b22	Let our journalists help you make sense of the...	
843	31d33510	Though the whole world relies on RT-PCR to “di...	

	title	rating	\
0	JK Rowling Confirms Stance Against Transgender...	TRUE	
1	MI Sec of State Official Caught On Video Telli...	FALSE	
2	What science can tell us about the links betwe...	TRUE	
3	Sarah Parker leaves legacy on Supreme Court	TRUE	
4	Current Actual Election Result Update: Preside...	FALSE	
..	...	...	
839	Re: Meeting the need for isolation space for h...	FALSE	
840	Breaking: Breonna Taylor’s boyfriend says SHE ...	FALSE	
841	A quote from Politifact: Gates never said that...	FALSE	
842	Before This Election, Newt Gingrich Believed t...	TRUE	
843	COVID19 PCR Tests are Scientifically Meaningless	FALSE	

	text_title	\
0	It’s been a long time coming, but finally we h...	
1	Constitutional Attorney Matthew DePerno is an ...	
2	The United States is witnessing a massive, dan...	
3	After three decades on the bench, Sarah Parker...	
4	Based on actual results and accounting for sta...	
..	...	
839	5 Million Muslim Children In Yemen Died due to...	
840	The bombshell claim comes from over 20 hours o...	
841	BILL GATES EXPLAINS THAT THE COVID VACCINE WIL...	
842	Let our journalists help you make sense of the...	
843	Though the whole world relies on RT-PCR to “di...	

	keywords
0	test, say, even, virus, year, pcr test, also, ...
1	heatwave picture, scorch saharan, seek relief,...
2	year, people, departure average, century avera...

```
3    say, year, child, romney, also, use path, mone...
4    world, mean believe, state, also, see, people,...
..                                                 ...
839  say, year, child, romney, also, use path, mone...
840  say, officer, people, year, shoot, year old, s...
841  vaccine, say, poverty, year, time, work, take,...
842  state, year ago, say, vote, mean, motorcycle m...
843  test, say, even, virus, year, pcr test, also, ...

[844 rows x 6 columns]
```

Vu qu'on va travailler sur text+keywords puis sur titre+keywords après sur la colonne de concaténation de titre et text+keywords, Donc on va d'abord concaténér :

- Texte et keywords
- Titre et keywords
- Titre+texte et keywords

et on va séléctionner ces dernières depuis le X_train et X_test pour apprendre et tester après

```
train_text_keywords = X_train.apply(lambda x : '{}
{}'.format(x['text'],x['keywords']),axis=1)
test_text_keywords = X_test.apply(lambda x : '{}
{}'.format(x['text'],x['keywords']),axis=1)

X_train['text_keywords'] = train_text_keywords
X_train_text_keywords = X_train['text_keywords']
X_train_text_keywords.reset_index(drop = True, inplace = True)

X_test['text_keywords'] = test_text_keywords
X_test_text_keywords = X_test['text_keywords']
X_test_text_keywords.reset_index(drop = True, inplace = True)

train_title_keywords = X_train.apply(lambda x : '{}
{}'.format(x['title'],x['keywords']),axis=1)
test_title_keywords = X_test.apply(lambda x : '{}
{}'.format(x['title'],x['keywords']),axis=1)


X_train['title_keywords'] = train_title_keywords
X_train_title_keywords = X_train['title_keywords']
X_train_title_keywords.reset_index(drop = True, inplace = True)

X_test['title_keywords'] = test_title_keywords
X_test_title_keywords = X_test['title_keywords']
X_test_title_keywords.reset_index(drop = True, inplace = True)

train_text_title_keywords = X_train.apply(lambda x : '{}
```

```
{}'.format(x['text_title'],x['keywords']),axis=1)
test_text_title_keywords = X_test.apply(lambda x : '{}
{}'.format(x['text_title'],x['keywords']),axis=1)


X_train['text_title_keywords'] = train_text_title_keywords
X_train_text_title_keywords = X_train['text_title_keywords']
X_train_text_title_keywords.reset_index(drop = True, inplace = True)

X_test['text_title_keywords'] = test_text_title_keywords
X_test_text_title_keywords = X_test['text_title_keywords']
X_test_text_title_keywords.reset_index(drop = True, inplace = True)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Etape 2 : Classification selon la colonne TEXT et KEYWORDS (concaténés) :

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.

```
# Utilisez la méthode ravel() pour transformer y_train en un tableau
unidimensionnel
y_train = np.ravel(y_train)

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))
]
```

```
#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))
    #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_keywords,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)

    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)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)

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())])

Exception ignored on calling ctypes callback function: <function
ThreadpoolController._find_libraries_with_dl_iterate_phdr.<locals>.mat
ch_library_callback at 0x7fbaa1d78d30>
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/threadpoolctl.py",
line 584, in match_library_callback
    self._make_controller_from_path(filepath)
  File "/usr/local/lib/python3.10/dist-packages/threadpoolctl.py",
line 725, in _make_controller_from_path
    lib_controller = lib_controller_class(
  File "/usr/local/lib/python3.10/dist-packages/threadpoolctl.py",
line 842, in __init__
    super().__init__(**kwargs)
  File "/usr/local/lib/python3.10/dist-packages/threadpoolctl.py",
line 810, in __init__
    self._dynlib = ctypes.CDLL(filepath, mode=_RTLD_NOLOAD)
  File "/usr/lib/python3.10/ctypes/__init__.py", line 374, in __init__
    self._handle = _dlopen(self._name, mode)
OSError:
/usr/local/lib/python3.10/dist-packages/numpy.libs/libopenblas64_p-r0-
2f7c42d4.3.18.so: cannot open shared object file: No such file or
directory

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', 0.9068042142230027, 0.03473994806063803),
('RF', 0.8903204565408253, 0.02837295958824581), ('CART',
0.8445127304653205, 0.035518762967657776), ('MultinomialNB',
0.8324626865671642, 0.0655043788410142), ('LogisticRegression',
0.8237269534679543, 0.047255651601879535), ('KNN', 0.7095039508340649,
0.06983686546327672)]
```

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_keywords_SVC = []
X_test_text_keywords_SVC = []

X_train_text_keywords_RandomForestClassifier = []
X_test_text_keywords_RandomForestClassifier = []


for name, pipeline in all_models :

X_train_text_keywords_SVC.append(pipeline.fit_transform(X_train_text_k
eywords).toarray())

X_test_text_keywords_SVC.append(pipeline.transform(X_test_text_keyword
s).toarray())

X_train_text_keywords_RandomForestClassifier.append(pipeline.fit_trans
form(X_train_text_keywords).toarray())

X_test_text_keywords_RandomForestClassifier.append(pipeline.transform(
X_test_text_keywords).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_keywords = eval('X_train_text_keywords_' +
model_name)
    X_test_text_keywords = eval('X_test_text_keywords_' + model_name)
    for i in range (len(X_train_text_keywords)):
      grid_search = GridSearchCV(model, params[model_name], n_jobs=-1,
verbose=1,scoring=score)
      print("grid search fait")
      grid_search.fit(X_train_text_keywords[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_keywords[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]))
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)

grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.850

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

Accuracy : 0.870
Classification Report
```

```
              precision    recall  f1-score   support

       FALSE    0.89610   0.83133   0.86250        83
        TRUE    0.84783   0.90698   0.87640        86

    accuracy                        0.86982       169
   macro avg    0.87196   0.86915   0.86945       169
weighted avg    0.87154   0.86982   0.86958       169

Ensemble des meilleurs paramètres :
     C: 1.0
     gamma: 'scale'
     kernel: 'linear'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.853

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

Accuracy : 0.888
Classification Report
              precision    recall  f1-score   support

       FALSE    0.92105   0.84337   0.88050        83
        TRUE    0.86022   0.93023   0.89385        86

    accuracy                        0.88757       169
   macro avg    0.89063   0.88680   0.88718       169
weighted avg    0.89009   0.88757   0.88730       169

Ensemble des meilleurs paramètres :
     C: 1.0
     gamma: 'scale'
     kernel: 'linear'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.867

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

Accuracy : 0.923
Classification Report
              precision    recall  f1-score   support

       FALSE    0.97297   0.86747   0.91720        83
        TRUE    0.88421   0.97674   0.92818        86

    accuracy                        0.92308       169
   macro avg    0.92859   0.92211   0.92269       169
```

```
weighted avg    0.92780   0.92308   0.92278       169

Ensemble des meilleurs paramètres :
     C: 1.0
     gamma: 'scale'
     kernel: 'linear'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.856

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

Accuracy : 0.893
Classification Report
              precision    recall  f1-score   support

       FALSE    0.91139   0.86747   0.88889        83
        TRUE    0.87778   0.91860   0.89773        86

    accuracy                        0.89349       169
   macro avg    0.89459   0.89304   0.89331       169
weighted avg    0.89429   0.89349   0.89339       169

Ensemble des meilleurs paramètres :
     C: 10
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.884

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

Accuracy : 0.899
Classification Report
              precision    recall  f1-score   support

       FALSE    0.90244   0.89157   0.89697        83
        TRUE    0.89655   0.90698   0.90173        86

    accuracy                        0.89941       169
   macro avg    0.89950   0.89927   0.89935       169
weighted avg    0.89944   0.89941   0.89939       169

Ensemble des meilleurs paramètres :
     C: 2
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
```

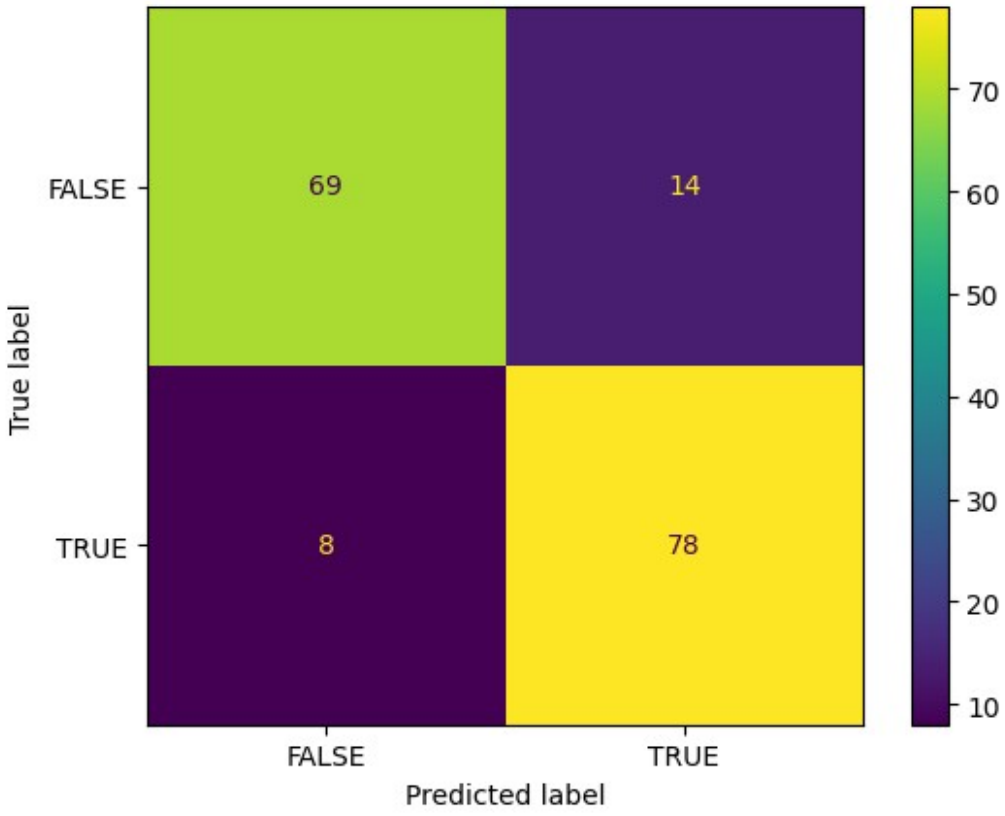
```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.890

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

Accuracy : 0.905
Classification Report
              precision    recall  f1-score   support

       FALSE    0.86813   0.95181   0.90805        83
        TRUE    0.94872   0.86047   0.90244        86

    accuracy                        0.90533       169
   macro avg    0.90842   0.90614   0.90524       169
weighted avg    0.90914   0.90533   0.90519       169

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 0.892

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

Accuracy : 0.923
Classification Report
              precision    recall  f1-score   support

       FALSE    0.90698   0.93976   0.92308        83
        TRUE    0.93976   0.90698   0.92308        86

    accuracy                        0.92308       169
   macro avg    0.92337   0.92337   0.92308       169
weighted avg    0.92366   0.92308   0.92308       169

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 0.886

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

Accuracy : 0.905
Classification Report
```

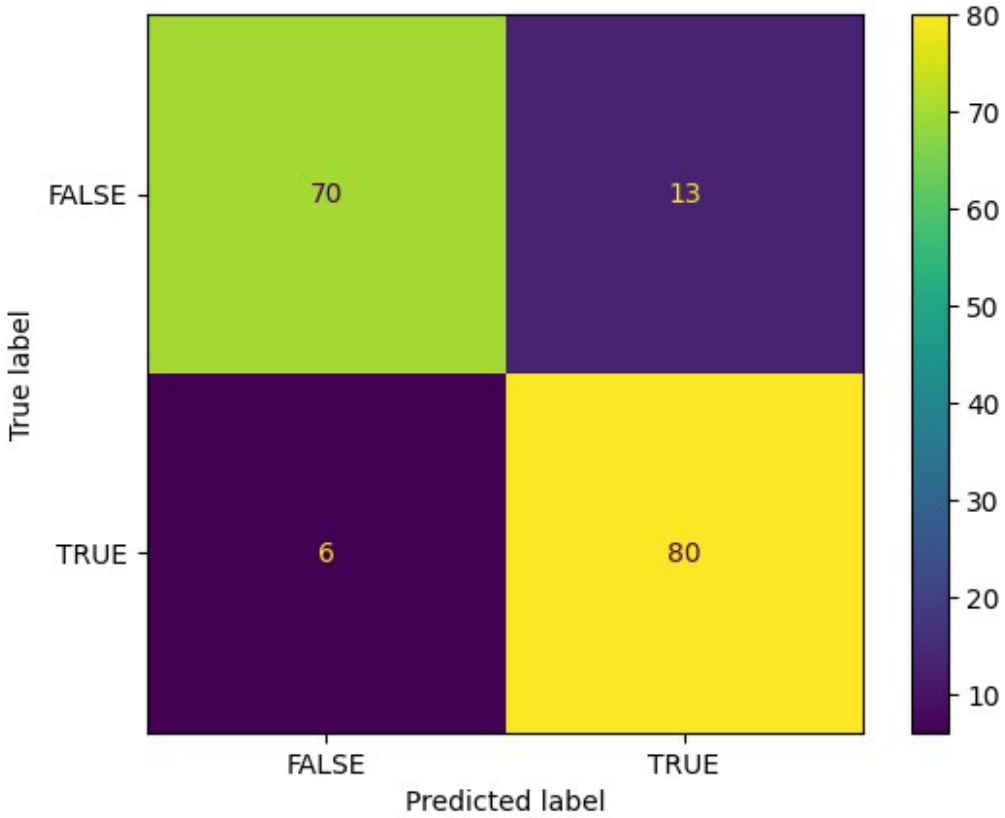
```
              precision    recall  f1-score   support

       FALSE    0.90361   0.90361   0.90361        83
        TRUE    0.90698   0.90698   0.90698        86

    accuracy                        0.90533       169
   macro avg    0.90530   0.90530   0.90530       169
weighted avg    0.90533   0.90533   0.90533       169

Ensemble des meilleurs paramètres :
     C: 2
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.868

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

Accuracy : 0.893
Classification Report
              precision    recall  f1-score   support

       FALSE    0.90123   0.87952   0.89024        83
        TRUE    0.88636   0.90698   0.89655        86

    accuracy                        0.89349       169
   macro avg    0.89380   0.89325   0.89340       169
weighted avg    0.89367   0.89349   0.89345       169

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 0.862

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

Accuracy : 0.911
Classification Report
              precision    recall  f1-score   support

       FALSE    0.97222   0.84337   0.90323        83
        TRUE    0.86598   0.97674   0.91803        86

    accuracy                        0.91124       169
```

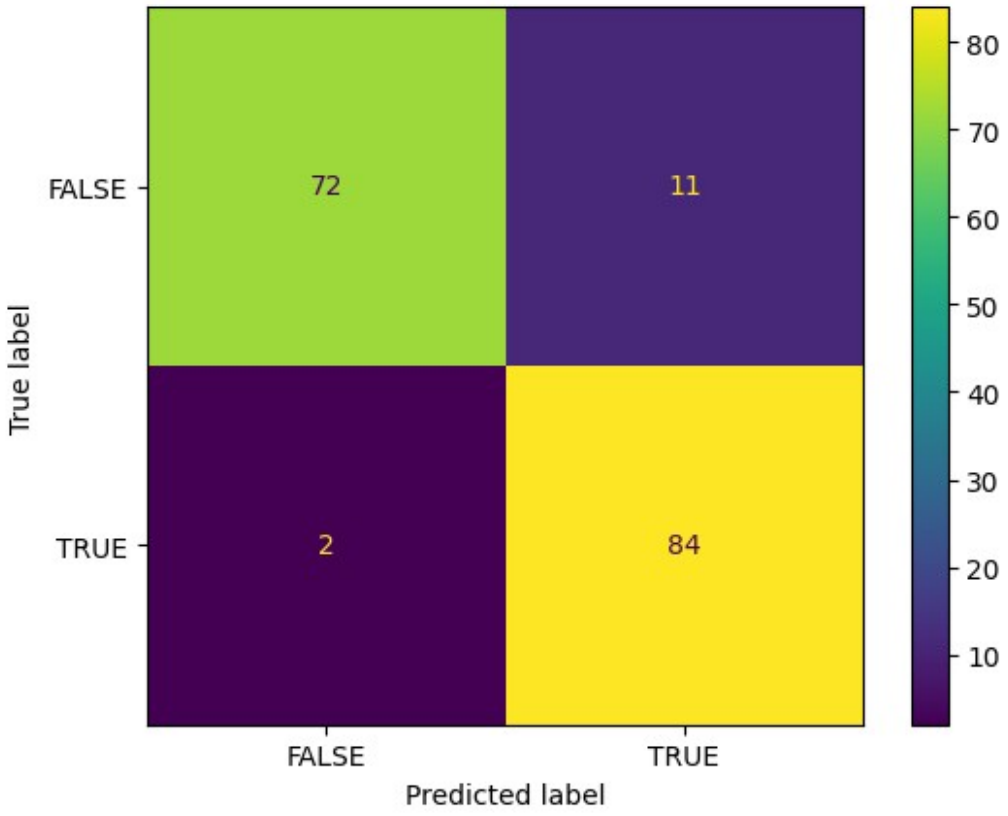
```
   macro avg    0.91910   0.91006   0.91063       169
weighted avg    0.91816   0.91124   0.91076       169

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 0.867

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

Accuracy : 0.911
Classification Report
              precision    recall  f1-score   support

       FALSE    0.86957   0.96386   0.91429        83
        TRUE    0.96104   0.86047   0.90798        86

    accuracy                        0.91124       169
   macro avg    0.91530   0.91216   0.91113       169
weighted avg    0.91611   0.91124   0.91107       169

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 0.867

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

Accuracy : 0.899
Classification Report
              precision    recall  f1-score   support

       FALSE    0.90244   0.89157   0.89697        83
        TRUE    0.89655   0.90698   0.90173        86

    accuracy                        0.89941       169
   macro avg    0.89950   0.89927   0.89935       169
weighted avg    0.89944   0.89941   0.89939       169

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

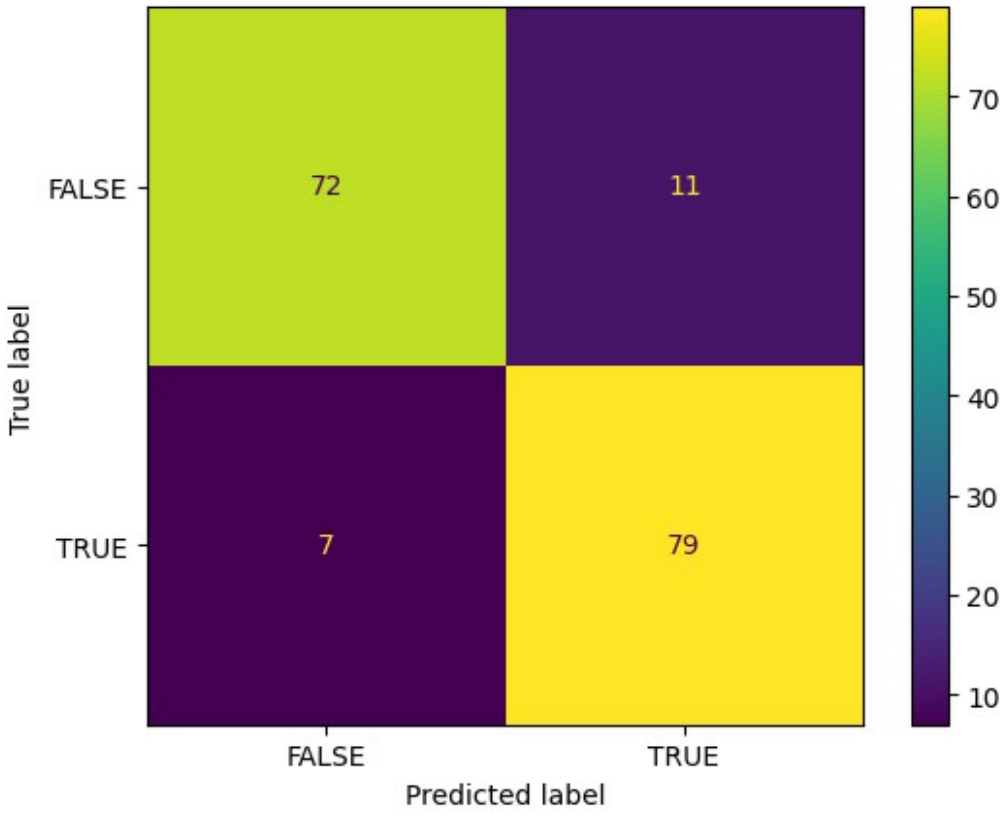
```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.879

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

Accuracy : 0.923
Classification Report
              precision    recall  f1-score   support

       FALSE    0.90698   0.93976   0.92308        83
        TRUE    0.93976   0.90698   0.92308        86

    accuracy                        0.92308       169
   macro avg    0.92337   0.92337   0.92308       169
weighted avg    0.92366   0.92308   0.92308       169

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 0.871

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

Accuracy : 0.893
Classification Report
              precision    recall  f1-score   support

       FALSE    0.84946   0.95181   0.89773        83
        TRUE    0.94737   0.83721   0.88889        86

    accuracy                        0.89349       169
   macro avg    0.89842   0.89451   0.89331       169
weighted avg    0.89928   0.89349   0.89323       169

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 0.876

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

Accuracy : 0.923
```

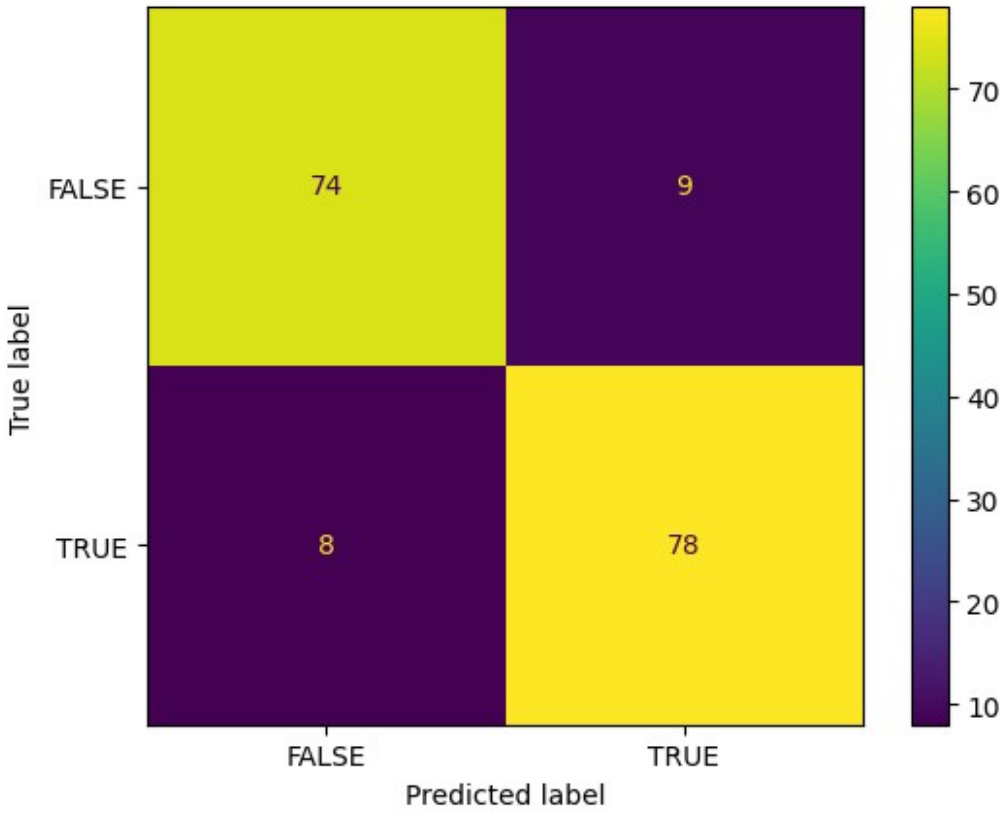
```
Classification Report
              precision    recall  f1-score   support

       FALSE    0.90698   0.93976   0.92308        83
        TRUE    0.93976   0.90698   0.92308        86

    accuracy                        0.92308       169
   macro avg    0.92337   0.92337   0.92308       169
weighted avg    0.92366   0.92308   0.92308       169

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 0.874

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

Accuracy : 0.911
Classification Report
              precision    recall  f1-score   support

       FALSE    0.90476   0.91566   0.91018        83
        TRUE    0.91765   0.90698   0.91228        86

    accuracy                        0.91124       169
   macro avg    0.91120   0.91132   0.91123       169
weighted avg    0.91132   0.91124   0.91125       169

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

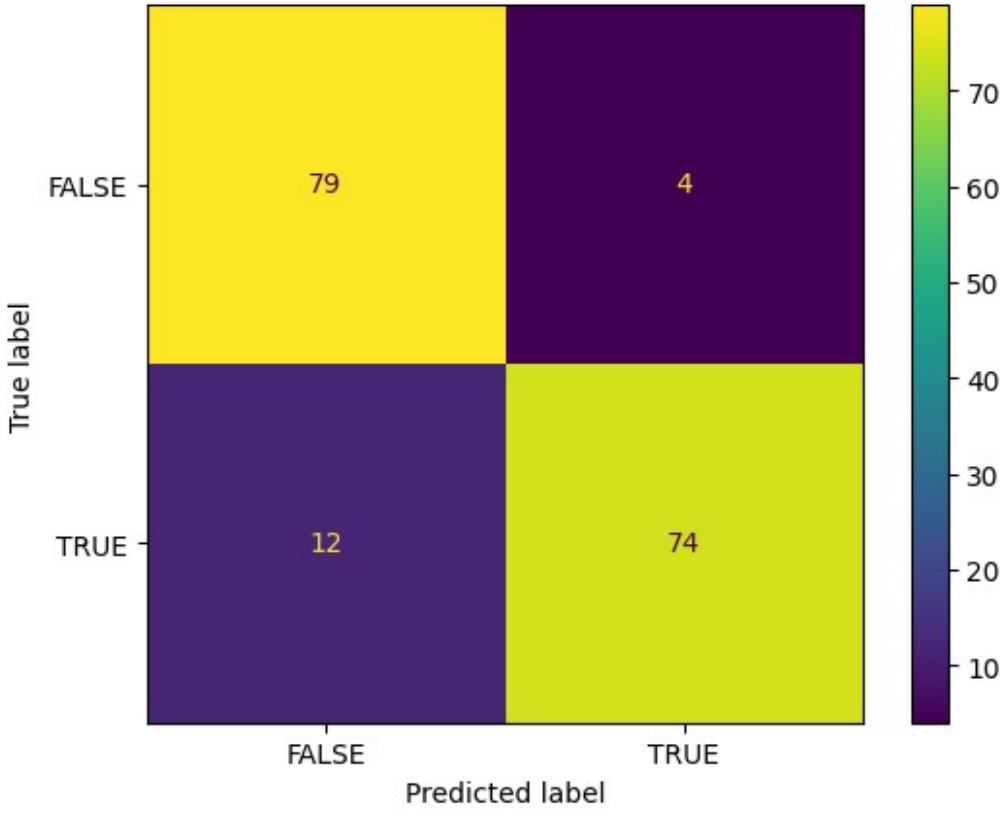

69
14
8
78
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

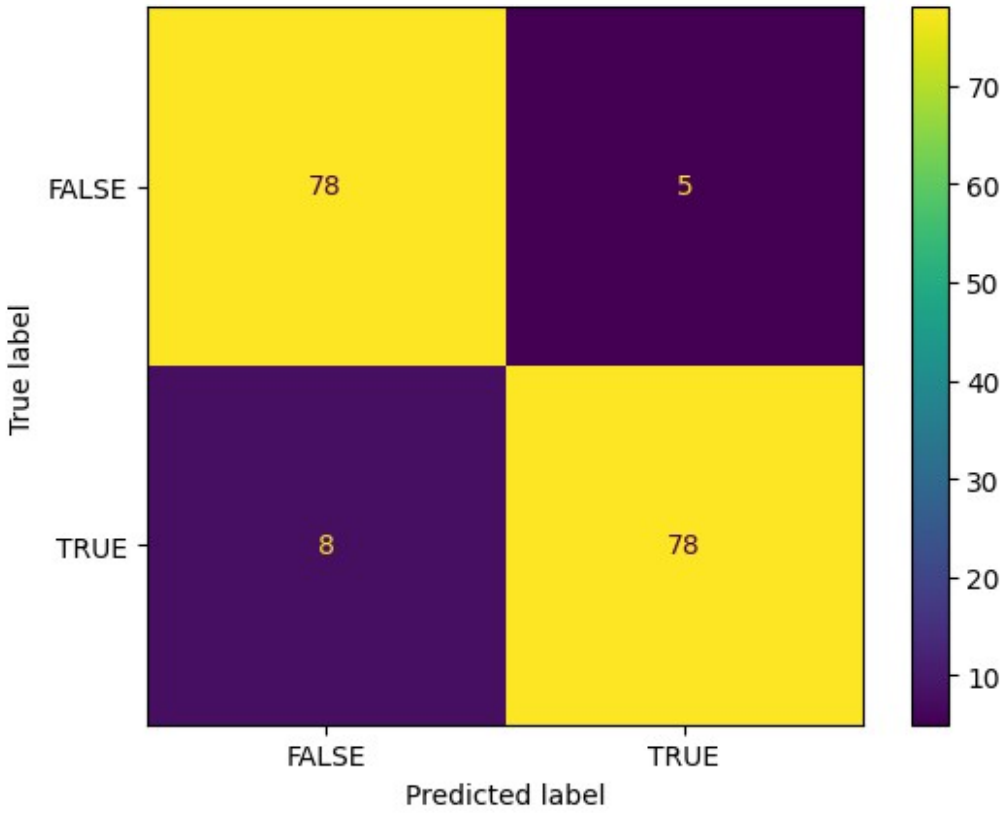

70
13
6
80
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10

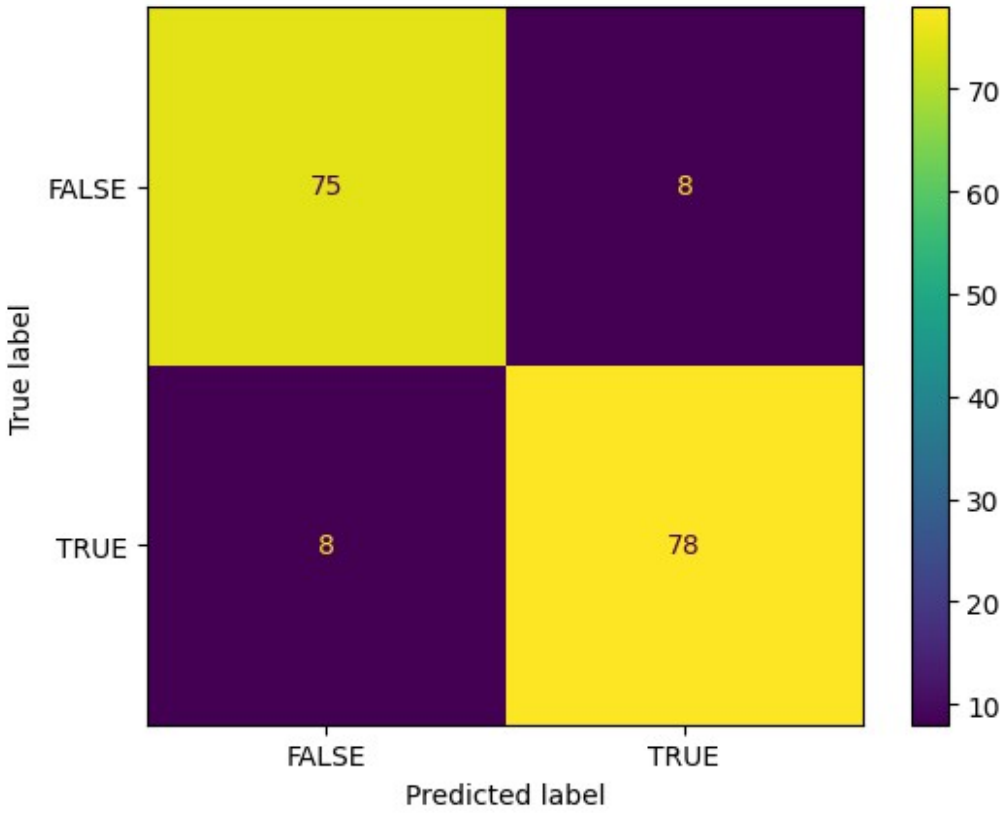

FALSE
TRUE
True label
72
11
2
84
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10

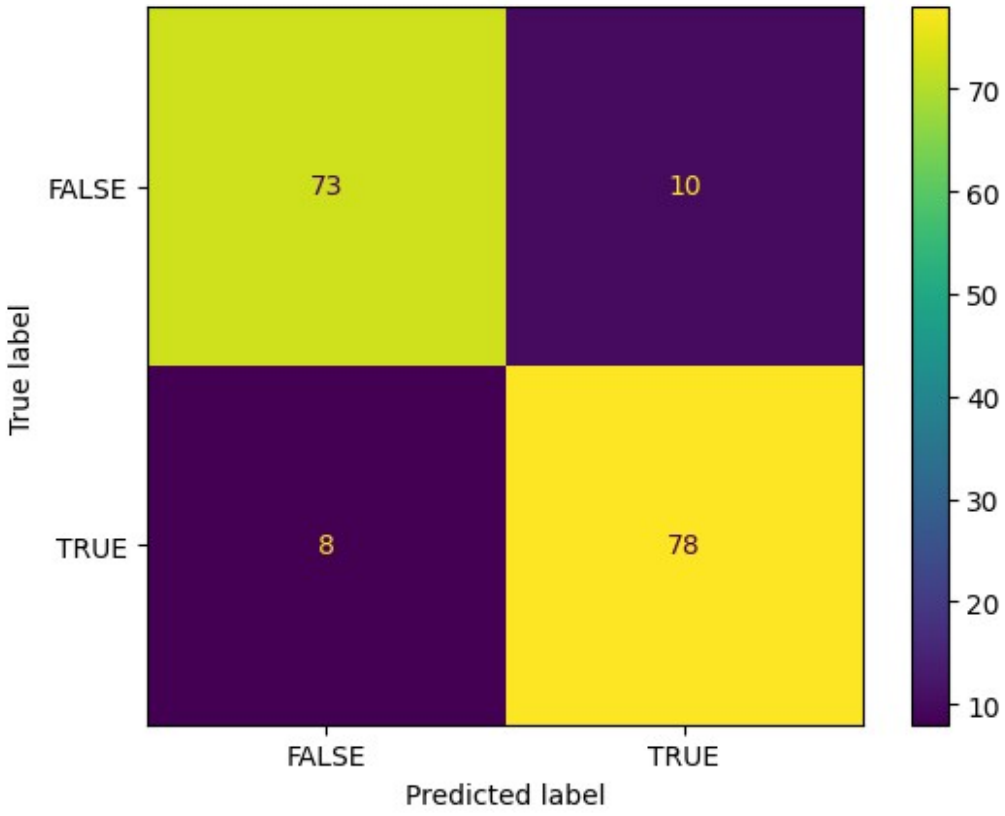

FALSE
TRUE
True label
72
11
7
79
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

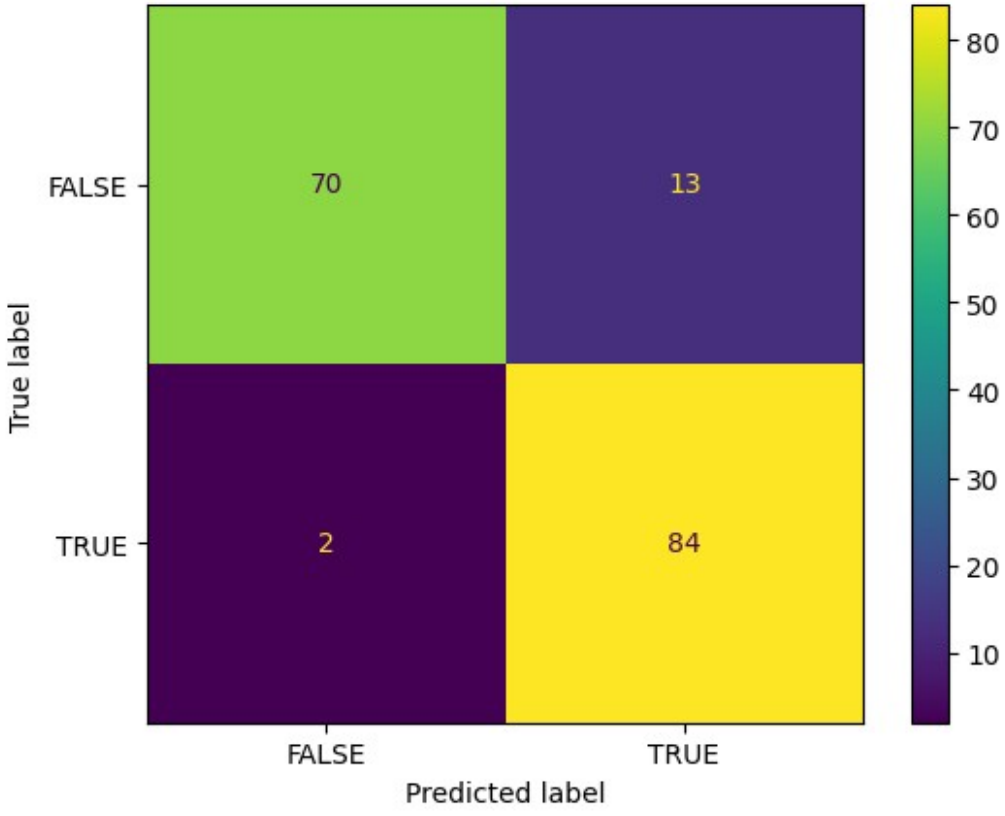

74
9
8
78
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

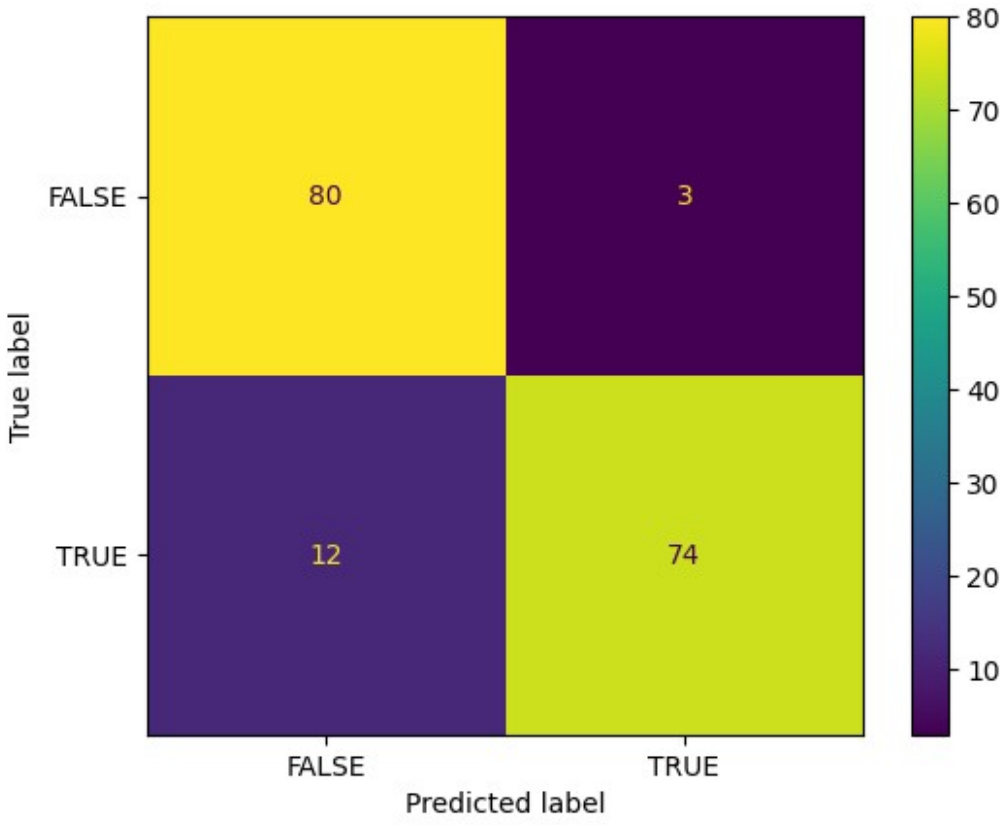

79
4
12
74
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

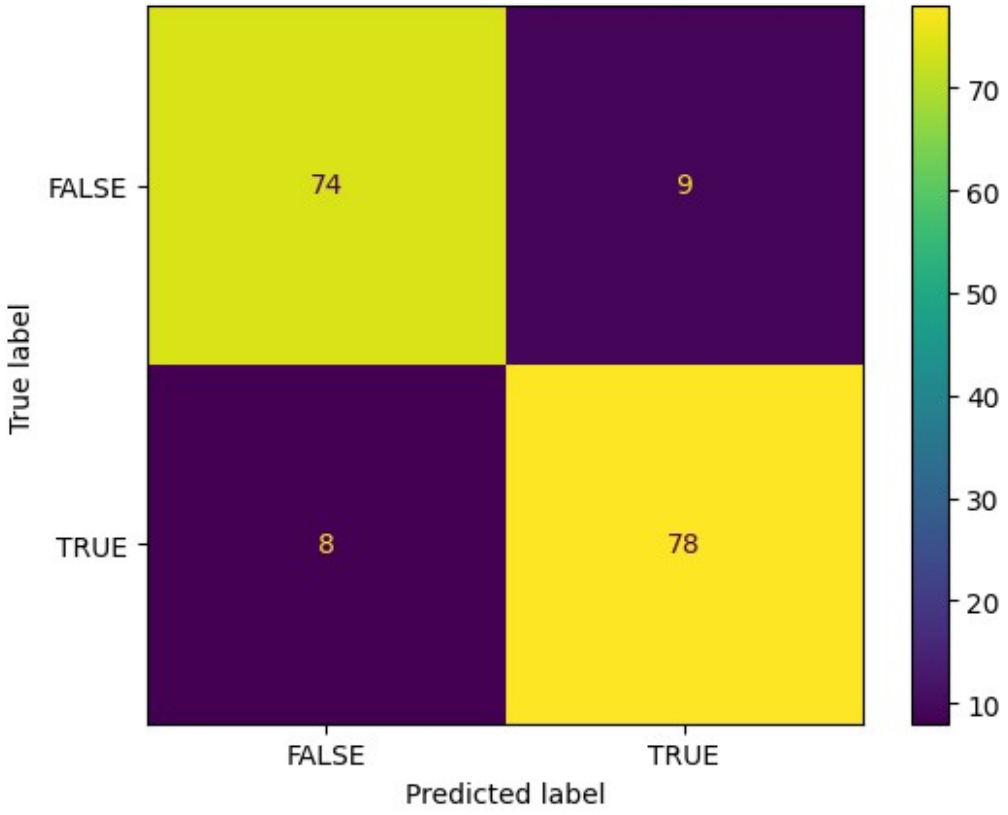

FALSE
TRUE
True label
78
5
8
78
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

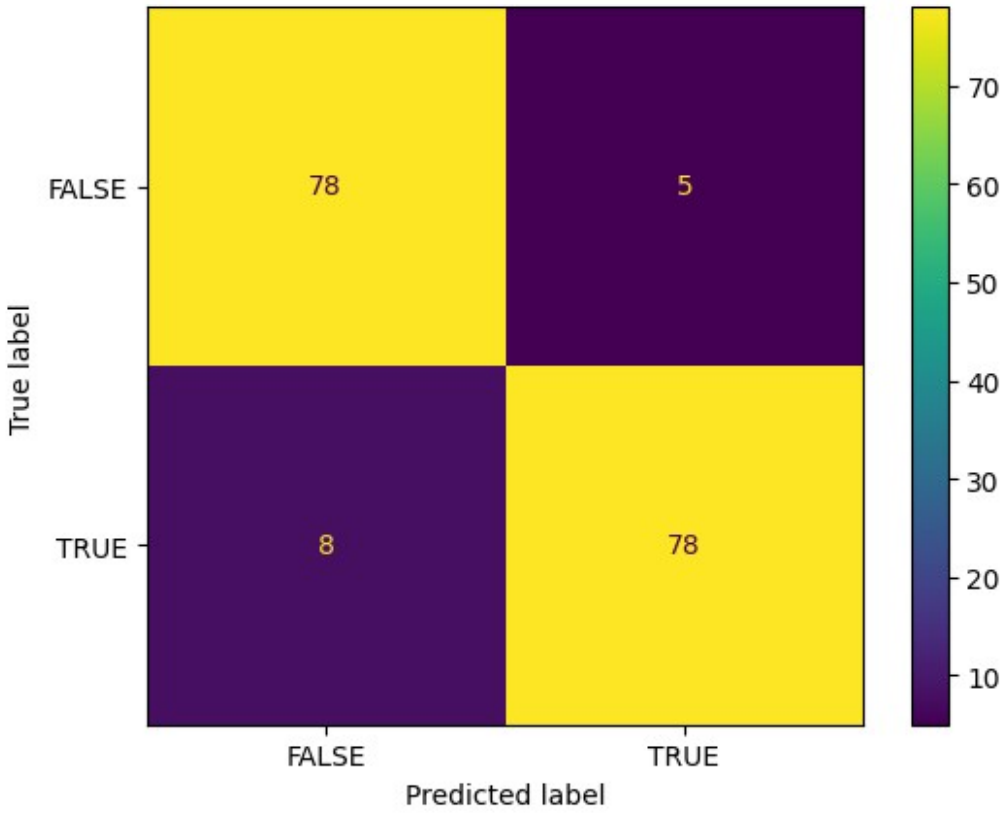

FALSE
TRUE
True label
75
8
8
78
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

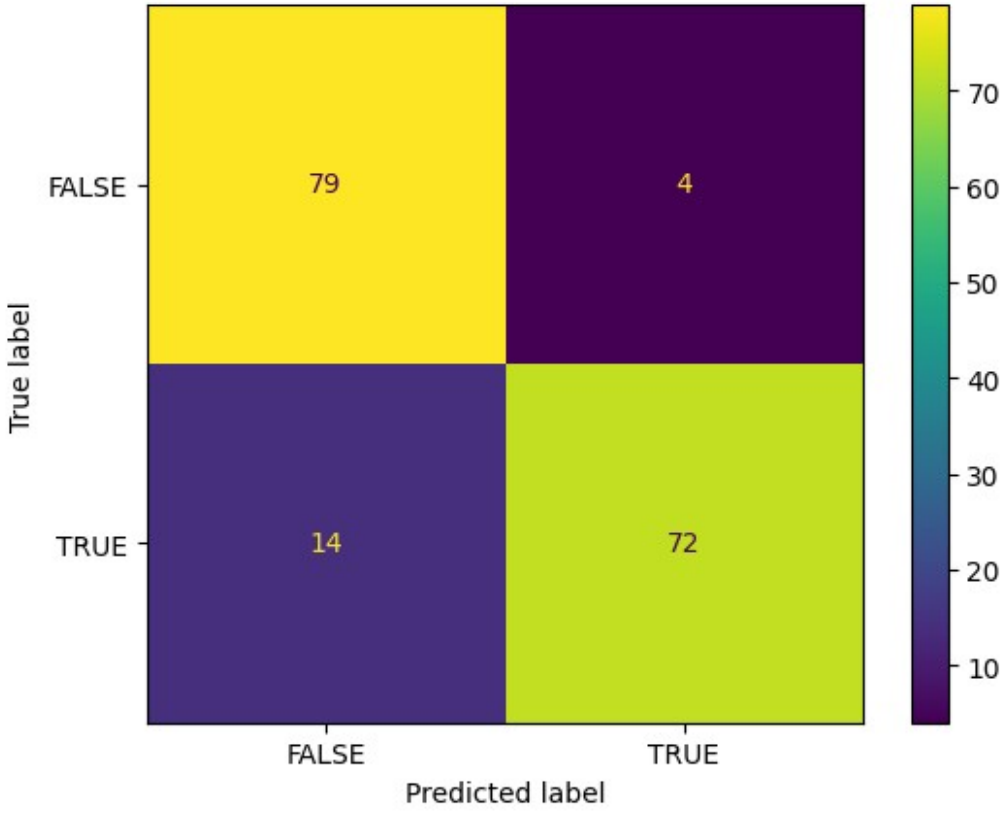

73
10
8
78
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

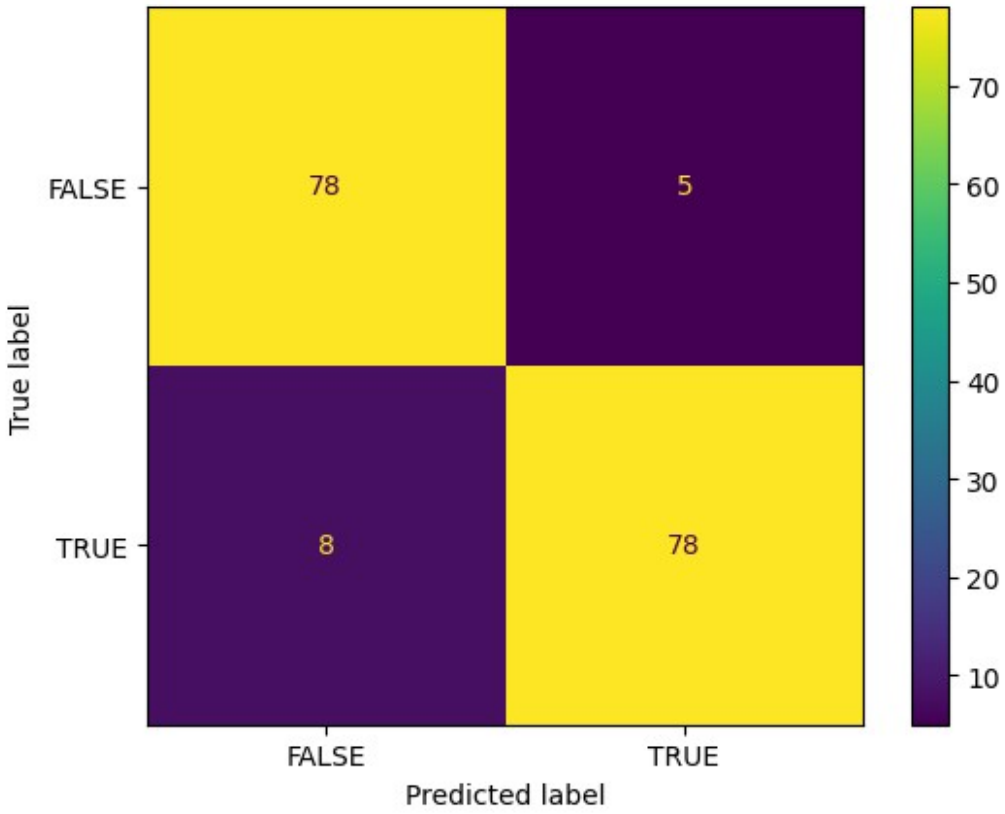

70
13
2
84
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10

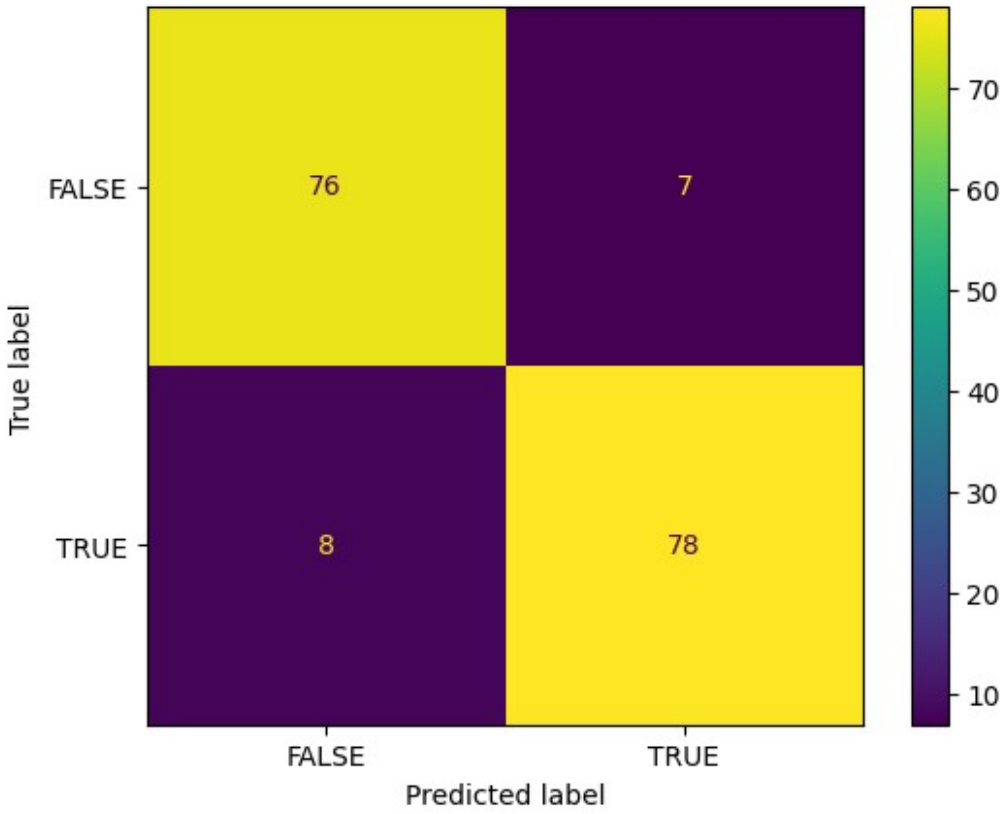

80
3
12
74
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10


74
9
8
78
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10


FALSE
TRUE
True label
78
5
8
78
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10


FALSE
TRUE
True label
79
4
14
72
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10


FALSE
TRUE
True label
78
5
8
78
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10


FALSE
TRUE
True label
76
7
8
78
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

Etape 3 : Classification selon la colonne TITRE et KEYWORDS (concaténés):

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.

```
# Utilisez la méthode ravel() pour transformer y_train en un tableau
unidimensionnel
y_train = np.ravel(y_train)

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))
]


#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))
    #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_title_keywords,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)

    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)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)

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', 0.8296532045654083, 0.044932550728791716),
('RF', 0.8132791922739244, 0.041321790598287424), ('CART',
0.7998902546093063, 0.04017857142092441), ('LogisticRegression',
0.7629280070237051, 0.034417449324786104), ('MultinomialNB',
```

```
0.7495171202809481, 0.03555476961902677), ('KNN', 0.6489244951712028,
0.0525423751584651)]
```

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_keywords_SVC = []
X_test_title_keywords_SVC = []

X_train_title_keywords_RandomForestClassifier = []
X_test_title_keywords_RandomForestClassifier = []


for name, pipeline in all_models :

X_train_title_keywords_SVC.append(pipeline.fit_transform(X_train_title
_keywords).toarray())

X_test_title_keywords_SVC.append(pipeline.transform(X_test_title_keywo
rds).toarray())

X_train_title_keywords_RandomForestClassifier.append(pipeline.fit_tran
sform(X_train_title_keywords).toarray())

X_test_title_keywords_RandomForestClassifier.append(pipeline.transform
(X_test_title_keywords).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_keywords = eval('X_train_title_keywords_' +
model_name)
    X_test_title_keywords = eval('X_test_title_keywords_' +
model_name)
    for i in range (len(X_train_title_keywords)):
      grid_search = GridSearchCV(model, params[model_name], n_jobs=-1,
verbose=1,scoring=score)
      print("grid search fait")
      grid_search.fit(X_train_title_keywords[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_keywords[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]))
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)

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

```
meilleur score 0.831

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

Accuracy : 0.899
Classification Report
              precision    recall  f1-score   support

       FALSE    0.83673   0.98795   0.90608        83
        TRUE    0.98592   0.81395   0.89172        86

    accuracy                        0.89941       169
   macro avg    0.91133   0.90095   0.89890       169
weighted avg    0.91265   0.89941   0.89877       169

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 0.831

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

Accuracy : 0.899
Classification Report
              precision    recall  f1-score   support

       FALSE    0.83673   0.98795   0.90608        83
        TRUE    0.98592   0.81395   0.89172        86

    accuracy                        0.89941       169
   macro avg    0.91133   0.90095   0.89890       169
weighted avg    0.91265   0.89941   0.89877       169

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 0.831

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

Accuracy : 0.893
Classification Report
              precision    recall  f1-score   support
```

```
       FALSE    0.83505   0.97590   0.90000        83
        TRUE    0.97222   0.81395   0.88608        86

    accuracy                        0.89349       169
   macro avg    0.90364   0.89493   0.89304       169
weighted avg    0.90485   0.89349   0.89291       169

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 0.831

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

Accuracy : 0.899
Classification Report
              precision    recall  f1-score   support

       FALSE    0.83673   0.98795   0.90608        83
        TRUE    0.98592   0.81395   0.89172        86

    accuracy                        0.89941       169
   macro avg    0.91133   0.90095   0.89890       169
weighted avg    0.91265   0.89941   0.89877       169

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 0.801

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

Accuracy : 0.834
Classification Report
              precision    recall  f1-score   support

       FALSE    0.83133   0.83133   0.83133        83
        TRUE    0.83721   0.83721   0.83721        86

    accuracy                        0.83432       169
   macro avg    0.83427   0.83427   0.83427       169
weighted avg    0.83432   0.83432   0.83432       169
```

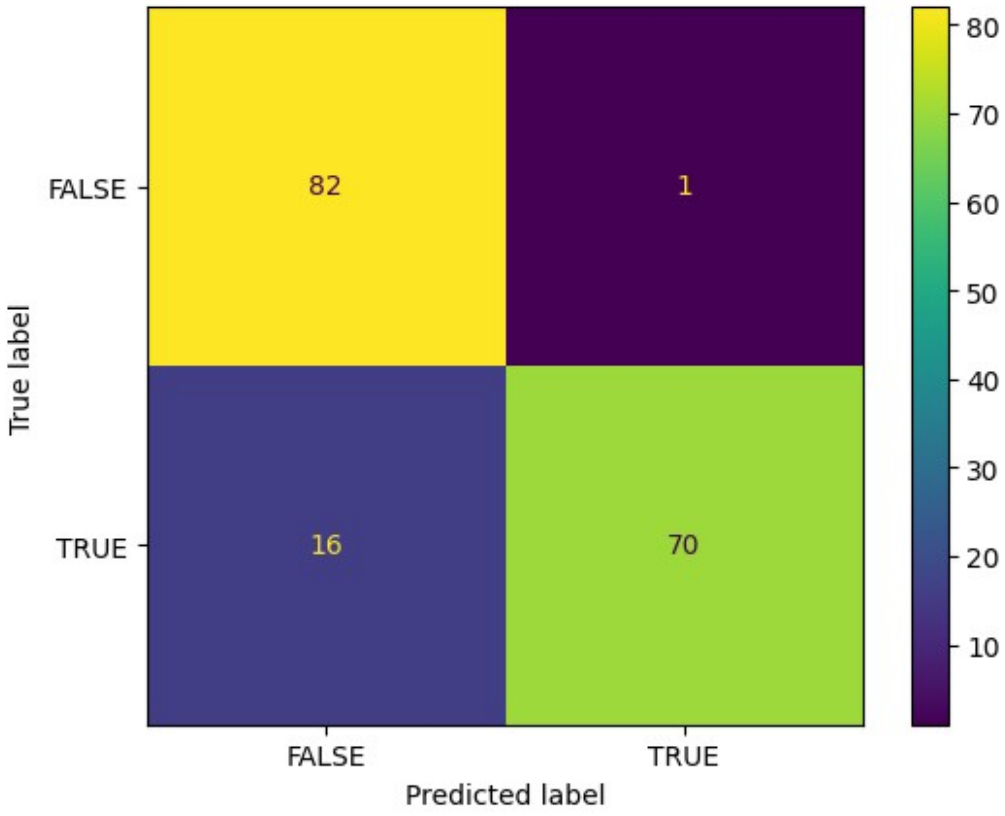
```
Ensemble des meilleurs paramètres :
     C: 1
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.790

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

Accuracy : 0.840
Classification Report
              precision    recall  f1-score   support

       FALSE    0.84146   0.83133   0.83636        83
        TRUE    0.83908   0.84884   0.84393        86

    accuracy                        0.84024       169
   macro avg    0.84027   0.84008   0.84015       169
weighted avg    0.84025   0.84024   0.84021       169

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 0.809

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

Accuracy : 0.893
Classification Report
              precision    recall  f1-score   support

       FALSE    0.90123   0.87952   0.89024        83
        TRUE    0.88636   0.90698   0.89655        86

    accuracy                        0.89349       169
   macro avg    0.89380   0.89325   0.89340       169
weighted avg    0.89367   0.89349   0.89345       169

Ensemble des meilleurs paramètres :
     C: 7
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

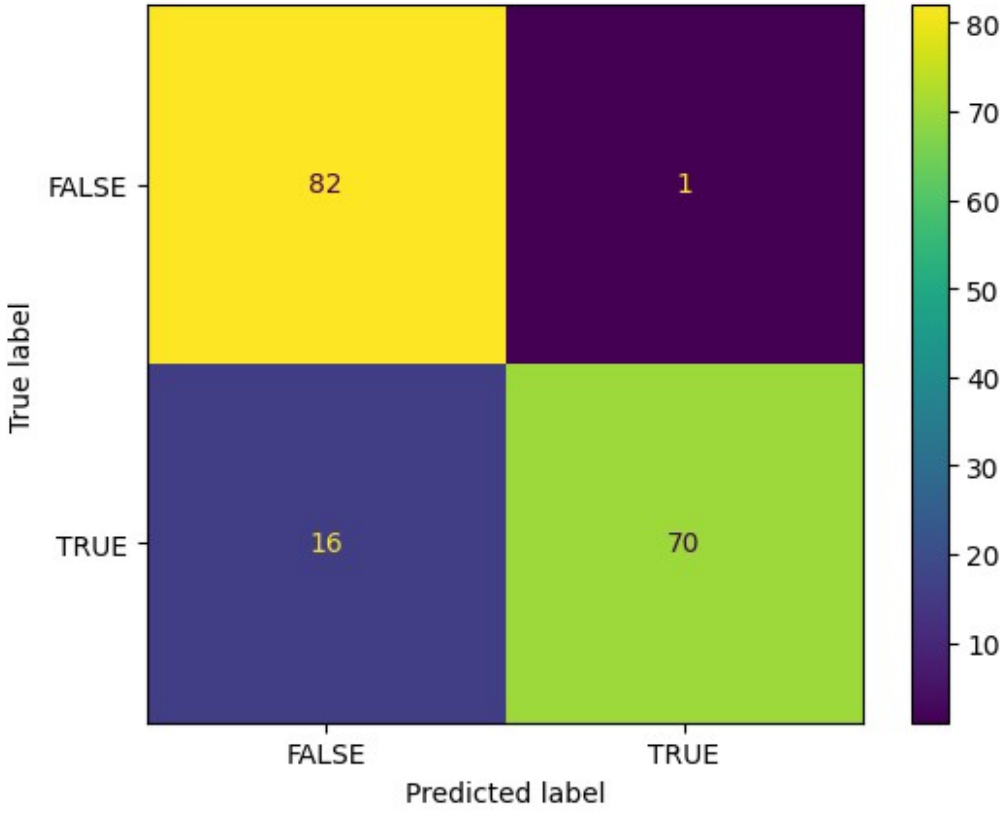
```
meilleur score 0.815

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

Accuracy : 0.882
Classification Report
              precision    recall  f1-score   support

       FALSE    0.89873   0.85542   0.87654        83
        TRUE    0.86667   0.90698   0.88636        86

    accuracy                        0.88166       169
   macro avg    0.88270   0.88120   0.88145       169
weighted avg    0.88242   0.88166   0.88154       169

Ensemble des meilleurs paramètres :
     C: 2
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.801

meilleur estimateur RandomForestClassifier(random_state=42)

Accuracy : 0.876
Classification Report
              precision    recall  f1-score   support

       FALSE    0.87805   0.86747   0.87273        83
        TRUE    0.87356   0.88372   0.87861        86

    accuracy                        0.87574       169
   macro avg    0.87581   0.87560   0.87567       169
weighted avg    0.87577   0.87574   0.87572       169

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 0.803

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

Accuracy : 0.876
Classification Report
              precision    recall  f1-score   support
```

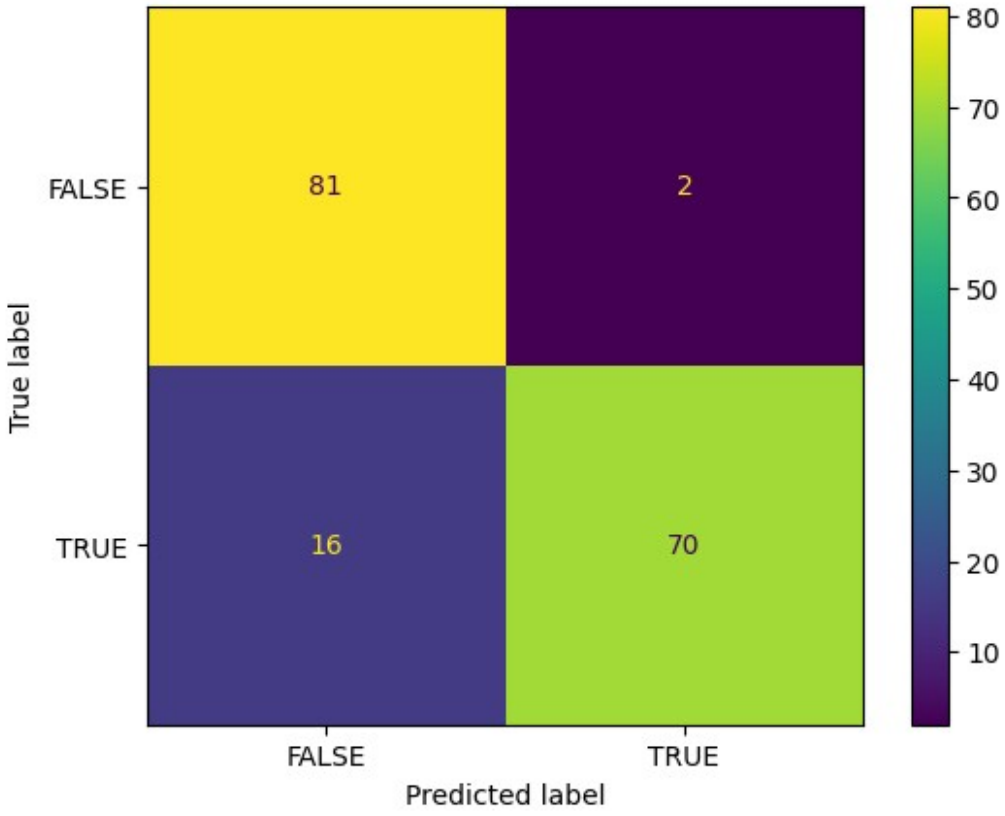
```
       FALSE    0.88750   0.85542   0.87117        83
        TRUE    0.86517   0.89535   0.88000        86

    accuracy                        0.87574       169
   macro avg    0.87633   0.87539   0.87558       169
weighted avg    0.87614   0.87574   0.87566       169

Ensemble des meilleurs paramètres :
     n_estimators: 10
     max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.804

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

Accuracy : 0.852
Classification Report
              precision    recall  f1-score   support

       FALSE    0.89189   0.79518   0.84076        83
        TRUE    0.82105   0.90698   0.86188        86

    accuracy                        0.85207       169
   macro avg    0.85647   0.85108   0.85132       169
weighted avg    0.85584   0.85207   0.85151       169

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 0.803

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

Accuracy : 0.858
Classification Report
              precision    recall  f1-score   support

       FALSE    0.91549   0.78313   0.84416        83
        TRUE    0.81633   0.93023   0.86957        86

    accuracy                        0.85799       169
   macro avg    0.86591   0.85668   0.85686       169
weighted avg    0.86503   0.85799   0.85709       169
```

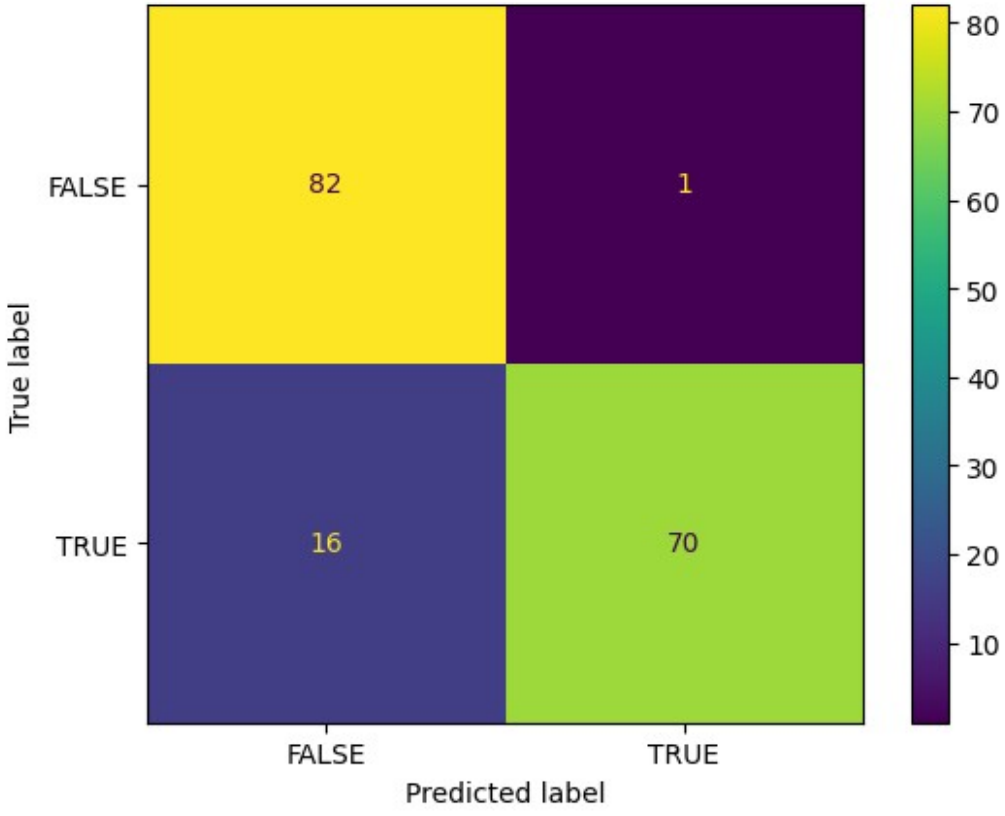
```
Ensemble des meilleurs paramètres :
     n_estimators: 50
     max_features: 'sqrt'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.824

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

Accuracy : 0.870
Classification Report
              precision    recall  f1-score   support

       FALSE    0.91781   0.80723   0.85897        83
        TRUE    0.83333   0.93023   0.87912        86

    accuracy                        0.86982       169
   macro avg    0.87557   0.86873   0.86905       169
weighted avg    0.87482   0.86982   0.86923       169

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 0.831

meilleur estimateur RandomForestClassifier(random_state=42)

Accuracy : 0.882
Classification Report
              precision    recall  f1-score   support

       FALSE    0.94366   0.80723   0.87013        83
        TRUE    0.83673   0.95349   0.89130        86

    accuracy                        0.88166       169
   macro avg    0.89020   0.88036   0.88072       169
weighted avg    0.88925   0.88166   0.88091       169

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 0.834
```

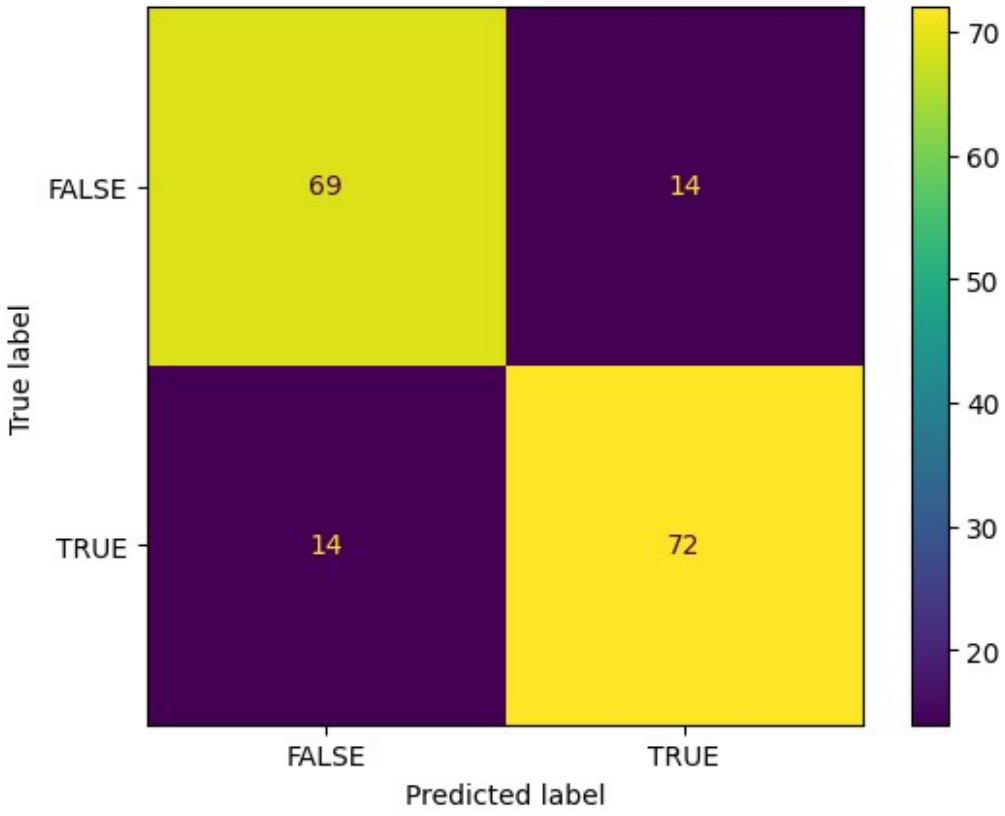
```
meilleur estimateur RandomForestClassifier(n_estimators=50,
random_state=42)

Accuracy : 0.870
Classification Report
              precision    recall  f1-score   support

       FALSE    0.94203   0.78313   0.85526        83
        TRUE    0.82000   0.95349   0.88172        86

    accuracy                        0.86982       169
   macro avg    0.88101   0.86831   0.86849       169
weighted avg    0.87993   0.86982   0.86873       169

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 0.834

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

Accuracy : 0.858
Classification Report
              precision    recall  f1-score   support

       FALSE    0.91549   0.78313   0.84416        83
        TRUE    0.81633   0.93023   0.86957        86

    accuracy                        0.85799       169
   macro avg    0.86591   0.85668   0.85686       169
weighted avg    0.86503   0.85799   0.85709       169

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

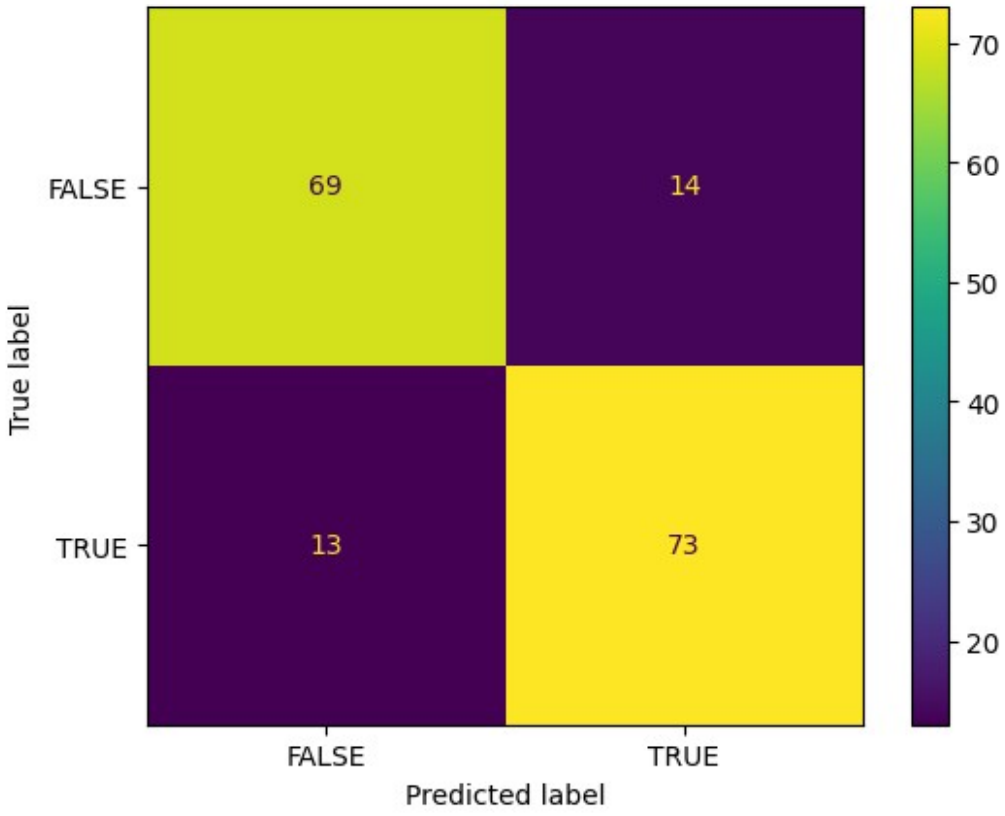

82
1
16
70
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10

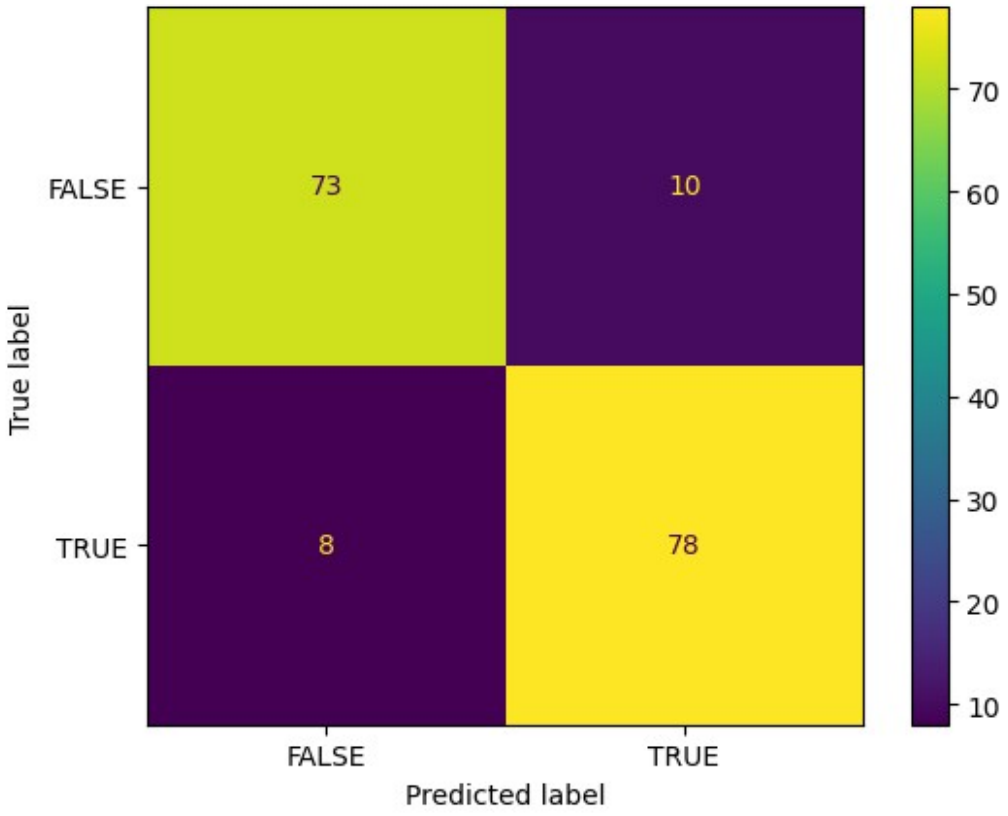

82
1
16
70
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10

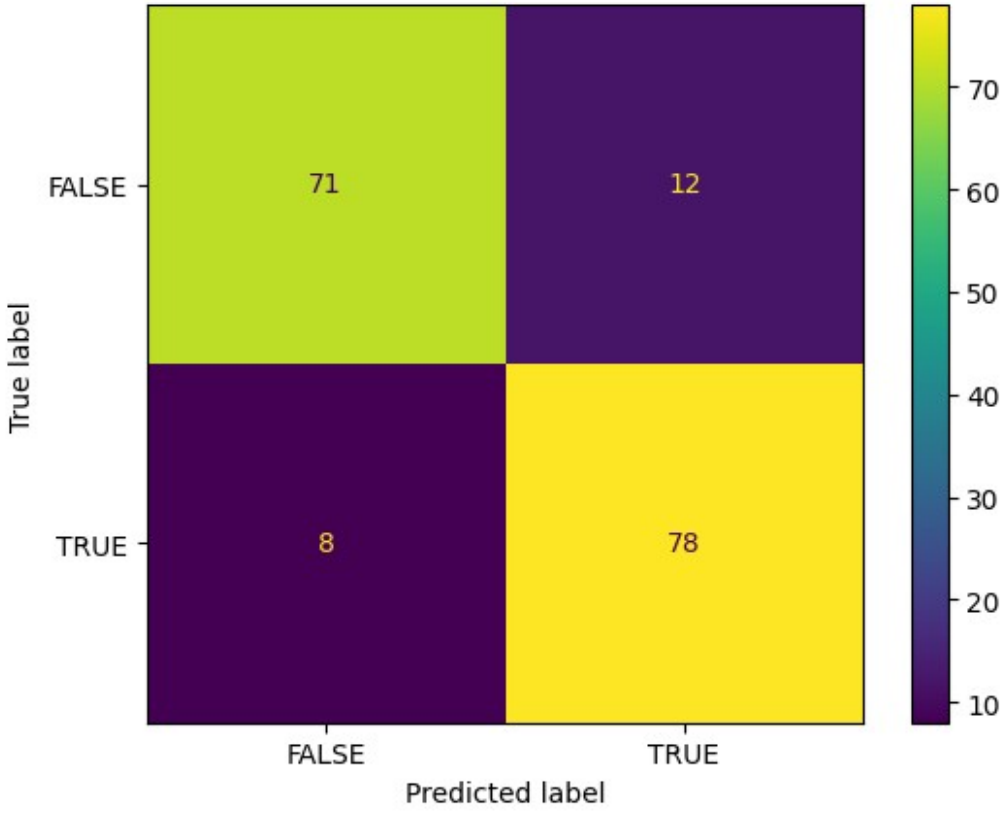

81
2
16
70
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10

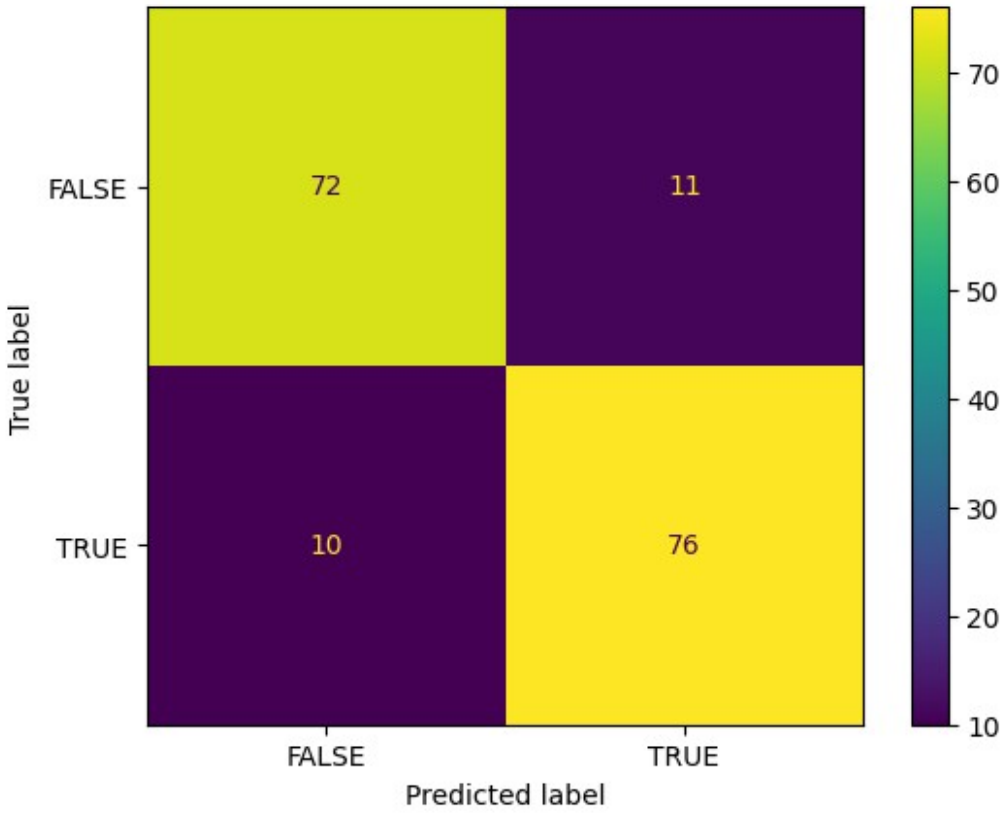

82
1
16
70
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10

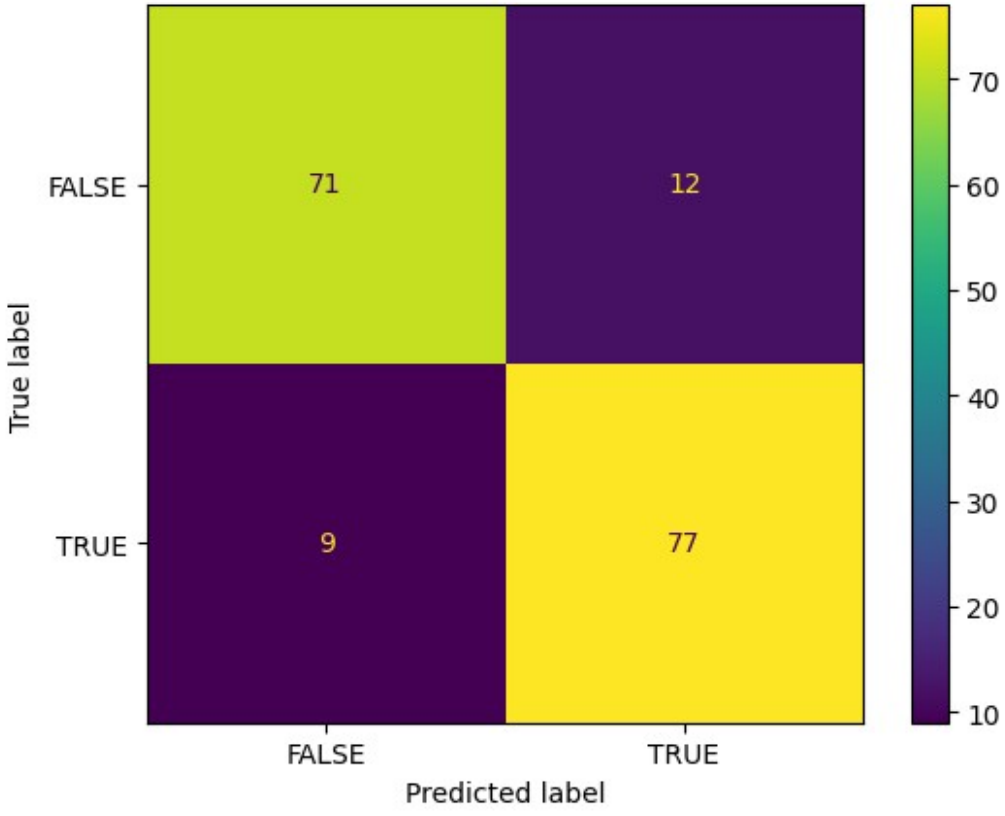

69
14
14
72
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20

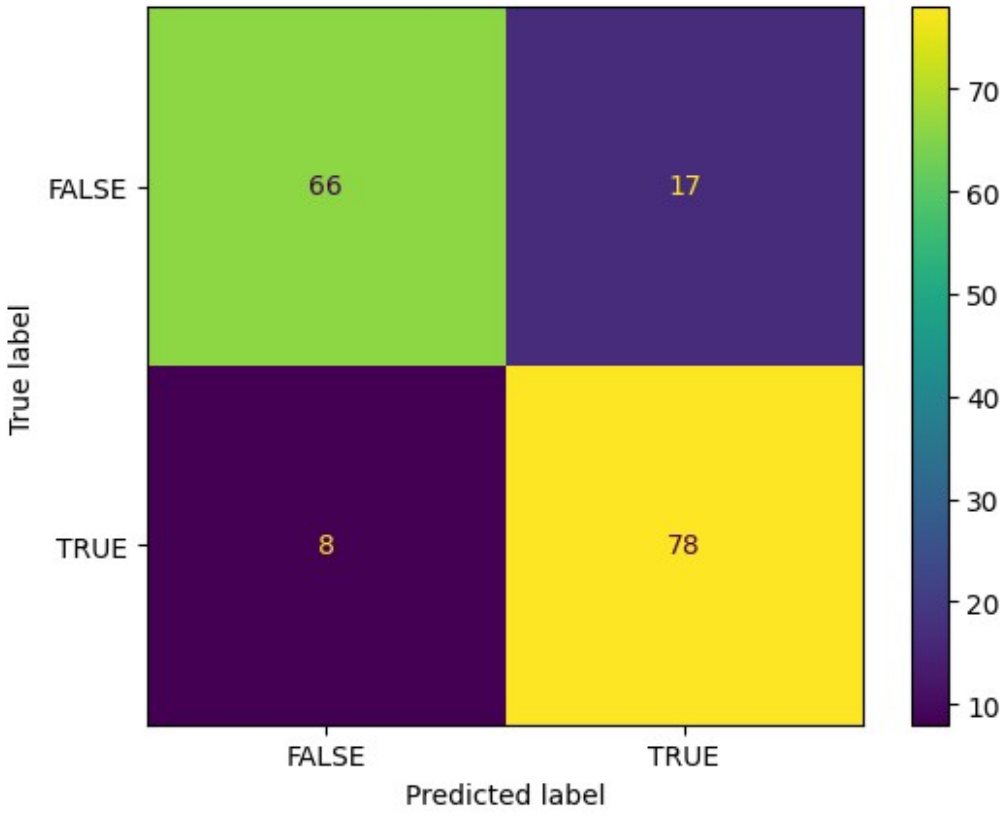

69
14
13
73
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20

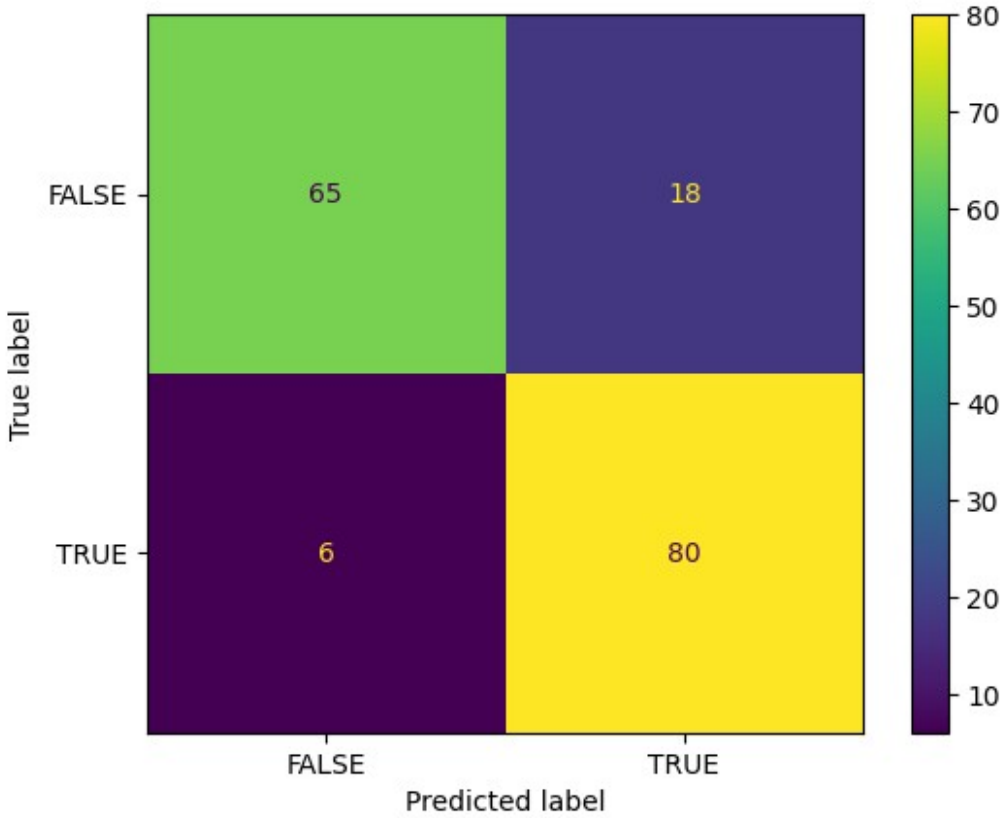

FALSE
TRUE
True label
73
10
8
78
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

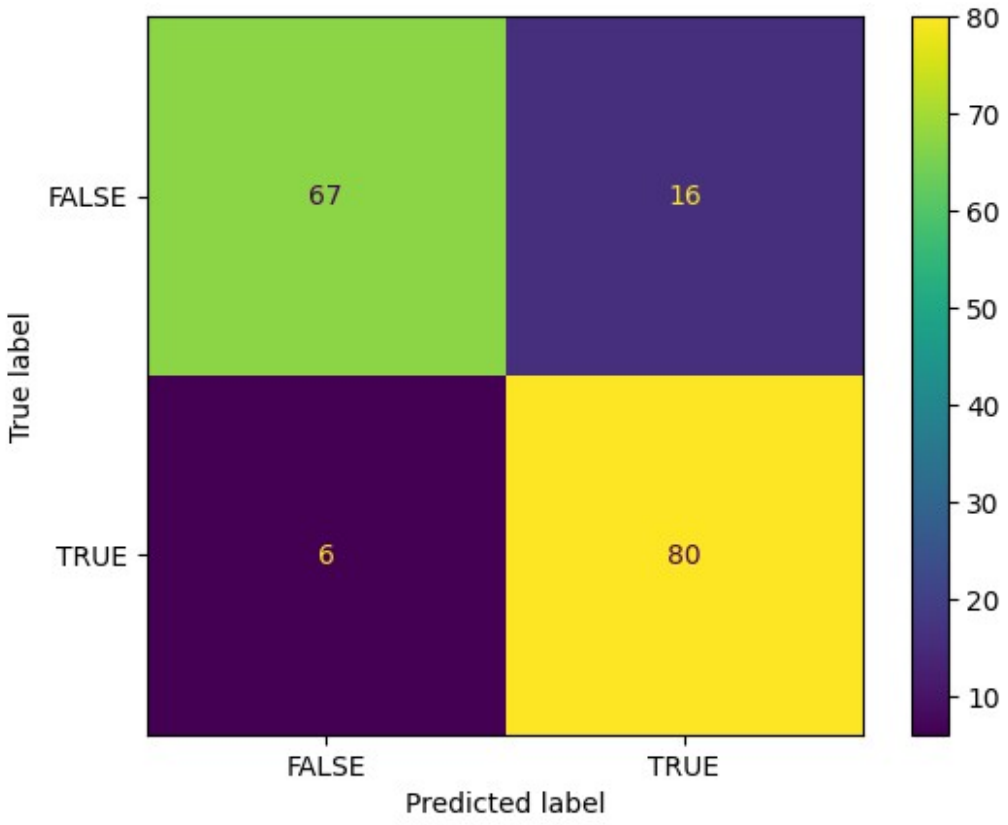

FALSE
TRUE
True label
71
12
8
78
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

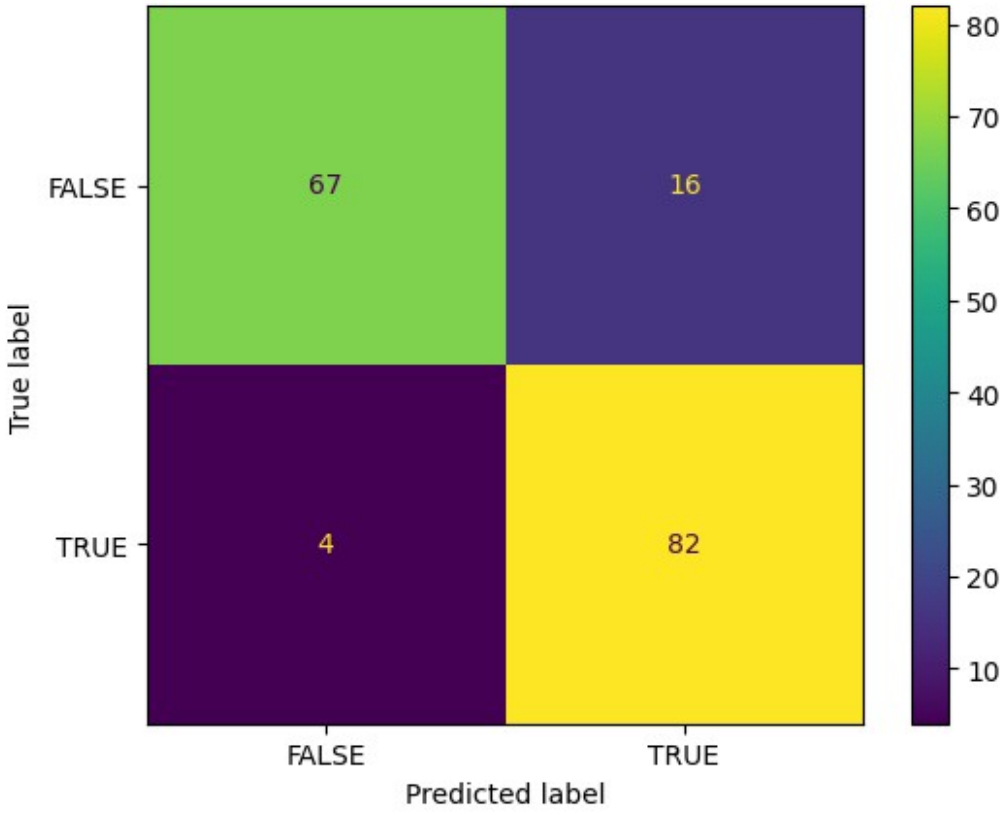

FALSE
TRUE
True label
72
11
10
76
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

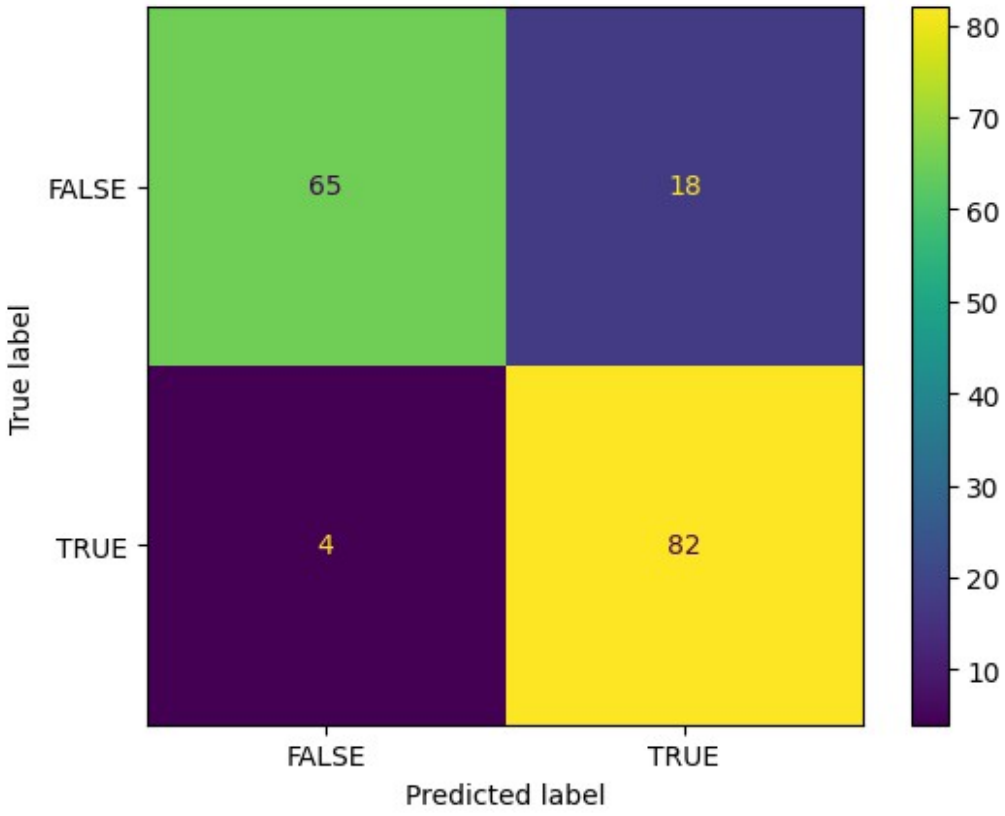

FALSE
TRUE
True label
71
12
9
77
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

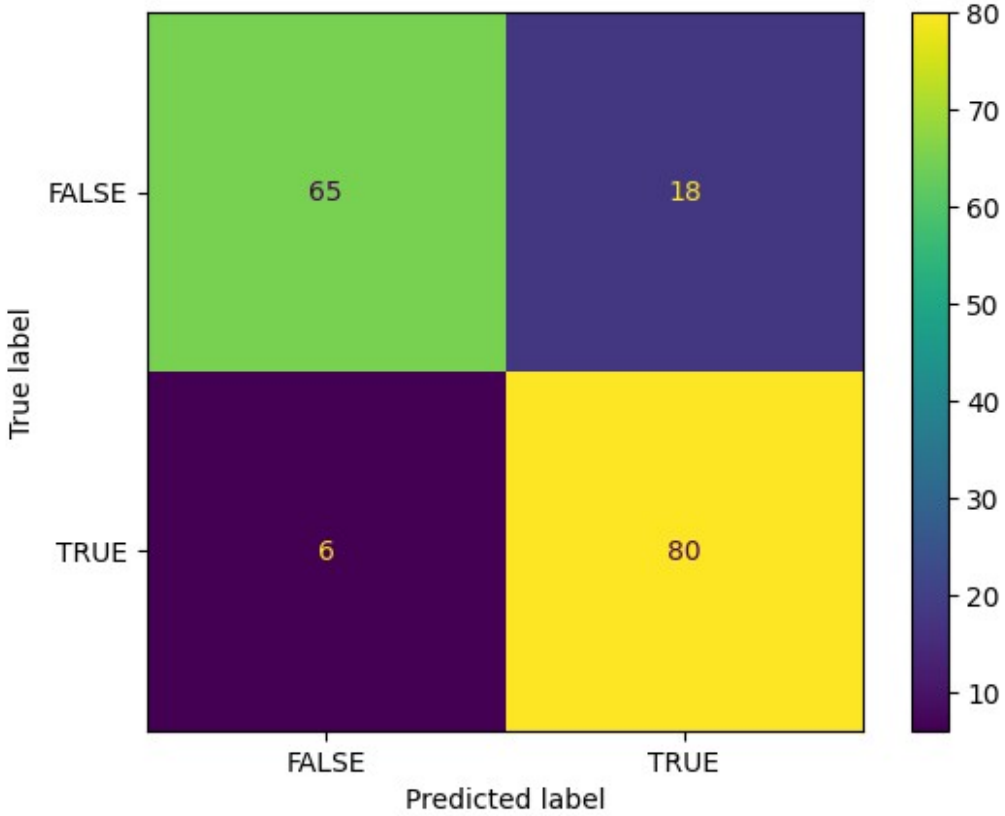

66
17
8
78
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10


65
18
6
80
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10


FALSE
TRUE
True label
67
16
6
80
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10


FALSE
TRUE
True label
67
16
4
82
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10


65
18
4
82
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10


65
18
6
80
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10

##**Etape 4 : Classification selon la colonne TEXT+TITRE et KEYWORDS (concaténés) :**

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.

```
# Utilisez la méthode ravel() pour transformer y_train en un tableau
unidimensionnel
y_train = np.ravel(y_train)

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))
]


#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))
    #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_title_keywords,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)

    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)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)

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', 0.8934372256365233, 0.04035996225541596),
('RF', 0.8829455662862159, 0.03710329845478108), ('CART',
0.8355794556628622, 0.04000678769199024), ('LogisticRegression',
```

```
0.8325943810359965, 0.03189829402977564), ('MultinomialNB',
0.8100965759438102, 0.0825013147911004), ('KNN', 0.685908691834943,
0.0685217034943254)]
```

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_keywords_SVC = []
X_test_text_title_keywords_SVC = []

X_train_text_title_keywords_RandomForestClassifier = []
X_test_text_title_keywords_RandomForestClassifier = []


for name, pipeline in all_models :

X_train_text_title_keywords_SVC.append(pipeline.fit_transform(X_train_
text_title_keywords).toarray())

X_test_text_title_keywords_SVC.append(pipeline.transform(X_test_text_t
itle_keywords).toarray())

X_train_text_title_keywords_RandomForestClassifier.append(pipeline.fit
_transform(X_train_text_title_keywords).toarray())

X_test_text_title_keywords_RandomForestClassifier.append(pipeline.tran
sform(X_test_text_title_keywords).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_keywords = eval('X_train_text_title_keywords_'
+ model_name)
    X_test_text_title_keywords = eval('X_test_text_title_keywords_' +
model_name)
    for i in range (len(X_train_text_title_keywords)):
      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_keywords[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_keywords[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]))
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

```
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.859

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

Accuracy : 0.870
Classification Report
              precision    recall  f1-score   support

       FALSE    0.91781   0.80723   0.85897        83
        TRUE    0.83333   0.93023   0.87912        86

    accuracy                        0.86982       169
   macro avg    0.87557   0.86873   0.86905       169
weighted avg    0.87482   0.86982   0.86923       169

Ensemble des meilleurs paramètres :
     C: 1.0
     gamma: 'scale'
     kernel: 'linear'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.843

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

Accuracy : 0.852
Classification Report
              precision    recall  f1-score   support

       FALSE    0.91429   0.77108   0.83660        83
        TRUE    0.80808   0.93023   0.86486        86

    accuracy                        0.85207       169
   macro avg    0.86118   0.85066   0.85073       169
weighted avg    0.86024   0.85207   0.85098       169

Ensemble des meilleurs paramètres :
     C: 1.0
     gamma: 'scale'
     kernel: 'linear'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.855

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

Accuracy : 0.888
```

```
Classification Report
              precision    recall  f1-score   support

       FALSE    0.91026   0.85542   0.88199        83
        TRUE    0.86813   0.91860   0.89266        86

    accuracy                        0.88757       169
   macro avg    0.88919   0.88701   0.88732       169
weighted avg    0.88882   0.88757   0.88742       169

Ensemble des meilleurs paramètres :
     C: 10
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.846

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

Accuracy : 0.882
Classification Report
              precision    recall  f1-score   support

       FALSE    0.88889   0.86747   0.87805        83
        TRUE    0.87500   0.89535   0.88506        86

    accuracy                        0.88166       169
   macro avg    0.88194   0.88141   0.88155       169
weighted avg    0.88182   0.88166   0.88162       169

Ensemble des meilleurs paramètres :
     C: 10
     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 : 0.882
Classification Report
              precision    recall  f1-score   support

       FALSE    0.87952   0.87952   0.87952        83
        TRUE    0.88372   0.88372   0.88372        86

    accuracy                        0.88166       169
```

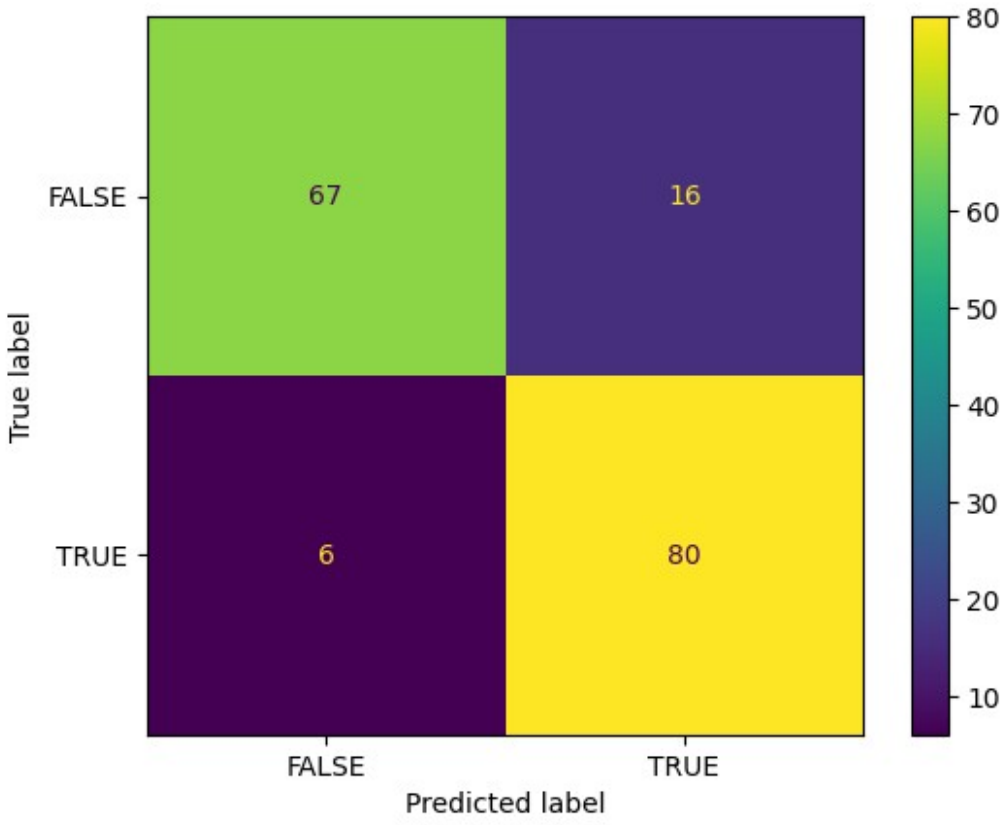
```
   macro avg    0.88162   0.88162   0.88162       169
weighted avg    0.88166   0.88166   0.88166       169

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 0.884

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

Accuracy : 0.893
Classification Report
              precision    recall  f1-score   support

       FALSE    0.86517   0.92771   0.89535        83
        TRUE    0.92500   0.86047   0.89157        86

    accuracy                        0.89349       169
   macro avg    0.89508   0.89409   0.89346       169
weighted avg    0.89562   0.89349   0.89342       169

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 0.886

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

Accuracy : 0.899
Classification Report
              precision    recall  f1-score   support

       FALSE    0.88372   0.91566   0.89941        83
        TRUE    0.91566   0.88372   0.89941        86

    accuracy                        0.89941       169
   macro avg    0.89969   0.89969   0.89941       169
weighted avg    0.89998   0.89941   0.89941       169

Ensemble des meilleurs paramètres :
     C: 2
     gamma: 'scale'
     kernel: 'rbf'
```

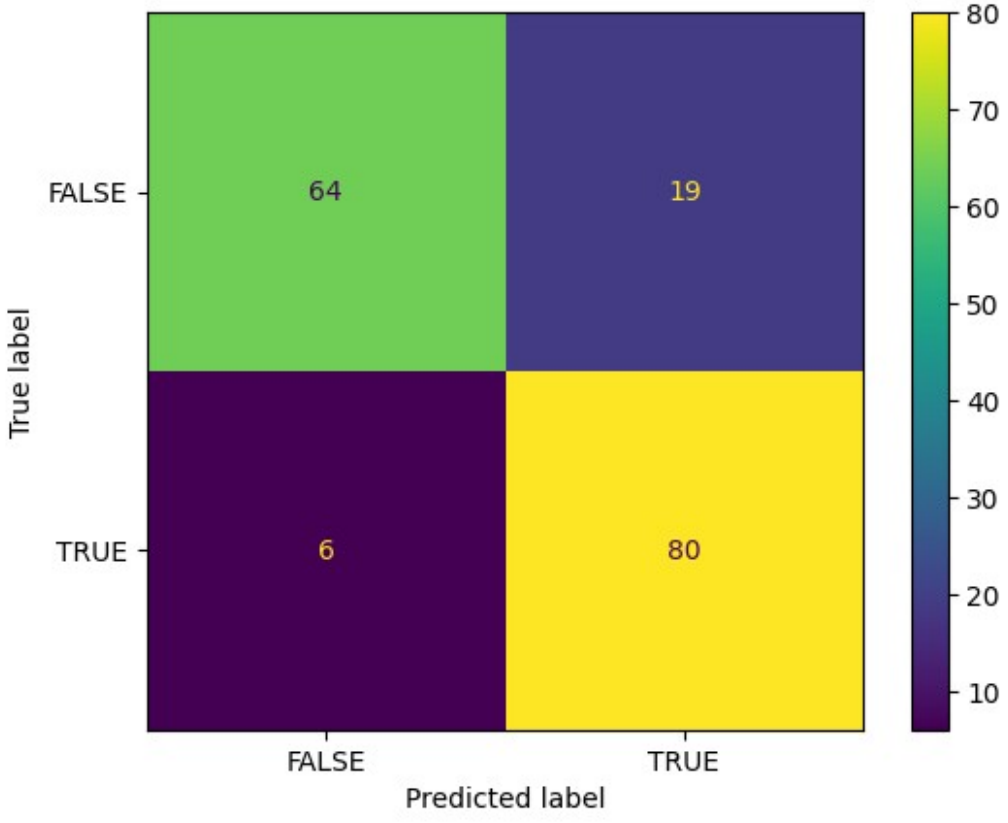
```
grid search fait
Fitting 5 folds for each of 18 candidates, totalling 90 fits
meilleur score 0.879

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

Accuracy : 0.882
Classification Report
              precision    recall  f1-score   support

       FALSE    0.87952   0.87952   0.87952        83
        TRUE    0.88372   0.88372   0.88372        86

    accuracy                        0.88166       169
   macro avg    0.88162   0.88162   0.88162       169
weighted avg    0.88166   0.88166   0.88166       169

Ensemble des meilleurs paramètres :
     C: 5
     gamma: 'scale'
     kernel: 'rbf'
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.858

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

Accuracy : 0.864
Classification Report
              precision    recall  f1-score   support

       FALSE    0.84091   0.89157   0.86550        83
        TRUE    0.88889   0.83721   0.86228        86

    accuracy                        0.86391       169
   macro avg    0.86490   0.86439   0.86389       169
weighted avg    0.86532   0.86391   0.86386       169

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 0.862

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

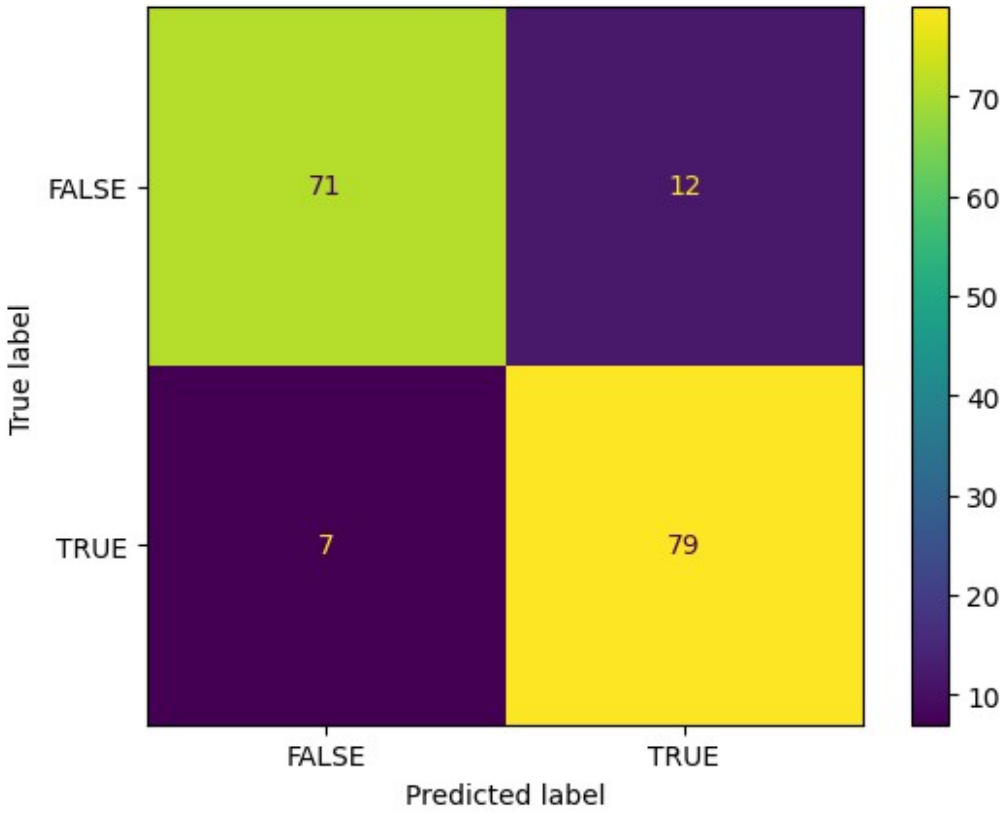
```
Accuracy : 0.858
Classification Report
              precision    recall  f1-score   support

       FALSE    0.85542   0.85542   0.85542        83
        TRUE    0.86047   0.86047   0.86047        86

    accuracy                        0.85799       169
   macro avg    0.85794   0.85794   0.85794       169
weighted avg    0.85799   0.85799   0.85799       169

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 0.856

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

Accuracy : 0.876
Classification Report
              precision    recall  f1-score   support

       FALSE    0.89744   0.84337   0.86957        83
        TRUE    0.85714   0.90698   0.88136        86

    accuracy                        0.87574       169
   macro avg    0.87729   0.87518   0.87546       169
weighted avg    0.87693   0.87574   0.87557       169

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 0.849

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

Accuracy : 0.882
Classification Report
              precision    recall  f1-score   support

       FALSE    0.84615   0.92771   0.88506        83
        TRUE    0.92308   0.83721   0.87805        86
```

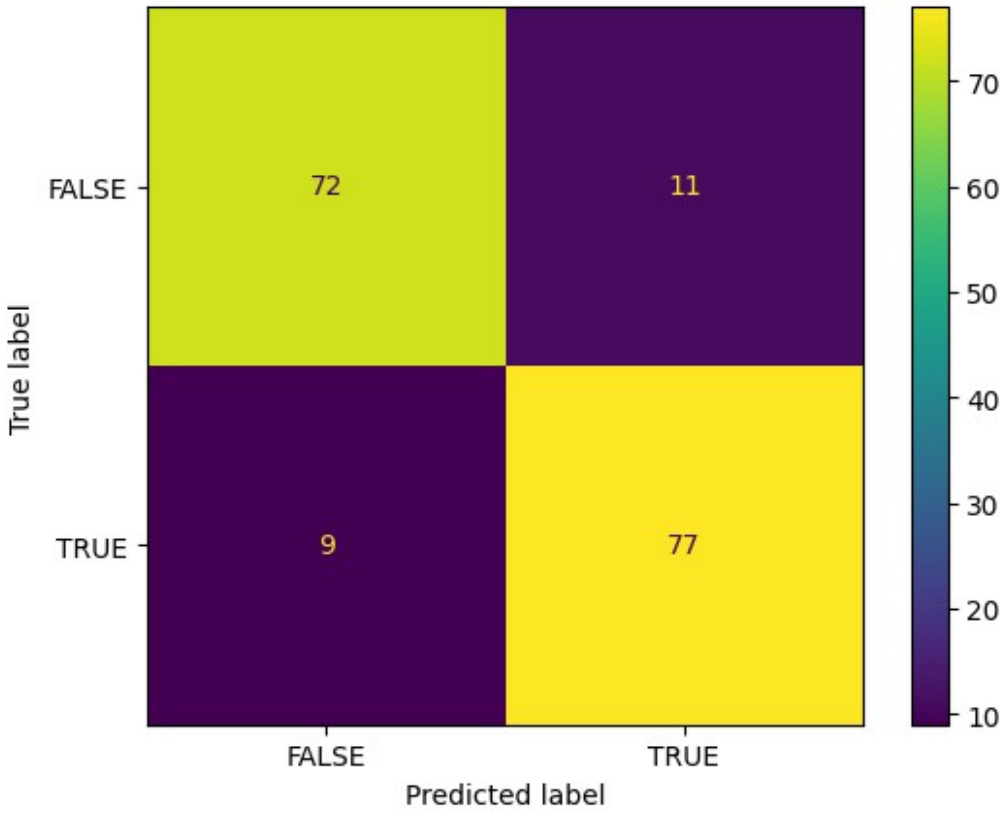
```
    accuracy                           0.88166       169
   macro avg    0.88462   0.88246   0.88155       169
weighted avg    0.88530   0.88166   0.88149       169

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 0.861

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

Accuracy : 0.888
Classification Report
              precision    recall  f1-score   support

       FALSE    0.84783   0.93976   0.89143        83
        TRUE    0.93506   0.83721   0.88344        86

    accuracy                           0.88757       169
   macro avg    0.89145   0.88848   0.88743       169
weighted avg    0.89222   0.88757   0.88736       169

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 0.862

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

Accuracy : 0.888
Classification Report
              precision    recall  f1-score   support

       FALSE    0.88095   0.89157   0.88623        83
        TRUE    0.89412   0.88372   0.88889        86

    accuracy                           0.88757       169
   macro avg    0.88754   0.88764   0.88756       169
weighted avg    0.88765   0.88757   0.88758       169

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

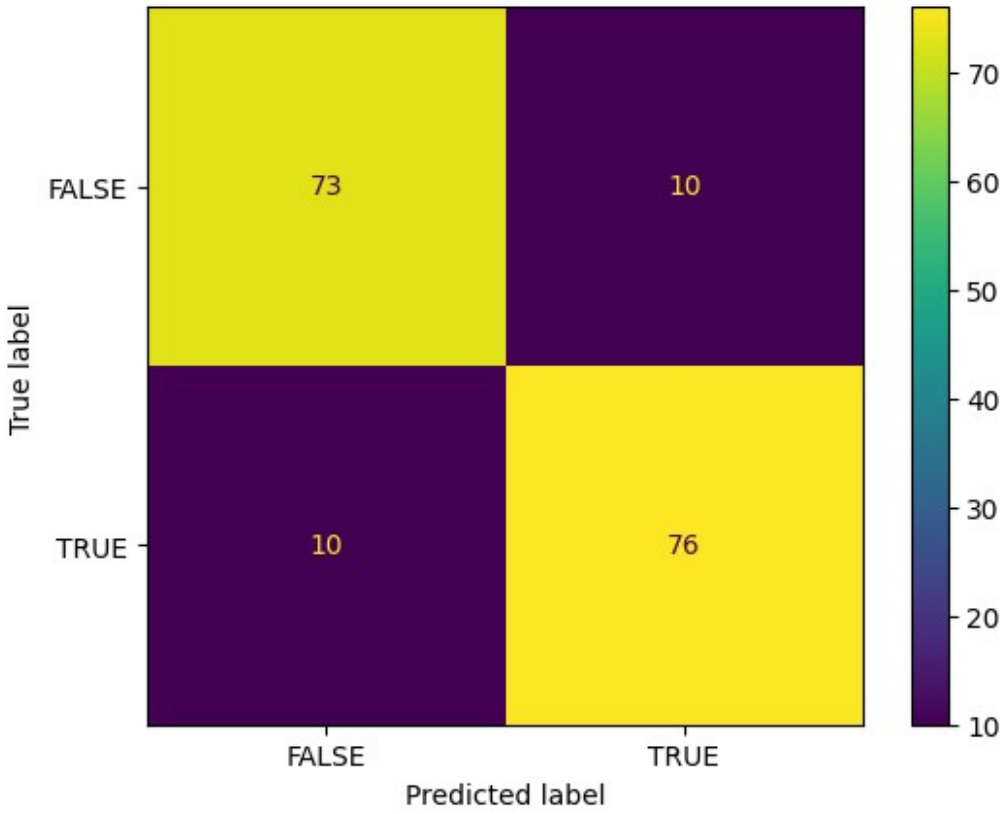
```
grid search fait
Fitting 5 folds for each of 8 candidates, totalling 40 fits
meilleur score 0.879

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

Accuracy : 0.917
Classification Report
              precision    recall  f1-score   support

       FALSE    0.88764   0.95181   0.91860        83
        TRUE    0.95000   0.88372   0.91566        86

    accuracy                        0.91716       169
   macro avg    0.91882   0.91776   0.91713       169
weighted avg    0.91937   0.91716   0.91711       169

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 0.865

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

Accuracy : 0.893
Classification Report
              precision    recall  f1-score   support

       FALSE    0.90123   0.87952   0.89024        83
        TRUE    0.88636   0.90698   0.89655        86

    accuracy                        0.89349       169
   macro avg    0.89380   0.89325   0.89340       169
weighted avg    0.89367   0.89349   0.89345       169

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

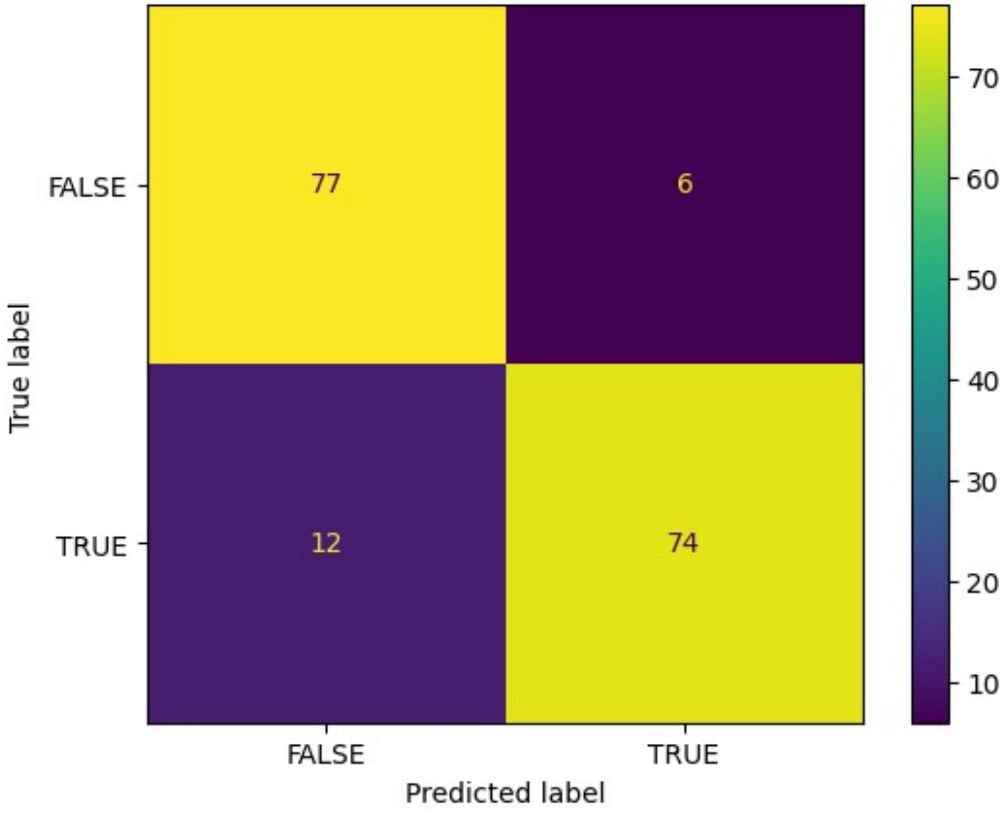

FALSE
TRUE
True label
67
16
6
80
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10

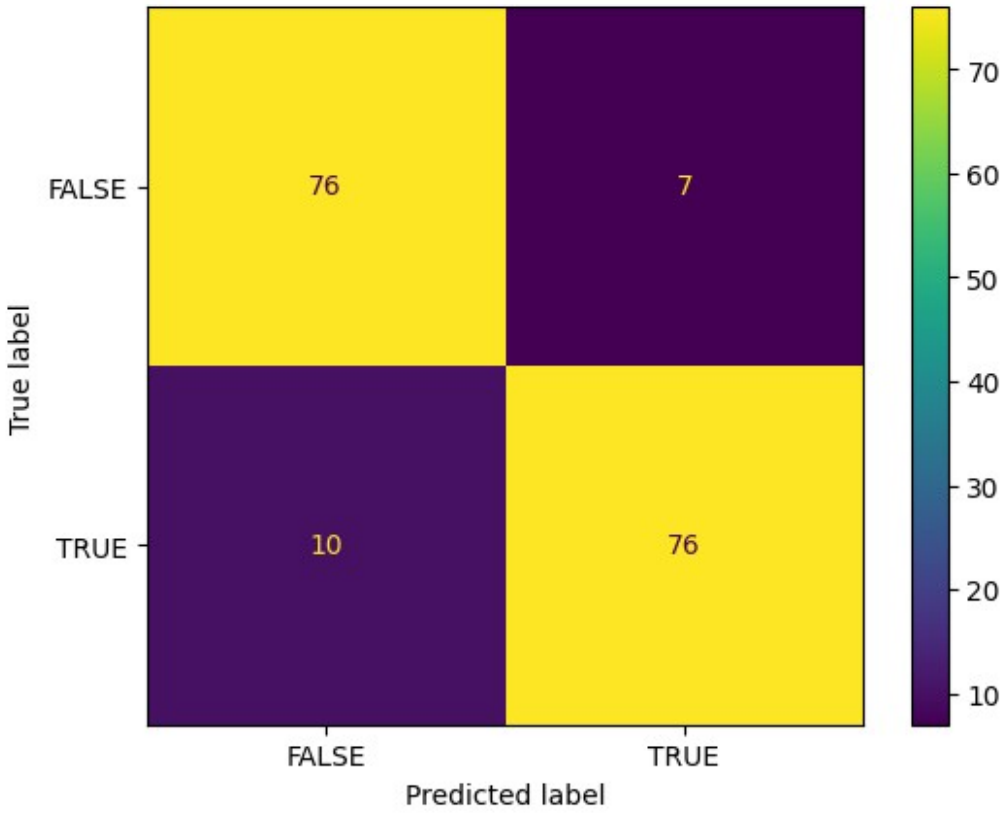

FALSE
TRUE
True label
64
19
6
80
FALSE
TRUE
Predicted label
80
70
60
50
40
30
20
10

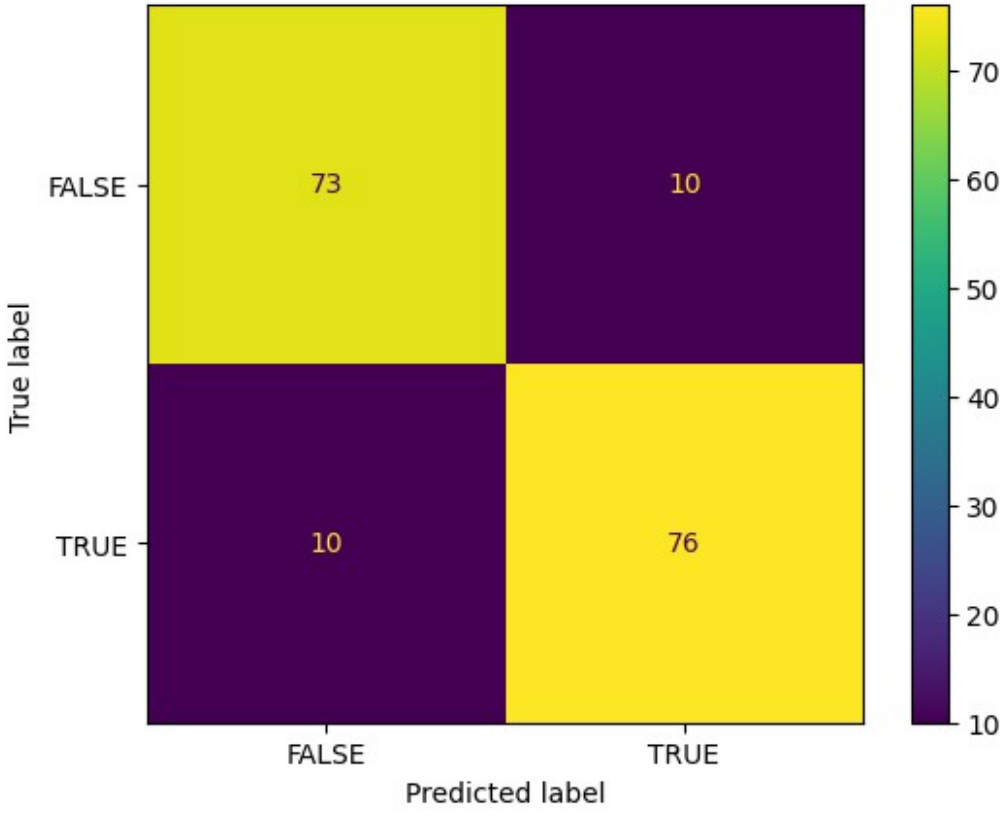

FALSE
TRUE
True label
71
12
7
79
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

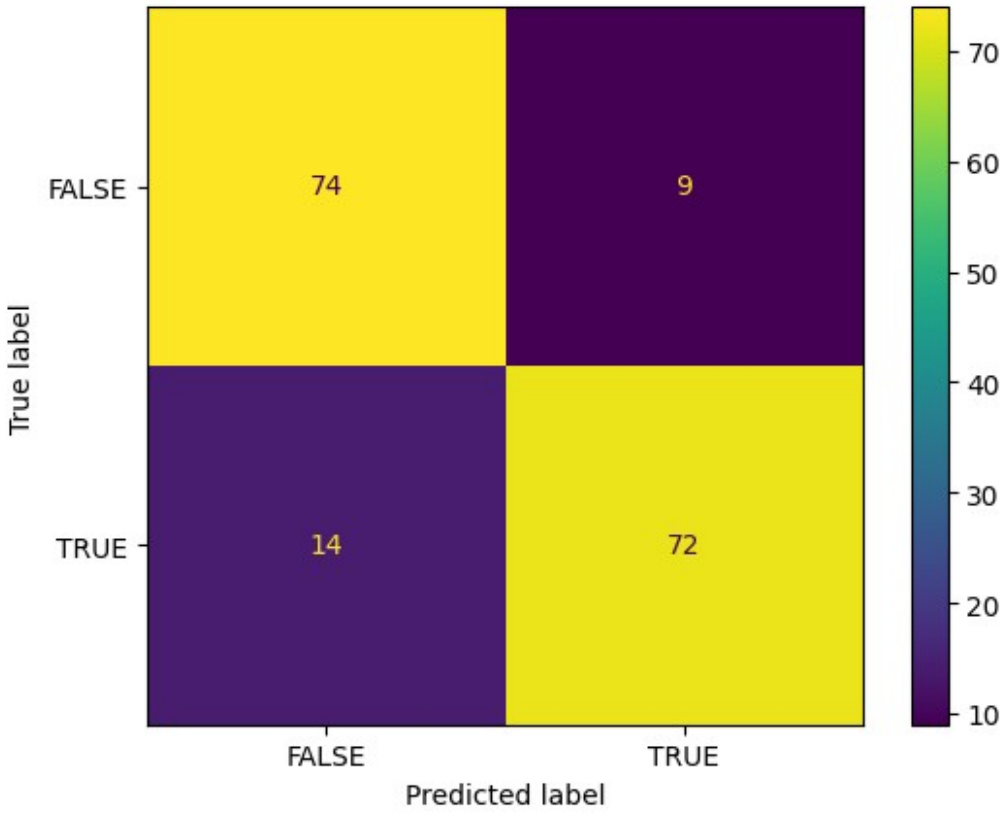

FALSE
TRUE
True label
72
11
9
77
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

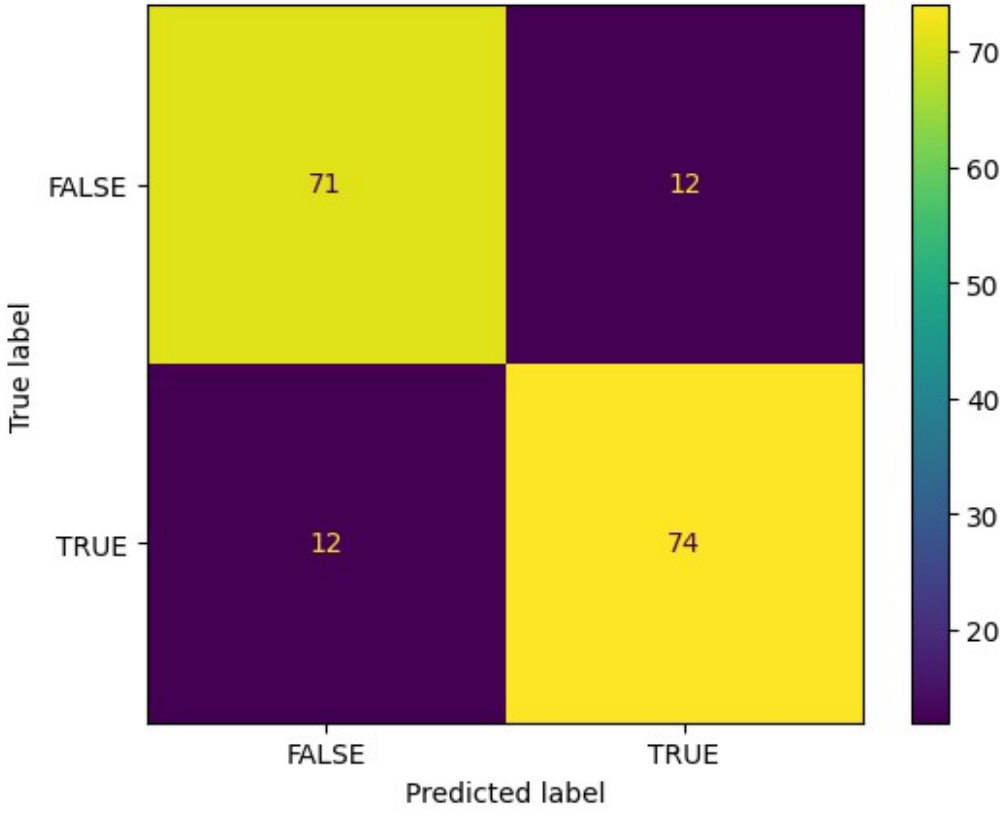

73
10
10
76
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

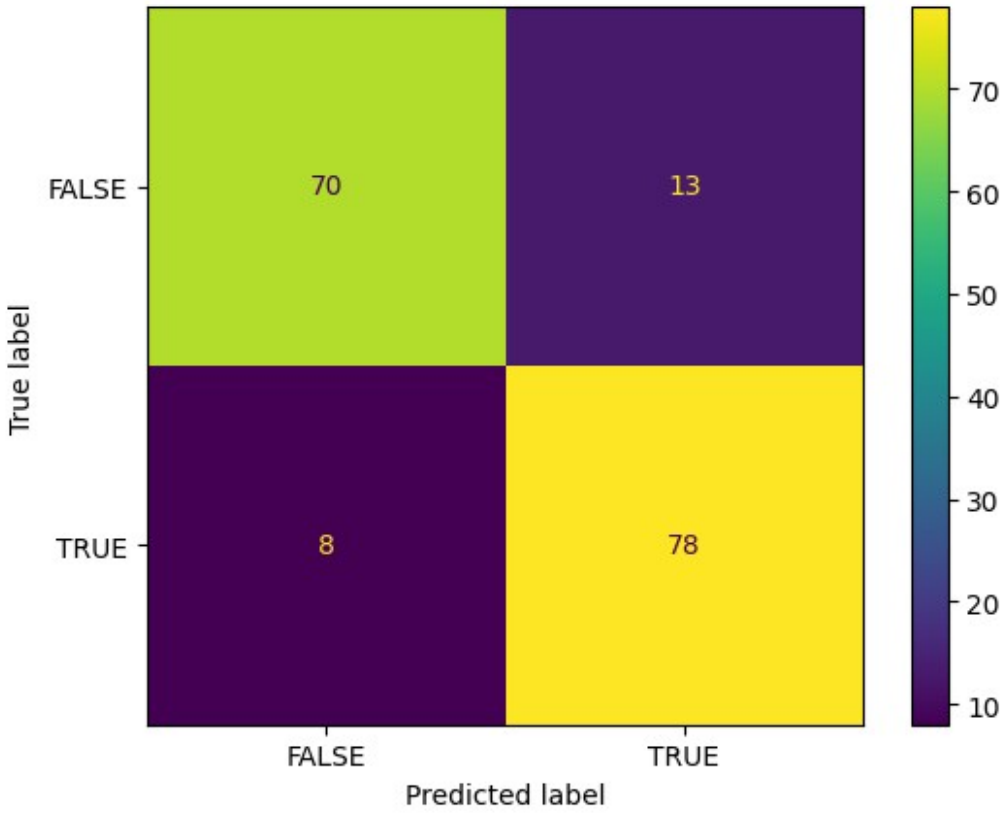

77
6
12
74
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

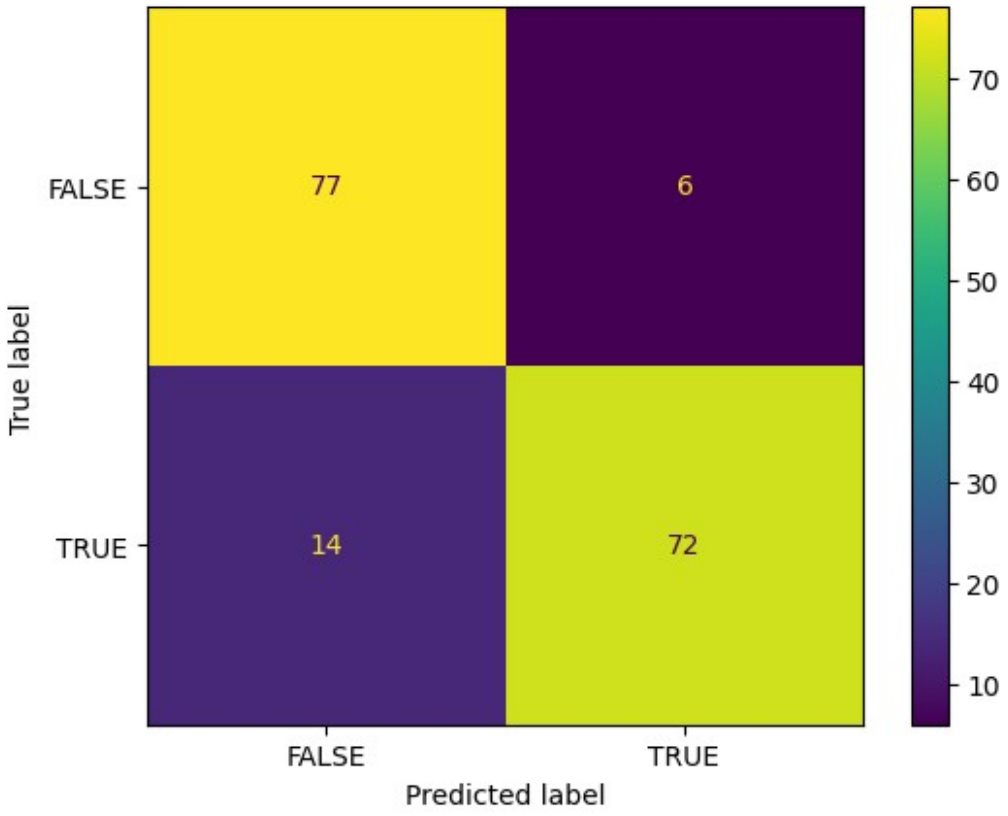

FALSE
TRUE
True label
76
7
10
76
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

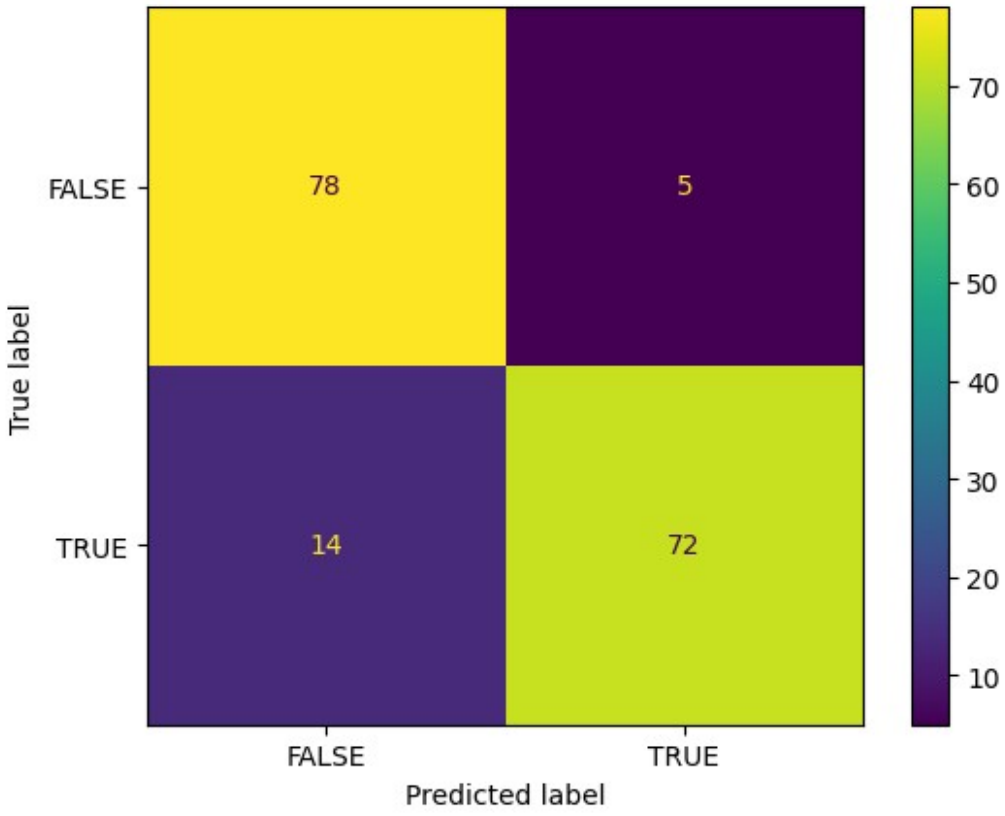

FALSE
TRUE
True label
73
10
10
76
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

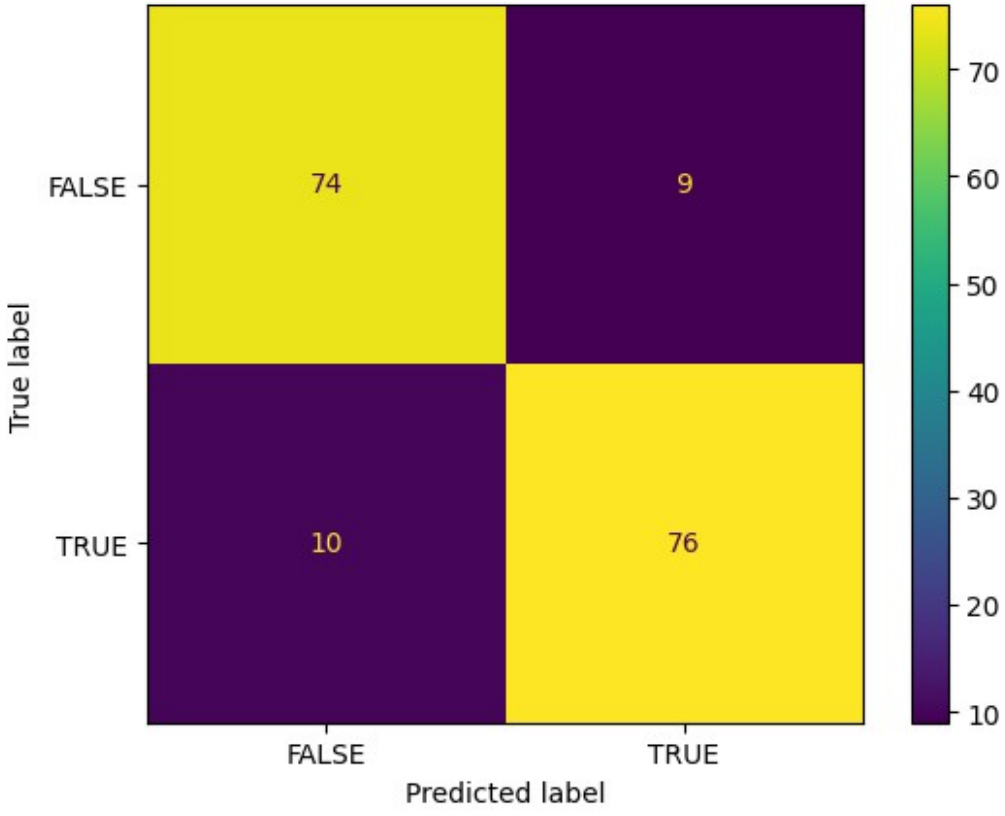

FALSE
TRUE
True label
74
9
14
72
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10

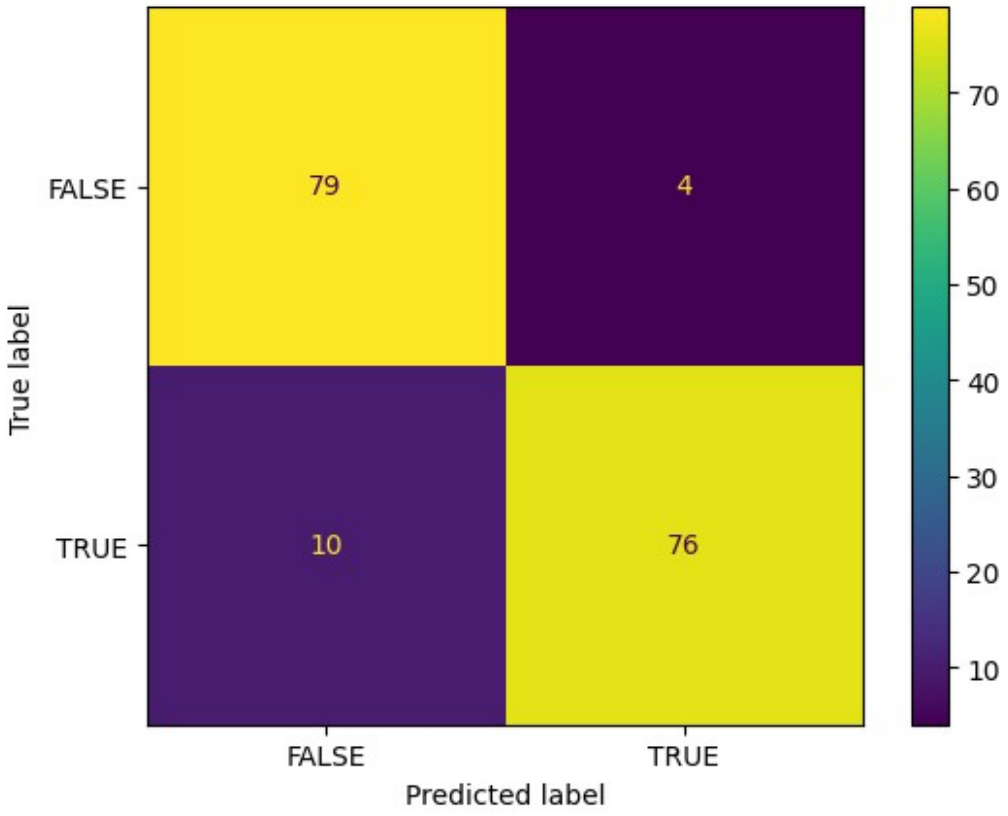

FALSE
TRUE
True label
71
12
12
74
FALSE
TRUE
Predicted label
70
60
50
40
30
20

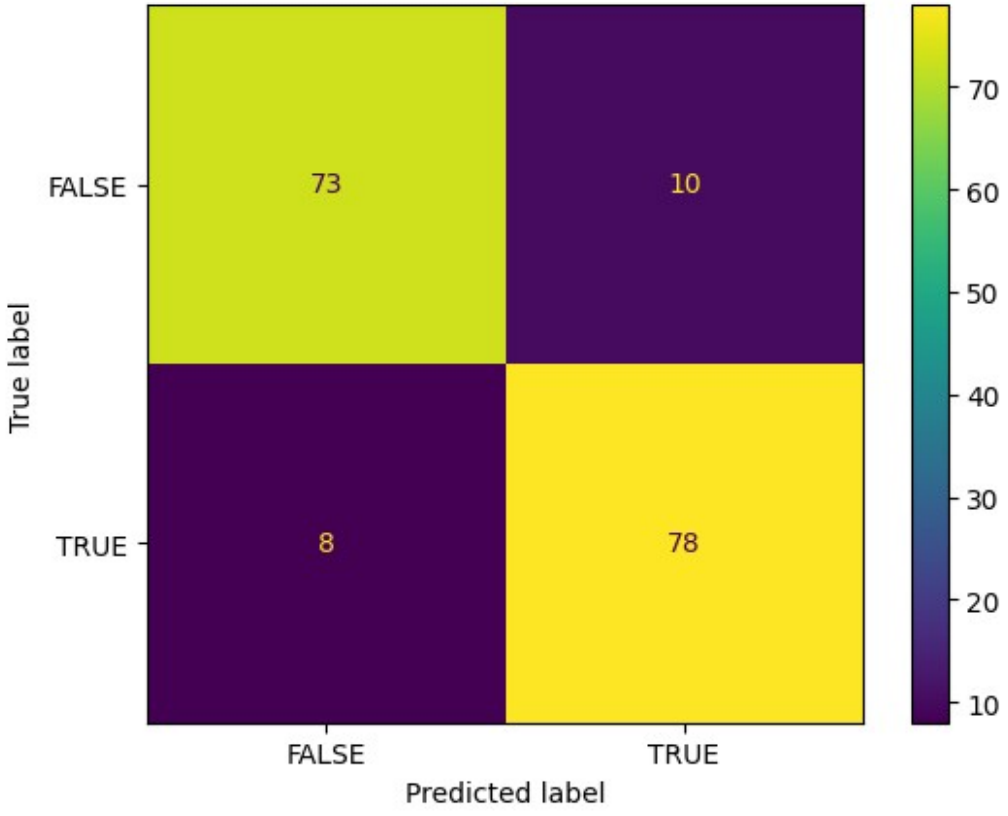

70
13
8
78
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10


77
6
14
72
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10


78
5
14
72
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10


74
9
10
76
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10


79
4
10
76
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10


73
10
8
78
FALSE
TRUE
True label
FALSE
TRUE
Predicted label
70
60
50
40
30
20
10