ss0n71dqf

May 2, 2025

Projet Machine Learning

HAI817I - 2024/2025

Classification d'assertions venant d'X (Twitter) selon leur rapport à la science

Groupe 8

Duban Mathis: 22102226

Gonzalez Oropeza Gilles : 21602817

Bernardon Vincent : 22009116

LONGLADE Mickaël : 22105047

Sujet

Ce projet s'inscrit dans le contexte de l'apprentissage supervisé, i.e. les données possèdent des labels. Il vise à trouver les modèles les plus performants pour prédire si des assertions (une assertion est une proposition que l'on avance et que l'on soutient comme vraie) faites par des hommes politiques (par exemple) sont vraies ou fausses.

0.1 Installation

Dans cette partie, nous allons installer toutes les librairies dont nous allons avoir besoin pour notre projet.

[ ]:
```
# Installation des librairies pour le projet
!pip install pandas numpy scipy gensim emoji nltk matplotlib seaborn␣
↪scikit-learn inflect googletrans==4.0.0-rc1 contractions pyspellchecker␣
↪optuna

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

# Librairies de manipulation de données (graphique, lecture de données,ect...)
import pandas as pd # Lecture de données
import numpy as np # Array
import seaborn as sns #
import matplotlib.pyplot as plt # Graphique
```

```python
import sys

# Librairies pour la fonction prepareText
import re # Regular expression
import nltk
import json
from nltk.corpus import stopwords #English stopwords
nltk.download('stopwords') # Téléchargement des stopwords (une seule fois)
from nltk.corpus import wordnet #Mots pour vérifier les suppressions de lettres␣
↪répétées
nltk.download('wordnet') # Téléchargement de mots existants
import emoji
import inflect # Transformation des chiffres en mots
import re
from googletrans import Translator # Traduction de langues
from nltk.stem import WordNetLemmatizer # Lemmatisation des mots
from nltk.stem import PorterStemmer # Racinanisation des mots
nltk.download('punkt') #Tagetisation des mots
nltk.download('punkt_tab') # Tokenisation des mots
nltk.download('averaged_perceptron_tagger')
nltk.download('averaged_perceptron_tagger_eng')
from nltk import pos_tag # Tagination des mots
from nltk.tokenize import word_tokenize
import unicodedata # Suppresion d'accent
import contractions # Transformation des contractions
from collections import Counter
from collections import defaultdict

#Libraries pour l'entraînement du modèle
from sklearn.feature_extraction.text import TfidfVectorizer # Vectorisation
from sklearn.preprocessing import MaxAbsScaler, StandardScaler # Vectorisation
from spellchecker import SpellChecker # dictionnaire phonétique
from sklearn.feature_extraction.text import CountVectorizer # Topic modelling
from sklearn.decomposition import LatentDirichletAllocation #LDA
from sklearn import preprocessing # Upsampling
from imblearn.over_sampling import SMOTE # Upsampling
from imblearn.combine import SMOTETomek
from imblearn.over_sampling import RandomOverSampler  # if resampleData doesn't␣
↪balance
import sklearn
from sklearn.preprocessing import LabelEncoder #Label encoder

from gensim.models.coherencemodel import CoherenceModel #Coherence model
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score # matrices de confusion
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b',␣
↪'#e377c2', '#7f7f7f', '#bcbd22', '#17becf'] # couleur pour les graphiques
```

```python
from sklearn.utils import resample # Upsampling
from wordcloud import WordCloud # Nuage de mot
from gensim.corpora.dictionary import Dictionary # évaluation de cohérenec
from gensim.models import LdaModel # LDA
import os

from sklearn.model_selection import train_test_split, GridSearchCV, KFold,␣
↪cross_val_score, cross_val_predict, RandomizedSearchCV
import optuna
from scipy import stats

# Classifiers
from sklearn.tree import DecisionTreeClassifier # Decision TREE classifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages
(2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(1.13.1)
Requirement already satisfied: gensim in /usr/local/lib/python3.11/dist-packages
(4.3.3)
Requirement already satisfied: emoji in /usr/local/lib/python3.11/dist-packages
(2.14.1)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages
(3.9.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-
packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-
packages (0.13.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-
packages (1.6.1)
Requirement already satisfied: inflect in /usr/local/lib/python3.11/dist-
packages (7.5.0)
Requirement already satisfied: googletrans==4.0.0-rc1 in
/usr/local/lib/python3.11/dist-packages (4.0.0rc1)
Requirement already satisfied: contractions in /usr/local/lib/python3.11/dist-
packages (0.1.73)
Requirement already satisfied: pyspellchecker in /usr/local/lib/python3.11/dist-
packages (0.8.2)
Requirement already satisfied: optuna in /usr/local/lib/python3.11/dist-packages
(4.3.0)
Requirement already satisfied: httpx==0.13.3 in /usr/local/lib/python3.11/dist-
packages (from googletrans==4.0.0-rc1) (0.13.3)
```

```
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (2025.4.26)
Requirement already satisfied: hstspreload in /usr/local/lib/python3.11/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (2025.1.1)
Requirement already satisfied: sniffio in /usr/local/lib/python3.11/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (1.3.1)
Requirement already satisfied: chardet==3.* in /usr/local/lib/python3.11/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (3.0.4)
Requirement already satisfied: idna==2.* in /usr/local/lib/python3.11/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (2.10)
Requirement already satisfied: rfc3986<2,>=1.3 in
/usr/local/lib/python3.11/dist-packages (from
httpx==0.13.3->googletrans==4.0.0-rc1) (1.5.0)
Requirement already satisfied: httpcore==0.9.* in
/usr/local/lib/python3.11/dist-packages (from
httpx==0.13.3->googletrans==4.0.0-rc1) (0.9.1)
Requirement already satisfied: h11<0.10,>=0.8 in /usr/local/lib/python3.11/dist-
packages (from httpcore==0.9.*->httpx==0.13.3->googletrans==4.0.0-rc1) (0.9.0)
Requirement already satisfied: h2==3.* in /usr/local/lib/python3.11/dist-
packages (from httpcore==0.9.*->httpx==0.13.3->googletrans==4.0.0-rc1) (3.2.0)
Requirement already satisfied: hyperframe<6,>=5.2.0 in
/usr/local/lib/python3.11/dist-packages (from
h2==3.*->httpcore==0.9.*->httpx==0.13.3->googletrans==4.0.0-rc1) (5.2.0)
Requirement already satisfied: hpack<4,>=3.0 in /usr/local/lib/python3.11/dist-
packages (from h2==3.*->httpcore==0.9.*->httpx==0.13.3->googletrans==4.0.0-rc1)
(3.0.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas) (2025.2)
Requirement already satisfied: smart-open>=1.8.1 in
/usr/local/lib/python3.11/dist-packages (from gensim) (7.1.0)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages
(from nltk) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages
(from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages
(from nltk) (4.67.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (4.57.0)
```

```
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-
packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: more_itertools>=8.5.0 in
/usr/local/lib/python3.11/dist-packages (from inflect) (10.7.0)
Requirement already satisfied: typeguard>=4.0.1 in
/usr/local/lib/python3.11/dist-packages (from inflect) (4.4.2)
Requirement already satisfied: textsearch>=0.0.21 in
/usr/local/lib/python3.11/dist-packages (from contractions) (0.0.24)
Requirement already satisfied: alembic>=1.5.0 in /usr/local/lib/python3.11/dist-
packages (from optuna) (1.15.2)
Requirement already satisfied: colorlog in /usr/local/lib/python3.11/dist-
packages (from optuna) (6.9.0)
Requirement already satisfied: sqlalchemy>=1.4.2 in
/usr/local/lib/python3.11/dist-packages (from optuna) (2.0.40)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.11/dist-packages
(from optuna) (6.0.2)
Requirement already satisfied: Mako in /usr/lib/python3/dist-packages (from
alembic>=1.5.0->optuna) (1.1.3)
Requirement already satisfied: typing-extensions>=4.12 in
/usr/local/lib/python3.11/dist-packages (from alembic>=1.5.0->optuna) (4.13.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages
(from smart-open>=1.8.1->gensim) (1.17.2)
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.11/dist-
packages (from sqlalchemy>=1.4.2->optuna) (3.2.1)
Requirement already satisfied: anyascii in /usr/local/lib/python3.11/dist-
packages (from textsearch>=0.0.21->contractions) (0.3.2)
Requirement already satisfied: pyahocorasick in /usr/local/lib/python3.11/dist-
packages (from textsearch>=0.0.21->contractions) (2.1.0)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data…
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
```

```
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
```

Importation du répertoire de travail sur Google Drive

[ ]:
```
from google.colab import drive
drive.mount('/content/gdrive/')
path='/content/gdrive/My Drive/M1_IMAGINE_ML/ML_Project/'
sys.path.append(path)
%cd $path
%pwd
```

```
Mounted at /content/gdrive/
/content/gdrive/My Drive/M1_IMAGINE_ML/ML_Project
```

[ ]:
```
'/content/gdrive/My Drive/M1_IMAGINE_ML/ML_Project'
```

Récupération des données du dataSet présent sur le répertoire Google Drive en ligne

[ ]:
```
# Importation des données
df=pd.read_csv('dataSet/data.csv', sep='\t')
# Lecture des 5 premières lignes pour confirmer la bonne récupération des␣
↪données
display (df.head())
```

```
   Unnamed: 0            tweet_id  \
0           0  316669998137483264
1           1  319090866545385472
2           2  322030931022065664
3           3  322694830620807168
4           4  328524426658328576

                                                text  science_related  \
0  Knees are a bit sore. i guess that's a sign th…                0
1          McDonald's breakfast stop then the gym                  0
2  Can any Gynecologist with Cancer Experience ex…                1
3  Couch-lock highs lead to sleeping in the couch…                1
4  Does daily routine help prevent problems with …                1

   scientific_claim  scientific_reference  scientific_context
0               0.0                   0.0                 0.0
1               0.0                   0.0                 0.0
2               1.0                   0.0                 0.0
3               1.0                   0.0                 0.0
```

```
4               1.0                     0.0                   0.0
```

Récupération de nos dictionnaires :

```
[ ]: # Récupération des monnaies
currency_file=pd.read_csv('dataSet/currency_translation.csv')
display (currency_file.head())
# Création d'un objet dictionnaire pour répertorier les monnaies
currency_dict = dict(zip(currency_file['Currency Symbol'],␣
 ↪currency_file['Currency Name']))
# Récupération des abbréviations (Slang)
slang_file=pd.read_csv('dataSet/slang_translation.csv')
display (slang_file.head())
# Création d'un objet dictionnaire pour répertorier les abbréviations
slang_dict = dict(zip(slang_file['Abbreviation'], slang_file['Meaning']))
```

```
  Currency Symbol Currency Name
0               $        dollar
1               €          euro
2               £         pound
3               ¥           yen
4                  indian rupee
```

```
  Abbreviation                  Meaning
0          4ao           for adults only
1          a.m             before midday
2           a3  anytime anywhere anyplace
3        aamof        as a matter of fact
4         acct                   account
```

0.2 Ingénierie des données

Dans cette partie, on s'interésse au pré-traitement des données. Afin que chaque élément de notre base soit utilisable et pertinent on va nettoyer, normaliser et transformer nos données afin qu'elles soient préparées et optimales pour nos analyses.

On va ici répertorier tous les éléments qui doivent être traités :

Élément	Exemple	Traitement à effectuer	Exemple après traitement
Emojis		Suppression des émojis ou remplacement par leur signification	basketball
Mention Twitter	@username	Remplacement par un Token	@MENTION
Hashtag	#example	Remplacement par un Token	@HASHTAG
URL	http://t.co/XGUfUDoL3B	Suppression de l'URL	””

Élément	Exemple	Traitement à effectuer	Exemple après traitement
Chiffre	13	Transformation en String	thirdteen
Majuscules	Hello	Suppression de la majuscule	hello
Ponctuation	!	Suppression de la ponctuation	””
Mots répétés	cool cool cool	Normalisation en une seule occurrence	cool
Lettres répétées	that’s greeeeeeeat!	Réduction des répétitions excessives	that’s great!
Abréviations	ngl, fr	Remplacement par la version complète	not gonna lie, for real
Stopwords (déterminants)	the, and, a	Suppression si non pertinent	””
Slang (Argot)	gonna, dunno, wanna	Remplacement par des mots standards	going to, do not know, want to
Langue étrangère	bonjour, gracias	Détection et traduction éventuelle	hello, thank you
Caractères spéciaux	§, $, ^	Suppression des caractères	””
Expressions courantes	btw, lol	Remplacement par la version complète	by the way, laughing out loud
Négation mal formatée	ain’t, dunno	Correction grammaticale	am not, do not know
Émojis en Unicode	`\U0001F60D` ( )	Conversion en texte lisible	smiling_face_with_hearteyes
Symboles de devises	10$, 10€	Normalisation (ex: “10 euros”)	10 dollars, 10 euros
Accents	cliché	Normalisation (Suppression des accents)	cliche
Heures	10AM, 13:30	Remplacement par un token	@TIME
Numéro de téléphone	+339208373	Remplacement par un token	@PHONENUMBER
Expression flottante	14,34 10,000	Conversion en texte lisible	fourteen thirty-four, ten thousand

On implémente ici nos fonctions que nous allons utiliser par la suite pour notre ingéniérie des données :

```
[ ]: #Fonction permettant de gérer les caractères spéciaux ayant un sens particulier (n° de tel commençant par +, format horaire, nombres...)
def removeSpecialCharacters(word, keepTokens):
```

```python
    # Replace '+' followed by digits (potential phone numbers) with␣
↪'PHONE_NUMBER'
    if keepTokens:
        word = re.sub(r'\+\d+', 'TOKENPHONENUMBER', word)
    else:
        word = re.sub(r'\+\d+', '', word)

    # Remove / between numbers : 10/10 -> 10 out of 10
    word = re.sub(r'(\d+)/(\d+)', r'\1 out of \2', word)

    # Replace time expressions in HH:MM format
    if keepTokens:
        word = re.sub(r'\b\d{1,2}:\d{2}\b', 'TOKENTIME', word)
    else:
        word = re.sub(r'\b\d{1,2}:\d{2}\b', '', word)

    # Replace numbers followed by 'k' with their full value (e.g., 41916514k →␣
↪41916514000 ,  or 5.5k -> 5500)
    word = re.sub(r'(\d+)k\b', lambda m: str(int(m.group(1)) * 1000), word)
    return word

#Fonction permet de supprimer les répétitions successives de lettres (cas␣
↪particuliers rencontrés)
def fixRepeat(word):
    # Reduce excessive repetition to exactly 2 occurrences
    repeat_regexp = re.compile(r'(\w*)(\w)\2{2,}(\w*)')
    repl = r'\1\2\2\3'

    if wordnet.synsets(word):
        return word

    repl_word = repeat_regexp.sub(repl, word)
    if repl_word != word:
        return fixRepeat(repl_word)

    # Try all combinations of removing one duplicate letter at a time
    candidates = set()
    for i in range(len(repl_word) - 1):
        if repl_word[i] == repl_word[i + 1]:
            candidates.add(repl_word[:i] + repl_word[i+1:])

    # Check if any of the candidates is a valid word
    for candidate in candidates:
        if wordnet.synsets(candidate):
            return candidate

    # No valid word is found, return the single-letter version
```

```
    single_letter_version = re.sub(r'(.)\1', r'\1', repl_word)
    return single_letter_version

# Fonction permettant de supprimer les accents
def remove_accents(text):
    return ''.join(c for c in unicodedata.normalize('NFD', text) if unicodedata.
↪category(c) != 'Mn')
```

On créer donc notre fonction **prepareText** permettant de préparer nos données brutes afin de les reformater correctement :

```
[ ]: stop_words_set = set(stopwords.words('english'))
translator = Translator() # initialisation du traducteur
lemmatizer = WordNetLemmatizer() # initialisation du lematiseur
stemmer = PorterStemmer() # initialisation du "racinisateur"
tokens = {"MENTION", "HASHTAG", "TIME", "PHONENUMBER"} # liste de nos token à␣
↪identifier

# Fonction permettant de préparer la chaîne de charactères passée en paramètre
def prepareText(text, keepTokens: bool = True, keepEmojis: bool = True,␣
↪numbersAsTokens: bool = False, translate = True):
    """
    Prépare la chaîne de caractère passée en paramètre

    Parameters
    ----------
    text : str
        La chaîne de caractères
    keepTokens : bool, optional
        True si on doit garder les token, False si on doit les supprimer␣
↪(defaut : True)
    keepEmojis : bool, optional
        True si on doit garder les emojis, False si on doit les supprimer␣
↪(defaut : True)
    numbersAsTokens : bool, optional
        True si on doit transformer les chiffres en token, False si on doit les␣
↪supprimer (defaut : False)
         Ce token n'est pas supprimé si keepToken vaut False
    translate : bool, optional
        True si on doit traduire le texte en anglais, False si on ne le fait␣
↪pas (defaut : True)

    Returns
    -------
    str
        La chaîne de caractère préparée
    """
```

```python
#Majuscule, suppression
data = str(text).lower()

#Suppression d'accent
data = remove_accents(data)

#Contraction, on corrige
data = contractions.fix(data)

#Emoji, transformation en String
if (keepEmojis):
    data = emoji.demojize(data)
else:
    data = emoji.replace_emoji(data, replace='')

#Mention Twitter, transformation en Token
if (keepTokens):
    data = re.sub(r'@\w+', 'TOKENMENTION', data)
else:
    data = re.sub(r'@\w+', '', data)

#Hashtag, transformation en Token
if (keepTokens):
    data = re.sub(r'#\w+', 'TOKENHASHTAG', data)
else:
    data = re.sub(r'#\w+', '', data)

#URL, on supprime
data = re.sub(r'https?://\S+|www\.\S+', '', data)


#Devise, remplacement par sa chaîne de caractères
for symbol, name in currency_dict.items():
    data = re.sub(rf'(\d+){re.escape(symbol)}', r'\1 ' + name, data)


#Special caracters that requires more attention than just remove
data = removeSpecialCharacters(data, keepTokens)

#Keep rating expressions (ex : 10/10)
rating_expressions = {}

def replace_match(match):
    key = f"RATING_{len(rating_expressions)}"  # Unique placeholder
    rating_expressions[key] = match.group(0)   # Store full match
    return key
```

```python
    # Replace rating expressions with placeholders
    data = re.sub(r'(\d+|ten|nine|eight|seven|six|five|four|three|two|one) out␣
↪of (\d+|ten|nine|eight|seven|six|five|four|three|two|one)', replace_match,␣
↪data)


    #Stopwords, suppression
    data = ' '.join([word for word in data.split() if word not in␣
↪stop_words_set])

    # Restore full rating expressions
    for key, value in rating_expressions.items():
        data = data.replace(key, value)

    #Ponctuation & charactères spéciaux, suppression
    data = re.sub(r'[^\w\s]', '', data)

    #Chiffre, transformation en String
    if (numbersAsTokens):
        words = data.split()
        data = ' '.join(["number" if word.isdigit() else word for word in␣
↪words])
    else:
        words = data.split()
        data = ' '.join([inflect.engine().number_to_words(word) if word.
↪isdigit() else word for word in words])

    #Heures, transformation en token
    if keepTokens:
        data = re.sub(r'\b(\d{1,2}([:h]\d{2})?\s*(am|pm)?)\b', 'TOKENTIME',␣
↪data)
    else:
        data = re.sub(r'\b(\d{1,2}([:h]\d{2})?\s*(am|pm)?)\b', '', data)

    #Abréviation (Slang)
    data = ' '.join([slang_dict.get(word, word) for word in data.split()])

    #Mots répétés
    data = re.sub(r'\b(\w+)(\s+\1\b)+', r'\1', data)
    #Lettres répétés
    data = ' '.join([fixRepeat(word) for word in data.split()])

    #remplacer TOKEN par @TOKENxxxx correspondant
    if (keepTokens):
        for token in tokens:
            data = re.sub(rf'TOKEN{token}', f'{token}', data)
```

```python
    #Traduction du tweet
    if translate:
        try:
            data = translator.translate(data, dest='en').text
        except Exception as e:
            pass

    return data
```

Exemple du passage de notre fonction

[ ]:
```python
#URL
display("http://t.co/XGUfUDoLJB")
display(prepareText("http://t.co/XGUfUDoLJB"))
print("\n")
#Chiffre
display("3")
display(prepareText("3"))
print("\n")
#Majuscule
display("Hello")
display(prepareText("Hello"))
print("\n")
#Ponctuation
display("Hello!")
display(prepareText("Hello!"))
print("\n")
#Abréviation
display("lol")
display(prepareText("lol"))
print("\n")
#StepWord
display("After planning the project, she carefully researched each step, ensuring the execution was smooth and timely")
display(prepareText("After planning the project, she carefully researched each step, ensuring the execution was smooth and timely"))
print("\n")
#Emojis
display(" ")
display("keepEmojis=True : " + prepareText(" ", keepEmojis=True))
display("keepEmojis=False : " + prepareText(" ", keepEmojis=False))
print("\n")
#Traductions (dernière étape)
display("      ")
display("translate=True : " + prepareText("      ", translate=True))
display("translate=False : " + prepareText("      ", translate=False))
print("\n")
```

```
#Mention Twitter
display("as @username said it's bad !")
display("keepTokens=True : " + prepareText("as @username said it's bad !",␣
↪keepTokens=True))
display("keepTokens=False : " + prepareText("as @username said it's bad !",␣
↪keepTokens=False))
print("\n")
#Hashtag
display("I went to the theater to see Dune 2 #Dune")
display(prepareText("I went to the theater to see Dune 2 #Dune"))
print("\n")
#Charactères spéciaux
display("§$£")
display(prepareText("§$£"))
print("\n")
#Devices
display("10$ 10£ 10€")
display(prepareText("10$ 10£ 10€"))
print("\n")
#Mot répétés
display("Cool Cool Cool Cool Hot Hot Hot")
display(prepareText("Cool Cool Cool Cool Hot Hot Hot"))
print("\n")
#Lettres répétées
display("Steaaaaaaaak tendeeeeers beeeeer goooooose   threeeeeeee woooooooooood␣
↪aggggggggressive")
display(prepareText("Steaaaaaaaak tendeeeeers beeeeer goooooose   threeeeeeee␣
↪woooooooooood aggggggggressive"))
print("\n")
#Accent
display("cliché")
display(prepareText("cliché"))
print("\n")
#Heures
display("10AM computer 10:30 potatoes 10h30")
display(prepareText("10AM computer 10:30 potatoes 10h30"))
print("\n")
#Numéro de téléphone
display("+33123456789")
display(prepareText("+33123456789"))
print("\n")
#Expression flottante
display("14,34 10,000")
display("numbersAsTokens=True : " + prepareText("14,34 10,000",␣
↪numbersAsTokens=True))
display("numbersAsTokens=True : " + prepareText("14,34 10,000",␣
↪numbersAsTokens=False))
```

```python
print("\n")
# 10/10
display("10/10")
display(prepareText("10/10"))
print("\n")
#Charactères spéciaux (numéro de téléphone, x/x , 10,0000)
display("N. Lutz ")
display(prepareText("N. Lutz"))
print("\n")
```

```
'http://t.co/XGUfUDoLJB'

''



'3'

'three'



'Hello'

'hello'



'Hello!'

'hello'



'lol'

'laughing out loud'



'After planning the project, she carefully researched each step, ensuring the␣
↪execution was smooth and timely'

'planning project carefully researched step ensuring execution smooth timely'



' '

'keepEmojis=True : smiling_face_with_hearteyes'
```

```
'keepEmojis=False : '


'      '

'translate=True : I went to the supermarket today and bought some fruit'

'translate=False :        '


"as @username said it's bad !"

'keepTokens=True : MENTION said bad'

'keepTokens=False : said bad'


'I went to the theater to see Dune 2 #Dune'

'went theater see dune two HASHTAG'


'§$£'

''


'10$ 10£ 10€'

'ten dollar ten pound ten euro'


'Cool Cool Cool Cool Hot Hot Hot'

'cool hot'


'Steaaaaaaaak tendeeeeers beeeeer goooooose   threeeeeeee wooooooooooood 
 ↪agggggggggressive'

'steak tenders beer goose three wood aggressive'


'cliché'
```

```
'cliche'
```

```
'10AM computer 10:30 potatoes 10h30'

'TIME computer TIME potatoes TIME'
```

```
'+33123456789'

'PHONENUMBER'
```

```
'14,34 10,000'

'numbersAsTokens=True : number'

'numbersAsTokens=True : one thousand, four hundred and thirty-four ten thousand'
```

```
'10/10'

'ten out of ten'
```

```
'N. Lutz '

'n lutz'
```

On créé une copie de notre set de données de base et on applique notre fonction sur tout nos tweets /! cette cellule prend un temps de calcul conséquent car elle créer une copie du CSV avec les données toutes formattées.

[ ]:
```
# File path for the specific dataset
file_path = 'dataSet/precomputed/dataPrepared1101.csv'

# On évite de traiter à nouveau les données si on a déjà le fichier des données␣
↪traitées
if os.path.exists(file_path):
    # If the file exists, load it
    dataPrepared = pd.read_csv(file_path)
    print("Le fichier dataPrepared1101 existe déjà. Chargement des données␣
↪depuis le disque.")
```

```
else:
    # If the file does not exist, compute it
    dataPrepared = df.copy()
    dataPrepared['text'] = dataPrepared['text'].apply(prepareText)

    # Save the computed data to disk
    dataPrepared.to_csv(file_path, index=False)
    print("Le fichier dataPrepared1101 n'existe pas. Les données ont été␣
 ↪calculées et enregistrées.")
```

```
Le fichier dataPrepared1101 existe déjà. Chargement des données depuis le
disque.
```

On supprime toutes les lignes contenant un tweet vide :

[ ]:
```
# Suppression de toutes les lignes vides
dataPrepared = dataPrepared[dataPrepared['text'] != '']
dataPrepared = dataPrepared.dropna(subset=['text'])

# Afficher le nombre de ligne ayant un tweet vide
print("Nombre de lignes contenant un tweet vide : ", len(df[df['text'] == '']))

# Afficher
print("5 premières lignes du dataset :")
display(dataPrepared.head())
```

```
Nombre de lignes contenant un tweet vide :  0
5 premières lignes du dataset :

   Unnamed: 0            tweet_id  \
0           0  316669998137483264
1           1  319090866545385472
2           2  322030931022065664
3           3  322694830620807168
4           4  328524426658328576

                                                text  science_related  \
0  knees bit sore guess sign recent treadmiling w…                0
1  mcdonalds breakfast stop gym basketbalflexed_b…                0
2  gynecologist cancer experience explain dangers…                1
3  couchlock highs lead sleeping couch got stop shit                1
4  daily routine help prevent problems bipolar di…                1

   scientific_claim  scientific_reference  scientific_context
0               0.0                   0.0                 0.0
1               0.0                   0.0                 0.0
2               1.0                   0.0                 0.0
3               1.0                   0.0                 0.0
4               1.0                   0.0                 0.0
```

Une fois notre premier traitement effectué on va effecter la dernière partie des traitements des données brutes la lemmatisation, racinisation et tagination. Ces étapes permettent d'affiner le texte pour que chaque mot soit réduit à sa forme de base, ce qui est essentiel pour de nombreuses applications de traitement de texte, comme la recherche d'informations ou l'analyse de sentiments.

Voici les étapes que l'on va faire après le formatage :

Lemmatisation : Cette technique consiste à réduire un mot à sa forme canonique (ou lemmé), c'est-à-dire à la forme sous laquelle il apparaît dans le dictionnaire. Exemple better deviendra good.

Racinisation : Cette méthode consiste à réduire un mot à sa racine, c'est-à-dire à enlever les suffixes (ou préfixes) pour obtenir une forme simplifiée du mot. Cela permet de mieux traiter les variations de mot comme runner qui devriendra run.

Tagination (ou étiquetage de parties du discours) : Cette technique consiste à identifier et à étiqueter chaque mot d'un texte en fonction de sa catégorie grammaticale (nom, verbe, adjectif, etc.)

On va commencer par appliquer une tokenisation et une taggenisation sur chacun de nos tweets. Pour cela on va définir 2 fonctions :

[ ]:
```
#Fonction permettant de récupérer le bon tag du mot passé en paramètre
def get_wordnet_pos(word):
    if word.startswith('J'):
        return wordnet.ADJ
    elif word.startswith('V'):
        return wordnet.VERB
    elif word.startswith('N'):
        return wordnet.NOUN
    elif word.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

#Fonction qui applique la tokenisation et une taggenisation sur une phrase␣
↪passée en paramètre
def lemmatize_taggenize_sentence(sentence):
    tokens = word_tokenize(sentence) # Tokenisation du texte
    tagged_tokens = pos_tag(tokens) # Étiquetage des mots (POS tagging)
    lemmatized = [lemmatizer.lemmatize(token, get_wordnet_pos(tag)) for token,␣
↪tag in tagged_tokens]
    return " ".join(lemmatized)
```

Essayons notre fonction :

[ ]:
```
print(dataPrepared['text'][0])
print(lemmatize_taggenize_sentence(dataPrepared['text'][0]))
```

```
knees bit sore guess sign recent treadmiling working
knee bite sore guess sign recent treadmiling work
```

On applique alors notre fonction sur nos données préparées :

```
[ ]: dataPrepared['text'].apply(lemmatize_taggenize_sentence)
```

```
[ ]: 0        knee bite sore guess sign recent treadmiling work
     1        mcdonalds breakfast stop gym basketbalflexed_b…
     2        gynecologist cancer experience explain danger …
     3            couchlock high lead sleep couch get stop shit
     4        daily routine help prevent problem bipolar dis…
                                    …
     1135     MENTION sorry one out of four million dead cov…
     1136     dear HASHTAG applicant kindly download enrolme…
     1137                          uber support team email address
     1138     house pass bill increase stimulus check two th…
     1139     MENTION HASHTAG renjun deserve well treatment …
     Name: text, Length: 1139, dtype: object
```

On va ajouter dans notre DataFrame un attribut contenant les tweets correctement traités en appliquant les opérations suivantes :

- Normalisation du texte : suppression des variations morphologiques.

-Réduction de la dimensionnalité : un même concept est représenté par un seul mot.

-Amélioration des performances des modèles : les algorithmes de Machine Learning comprennent mieux les relations entre les mots.

La partie subtile c'est que ce genre de traitement peuvent influencer complétement les mots post-traitement. Par exemple unhappiness doit devenir unhappy, pour éviter ce genre d'erreur on doit appliquer une dernière transformation :

```
[ ]: #Fonction qui permet de ne pas perdre le sens d'un mot traité (i.e unhap)
def refine_stem_lemmatize(token, tag):
    try:
        # Racinisation
        stemmed = stemmer.stem(token)

        # Vérification du préfixe "un"
        if stemmed.startswith("un") and len(stemmed) > 2:  # Vérifie que "un"␣
↪n'est pas seul
            root = stemmed[2:]  # Retire le préfixe "un"
            if wordnet.synsets(root):  # Vérifie si la racine sans "un" est␣
↪valide dans WordNet
                return f"not {root}"

        # Vérification de validité du mot racinisé
        if not wordnet.synsets(stemmed):
            stemmed = token  # Si le mot racinisé est incompréhensible, garde␣
↪l'original
```

```
        # Lemmatisation
        lemmatized = lemmatizer.lemmatize(stemmed, get_wordnet_pos(tag))
        return lemmatized
    except Exception as e:
        # Afficher le mot problématique et son erreur
        print(f"Erreur avec le mot : '{token}' - Exception : {e}")
        raise e  # Propager l'exception pour un traitement éventuel



#Fonction qui ajoute dans le dataSet un colonne contenant le texte traité
def process_text_column(text):

    # Tokenisation et traitement
    tokens = word_tokenize(text)
    tagged_tokens = pos_tag(tokens)
    processed_tokens = [refine_stem_lemmatize(token, tag) for token, tag in␣
 ↪tagged_tokens]
    return " ".join(processed_tokens)


# Créer une colonne vide pour stocker les textes transformés
dataPrepared['processed_text'] = ""

# Boucle for avec iterrows
for index, row in dataPrepared.iterrows():
    text = row['text']  # Récupérer le texte original

    if pd.notnull(text) and text.strip() != "":  # Vérifier que le texte est␣
 ↪valide
        try:
            # Appliquer la fonction process_text_column
            dataPrepared.at[index, 'processed_text'] = process_text_column(text)
        except Exception as e:
            print(f"Erreur à l'index {index} avec le texte : {text}")
            print(f"Exception : {e}")
            dataPrepared.at[index, 'processed_text'] = ""  # Insérer une chaîne␣
 ↪vide en cas d'erreur
    else:
        dataPrepared.at[index, 'processed_text'] = ""  # Gérer les textes nuls␣
 ↪ou vides
```

On teste notre fonction de traitement final :

[ ]:
```
text = "The runners were running faster than the dogs unhappiness displacement␣
 ↪inflexibility irresponsible kindness impossible"
tokens = word_tokenize(text)
```

```
tagged_tokens = pos_tag(tokens)
stemmed_then_lemmatized = [refine_stem_lemmatize(token, tag) for token, tag in
    tagged_tokens]
print("racinetisation puis lemmatization : ")
print(" ".join(stemmed_then_lemmatized))
print("\n")

stemmed_tokens = [stemmer.stem(token) for token in tokens]
lemmatized = [lemmatizer.lemmatize(token, get_wordnet_pos(tag)) for token, tag
    in tagged_tokens]
print("racinetisation : ")
print(" ".join(stemmed_tokens))
print("\n")
print("lemmatization : ")
print(" ".join(lemmatized))
```

```
racinetisation puis lemmatization :
The runner be run faster than the dog unhappiness displacement inflexibility
irresponsible kind impossible


racinetisation :
the runner were run faster than the dog unhappi displac inflex irrespons kind
imposs


lemmatization :
The runner be run faster than the dog unhappiness displacement inflexibility
irresponsible kindness impossible
```

On remplace dans notre dataSet la colonne processed_text contenant le texte filtré et traité par text afin d'obtenir qu'un seul attribut :

[ ]:
```
#Ajout de l'attribut processed_text sur chaque ligne de notre dataSet
dataPrepared['text'] = dataPrepared['processed_text']
#Suppression de l'attribut processed_text
dataPrepared = dataPrepared.drop(columns=['processed_text'])

# Sauvegarde des données dans un CSV
dataPrepared.to_csv('dataSet/dataPrepared.csv', index=False)

#Affichage des 5 premières lignes
display(dataPrepared.head())
```

```
   Unnamed: 0            tweet_id  \
0           0  316669998137483264
1           1  319090866545385472
2           2  322030931022065664
```

```
3           3  322694830620807168
4           4  328524426658328576

                                                text  science_related  \
0  knee bite sore guess sign recent treadmiling work                0
1  mcdonalds breakfast stop gym basketbalflexed_b…                0
2  gynecologist cancer experience explain danger …                1
3      couchlock high lead sleep couch get stop shit                1
4  daily routine help prevent problem bipolar dis…                1

   scientific_claim  scientific_reference  scientific_context
0               0.0                   0.0                 0.0
1               0.0                   0.0                 0.0
2               1.0                   0.0                 0.0
3               1.0                   0.0                 0.0
4               1.0                   0.0                 0.0
```

0.3 Vectorisation via TF-IDF

Dans cette partie, on souhaite continuer notre travail visant à préparer nos données pour les envoyer aux algorithmes d'apprentissage automatique. Pour cela, on va effectuer une vectorisation via la méthode TF-IDF.

Le principe de la vectorisation est de convertir des données textuelles en une représentation numérique. Cela va permettre aux algorithmes d'apprentissage automatique de comprendre et de traiter le langage humain à partir de nos données préparées.

Dans notre cas, on a décidé d'utiliser les n-grammes (séquence de n mot répétées), cela va nous permettre de récupérer les relations entre les mots et ainsi détecter les mots qui pourraient potentiellement nous conduire vers une fake news ou nous indiquer les mots démontrant qu'un tweet est scientifique.

[ ]:
```python
# Récupération des données préparées
dataPrepared = pd.read_csv('dataSet/precomputed/dataPrepared1101.csv')

# Suppression des lignes vides
dataPrepared = dataPrepared[dataPrepared['text'] != '']
dataPrepared = dataPrepared.dropna(subset=['text'])
```

[ ]:
```python
def vectorize_data(text_series):
    vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=0.5, max_df=0.9)
    scaled = MaxAbsScaler().fit_transform(vectorizer.fit_transform(text_series))
    return scaled
#Exemple d'appel
#X = vectorize_data(filtered_data['text'])
```

##**Topic Modelling via LDA**

Une fois nos données vectorisées et compréhensibles pour la machine, nous allons appliquer le principe de *Topic Modelling*.

Le topic modelling est une technique d'apprentissage automatique non supervisé qui identifie et extrait des thèmes ou des sujets latents à partir d'un ensemble de documents textuels (dans notre cas notre ensemble de tweet). Le but de cette étape est d'aider notre futur modèle à labelliser ses données tout en identifiant les sujets principaux.

Dans notre cas, nous utilisons LDA (Latent Dirichlet Allocation), cette technique cherche à découvrir des thématiques cachées (topics) dans un ensemble de documents. On va appliquer le LDA afin d'identifier les topics de chaque tweets et les insérer dans un attribut nommé 'Topic'.

Afin d'obtenir des topics pertinents, il faut faire attention à la répartition de ces derniers, si nous avons un topic trop dominant (ex:30%), cela va écraser la représentation des autres topics. On va alors jouer sur le nombre de topics à représenter afin d'obtenir une réparition plus partagée. De plus nous devons faire attention que chaque topic ait du sens. Le but de cette étape est purement statistique.

[ ]:
```
#Fonction permettant d'afficher les mots les plus courant d'un topic en question
def print_top_words(model, feature_names, n_top_words=10):
    for topic_idx, topic in enumerate(model.components_):
        top_words = [feature_names[i] for i in topic.argsort()[:-n_top_words -␣
 ↪1:-1]]
        display(f"Topic {topic_idx+1}: {', '.join(top_words)}")



# Transformer TF-IDF en une Matrice Exploitable par LDA
count_vectorizer = CountVectorizer(ngram_range=(1, 2), min_df=5, max_df=0.9)
count_matrix = count_vectorizer.fit_transform(dataPrepared['text'])



# Configuration du modèle LDA pour l'appliquer
n_topics = 15  # Nombre de topics à identifier ( variable à ajuster pour avoir␣
 ↪+/- de topic majeur à identifier)



# Préparer les données tokenisées
tokenized_texts = [text.split() for text in dataPrepared['text']]

# Créer un dictionnaire et un corpus
dictionary = Dictionary(tokenized_texts)
corpus = [dictionary.doc2bow(text) for text in tokenized_texts]

# Entraîner un modèle LDA
lda_model_gensim = LdaModel(corpus=corpus, num_topics=n_topics,␣
 ↪id2word=dictionary, passes=10, random_state=42)
```

```python
lda_model = LatentDirichletAllocation(n_components=n_topics, random_state=42,
    max_iter=10)

# Entraînement du modèle
lda_topics = lda_model.fit_transform(count_matrix)


# Obtenir les mots les plus représentatifs de chaque topic
feature_names = count_vectorizer.get_feature_names_out()


# Associer chaque tweet à son topic dominant
topic_assignments = np.argmax(lda_topics, axis=1)

# Ajouter au DataFrame
dataPrepared['Topic'] = topic_assignments

# Afficher pour chaque répartition son nombre d'itération et son occurence
    normalisée dans le même tableau
topic_counts = dataPrepared['Topic'].value_counts(normalize=True)

# Convertir les occurrences en pourcentages
topic_counts_percent = dataPrepared['Topic'].value_counts()

# Créer un tableau avec le nombre d'occurrences et les pourcentages
topic_stats = pd.DataFrame({
    'Représentation (%)': topic_counts,
    'Occurence': topic_counts_percent
})

display("Visualisation des recurrences et de la répartition des topics :")

# Afficher le tableau
display(topic_stats)


display("Affichage des topics principaux représenté dans nos tweets :")
# Afficher les topics principaux représenté 1,3,0,2,4
print_top_words(lda_model, feature_names, n_top_words=10)


display("Visualisation de la répartition des topics via un graph normalisé :")
# Visualisation de la répartition des topics via un graph normalisé
plt.figure(figsize=(10, 6))
topic_counts.plot(kind='bar', color=colors)
plt.title('Répartition des Topics Dominants', fontsize=16)
plt.xlabel('Topic', fontsize=12)
```

```python
plt.ylabel('Proportion (%)', fontsize=12)
plt.xticks(rotation=0)
plt.show()
```

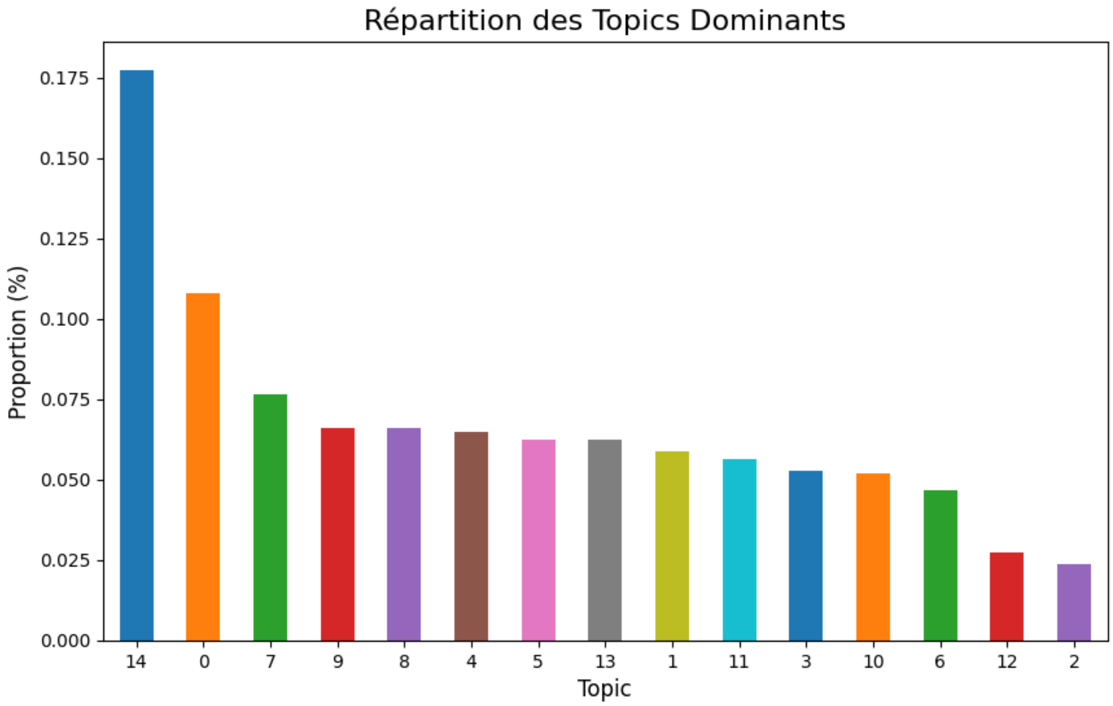
```
'Visualisation des recurrences et de la répartition des topics :'

       Représentation (%)  Occurence
Topic
14               0.177349        202
0                0.107989        123
7                0.076383         87
9                0.065847         75
8                0.065847         75
4                0.064969         74
5                0.062335         71
13               0.062335         71
1                0.058824         67
11               0.056190         64
3                0.052678         60
10               0.051800         59
6                0.046532         53
12               0.027217         31
2                0.023705         27

'Affichage des topics principaux représenté dans nos tweets :'

'Topic 1: mention, reports, via, via mention, hashtag, research, mention
 ↪reports, way, trump, cause'

'Topic 2: hundred, one, and, hundred and, thousand, twenty, eight, five, one
 ↪hundred, one thousand'

'Topic 3: three, fifty, four, thousand, and fifty, seven, six, and, three
 ↪thousand, evidence'

'Topic 4: science, good, mention, look, time, important, take, real, the, story'

'Topic 5: mention, stop, hashtag, get, mention stop, brain, find, us, big, let'

'Topic 6: hashtag, mention, new, cancer, every, get, see, oral, video, much'

'Topic 7: two, and, two thousand, thousand, thousand and, mention, hashtag,
 ↪eighteen, us, and eighteen'

'Topic 8: hashtag, stop, mention, support, us, need, would, climate, go, change'

'Topic 9: mention, study, lead, could, stop, new, support, twitter, may, help'

'Topic 10: people, stop, support, mention, life, like, today, sleep, pain,
 ↪program'

'Topic 11: stop, free, men, god, play, join, increase, show, think, sex'

'Topic 12: mention, stop, associated, it, got, seems, trade, supports, game, use'
```

```
'Topic 13: leads, day, one, ninety, increase, five, and ninety, security, forty, two'
```

```
'Topic 14: hashtag, treat, mention, know, weight, game, scientists, ten, must, lower'
```

```
'Topic 15: hashtag, mention, support, mention hashtag, stop, support hashtag, new, hashtag mention, please, today'
```

```
'Visualisation de la répartition des topics via un graph normalisé :'
```



On observe que nous avons des topics ayant une réparition +/- équivalente, ce qui signifie que ces 7 topics sont à peu près représentés de la même manière dans nos tweets.

Evaluons la cohérence de nos topics

[ ]:
```
#Evaluons la cohérence de nos topics
coherence_model = CoherenceModel(model=lda_model_gensim, texts=tokenized_texts, dictionary=dictionary, coherence='c_v')
coherence_score = coherence_model.get_coherence()

print(f"Score de cohérence des topics : {coherence_score}")
```

```
Score de cohérence des topics : 0.41984850392502715
```

Visualisation du nuage des mots de chaque topics

[ ]:
```python
n_topics = len(lda_model.components_)


n_rows = 2
n_cols = (n_topics // n_rows) + (n_topics % n_rows > 0)


fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(n_cols * 5,␣
↪n_rows * 5))
axes = axes.flatten()
# Boucle sur chaque topic pour créer le nuage de mots
for topic_idx, topic in enumerate(lda_model.components_):
    # Générer le nuage de mots pour chaque topic
    wordcloud = WordCloud(width=800, height=400).generate(" ".
↪join([feature_names[i] for i in topic.argsort()[:-15 - 1:-1]]))
    # Afficher le nuage de mots dans le sous-graphe approprié
    axes[topic_idx].imshow(wordcloud, interpolation="bilinear")
    axes[topic_idx].axis("off")  # Retirer les axes
    axes[topic_idx].set_title(f"Topic {topic_idx + 1}")
for i in range(n_topics, len(axes)):
    axes[i].axis('off')
plt.tight_layout()
plt.show()
```

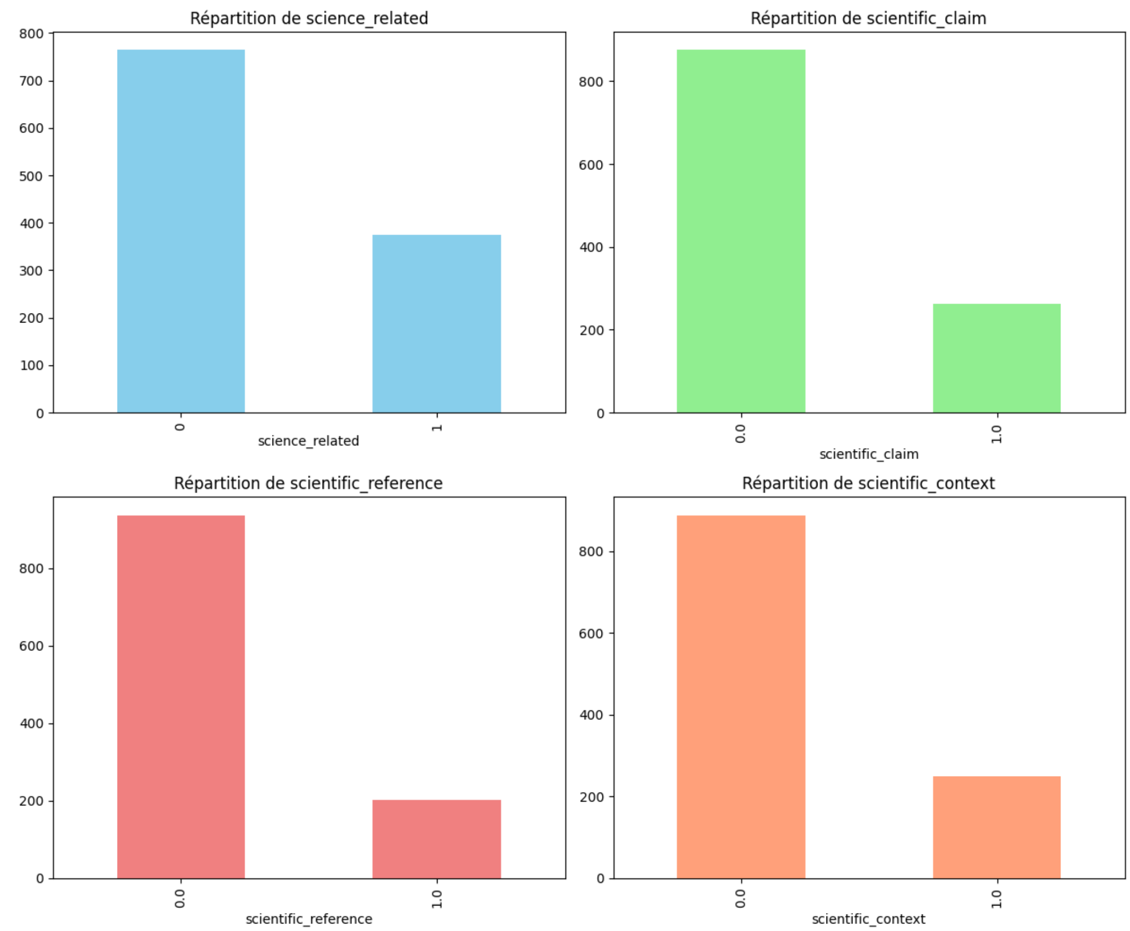
0.4 Upsampling

Après avoir effectué notre topic modelling afin d'identifier les idées principales de nos tweets, nous allons appliquer un Upsampling.

Dans le domaine des méthodes d'apprentissage automatique, l'upsampling est utilisé afin d'équilibrer nos classes désséquilibrées en augmentant les classes qui sont sous représentés. De ce fait, nous aurons une meilleure répartion de nos classes (**science_related, scientific_claim, scientific_reference, scientific_context**). Voici un extrait de chaque classe dans notre dataset préparé avant l'upsampling :

[ ]:
```python
#affichage de chaque classe dans un graph
fig, axes = plt.subplots(2, 2, figsize=(12, 10))  # 2 lignes, 2 colonnes pour␣
↪les sous-graphiques
```

```
# Afficher la répartition de chaque colonne dans un sous-graphe
dataPrepared['science_related'].value_counts().plot(kind='bar', ax=axes[0, 0],␣
 ↪color='skyblue')
axes[0, 0].set_title('Répartition de science_related')

dataPrepared['scientific_claim'].value_counts().plot(kind='bar', ax=axes[0, 1],␣
 ↪color='lightgreen')
axes[0, 1].set_title('Répartition de scientific_claim')

dataPrepared['scientific_reference'].value_counts().plot(kind='bar', ax=axes[1,␣
 ↪0], color='lightcoral')
axes[1, 0].set_title('Répartition de scientific_reference')

dataPrepared['scientific_context'].value_counts().plot(kind='bar', ax=axes[1,␣
 ↪1], color='lightsalmon')
axes[1, 1].set_title('Répartition de scientific_context')


plt.tight_layout()
plt.show()
#print la répartition
```



[ ]:
```python
# Définition de notre fonction d'upsampling, utilisant SMOTE
def resampleData(X, y):
    combined = SMOTETomek(random_state=42)
    X_resampled, y_resampled = combined.fit_resample(X, y)
    return X_resampled, y_resampled
```

Appliquons l'upsampling :

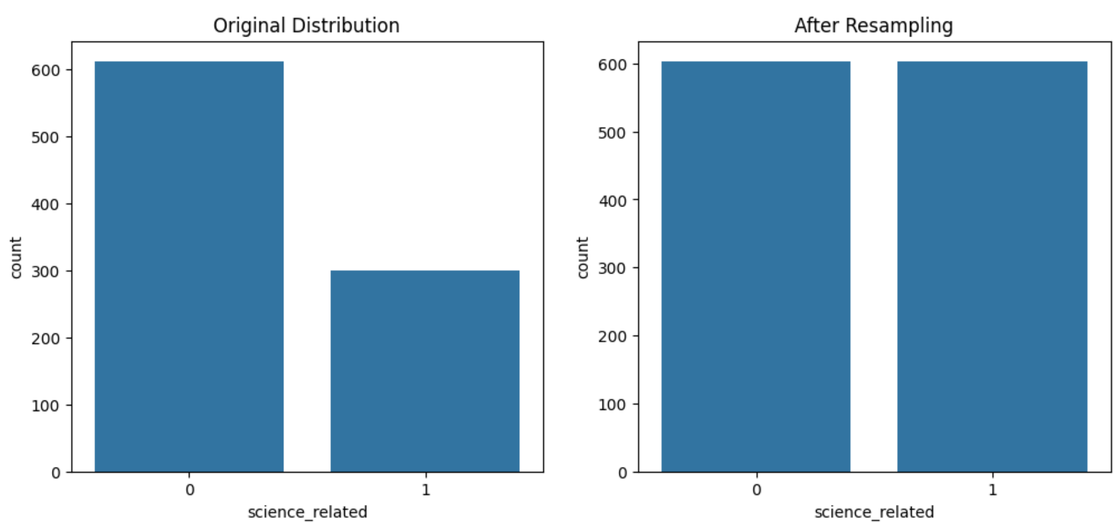
[ ]:
```python
data_lvl1 = dataPrepared.copy()
y = data_lvl1['science_related']
X_text = data_lvl1['text']

#Vectorisation TF-IDF + Scaling
vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=5, max_df=0.9)
X_vectorized = vectorizer.fit_transform(X_text)

scaler = MaxAbsScaler()
X_scaled = scaler.fit_transform(X_vectorized)
```

```
#Split train/test
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42, stratify=y)

X_resampled, y_resampled = resampleData(X_train, y_train)
```

On visualise bien le fait que l'upsampling a bien rééquilibré nos classes et que désormais nous n'avons plus de classes minoritaires.

[ ]:
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.countplot(x=y_train)
plt.title('Original Distribution')

plt.subplot(1, 2, 2)
sns.countplot(x=y_resampled)
plt.title('After Resampling')
```

[ ]:
```
Text(0.5, 1.0, 'After Resampling')
```



1 Classification

[ ]:
```
```

Le but de la classification est de permettre de déterminer le contexte de nos tweets en fonction de leurs caractéristiques.

Pour obtenir les meuilleurs résultats possibles, nous allons pour établir 4 méthodes de classification :

-Decision Tree

-Naïve Bayes

-SVC (Support Vector Clustering)

-KNN (k-nearest neighbors)

On rappelle nos classes de tweets :

- science_related
- scientific_claim
- scientific_reference
- scientific_context

Puis nous allons les tester sur 3 tâches de classification :

- {SCIENTIFIQUE} vs. {NON SCIENTIFIQUE} (2 classes, pour la classification de niveau 1)
- {CLAIM, REF} vs. {CONTEXT} (deux classes pour la classification de niveau 2) -{CLAIM} vs. {REF} vs. {CONTEXT} (trois classes pour la classification niveau 3)

Une fois ce travail réalisé, nous pourrons évaluer les performances de chaque classifieur via plusieurs métriques différentes.

On définis nos fonctions d'affichage de courbes :

[ ]:
```
#Fonction d'affichage des courbes
def plot_curves_confusion(confusion_matrix, class_names):
    plt.figure(1, figsize=(16, 6))
    plt.gcf().subplots_adjust(left=0.125, bottom=0.2, right=1, top=0.9,␣
 ↪wspace=0.25, hspace=0)

    # Matrice de confusion
    plt.subplot(1, 3, 3)
    sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap='Blues',␣
 ↪xticklabels=class_names, yticklabels=class_names)
    plt.xlabel('Predicted', fontsize=12)
    plt.title("Confusion matrix")
    plt.ylabel('True', fontsize=12)
    plt.show()


def plot_curves(scores):
    plt.figure(1, figsize=(16, 6))
    plt.gcf().subplots_adjust(left=0.125, bottom=0.2, right=1, top=0.9,␣
 ↪wspace=0.25, hspace=0)

    # Plot loss
```

```
    plt.subplot(121)
    plt.title('Cross Entropy Loss')
    plt.plot(scores, color='blue')
    plt.ylabel('Loss')
    plt.xlabel('Fold')

    # Plot accuracy
    plt.subplot(122)
    plt.title('Classification Accuracy')
    plt.plot(1 - scores, color='red')
    plt.ylabel('Error Rate')
    plt.xlabel('Fold')

    plt.show()

def plot_curves_results(naive_scores, svc_scores, decision_scores, knn_scores):
  classifiers = ['Naive Bayes', 'SVC', 'Decision Tree', 'KNN']

  fold_scores = [naive_scores, svc_scores, decision_scores, knn_scores]

  # Scores moyens
  plt.figure(figsize=(8, 5))
  mean_scores = [score.mean() for score in fold_scores]
  plt.bar(classifiers, mean_scores, color=['blue', 'orange', 'green', 'red'])
  plt.title('Scores moyens des classifieurs')
  plt.xlabel('Classifieurs')
  plt.ylabel('Score moyen')
  plt.show()
```

2 Classification {SCI} vs {NON-SCI} (NIVEAU 1)

On définit nos données d'entrainement pour la classification de niveau 1 :

[ ]:
```
#Copie de nos données d'entrées
data_lvl1 = dataPrepared.copy()
y = data_lvl1['science_related']
X_text = data_lvl1['text']

#Vectorisation TF-IDF + Scaling
vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=5, max_df=0.9)
X_vectorized = vectorizer.fit_transform(X_text)

scaler = MaxAbsScaler()
X_scaled = scaler.fit_transform(X_vectorized)

#Split train/test
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42, stratify=y)

# Upsampling du jeu d'entraînement SEULEMENT
print(f"sizes of the classes before resampling : {Counter(y_train)}")

X_resampled, y_resampled = resampleData(X_train, y_train)

print(f"sizes of the classes after resampling : {Counter(y_resampled)}")
```

```
sizes of the classes before resampling : Counter({0: 612, 1: 299})
sizes of the classes after resampling : Counter({0: 603, 1: 603})
```

##**Recherches des paramètres optimaux des classifieurs**

Nous allons utiliser GridSearchCV afin de trouver les paramètres optimaux pour les classifieurs. Nous avons expérimenté avec une petite plage de recherche peu précise au cours du projet afin d'itérer rapidement. Après avoir acquis plus de connaissances, nous avons décidé de pousser la recherche de paramètres ici.

Bien entendu, les résultats sont stockés dans un fichier pour éviter de les recalculer à chaque fois.

```
[ ]: def perform_gridsearch_and_plot(X_resampled, y_resampled, X_test, y_test,␣
↪level, filename):
    if os.path.exists(filename):
        with open(filename, 'r') as f:
            best_params = json.load(f)
        print(f"Loaded best parameters from {filename}")
    else:
        best_params = {}

        # Decision Tree
        param_grid_dt = {
            'criterion': ['gini', 'entropy'],
            'max_depth': [3, 5, 7, 10, 20, 30, 40, 50, None],
            'min_samples_split': [2, 5, 10, 15, 20, 25, 30],
            'min_samples_leaf': [1, 2, 4, 6, 8, 10]
        }

        dt = DecisionTreeClassifier(random_state=42)
        grid_dt = GridSearchCV(dt, param_grid_dt, cv=5, scoring='f1_macro',␣
↪n_jobs=-1)
        grid_dt.fit(X_resampled, y_resampled)

        results = grid_dt.cv_results_
        params = results['params']
        scores = results['mean_test_score']
```

```
    gini_scores = [s for s, p in zip(scores, params) if p['criterion'] ==␣
↪'gini']
    entropy_scores = [s for s, p in zip(scores, params) if p['criterion']␣
↪== 'entropy']

    plt.figure(figsize=(10, 6))
    plt.plot(range(len(gini_scores)), gini_scores, label='Gini',␣
↪color='steelblue')
    plt.plot(range(len(entropy_scores)), entropy_scores, label='Entropy',␣
↪color='seagreen')
    plt.title(f'Decision Tree (Level {level}) - Mean F1 Macro Score␣
↪Comparison', fontsize=14)
    plt.xlabel('Parameter Combination Index', fontsize=12)
    plt.ylabel('Mean F1 Macro Score', fontsize=12)
    plt.legend()
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.show()

    best_params["DecisionTree"] = grid_dt.best_params_

    # Naive Bayes
    param_grid_nb = {'alpha': np.logspace(-3, 3, num=100), 'fit_prior':␣
↪[True, False]}
    nb = MultinomialNB()
    grid_nb = GridSearchCV(nb, param_grid_nb, cv=5, scoring='f1_macro')
    grid_nb.fit(X_resampled, y_resampled)

    scores_nb_true = grid_nb.cv_results_['mean_test_score'][::2]
    scores_nb_false = grid_nb.cv_results_['mean_test_score'][1::2]
    alphas = param_grid_nb['alpha']

    plt.figure(figsize=(8, 6))
    plt.plot(alphas, scores_nb_true, label='fit_prior=True', color='teal')
    plt.plot(alphas, scores_nb_false, label='fit_prior=False',␣
↪color='darkorange')
    plt.title(f'Multinomial Naive Bayes (Level {level}) - F1 Score vs␣
↪Alpha', fontsize=14)
    plt.xlabel('Alpha', fontsize=12)
    plt.xscale('log')
    plt.ylabel('Mean F1 Macro Score', fontsize=12)
    plt.legend()
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.show()
```

```python
    best_params["MultinomialNB"] = grid_nb.best_params_

    # KNN
    param_grid_knn = {
        'n_neighbors': list(range(1, 31)),
        'weights': ['uniform', 'distance'],
        'metric': ['euclidean', 'manhattan', 'minkowski']
      }
    knn = KNeighborsClassifier()
    grid_knn = GridSearchCV(knn, param_grid_knn, cv=5, scoring='f1_macro')
    grid_knn.fit(X_resampled, y_resampled)

    # Organize results by metric and weights
    scores_by_metric = defaultdict(lambda: defaultdict(list))

    for params, score in zip(grid_knn.cv_results_['params'], grid_knn.
↪cv_results_['mean_test_score']):
        metric = params['metric']
        weights = params['weights']
        n_neighbors = params['n_neighbors']
        scores_by_metric[metric][weights].append((n_neighbors, score))

    # Plot for each metric
    fig, axs = plt.subplots(1, 3, figsize=(18, 5), sharey=True)
    metrics = ['euclidean', 'manhattan', 'minkowski']
    colors = ['royalblue', 'darkorange']

    for i, metric in enumerate(metrics):
        ax = axs[i]
        for j, weights in enumerate(['uniform', 'distance']):
            data = sorted(scores_by_metric[metric][weights], key=lambda x:␣
↪x[0])
            neighbors = [x[0] for x in data]
            scores = [x[1] for x in data]
            ax.plot(neighbors, scores, label=f'weights={weights}',␣
↪color=colors[j])
        ax.set_title(f'KNN - metric={metric}', fontsize=13)
        ax.set_xlabel('n_neighbors', fontsize=11)
        ax.set_ylabel('Mean F1 Macro Score', fontsize=11)
        ax.legend()
        ax.grid(alpha=0.3)

    plt.tight_layout()
    plt.show()


    best_params["KNN"] = grid_knn.best_params_
```

```
    # SVC
    c_values = np.logspace(-3, 3, num=100)
    param_grid_svc = {
        'C': c_values,
        'kernel': ['linear', 'rbf', 'poly'],
        'gamma': ['auto', 'scale']
    }

    svc = SVC(random_state=42)
    grid_svc = GridSearchCV(svc, param_grid_svc, cv=5, scoring='f1_macro',␣
↪n_jobs=-1)
    grid_svc.fit(X_resampled, y_resampled)

    results = grid_svc.cv_results_
    params = results['params']
    scores = results['mean_test_score']

    for gamma_value in ['auto', 'scale']:
      plt.figure(figsize=(10, 6))
      for kernel, color in zip(['linear', 'rbf'], ['blue', 'green']): #␣
↪kernel = poly yielded consistantly very inferior results so we removed it
          kernel_gamma_scores = [
              s for s, p in zip(scores, params)
              if p['kernel'] == kernel and p['gamma'] == gamma_value
          ]
          plt.plot(c_values, kernel_gamma_scores, label=f'kernel={kernel}',␣
↪color=color)
      plt.title(f'SVC (Level {level}) - gamma={gamma_value}', fontsize=14)
      plt.xlabel('C', fontsize=12)
      plt.xscale('log')
      plt.ylabel('Mean F1 Macro Score', fontsize=12)
      plt.legend()
      plt.grid(alpha=0.3)
      plt.tight_layout()
      plt.show()


    best_params["SVC"] = grid_svc.best_params_

    # Save the best parameters to file
    with open(filename, 'w') as f:
        json.dump(best_params, f, indent=4)
    print(f"Saved best parameters to {filename}")

# Display all best parameters
for model_name, params in best_params.items():
```

```
        print(f"Best parameters for {model_name} (Level {level}): {params}")
```

[ ]:
```
perform_gridsearch_and_plot(X_resampled, y_resampled, X_test, y_test, 1,␣
↪"dataSet/lvl1_parameters.json")
```

```
Loaded best parameters from dataSet/lvl1_parameters.json
Best parameters for DecisionTree (Level 1): {'criterion': 'entropy',
'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 25}
Best parameters for MultinomialNB (Level 1): {'alpha': 0.06579332246575682,
'fit_prior': True}
Best parameters for KNN (Level 1): {'metric': 'euclidean', 'n_neighbors': 2,
'weights': 'uniform'}
Best parameters for SVC (Level 1): {'C': 7.56463327554629, 'gamma': 'scale',
'kernel': 'rbf'}
```

[ ]:
```
# Définition de la fonction qui permet seulement de charger les paramètres␣
↪sauvegardés
def load_models_from_file(filename):
    with open(filename, 'r') as f:
        best_params = json.load(f)

    models = {}

    if "DecisionTree" in best_params:
        models["DecisionTree"] =␣
↪DecisionTreeClassifier(**best_params["DecisionTree"], random_state=42)

    if "MultinomialNB" in best_params:
        models["MultinomialNB"] = MultinomialNB(**best_params["MultinomialNB"])

    if "KNN" in best_params:
        models["KNN"] = KNeighborsClassifier(**best_params["KNN"])

    if "SVC" in best_params:
        models["SVC"] = SVC(**best_params["SVC"], random_state=42)

    return models

lvl1_best_params = load_models_from_file("dataSet/lvl1_parameters.json")

print(lvl1_best_params)
```

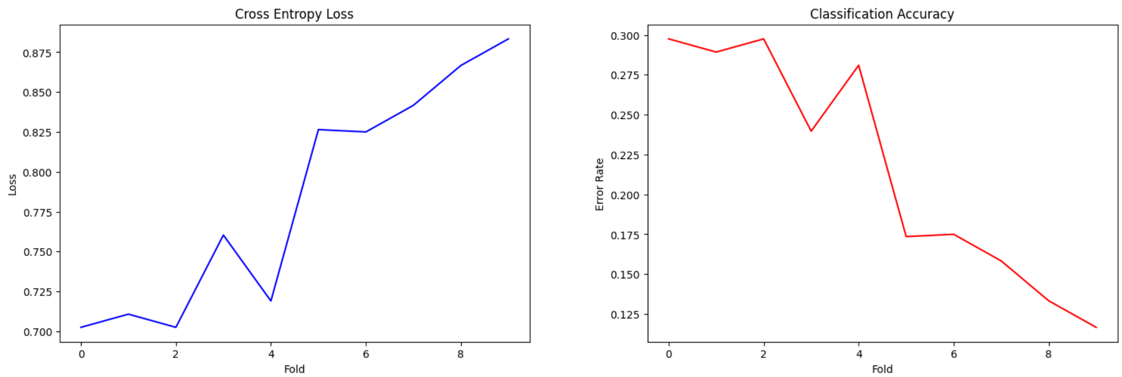
```
{'DecisionTree': DecisionTreeClassifier(criterion='entropy',
min_samples_split=25,
                       random_state=42), 'MultinomialNB':
MultinomialNB(alpha=0.06579332246575682), 'KNN':
KNeighborsClassifier(metric='euclidean', n_neighbors=2), 'SVC':
```

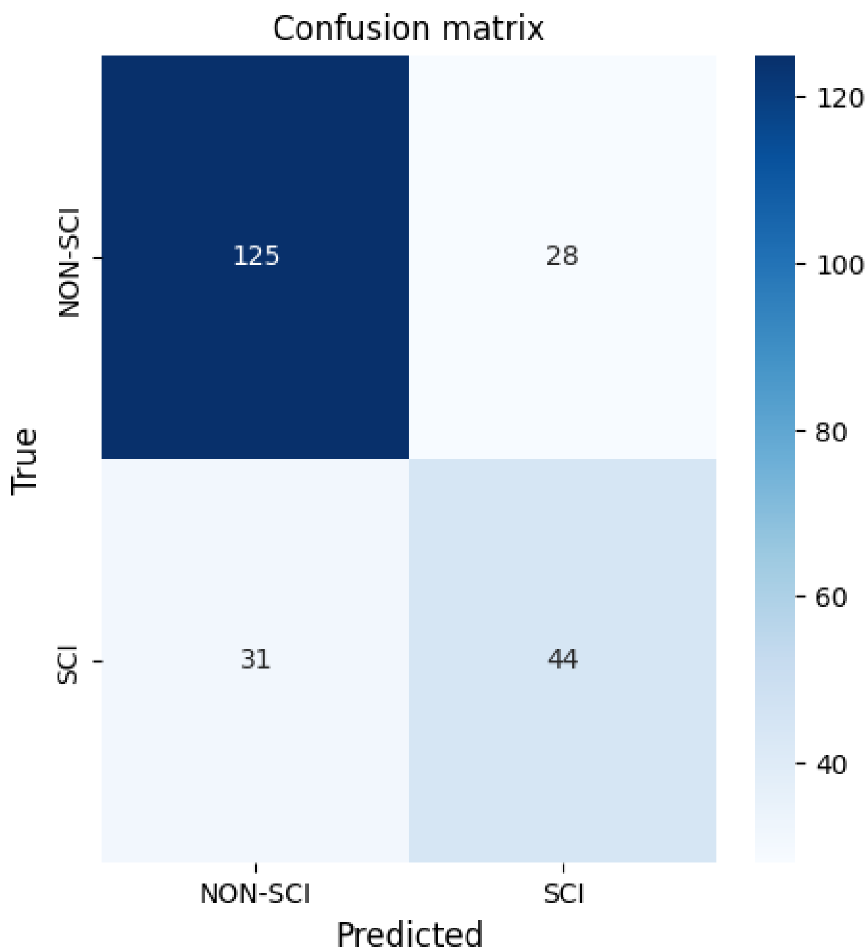
```
SVC(C=7.56463327554629, random_state=42)}
```

##**Decision Tree**

```
[ ]: best_tree = lvl1_best_params["DecisionTree"]

print("Paramètres :", best_tree.get_params())

best_tree.fit(X_resampled, y_resampled)

# Cross-validation sur données équilibrées
cv_scores = cross_val_score(best_tree, X_resampled, y_resampled, cv=10)
print("Scores CV :", cv_scores)
print("Moyenne CV :", cv_scores.mean())

# Prédiction sur le vrai test (non modifié)
y_pred = best_tree.predict(X_test)
print("\n Accuracy (test) :", accuracy_score(y_test, y_pred))
print(" Classification Report (test) :")
print(classification_report(y_test, y_pred))

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Tes fonctions de visualisation (si elles existent)
plot_curves(cv_scores)
plot_curves_confusion(conf_matrix, ['NON-SCI', 'SCI'])
```

```
Paramètres : {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy',
'max_depth': None, 'max_features': None, 'max_leaf_nodes': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 25,
'min_weight_fraction_leaf': 0.0, 'monotonic_cst': None, 'random_state': 42,
'splitter': 'best'}
Scores CV : [0.70247934 0.7107438  0.70247934 0.76033058 0.71900826 0.82644628
 0.825      0.84166667 0.86666667 0.88333333]
Moyenne CV : 0.783815426997245

 Accuracy (test) : 0.7412280701754386
 Classification Report (test) :
              precision    recall  f1-score   support

           0       0.80      0.82      0.81       153
           1       0.61      0.59      0.60        75

    accuracy                           0.74       228
   macro avg       0.71      0.70      0.70       228
weighted avg       0.74      0.74      0.74       228
```

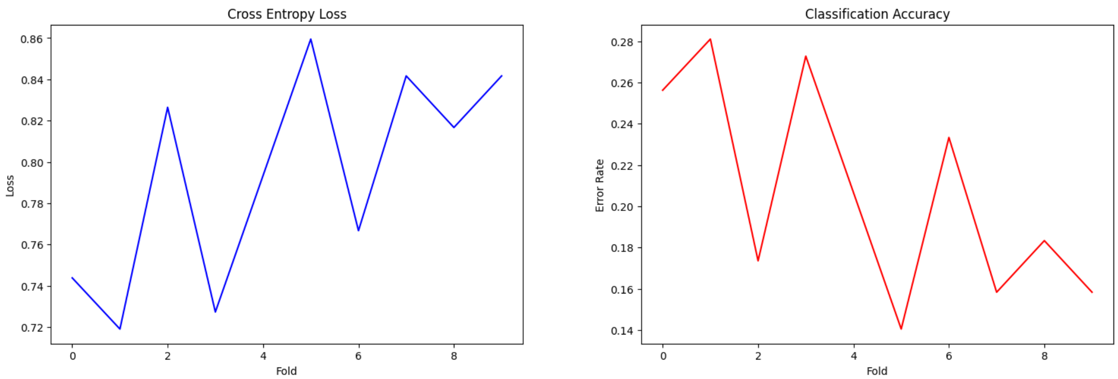

Cross Entropy Loss
Loss
Fold
Classification Accuracy
Error Rate
Fold

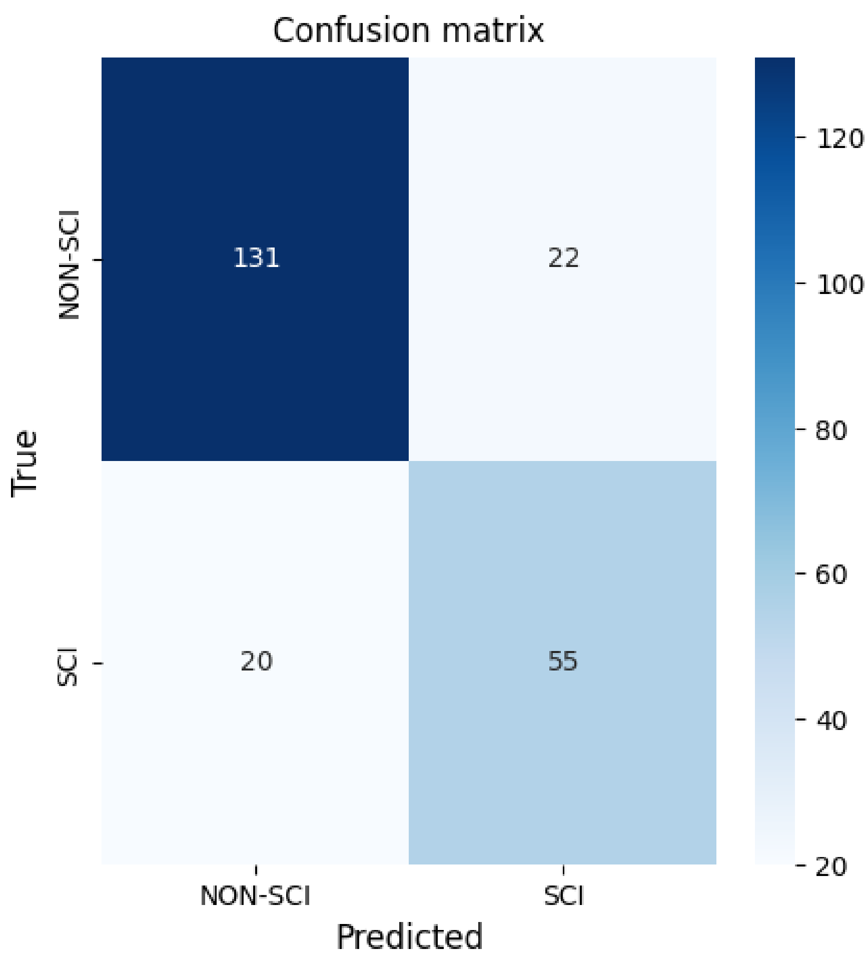

Confusion matrix
True
Predicted
NON-SCI
SCI
125
28
31
44

##Naive bayes

```
[ ]: best_naive_bayes_classifier = lvl1_best_params["MultinomialNB"]

print("Paramètres :", best_naive_bayes_classifier.get_params())

best_naive_bayes_classifier.fit(X_resampled, y_resampled)

naive_scores = cross_val_score(best_naive_bayes_classifier, X_resampled,␣
 ↪y_resampled, cv=10)
print("Scores CV :", naive_scores)
print("Moyenne CV :", naive_scores.mean())


y_pred_test = best_naive_bayes_classifier.predict(X_test)

# Rapports
print("\n Accuracy (test) :", accuracy_score(y_test, y_pred_test))
print("Classification Report (test) :")
print(classification_report(y_test, y_pred_test))

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred_test)

# Visualisation
plot_curves(naive_scores)
plot_curves_confusion(conf_matrix, ['NON-SCI', 'SCI'])
```

```
Paramètres : {'alpha': 0.06579332246575682, 'class_prior': None, 'fit_prior':
True, 'force_alpha': True}
Scores CV : [0.74380165 0.71900826 0.82644628 0.72727273 0.79338843 0.85950413
 0.76666667 0.84166667 0.81666667 0.84166667]
Moyenne CV : 0.7936088154269972

 Accuracy (test) : 0.8157894736842105
Classification Report (test) :
              precision    recall  f1-score   support

           0       0.87      0.86      0.86       153
           1       0.71      0.73      0.72        75

    accuracy                           0.82       228
   macro avg       0.79      0.79      0.79       228
weighted avg       0.82      0.82      0.82       228
```

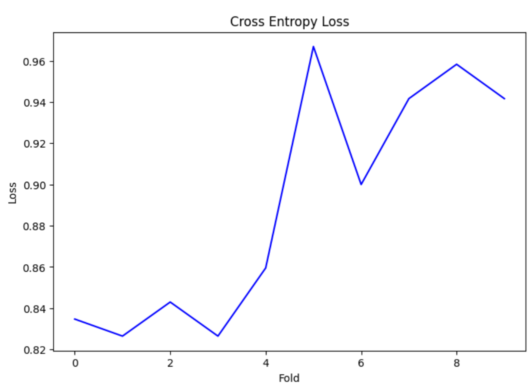

Cross Entropy Loss
Loss
Fold
Classification Accuracy
Error Rate
Fold

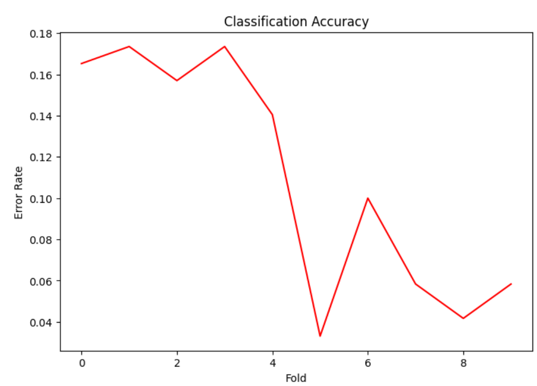

Confusion matrix
True
NON-SCI
SCI
131
22
20
55
NON-SCI
SCI
Predicted

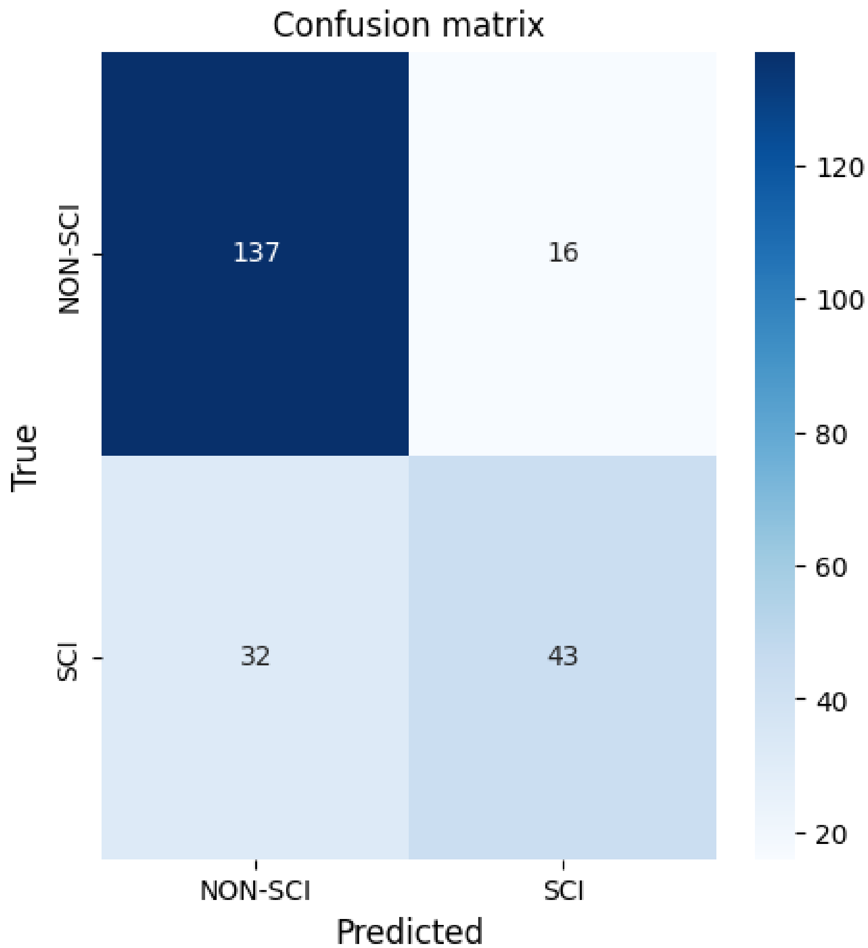
##**SVC**

[ ]:
```python
best_svc_classifier = lvl1_best_params["SVC"]

print("Paramètres :", best_svc_classifier.get_params())

best_svc_classifier.fit(X_resampled, y_resampled)

svc_scores = cross_val_score(best_svc_classifier, X_resampled, y_resampled,␣
↪cv=10)
print("Scores de validation croisée :", svc_scores)
print("Moyenne :", svc_scores.mean())

y_pred_test = best_svc_classifier.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred_test)
plot_curves(svc_scores)
plot_curves_confusion(conf_matrix, ['NON-SCI', 'SCI'])
```

```
Paramètres : {'C': 7.56463327554629, 'break_ties': False, 'cache_size': 200,
'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree':
3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False,
'random_state': 42, 'shrinking': True, 'tol': 0.001, 'verbose': False}
Scores de validation croisée : [0.83471074 0.82644628 0.84297521 0.82644628
0.85950413 0.96694215
 0.9        0.94166667 0.95833333 0.94166667]
Moyenne : 0.8898691460055096
```





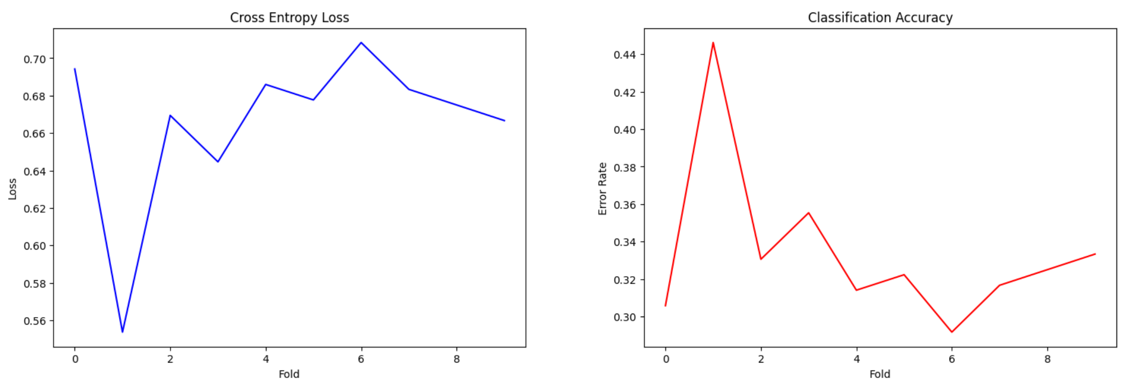


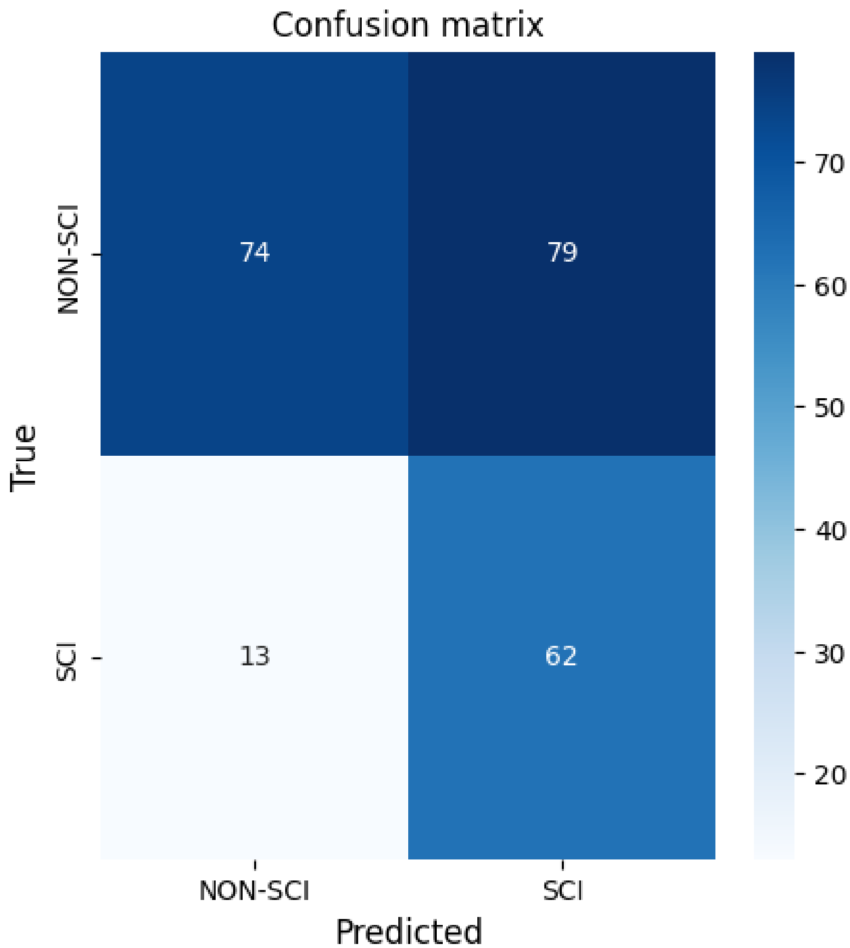
##KNN

```
[ ]: best_knn_classifier = lvl1_best_params["KNN"]

print("Paramètres :", best_knn_classifier.get_params())

best_knn_classifier.fit(X_resampled, y_resampled)

knn_scores = cross_val_score(best_knn_classifier, X_resampled, y_resampled,␣
 ↪cv=10)
print("Scores de validation croisée :", knn_scores)
print("Moyenne :", knn_scores.mean())


y_pred_test = best_knn_classifier.predict(X_test)
print("\n Accuracy (test) :", accuracy_score(y_test, y_pred_test))
```

```python
print("Rapport de classification :\n", classification_report(y_test,␣
↪y_pred_test))


conf_matrix = confusion_matrix(y_test, y_pred_test)


plot_curves(knn_scores)
plot_curves_confusion(conf_matrix, ['NON-SCI', 'SCI'])
```

```
Paramètres : {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'euclidean',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 2, 'p': 2, 'weights':
'uniform'}
Scores de validation croisée : [0.69421488 0.55371901 0.66942149 0.6446281
0.68595041 0.67768595
 0.70833333 0.68333333 0.675      0.66666667]
Moyenne : 0.6658953168044077

 Accuracy (test) : 0.5964912280701754
Rapport de classification :
               precision    recall  f1-score   support

           0       0.85      0.48      0.62       153
           1       0.44      0.83      0.57        75

    accuracy                           0.60       228
   macro avg       0.65      0.66      0.60       228
weighted avg       0.72      0.60      0.60       228
```

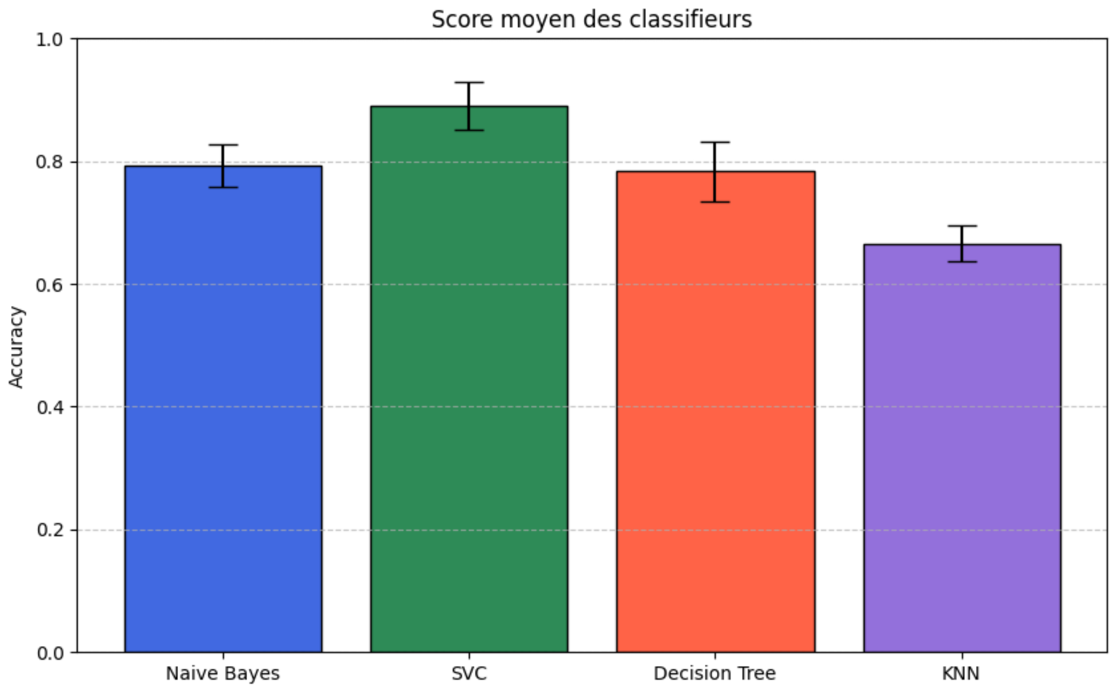




##**Evaluation des classificeurs (Niveau 1)**

[ ]:
```
def compute_mean_and_ci(scores):
    mean = scores.mean()
    std = scores.std()
    n = len(scores)
    ci_width = stats.t.ppf(0.975, df=n-1) * (std / np.sqrt(n))  # Half-width (±)
    return mean, ci_width

def print_accuracy_with_pm(name, scores):
    mean, ci_width = compute_mean_and_ci(scores)
    pm_percent = (ci_width / mean) * 100
    print(f"{name} : {mean:.4f} ± {pm_percent:.2f}%")

def plot_curves_results_with_ci(scores_list, names=None, colors=None):
    means = []
    cis = []
```

```
    for scores in scores_list:
        mean, ci_width = compute_mean_and_ci(scores)
        means.append(mean)
        cis.append(ci_width)

    if names is None:
        names = [f"Model {i+1}" for i in range(len(scores_list))]

    if colors is None:
        colors = ['royalblue', 'seagreen', 'tomato', 'mediumpurple']

    # Plot
    x = np.arange(len(names))
    plt.figure(figsize=(10,6))
    bars = plt.bar(x, means, yerr=cis, capsize=8, color=colors,␣
 ↪edgecolor='black')

    plt.xticks(x, names)
    plt.ylabel('Accuracy')
    plt.title('Score moyen des classifieurs')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.ylim(0, 1)

    plt.show()


# Plot courbes
plot_curves_results_with_ci(
    [naive_scores, svc_scores, cv_scores, knn_scores],
    names=['Naive Bayes', 'SVC', 'Decision Tree', 'KNN']
)

# Print précision ± confiance
print_accuracy_with_pm("Naive Bayes", naive_scores)
print_accuracy_with_pm("SVC", svc_scores)
print_accuracy_with_pm("Decision Tree", cv_scores)
print_accuracy_with_pm("KNN", knn_scores)
```




```
Naive Bayes : 0.7936 ± 4.39%
SVC : 0.8899 ± 4.43%
Decision Tree : 0.7838 ± 6.25%
KNN : 0.6659 ± 4.37%
```

3 Classification {CLAIM, REF} vs CONTEXT (Niveau 2)

On définit nos données d'entraînement pour cette classification de niveau 2 :

```
[ ]: # Copie des données pour le niveau 2
data_lvl2 = dataPrepared.copy()
# Combiner 'scientific_claim' et 'scientific_reference' en une seule colonne
data_lvl2['claim_or_ref'] = data_lvl2.apply(lambda row: 1 if␣
 ↪row['scientific_claim'] == 1 or row['scientific_reference'] == 1 else 0,␣
 ↪axis=1)
y = data_lvl2['claim_or_ref'] # la colonne cible pour le niveau 2
X_text = data_lvl2['text']

# Vectorisation TF-IDF + Scaling =========
vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=5, max_df=0.9)
X_vectorized = vectorizer.fit_transform(X_text)

scaler = MaxAbsScaler()
X_scaled = scaler.fit_transform(X_vectorized)
```

```
# Split train/test =========
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42, stratify=y)

print(f"sizes of the classes before resampling : {Counter(y_train)}")

X_resampled, y_resampled = resampleData(X_train, y_train)

print(f"sizes of the classes after resampling : {Counter(y_resampled)}")
```

```
sizes of the classes before resampling : Counter({0: 638, 1: 273})
sizes of the classes after resampling : Counter({0: 634, 1: 634})
```

##**Recherche des paramètres optimaux des classifieurs**

[ ]:
```
perform_gridsearch_and_plot(X_resampled, y_resampled, X_test, y_test, 2,␣
↪"dataSet/lvl2_parameters.json")
```

```
Loaded best parameters from dataSet/lvl2_parameters.json
Best parameters for DecisionTree (Level 2): {'criterion': 'entropy',
'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 20}
Best parameters for MultinomialNB (Level 2): {'alpha': 0.0026560877829466868,
'fit_prior': True}
Best parameters for KNN (Level 2): {'metric': 'manhattan', 'n_neighbors': 8,
'weights': 'distance'}
Best parameters for SVC (Level 2): {'C': 17.47528400007683, 'gamma': 'scale',
'kernel': 'rbf'}
```

[ ]:
```
lvl2_best_params = load_models_from_file("dataSet/lvl2_parameters.json")

print(lvl2_best_params)
```

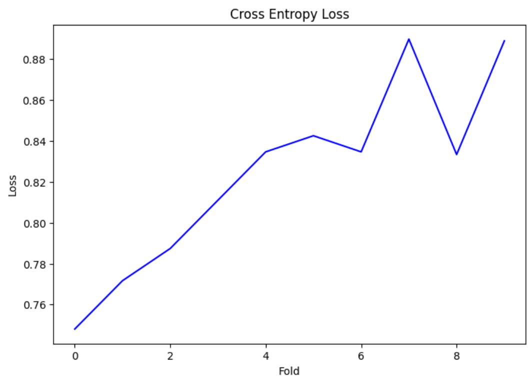
```
{'DecisionTree': DecisionTreeClassifier(criterion='entropy',
min_samples_split=20,
                       random_state=42), 'MultinomialNB':
MultinomialNB(alpha=0.0026560877829466868), 'KNN':
KNeighborsClassifier(metric='manhattan', n_neighbors=8, weights='distance'),
'SVC': SVC(C=17.47528400007683, random_state=42)}
```

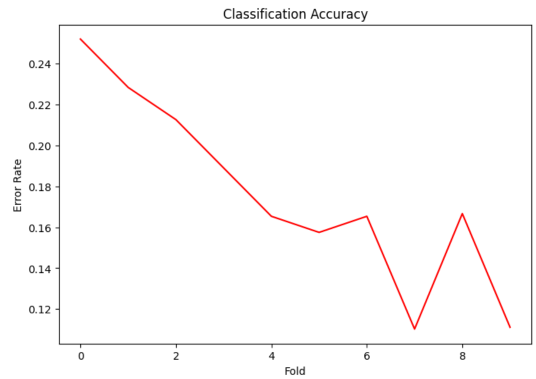
##Decision Tree

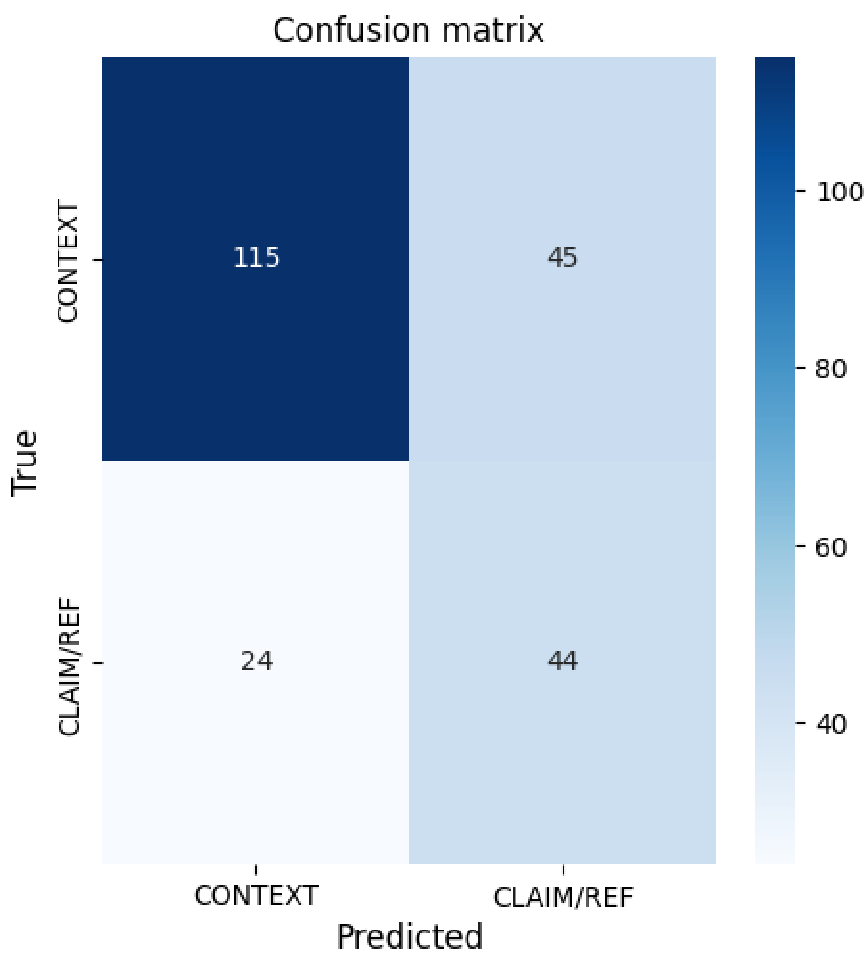
[ ]:
```
best_tree = lvl2_best_params["DecisionTree"]


print("Paramètres :", best_tree.get_params())

best_tree.fit(X_resampled, y_resampled)
```

```python
# Cross-validation
cv_scores = cross_val_score(best_tree, X_resampled, y_resampled, cv=10)
print("Scores CV :", cv_scores)
print("Moyenne CV :", cv_scores.mean())

# Prédiction
y_pred = best_tree.predict(X_test)
print("\n Accuracy (test) :", accuracy_score(y_test, y_pred))
print("Classification Report (test) :")
print(classification_report(y_test, y_pred))


conf_matrix = confusion_matrix(y_test, y_pred)


plot_curves(cv_scores)
plot_curves_confusion(conf_matrix, ['CONTEXT', 'CLAIM/REF'])
```

```
Paramètres : {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy',
'max_depth': None, 'max_features': None, 'max_leaf_nodes': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 20,
'min_weight_fraction_leaf': 0.0, 'monotonic_cst': None, 'random_state': 42,
'splitter': 'best'}
Scores CV : [0.7480315  0.77165354 0.78740157 0.81102362 0.83464567 0.84251969
 0.83464567 0.88976378 0.83333333 0.88888889]
Moyenne CV : 0.8241907261592301

 Accuracy (test) : 0.6973684210526315
Classification Report (test) :
              precision    recall  f1-score   support

           0       0.83      0.72      0.77       160
           1       0.49      0.65      0.56        68

    accuracy                           0.70       228
   macro avg       0.66      0.68      0.66       228
weighted avg       0.73      0.70      0.71       228
```


Cross Entropy Loss
Loss
Fold
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0
2
4
6
8


Classification Accuracy
Error Rate
Fold
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0
2
4
6
8

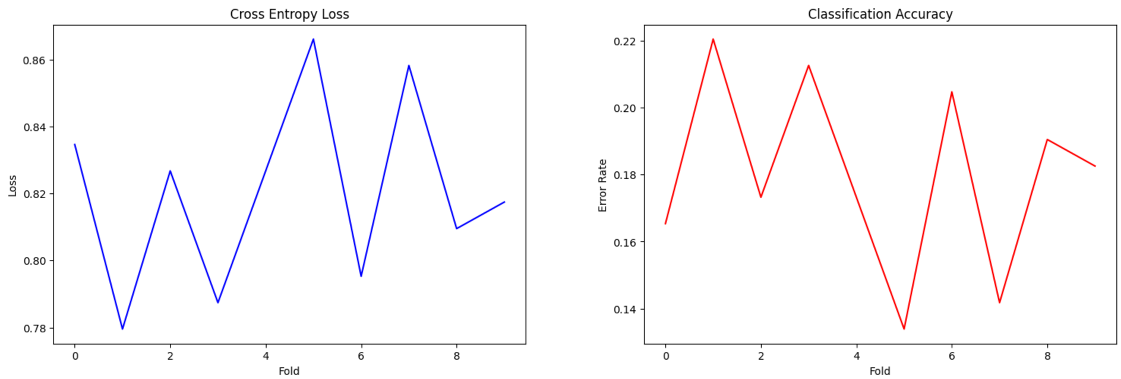

Confusion matrix
True
Predicted
CONTEXT
CLAIM/REF
CONTEXT
CLAIM/REF
115
45
24
44
100
80
60
40

##Naive bayes

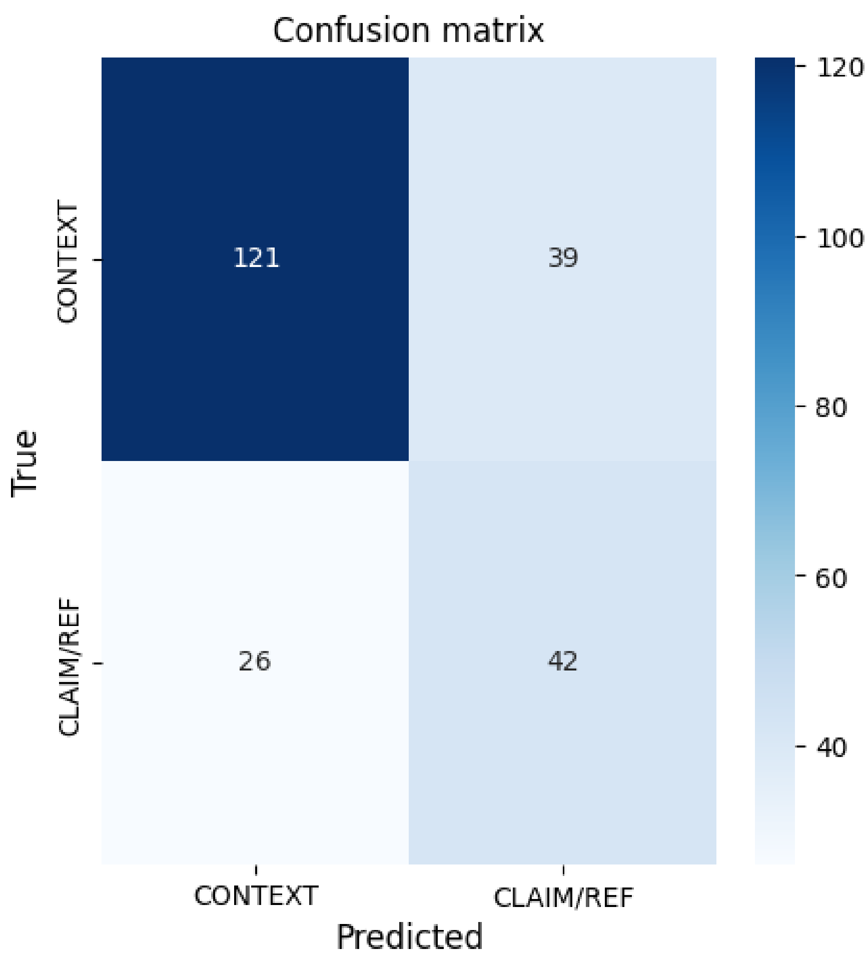
[ ]:

```python
best_naive_bayes_classifier = lvl2_best_params["MultinomialNB"]

print("Paramètres :", best_naive_bayes_classifier.get_params())

best_naive_bayes_classifier.fit(X_resampled, y_resampled)

naive_scores = cross_val_score(best_naive_bayes_classifier, X_resampled,␣
 ↪y_resampled, cv=10)
print("Scores CV :", naive_scores)
print("Moyenne CV :", naive_scores.mean())


y_pred_test = best_naive_bayes_classifier.predict(X_test)


print("\n Accuracy (test) :", accuracy_score(y_test, y_pred_test))
print("Classification Report (test) :")
print(classification_report(y_test, y_pred_test))


conf_matrix = confusion_matrix(y_test, y_pred_test)

plot_curves(naive_scores)
plot_curves_confusion(conf_matrix, ['CONTEXT', 'CLAIM/REF']) # Updated labels␣
 ↪for level 2
```

```
Paramètres : {'alpha': 0.0026560877829466868, 'class_prior': None, 'fit_prior':
True, 'force_alpha': True}
Scores CV : [0.83464567 0.77952756 0.82677165 0.78740157 0.82677165 0.86614173
 0.79527559 0.85826772 0.80952381 0.81746032]
Moyenne CV : 0.8201787276590427

 Accuracy (test) : 0.7149122807017544
Classification Report (test) :
              precision    recall  f1-score   support

           0       0.82      0.76      0.79       160
           1       0.52      0.62      0.56        68

    accuracy                           0.71       228
   macro avg       0.67      0.69      0.68       228
weighted avg       0.73      0.71      0.72       228
```



Cross Entropy Loss
Loss
Fold
0.86
0.84
0.82
0.80
0.78
0
2
4
6
8
Classification Accuracy
Error Rate
Fold
0.22
0.20
0.18
0.16
0.14



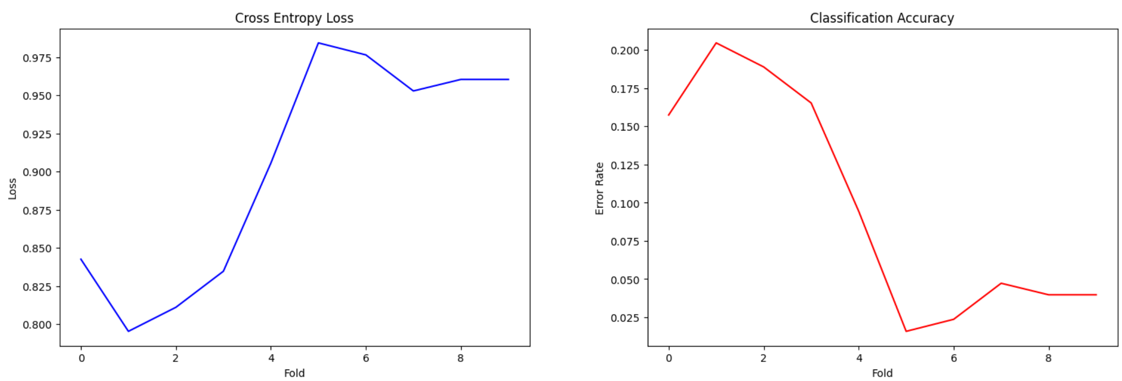
Confusion matrix
True
Predicted
CONTEXT
CLAIM/REF
121
39
26
42
CONTEXT
CLAIM/REF
120
100
80
60
40

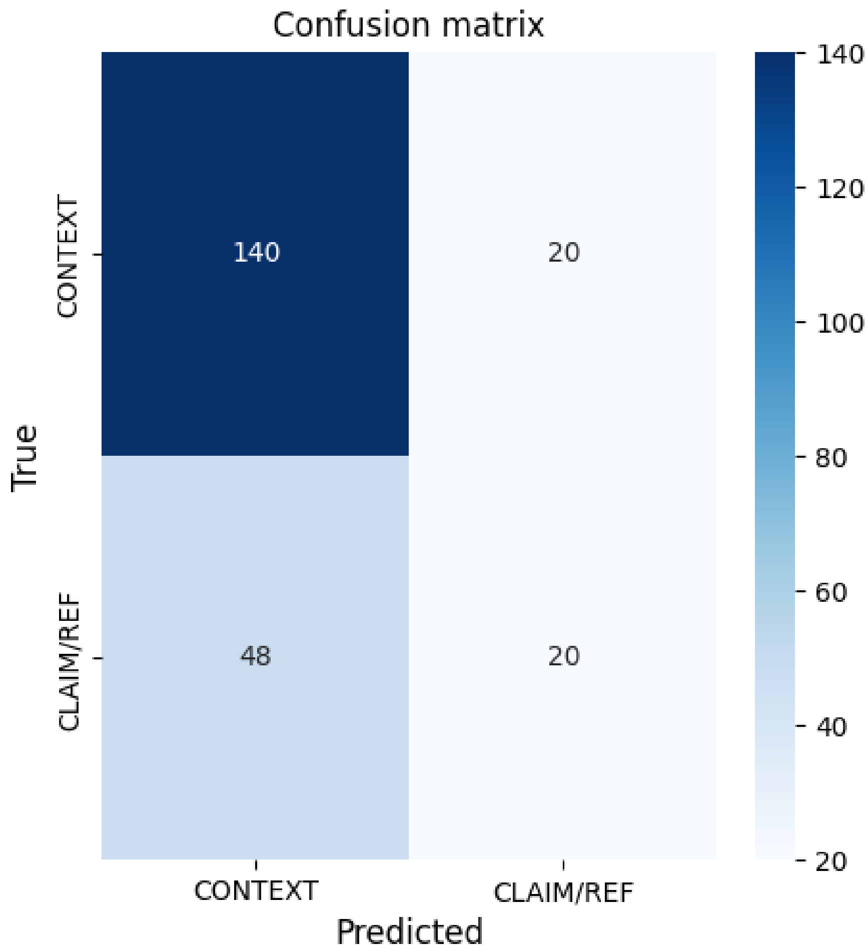
##SVC

[ ]:

```python
# --- Step 1: RESAMPLE THE TRAINING DATA TO BALANCE CLASSES ---

# Optional: use imblearn if your resampleData() doesn't work well
# ros = RandomOverSampler(random_state=42)
# X_resampled, y_resampled = ros.fit_resample(X_train, y_train)

# Convert to dense format if sparse
X_resampled_dense = X_resampled.toarray() if hasattr(X_resampled, "toarray")␣
↪else X_resampled
X_test_dense = X_test.toarray() if hasattr(X_test, "toarray") else X_test

# --- Step 2: SCALE BOTH TRAIN AND TEST DATA USING SAME SCALER ---
scaler = StandardScaler()
X_resampled_scaled = scaler.fit_transform(X_resampled_dense)  # fit on train
X_test_scaled = scaler.transform(X_test_dense)  # transform test with same␣
↪scaler

# --- Step 3: HYPERPARAMETER SEARCH FOR SVC ---




# --- Step 4: VALIDATE WITH CROSS-VAL ON TRAIN ---

best_svc_classifier = lvl2_best_params["SVC"]

print("Paramètres :", best_svc_classifier.get_params())

best_svc_classifier.fit(X_resampled_scaled, y_resampled)


svc_scores = cross_val_score(best_svc_classifier, X_resampled_scaled,␣
↪y_resampled, cv=10)
print("Cross-val scores:", svc_scores)
print("Mean:", svc_scores.mean())

# --- Step 5: EVALUATE ON TEST SET ---

y_pred_test = best_svc_classifier.predict(X_test_scaled)

print("\nAccuracy (test):", accuracy_score(y_test, y_pred_test))
print("Classification Report (test):")
print(classification_report(y_test, y_pred_test))

conf_matrix = confusion_matrix(y_test, y_pred_test)
plot_curves(svc_scores)
plot_curves_confusion(conf_matrix, ['CONTEXT', 'CLAIM/REF'])
```

```
Paramètres : {'C': 17.47528400007683, 'break_ties': False, 'cache_size': 200,
'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree':
3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False,
'random_state': 42, 'shrinking': True, 'tol': 0.001, 'verbose': False}
Cross-val scores: [0.84251969 0.79527559 0.81102362 0.83464567 0.90551181
0.98425197
 0.97637795 0.95275591 0.96031746 0.96031746]
Mean: 0.902299712535933

Accuracy (test): 0.7017543859649122
Classification Report (test):
              precision    recall  f1-score   support

           0       0.74      0.88      0.80       160
           1       0.50      0.29      0.37        68

    accuracy                           0.70       228
   macro avg       0.62      0.58      0.59       228
weighted avg       0.67      0.70      0.68       228
```



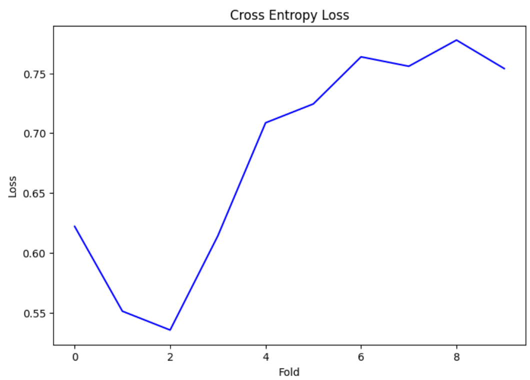


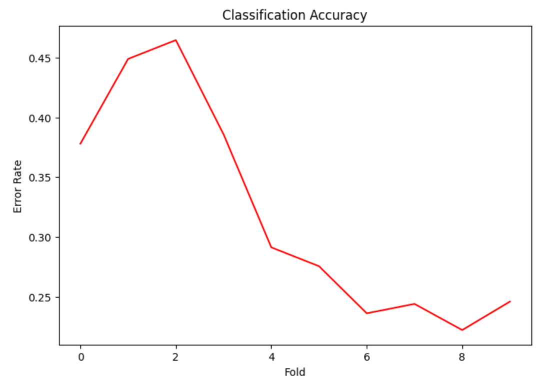
##KNN

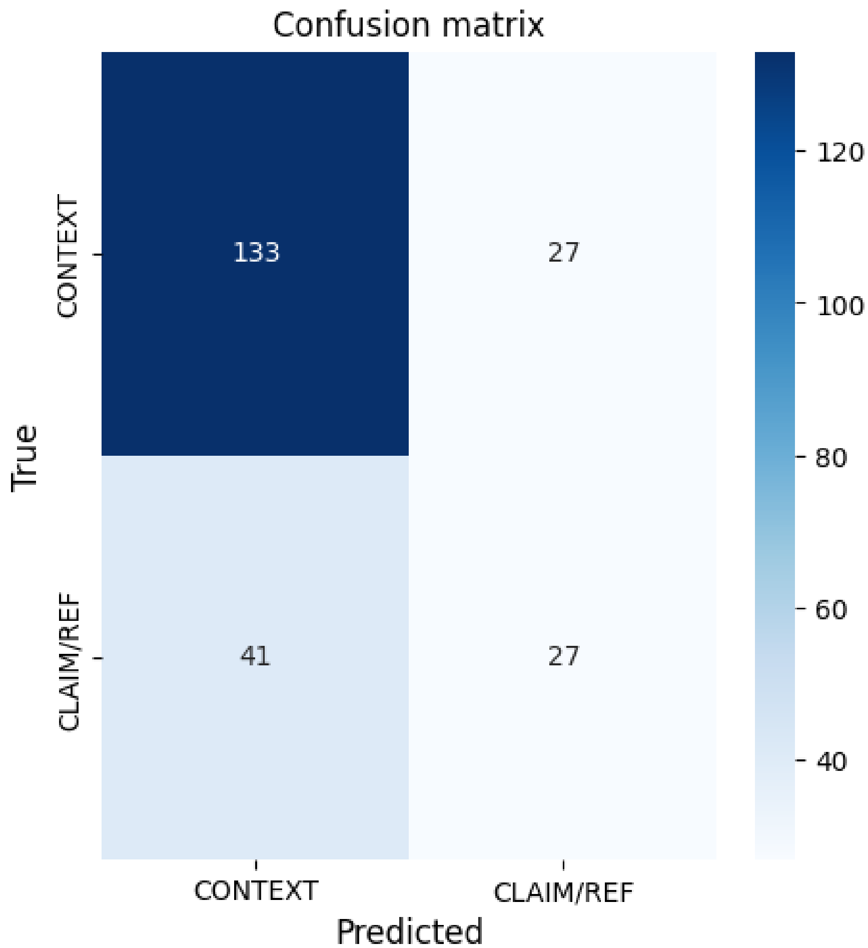
```
[ ]: best_knn_classifier = lvl2_best_params["KNN"]

print("Paramètres :", best_knn_classifier.get_params())

best_knn_classifier.fit(X_resampled, y_resampled)

knn_scores = cross_val_score(best_knn_classifier, X_resampled, y_resampled,␣
  ↪cv=10)
print("Scores de validation croisée :", knn_scores)
print("Moyenne :", knn_scores.mean())


y_pred_test = best_knn_classifier.predict(X_test_scaled)
print("\n Accuracy (test) :", accuracy_score(y_test, y_pred_test))
```

```python
print("Rapport de classification :\n", classification_report(y_test,␣
 ↪y_pred_test))


conf_matrix = confusion_matrix(y_test, y_pred_test)

plot_curves(knn_scores)
plot_curves_confusion(conf_matrix, ['CONTEXT', 'CLAIM/REF'])
```

```
Paramètres : {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'manhattan',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 8, 'p': 2, 'weights':
'distance'}
Scores de validation croisée : [0.62204724 0.5511811  0.53543307 0.61417323
0.70866142 0.72440945
 0.76377953 0.75590551 0.77777778 0.75396825]
Moyenne : 0.6807336582927135

 Accuracy (test) : 0.7017543859649122
Rapport de classification :
               precision    recall  f1-score   support

           0       0.76      0.83      0.80       160
           1       0.50      0.40      0.44        68

    accuracy                           0.70       228
   macro avg       0.63      0.61      0.62       228
weighted avg       0.69      0.70      0.69       228
```

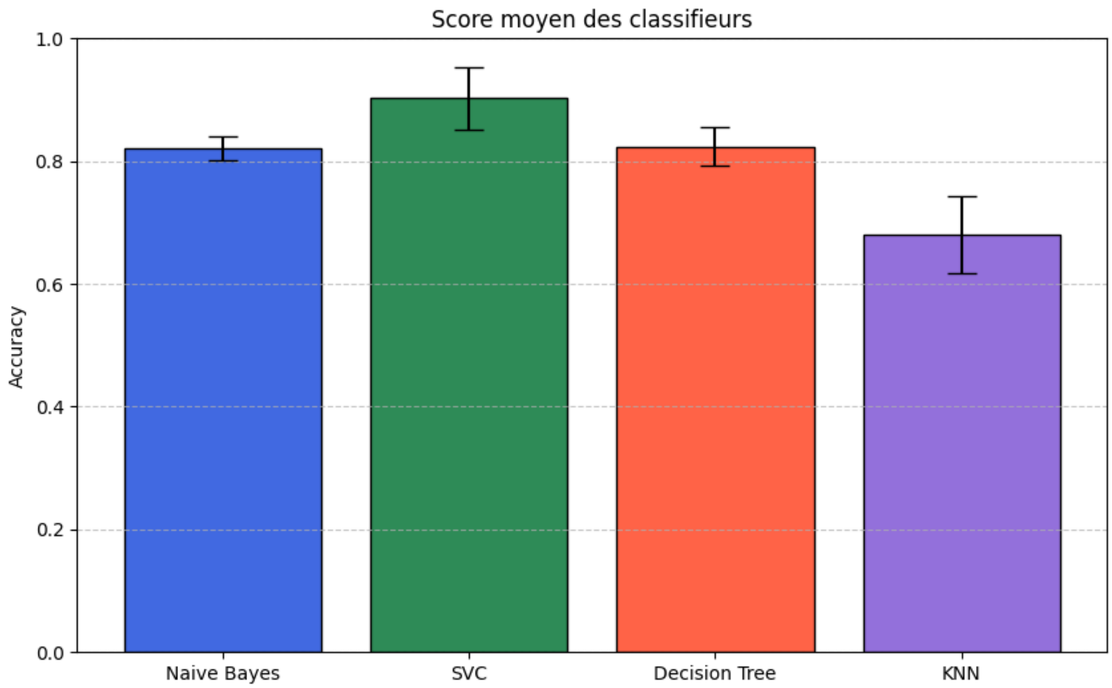






3.1 Evaluation des classifieurs (niveau 2)

[ ]:
```
#Evaluation des classifieurs

# Plot courbes
plot_curves_results_with_ci(
    [naive_scores, svc_scores, cv_scores, knn_scores],
    names=['Naive Bayes', 'SVC', 'Decision Tree', 'KNN']
)

# Print précision ± confiance
print_accuracy_with_pm("Naive Bayes", naive_scores)
print_accuracy_with_pm("SVC", svc_scores)
print_accuracy_with_pm("Decision Tree", cv_scores)
print_accuracy_with_pm("KNN", knn_scores)
```




```
Naive Bayes : 0.8202 ± 2.36%
SVC : 0.9023 ± 5.57%
Decision Tree : 0.8242 ± 3.80%
KNN : 0.6807 ± 9.15%
```

4 Classification {CLAIM} VS {REF} VS {CONTEXT} (niveau 3)

On définit nos données d'entraînement pour cette classification de niveau 3 :

[ ]:
```
data_lvl3 = dataPrepared.copy()


def get_level3_label(row):
    if row['scientific_claim'] == 1:
        return 'CLAIM'
    elif row['scientific_reference'] == 1:
        return 'REF'
    elif row['scientific_context'] == 1:
        return 'CONTEXT'
    else:
        return 'NON-SCI'

def apply_level3_label(data):
```

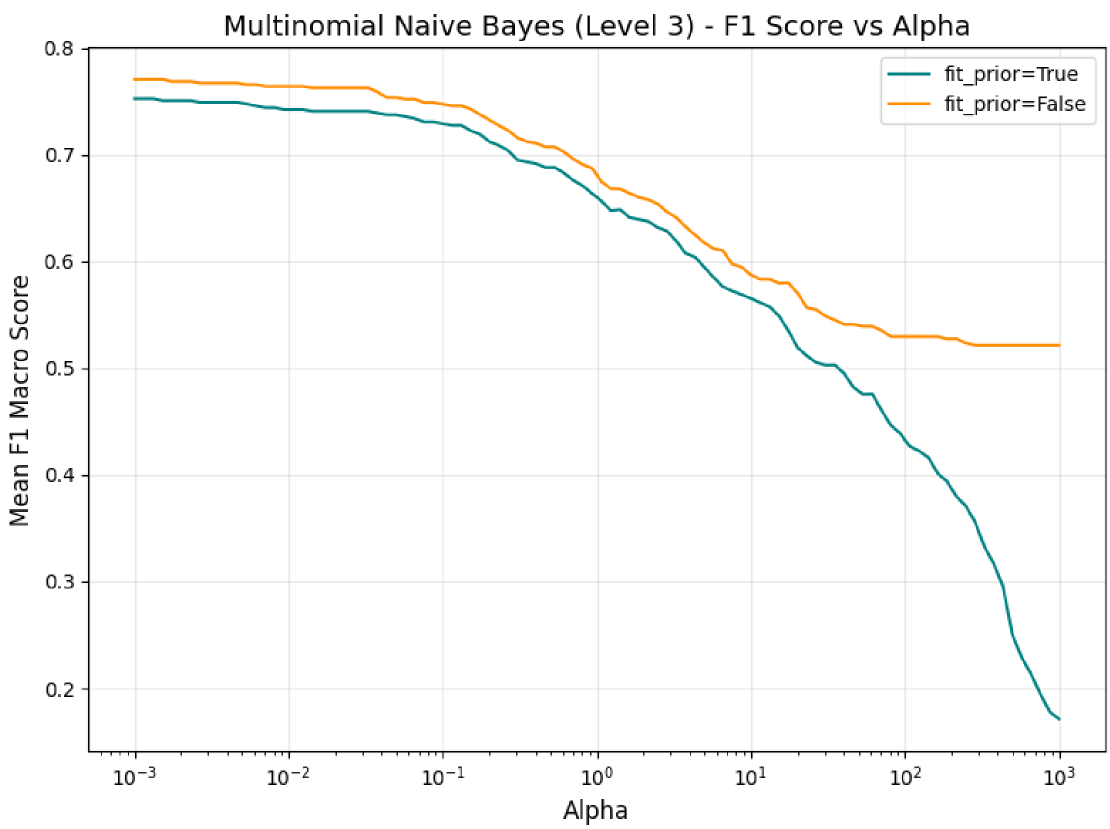
```
    data['level3_label'] = data.apply(get_level3_label, axis=1)
    # drop all non sci
    data = data[data['level3_label'] != 'NON-SCI']
    return data

data_lvl3 = apply_level3_label(data_lvl3)
y = data_lvl3['level3_label']
X_text = data_lvl3['text']

vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=5, max_df=0.9)
X_vectorized = vectorizer.fit_transform(X_text)

scaler = MaxAbsScaler()
X_scaled = scaler.fit_transform(X_vectorized)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

print(f"sizes of the classes before resampling : {Counter(y_train)}")

X_resampled, y_resampled = resampleData(X_train, y_train)

print(f"sizes of the classes after resampling : {Counter(y_resampled)}")
```

```
sizes of the classes before resampling : Counter({'CLAIM': 210, 'REF': 62,
'CONTEXT': 27})
sizes of the classes after resampling : Counter({'CONTEXT': 210, 'CLAIM': 207,
'REF': 207})
```

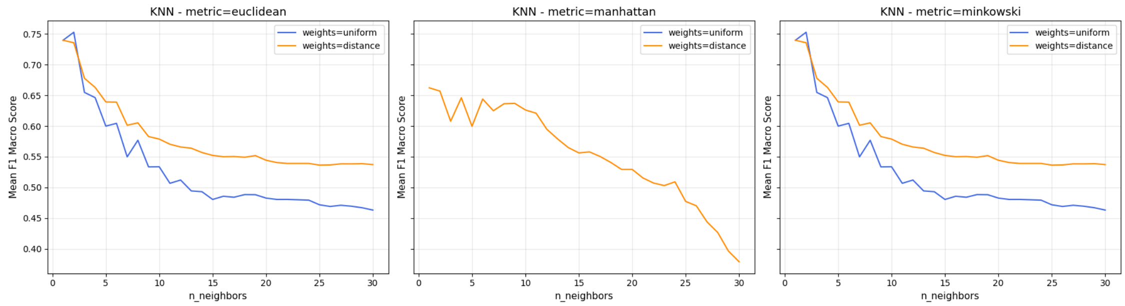
##**Recherche des paramètres optimaux des classifieurs**

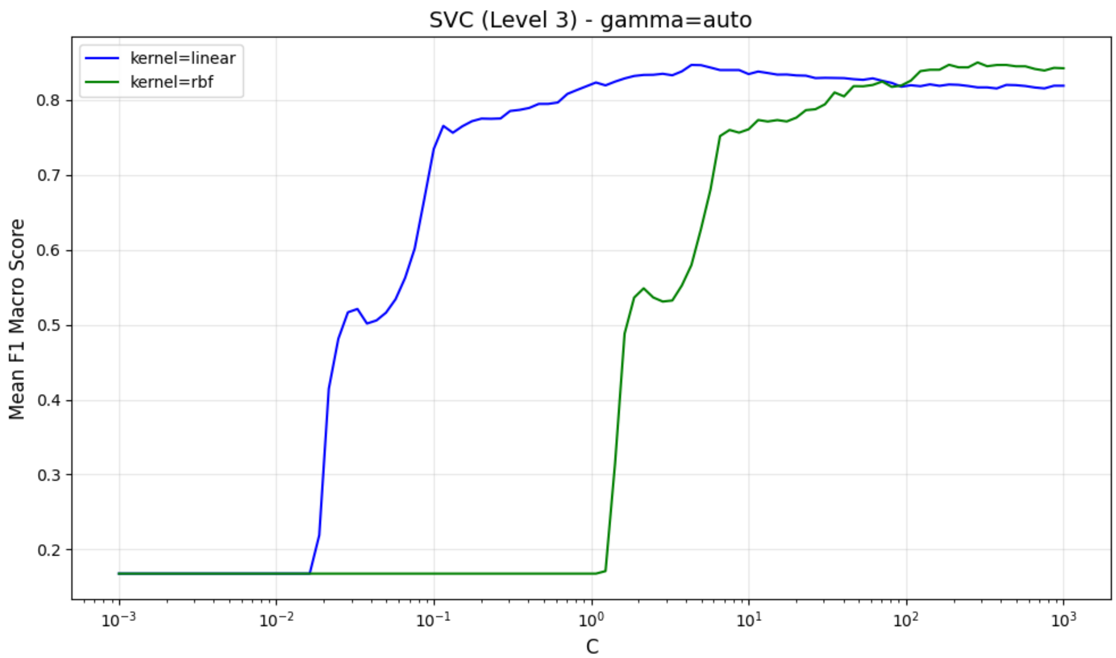
[ ]:
```
perform_gridsearch_and_plot(X_resampled, y_resampled, X_test, y_test, 3,␣
 ↪"dataSet/lvl3_parameters.json")
```

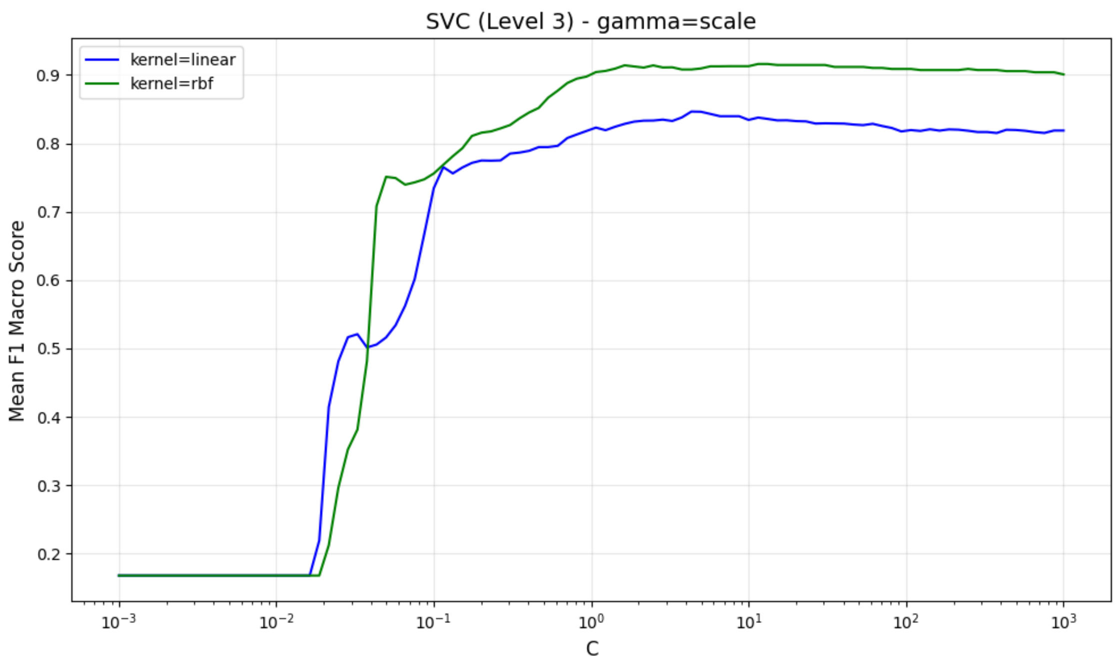
Decision Tree (Level 3) - Mean F1 Macro Score Comparison
Gini
Entropy
Mean F1 Macro Score
Parameter Combination Index
0.75
0.70
0.65
0.60
0.55
0
50
100
150
200
250
300
350



Multinomial Naive Bayes (Level 3) - F1 Score vs Alpha
fit_prior=True
fit_prior=False
Mean F1 Macro Score
Alpha
0.8
0.7
0.6
0.5
0.4
0.3
0.2
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}


KNN - metric=euclidean
KNN - metric=manhattan
KNN - metric=minkowski
weights=uniform
weights=distance
Mean F1 Macro Score
n_neighbors


SVC (Level 3) - gamma=auto
kernel=linear
kernel=rbf
Mean F1 Macro Score
C




```
Saved best parameters to dataSet/lvl3_parameters.json
Best parameters for DecisionTree (Level 3): {'criterion': 'entropy',
'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best parameters for MultinomialNB (Level 3): {'alpha': 0.001, 'fit_prior':
False}
Best parameters for KNN (Level 3): {'metric': 'euclidean', 'n_neighbors': 2,
'weights': 'uniform'}
Best parameters for SVC (Level 3): {'C': 11.497569953977356, 'gamma': 'scale',
'kernel': 'rbf'}
```

[ ]:
```
lvl3_best_params = load_models_from_file("dataSet/lvl3_parameters.json")

print(lvl3_best_params)
```

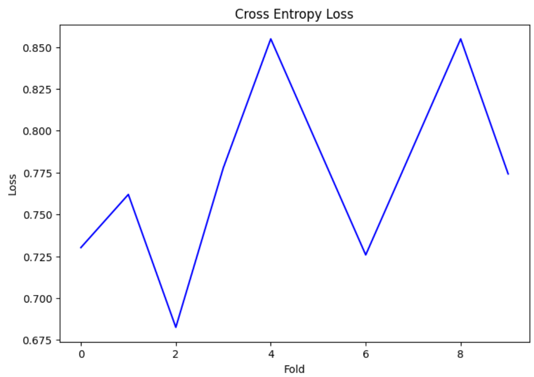
```
{'DecisionTree': DecisionTreeClassifier(criterion='entropy', random_state=42),
'MultinomialNB': MultinomialNB(alpha=0.001, fit_prior=False), 'KNN':
KNeighborsClassifier(metric='euclidean', n_neighbors=2), 'SVC':
SVC(C=11.497569953977356, random_state=42)}
```

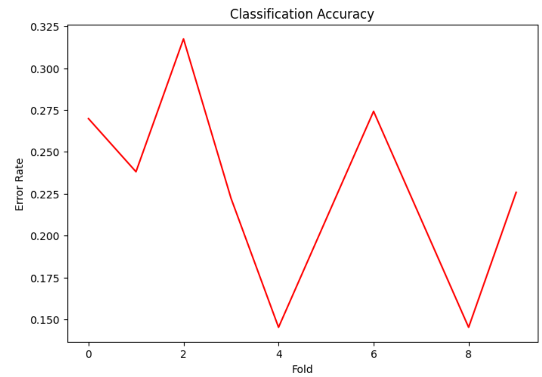
4.1 Decision tree

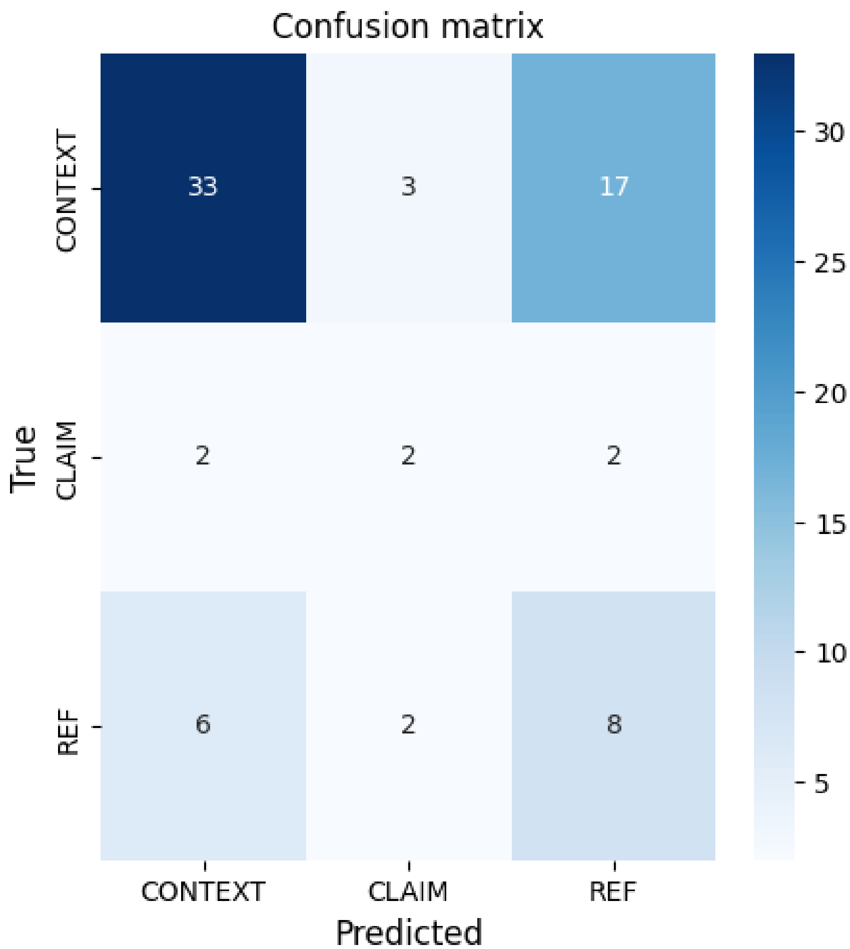
[ ]:
```
best_tree = lvl3_best_params["DecisionTree"]

print("Paramètres :", best_tree.get_params())

best_tree.fit(X_resampled, y_resampled)
```

```python
# Cross-validation
cv_scores = cross_val_score(best_tree, X_resampled, y_resampled, cv=10)
print("Scores CV :", cv_scores)
print("Moyenne CV :", cv_scores.mean())

# Prediction
y_pred = best_tree.predict(X_test)
print("\n Accuracy (test) :", accuracy_score(y_test, y_pred))
print("Classification Report (test) :")
print(classification_report(y_test, y_pred))

conf_matrix = confusion_matrix(y_test, y_pred)

plot_curves(cv_scores)
plot_curves_confusion(conf_matrix, ['CONTEXT', 'CLAIM', 'REF'])
```

```
Paramètres : {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy',
'max_depth': None, 'max_features': None, 'max_leaf_nodes': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'monotonic_cst': None, 'random_state': 42,
'splitter': 'best'}
Scores CV : [0.73015873 0.76190476 0.68253968 0.77777778 0.85483871 0.79032258
 0.72580645 0.79032258 0.85483871 0.77419355]
Moyenne CV : 0.7742703533026114

 Accuracy (test) : 0.5733333333333334
Classification Report (test) :
              precision    recall  f1-score   support

       CLAIM       0.80      0.62      0.70        53
     CONTEXT       0.29      0.33      0.31         6
         REF       0.30      0.50      0.37        16

    accuracy                           0.57        75
   macro avg       0.46      0.49      0.46        75
weighted avg       0.65      0.57      0.60        75
```


Cross Entropy Loss
Loss
Fold

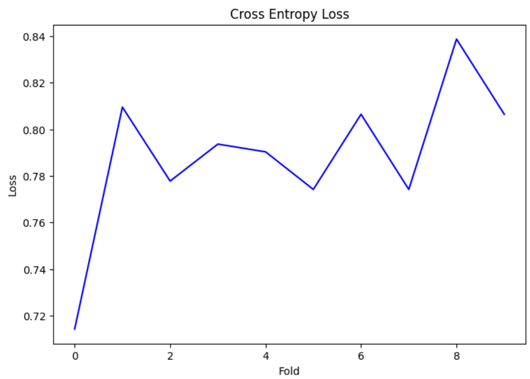

Classification Accuracy
Error Rate
Fold

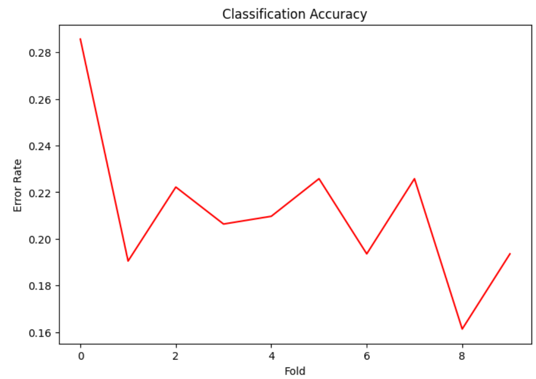

Confusion matrix
True
Predicted
CONTEXT
CLAIM
REF
33
3
17
2
2
2
6
2
8

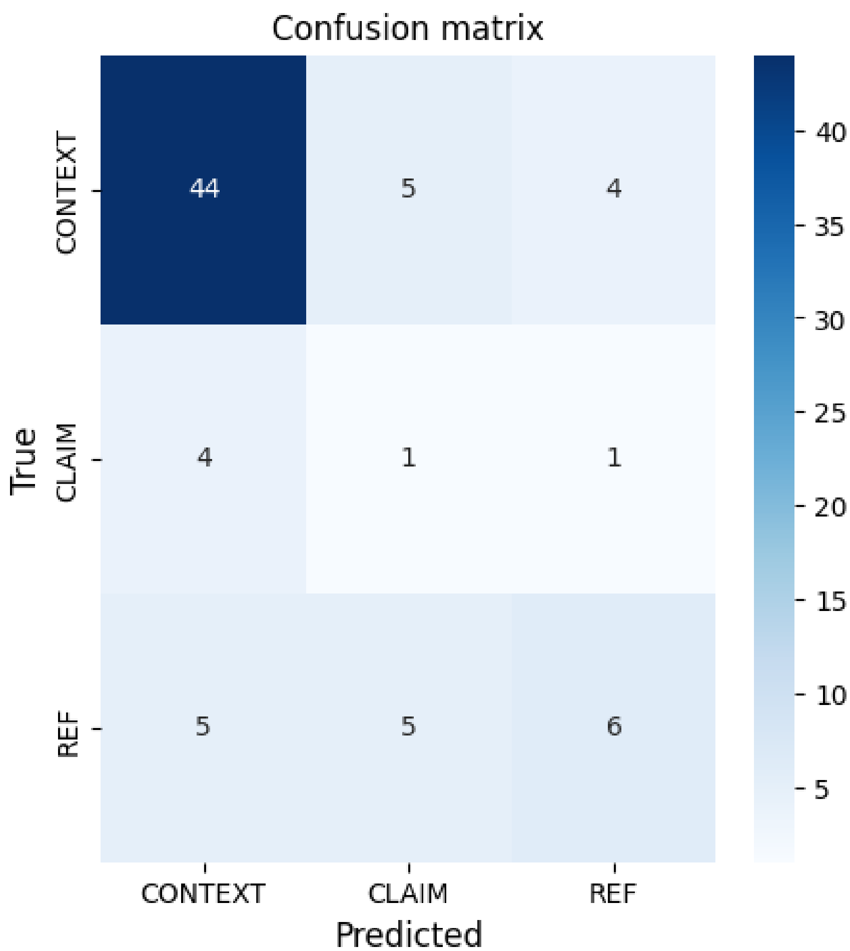
4.2 Naive bayes

[ ]:
```
best_naive_bayes_classifier = lvl3_best_params["MultinomialNB"]

print("Paramètres :", best_naive_bayes_classifier.get_params())

best_naive_bayes_classifier.fit(X_resampled, y_resampled)

naive_scores = cross_val_score(best_naive_bayes_classifier, X_resampled,␣
 ↪y_resampled, cv=10)
print("Scores CV :", naive_scores)
print("Moyenne CV :", naive_scores.mean())

y_pred_test = best_naive_bayes_classifier.predict(X_test)

print("\n Accuracy (test) :", accuracy_score(y_test, y_pred_test))
print("Classification Report (test) :")
print(classification_report(y_test, y_pred_test))

conf_matrix = confusion_matrix(y_test, y_pred_test)

plot_curves(naive_scores)
plot_curves_confusion(conf_matrix, ['CONTEXT', 'CLAIM', 'REF'])
```

```
Paramètres : {'alpha': 0.001, 'class_prior': None, 'fit_prior': False,
'force_alpha': True}
Scores CV : [0.71428571 0.80952381 0.77777778 0.79365079 0.79032258 0.77419355
 0.80645161 0.77419355 0.83870968 0.80645161]
Moyenne CV : 0.7885560675883256

 Accuracy (test) : 0.68
Classification Report (test) :
              precision    recall  f1-score   support

       CLAIM       0.83      0.83      0.83        53
     CONTEXT       0.09      0.17      0.12         6
         REF       0.55      0.38      0.44        16

    accuracy                           0.68        75
   macro avg       0.49      0.46      0.46        75
weighted avg       0.71      0.68      0.69        75
```


Cross Entropy Loss
Loss
Fold


Classification Accuracy
Error Rate
Fold

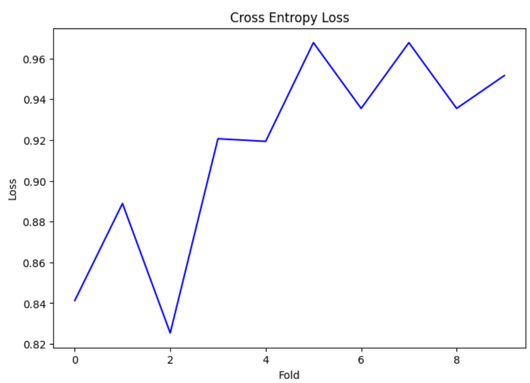

Confusion matrix
True
Predicted
CONTEXT
CLAIM
REF

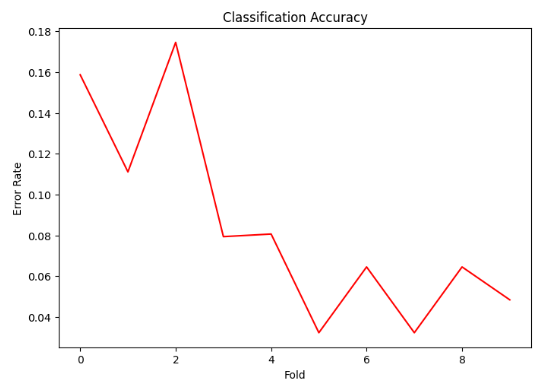
4.3 SVC

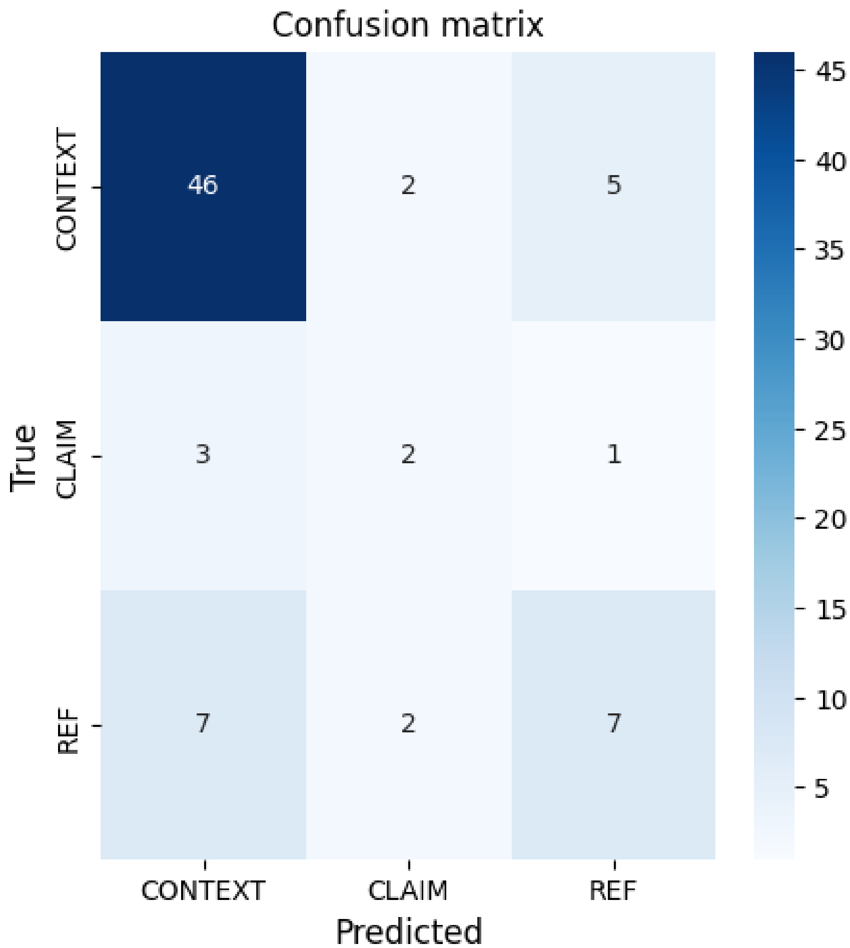
[ ]:
```
# ##SVC

X_resampled, y_resampled = resampleData(X_train, y_train)
print("Resampled class distribution:", Counter(y_resampled))

X_resampled_dense = X_resampled.toarray() if hasattr(X_resampled, "toarray")␣
↪else X_resampled
X_test_dense = X_test.toarray() if hasattr(X_test, "toarray") else X_test

scaler = StandardScaler()
X_resampled_scaled = scaler.fit_transform(X_resampled_dense)
X_test_scaled = scaler.transform(X_test_dense)
clf_SVC = SVC()



best_svc_classifier = lvl3_best_params["SVC"]

print("Paramètres :", best_svc_classifier.get_params())

best_svc_classifier.fit(X_resampled_scaled, y_resampled)

svc_scores = cross_val_score(best_svc_classifier, X_resampled_scaled,␣
↪y_resampled, cv=10)
print("Cross-val scores:", svc_scores)
print("Mean:", svc_scores.mean())

# --- Step 5: EVALUATE ON TEST SET ---

y_pred_test = best_svc_classifier.predict(X_test_scaled)

print("\nAccuracy (test):", accuracy_score(y_test, y_pred_test))
print("Classification Report (test):")
print(classification_report(y_test, y_pred_test))

conf_matrix = confusion_matrix(y_test, y_pred_test)
plot_curves(svc_scores)
plot_curves_confusion(conf_matrix, ['CONTEXT', 'CLAIM', 'REF'])
```

```
Resampled class distribution: Counter({'CONTEXT': 210, 'CLAIM': 207, 'REF':
207})
Paramètres : {'C': 11.497569953977356, 'break_ties': False, 'cache_size': 200,
'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree':
3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False,
'random_state': 42, 'shrinking': True, 'tol': 0.001, 'verbose': False}
Cross-val scores: [0.84126984 0.88888889 0.82539683 0.92063492 0.91935484
```

```
0.96774194
 0.93548387 0.96774194 0.93548387 0.9516129 ]
Mean: 0.9153609831029186

Accuracy (test): 0.7333333333333333
Classification Report (test):
              precision    recall  f1-score   support

       CLAIM       0.82      0.87      0.84        53
     CONTEXT       0.33      0.33      0.33         6
         REF       0.54      0.44      0.48        16

    accuracy                           0.73        75
   macro avg       0.56      0.55      0.55        75
weighted avg       0.72      0.73      0.73        75
```





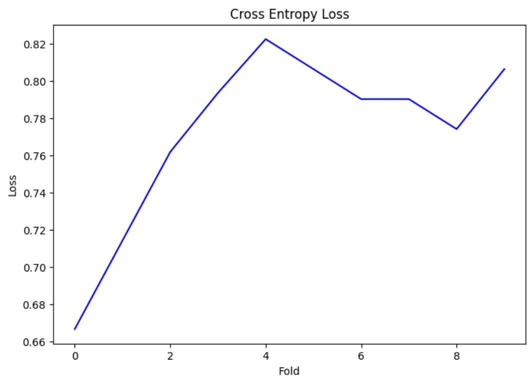


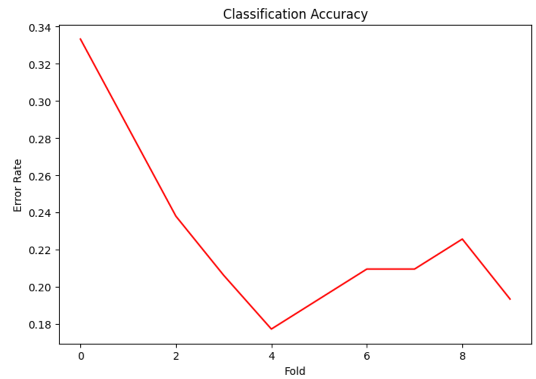
4.4 KNN

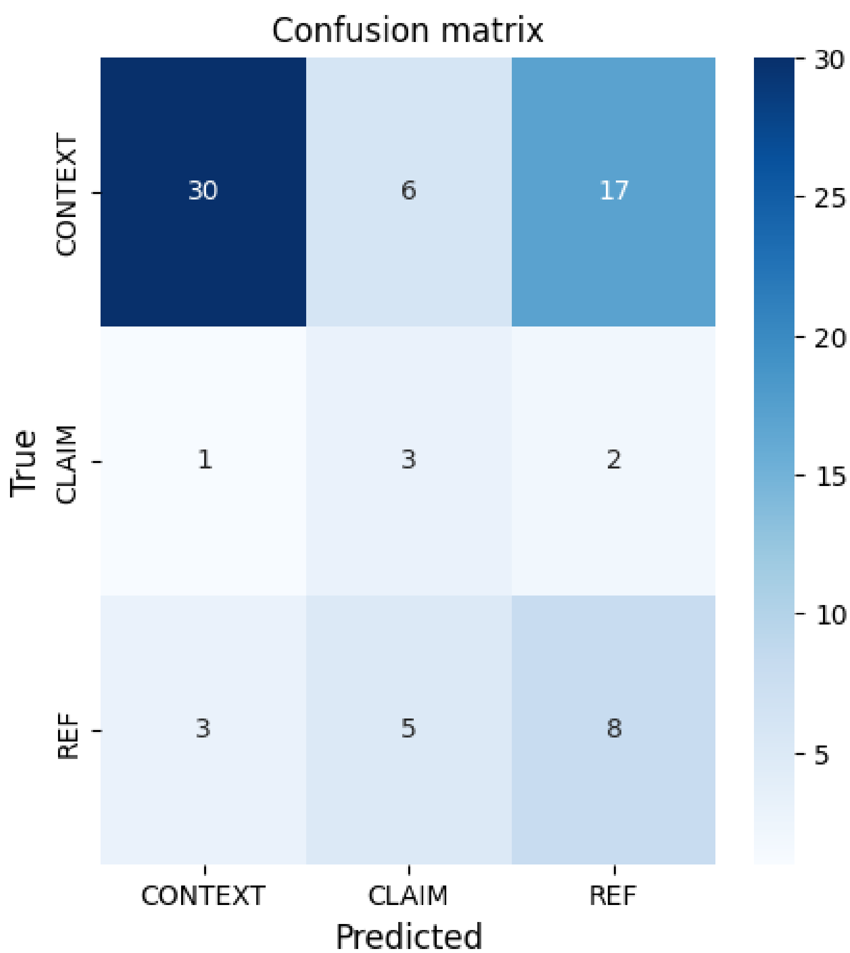
```
[ ]: from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV, KFold, cross_val_score,␣
 ↪cross_val_predict, train_test_split
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
from sklearn.preprocessing import StandardScaler


# Assuming X_train, X_test, y_train, y_test from previous level 3 setup
X_resampled, y_resampled = resampleData(X_train, y_train)  # use yours
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.toarray())
X_test_scaled = scaler.transform(X_test.toarray())
```

```python
best_knn_classifier = lvl3_best_params["KNN"]

print("Paramètres :", best_knn_classifier.get_params())

best_knn_classifier.fit(X_resampled, y_resampled)

knn_scores = cross_val_score(best_knn_classifier, X_resampled, y_resampled,␣
 ↪cv=10)
print("Scores de validation croisée :", knn_scores)
print("Moyenne :", knn_scores.mean())

y_pred_test = best_knn_classifier.predict(X_test)
print("\n Accuracy (test) :", accuracy_score(y_test, y_pred_test))
print("Rapport de classification :\n", classification_report(y_test,␣
 ↪y_pred_test))

conf_matrix = confusion_matrix(y_test, y_pred_test)

plot_curves(knn_scores)
plot_curves_confusion(conf_matrix, ['CONTEXT', 'CLAIM', 'REF'])
```

```
Paramètres : {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'euclidean',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 2, 'p': 2, 'weights':
'uniform'}
Scores de validation croisée : [0.66666667 0.71428571 0.76190476 0.79365079
0.82258065 0.80645161
 0.79032258 0.79032258 0.77419355 0.80645161]
Moyenne : 0.7726830517153098

 Accuracy (test) : 0.5466666666666666
Rapport de classification :
               precision    recall  f1-score   support

       CLAIM       0.88      0.57      0.69        53
     CONTEXT       0.21      0.50      0.30         6
         REF       0.30      0.50      0.37        16

    accuracy                           0.55        75
   macro avg       0.46      0.52      0.45        75
weighted avg       0.70      0.55      0.59        75
```

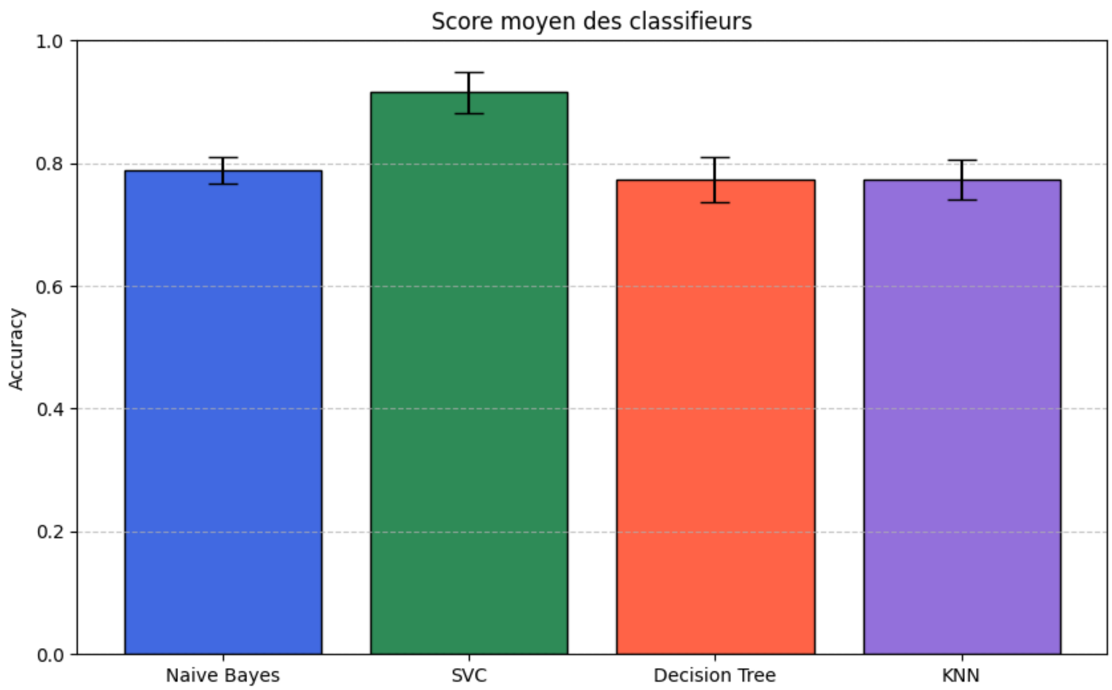

Cross Entropy Loss
Loss
0.82
0.80
0.78
0.76
0.74
0.72
0.70
0.68
0.66
0
2
4
6
8
Fold


Classification Accuracy
Error Rate
0.34
0.32
0.30
0.28
0.26
0.24
0.22
0.20
0.18
0
2
4
6
8
Fold


Confusion matrix
True
CONTEXT
CLAIM
REF
30
6
17
1
3
2
3
5
8
CONTEXT
CLAIM
REF
Predicted
30
25
20
15
10
5

4.5 Evaluation des classifieurs (Niveau 3)

[ ]:
```python
# Evaluation des classifieurs

# Plot courbes
plot_curves_results_with_ci(
    [naive_scores, svc_scores, cv_scores, knn_scores],
    names=['Naive Bayes', 'SVC', 'Decision Tree', 'KNN']
)

# Print précision ± confiance
print_accuracy_with_pm("Naive Bayes", naive_scores)
print_accuracy_with_pm("SVC", svc_scores)
print_accuracy_with_pm("Decision Tree", cv_scores)
print_accuracy_with_pm("KNN", knn_scores)
```




```
Naive Bayes : 0.7886 ± 2.82%
SVC : 0.9154 ± 3.66%
Decision Tree : 0.7743 ± 4.74%
KNN : 0.7727 ± 4.20%
```

5 Optimisation #1 : Utiliser un dataSet plus adapté en faisant varier prepareText

Actuellement, quand nous avions lancé notre fonction prepareText, nous avions appliqué tous les paramètres possibles. Cependant, il est possible que nous obtenions un dataSet de meilleure qualité suivant les paramètres appliqués. Pour cela, nous allons faire tourner notre fonction de préparation de données avec chaque combinaison de paramètre (2 possibilités) et par la suite, nous utiliserons des outils statistiques afin de déterminer quel est le meilleur dataSet à utiliser.

Comme précédement, on va stocker les prétraitements sous forme de fichier afin de gagner du temps lors des réexécutions.

[ ]:
```python
# Define preprocessing parameter combinations
combinations = {
    "0000": {"keepTokens": False, "keepEmojis": False, "numbersAsTokens":␣
 ↪False, "translate": False},
    "0001": {"keepTokens": False, "keepEmojis": False, "numbersAsTokens":␣
 ↪False, "translate": True},
    "0010": {"keepTokens": False, "keepEmojis": True, "numbersAsTokens": False,␣
 ↪"translate": False},
    "0011": {"keepTokens": False, "keepEmojis": True, "numbersAsTokens": False,␣
 ↪"translate": True},
    "0100": {"keepTokens": False, "keepEmojis": False, "numbersAsTokens": True,␣
 ↪"translate": False},
    "0101": {"keepTokens": False, "keepEmojis": False, "numbersAsTokens": True,␣
 ↪"translate": True},
    "0110": {"keepTokens": False, "keepEmojis": True, "numbersAsTokens": True,␣
 ↪"translate": False},
    "0111": {"keepTokens": False, "keepEmojis": True, "numbersAsTokens": True,␣
 ↪"translate": True},
    "1000": {"keepTokens": True, "keepEmojis": False, "numbersAsTokens": False,␣
 ↪"translate": False},
    "1001": {"keepTokens": True, "keepEmojis": False, "numbersAsTokens": False,␣
 ↪"translate": True},
    "1010": {"keepTokens": True, "keepEmojis": True, "numbersAsTokens": False,␣
 ↪"translate": False},
    "1011": {"keepTokens": True, "keepEmojis": True, "numbersAsTokens": False,␣
 ↪"translate": True},
    "1100": {"keepTokens": True, "keepEmojis": False, "numbersAsTokens": True,␣
 ↪"translate": False},
    "1101": {"keepTokens": True, "keepEmojis": False, "numbersAsTokens": True,␣
 ↪"translate": True},
    "1110": {"keepTokens": True, "keepEmojis": True, "numbersAsTokens": True,␣
 ↪"translate": False},
    "1111": {"keepTokens": True, "keepEmojis": True, "numbersAsTokens": True,␣
 ↪"translate": True}
}
```

```python
# Ensure dataset folder exists
os.makedirs("dataSet", exist_ok=True)

# Dictionary to store all dataframes
dataPrepared = {}

# Process all datasets
for key, params in combinations.items():
    file_path = f"dataSet/precomputed/dataPrepared{key}.csv"

    if os.path.exists(file_path):
        print(f"Loading existing dataset: {file_path}")
        dataPrepared[key] = pd.read_csv(file_path)  # Store in dictionary
    else:
        print(f"Processing and saving: {file_path}")
        dataPrepared[key] = df.copy()
        dataPrepared[key]["text"] = dataPrepared[key]["text"].apply(lambda x: 
 ↪prepareText(x, **params))
        dataPrepared[key].to_csv(file_path, index=False)

print("\nAll 16 datasets are ready and stored in `dataPrepared` dictionary!")
```

```
Loading existing dataset: dataSet/precomputed/dataPrepared0000.csv
Loading existing dataset: dataSet/precomputed/dataPrepared0001.csv
Loading existing dataset: dataSet/precomputed/dataPrepared0010.csv
Loading existing dataset: dataSet/precomputed/dataPrepared0011.csv
Loading existing dataset: dataSet/precomputed/dataPrepared0100.csv
Loading existing dataset: dataSet/precomputed/dataPrepared0101.csv
Loading existing dataset: dataSet/precomputed/dataPrepared0110.csv
Loading existing dataset: dataSet/precomputed/dataPrepared0111.csv
Loading existing dataset: dataSet/precomputed/dataPrepared1000.csv
Loading existing dataset: dataSet/precomputed/dataPrepared1001.csv
Loading existing dataset: dataSet/precomputed/dataPrepared1010.csv
Loading existing dataset: dataSet/precomputed/dataPrepared1011.csv
Loading existing dataset: dataSet/precomputed/dataPrepared1100.csv
Loading existing dataset: dataSet/precomputed/dataPrepared1101.csv
Loading existing dataset: dataSet/precomputed/dataPrepared1110.csv
Loading existing dataset: dataSet/precomputed/dataPrepared1111.csv

All 16 datasets are ready and stored in `dataPrepared` dictionary!
```

On applique la vectorisation sur chaque dataSet :

[ ]:
```python
for key, dataset in dataPrepared.items():
    print(f"Dataset: {key}")
    file_path = f'dataSet/vectorized_{key}.csv'
    if os.path.exists(file_path):
```

```
        print(f"Le fichier {file_path} existe déjà, skipping vectorization for␣
↪this dataset.")
        continue
    # Nettoyage du dataset courant
    dataset = dataset[dataset['text'] != '']
    dataset = dataset.dropna(subset=['text'])
    dataset['text'] = dataset['text'].fillna('').astype(str)
    dataset['text'] = dataset['text'].apply(lemmatize_taggenize_sentence)
    # Traitement avec gestion d'erreur
    processed_text = []
    for index, text in dataset['text'].items():
        if pd.notnull(text) and text.strip() != "":
            try:
                processed_text.append(process_text_column(text))
            except Exception as e:
                print(f"Erreur à l'index {index} avec le texte : {text}")
                print(f"Exception : {e}")
                processed_text.append("")
        else:
            processed_text.append("")
    dataset['text'] = processed_text  # Remplace directement
    # TF-IDF vectorization
    vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=5, max_df=0.9)
    vectorized = vectorizer.fit_transform(dataset['text'])
    # Scaling
    scaled = MaxAbsScaler().fit_transform(vectorized)
    # Conversion + sauvegarde
    vectorized_df = pd.DataFrame(scaled.toarray(), columns=vectorizer.
↪get_feature_names_out())
    vectorized_df.to_csv(file_path, index=False)
    print(f"Vectorized dataset {key} saved.")
    display(vectorized_df.head())
    print("\n" + "-"*50 + "\n")
```

```
Dataset: 0000
Le fichier dataSet/vectorized_0000.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 0001
Le fichier dataSet/vectorized_0001.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 0010
Le fichier dataSet/vectorized_0010.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 0011
Le fichier dataSet/vectorized_0011.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 0100
```

```
Le fichier dataSet/vectorized_0100.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 0101
Le fichier dataSet/vectorized_0101.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 0110
Le fichier dataSet/vectorized_0110.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 0111
Le fichier dataSet/vectorized_0111.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 1000
Le fichier dataSet/vectorized_1000.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 1001
Le fichier dataSet/vectorized_1001.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 1010
Le fichier dataSet/vectorized_1010.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 1011
Le fichier dataSet/vectorized_1011.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 1100
Le fichier dataSet/vectorized_1100.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 1101
Le fichier dataSet/vectorized_1101.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 1110
Le fichier dataSet/vectorized_1110.csv existe déjà, skipping vectorization for
this dataset.
Dataset: 1111
Le fichier dataSet/vectorized_1111.csv existe déjà, skipping vectorization for
this dataset.
```

On va applique notre Topic Modelling sur chaque dataSet :

[ ]:
```
def apply_lda_and_save(data, key, n_topics=15):
    file_path_lda = f"dataSet/lda_results_{key}.csv"

    if os.path.exists(file_path_lda):
        print(f"LDA results already exist for dataset {key}. Skipping LDA.")
        return pd.read_csv(file_path_lda)

    print(f"Applying LDA to dataset {key}...")

    count_vectorizer = CountVectorizer(ngram_range=(1, 2), min_df=5, max_df=0.9)
```

```
    count_matrix = count_vectorizer.fit_transform(data['text'])

    lda_model = LatentDirichletAllocation(n_components=n_topics,␣
↪random_state=42, max_iter=10)
    lda_topics = lda_model.fit_transform(count_matrix)
    topic_assignments = np.argmax(lda_topics, axis=1)
    data['Topic'] = topic_assignments

    data.to_csv(file_path_lda, index=False)
    print(f"LDA results for dataset {key} saved.")
    return data


for key in dataPrepared:
    dataPrepared[key] = apply_lda_and_save(dataPrepared[key], key)
```

```
LDA results already exist for dataset 0000. Skipping LDA.
LDA results already exist for dataset 0001. Skipping LDA.
LDA results already exist for dataset 0010. Skipping LDA.
LDA results already exist for dataset 0011. Skipping LDA.
LDA results already exist for dataset 0100. Skipping LDA.
LDA results already exist for dataset 0101. Skipping LDA.
LDA results already exist for dataset 0110. Skipping LDA.
LDA results already exist for dataset 0111. Skipping LDA.
LDA results already exist for dataset 1000. Skipping LDA.
LDA results already exist for dataset 1001. Skipping LDA.
LDA results already exist for dataset 1010. Skipping LDA.
LDA results already exist for dataset 1011. Skipping LDA.
LDA results already exist for dataset 1100. Skipping LDA.
LDA results already exist for dataset 1101. Skipping LDA.
LDA results already exist for dataset 1110. Skipping LDA.
LDA results already exist for dataset 1111. Skipping LDA.
```

On a maintenant tous les dataSet vectorisés, on va chercher quel est le dataSet optimal. Pour cela, on va évaluer la cohérence de chaque dataSet :

[ ]:
```
def compute_topic_coherence(texts, n_topics=15):
    stop_words = set(stopwords.words('english'))
    tokenized_texts = [[word for word in doc.lower().split() if word not in␣
↪stop_words]
                       for doc in texts]

    # Create Dictionary and Corpus
    dictionary = Dictionary(tokenized_texts)
    corpus = [dictionary.doc2bow(text) for text in tokenized_texts]

    # Gensim LDA model (not scikit-learn)
    lda_model = LdaModel(corpus=corpus,
```

```
                         id2word=dictionary,
                         num_topics=n_topics,
                         random_state=42,
                         passes=10)

    # Compute coherence
    coherence_model = CoherenceModel(model=lda_model, texts=tokenized_texts,␣
↪dictionary=dictionary, coherence='c_v')
    coherence_score = coherence_model.get_coherence()
    return coherence_score
coherence_scores = {}

max_score=0
max_key=""
for key in dataPrepared:
    df = dataPrepared[key]
    coherence = compute_topic_coherence(df['text'].tolist(), n_topics=15)
    coherence_scores[key] = coherence
    print(f"Dataset '{key}' → Coherence Score: {coherence:.4f}")
    if(coherence>max_score):
        max_score=coherence
        max_key=key


print(f"The best option is {max_key} with score {max_score}")
```

```
Dataset '0000' → Coherence Score: 0.3879
Dataset '0001' → Coherence Score: 0.3921
Dataset '0010' → Coherence Score: 0.4180
Dataset '0011' → Coherence Score: 0.4029
Dataset '0100' → Coherence Score: 0.4220
Dataset '0101' → Coherence Score: 0.4098
Dataset '0110' → Coherence Score: 0.4221
Dataset '0111' → Coherence Score: 0.4247
Dataset '1000' → Coherence Score: 0.3653
Dataset '1001' → Coherence Score: 0.3843
Dataset '1010' → Coherence Score: 0.3812
Dataset '1011' → Coherence Score: 0.3644
Dataset '1100' → Coherence Score: 0.3682
Dataset '1101' → Coherence Score: 0.3705
Dataset '1110' → Coherence Score: 0.3621
Dataset '1111' → Coherence Score: 0.3616
The best option is 0111 with score 0.42465753951006896
```

On va utiliser alors pour chaque niveau les classifieurs qui on eu le meilleur score.

6 Optimisation #2 : on va utiliser Optuna, un outil de recherche d'hyperparamètres plus performant

En plus du dataSet optimisé, on va non pas utiliser SearchGridCV mais optuna qui est à priori plus efficace pour essyer d'obtenir une meilleur accuracy.

Pour chaque niveau de classification, on va appliquer nos 2 optimisations pour le meilleur classifieur obtenu précédement (ne montrant aucun signe de surapprentissage).

6.0.1 Niveau 1 : SVC

On utilise notre meilleur dataSet avec optuna sur le clasifieur SVC pour voir comment nos résultats sont influencés.

[ ]:
```
import pandas as pd
from sklearn.svm import SVC
import optuna

# -- Copie de tes données (adjust to your actual data loading)
data_lvl1 = dataPrepared["0111"]  # Use the best dataset from optimization #1
y = data_lvl1['science_related']
X_text = data_lvl1['text']

# -- Vectorisation TF-IDF + Scaling (unchanged)
vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=5, max_df=0.9)
X_vectorized = vectorizer.fit_transform(X_text)
scaler_tfidf = MaxAbsScaler()
X_scaled = scaler_tfidf.fit_transform(X_vectorized)

# -- Split (unchanged)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42, stratify=y)

# -- Standard Scaling for the model (unchanged)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.toarray())
X_test_scaled = scaler.transform(X_test.toarray())

# -- Upsampling to balance (unchanged)
X_df = pd.DataFrame(X_train_scaled)
X_df['label'] = y_train.values
df_majority = X_df[X_df['label'] == 0]
df_minority = X_df[X_df['label'] == 1]
df_minority_upsampled = resample(
    df_minority,
    replace=True,
    n_samples=len(df_majority),
    random_state=42
```

```
)
df_upsampled = pd.concat([df_majority, df_minority_upsampled])
X_train_balanced = df_upsampled.drop('label', axis=1).values
y_train_balanced = df_upsampled['label'].values

# -- Optuna: optimization of the SVC
def objective(trial):
    C = trial.suggest_float("C", 1e-3, 1e3, log=True)
    kernel = trial.suggest_categorical("kernel", ["linear", "rbf", "poly"])
    gamma = trial.suggest_categorical("gamma", ["scale", "auto"])
    svc = SVC(C=C, kernel=kernel, gamma=gamma, random_state=42)  # Use the best␣
↪dataset from optimization #1
    scores = cross_val_score(svc, X_train_balanced, y_train_balanced,␣
↪cv=KFold(n_splits=10, shuffle=True, random_state=42), scoring='accuracy')
    return scores.mean()

# -- Create and launch the study
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100) # Adjust n_trials as needed

print("Best hyperparameters:", study.best_params)
print("Best accuracy:", study.best_value)

# Train the model with best hyperparameters
best_svc = SVC(**study.best_params, random_state=42)
best_svc.fit(X_train_balanced, y_train_balanced)

# Evaluate on the test set
y_pred = best_svc.predict(X_test_scaled)
print(classification_report(y_test,y_pred))

#Affiche la matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
plot_curves_confusion(conf_matrix, ['NON-SCI', 'SCI'])
```

```
[I 2025-05-02 12:32:45,270] A new study created in memory with name: no-
name-a7fa08bd-213f-43d8-a1be-635158a4ea55
[I 2025-05-02 12:32:49,764] Trial 0 finished with value: 0.9002998800479807 and
parameters: {'C': 5.359651279643348, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9002998800479807.
[I 2025-05-02 12:32:52,177] Trial 1 finished with value: 0.9060375849660135 and
parameters: {'C': 10.163435890730248, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 1 with value: 0.9060375849660135.
[I 2025-05-02 12:32:53,846] Trial 2 finished with value: 0.8382513661202186 and
parameters: {'C': 0.2227200153081087, 'kernel': 'linear', 'gamma': 'scale'}.
Best is trial 1 with value: 0.9060375849660135.
[I 2025-05-02 12:32:56,121] Trial 3 finished with value: 0.8962148473943756 and
```

```
parameters: {'C': 3.1416879277220713, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 1 with value: 0.9060375849660135.
[I 2025-05-02 12:32:58,259] Trial 4 finished with value: 0.9076636012261762 and
parameters: {'C': 20.038327195311233, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 4 with value: 0.9076636012261762.
[I 2025-05-02 12:33:00,126] Trial 5 finished with value: 0.8259629481540717 and
parameters: {'C': 0.0021405687827504567, 'kernel': 'linear', 'gamma': 'scale'}.
Best is trial 4 with value: 0.9076636012261762.
[I 2025-05-02 12:33:02,292] Trial 6 finished with value: 0.8316939890710383 and
parameters: {'C': 2.5852939367595047, 'kernel': 'linear', 'gamma': 'scale'}.
Best is trial 4 with value: 0.9076636012261762.
[I 2025-05-02 12:33:05,900] Trial 7 finished with value: 0.4878315340530455 and
parameters: {'C': 0.008159188560539349, 'kernel': 'rbf', 'gamma': 'auto'}. Best
is trial 4 with value: 0.9076636012261762.
[I 2025-05-02 12:33:08,292] Trial 8 finished with value: 0.8881114220978276 and
parameters: {'C': 466.6879517122993, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 4 with value: 0.9076636012261762.
[I 2025-05-02 12:33:10,514] Trial 9 finished with value: 0.9002998800479807 and
parameters: {'C': 5.571920956736447, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 4 with value: 0.9076636012261762.
[I 2025-05-02 12:33:12,679] Trial 10 finished with value: 0.8283419965347194 and
parameters: {'C': 748.8454617876046, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 4 with value: 0.9076636012261762.
[I 2025-05-02 12:33:14,959] Trial 11 finished with value: 0.90848993735839 and
parameters: {'C': 43.88233762014047, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:17,758] Trial 12 finished with value: 0.9068639210982272 and
parameters: {'C': 138.21317371011222, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:20,073] Trial 13 finished with value: 0.90848993735839 and
parameters: {'C': 50.3342934177376, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:22,700] Trial 14 finished with value: 0.6993002798880448 and
parameters: {'C': 0.2519644331112752, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:25,088] Trial 15 finished with value: 0.8881114220978276 and
parameters: {'C': 68.14731075436232, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:27,235] Trial 16 finished with value: 0.906850593096095 and
parameters: {'C': 59.07649868262491, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:30,180] Trial 17 finished with value: 0.7998067439690789 and
parameters: {'C': 0.41720149251149075, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:32,322] Trial 18 finished with value: 0.8538118086098893 and
parameters: {'C': 0.04106817270318032, 'kernel': 'linear', 'gamma': 'auto'}.
Best is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:34,698] Trial 19 finished with value: 0.8881114220978276 and
```

```
parameters: {'C': 250.01998631147634, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:36,818] Trial 20 finished with value: 0.9076702652272426 and
parameters: {'C': 28.55272747500124, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:38,980] Trial 21 finished with value: 0.9060309209649473 and
parameters: {'C': 24.76830914319448, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:41,111] Trial 22 finished with value: 0.90848993735839 and
parameters: {'C': 40.38618694932617, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:43,946] Trial 23 finished with value: 0.8757896841263495 and
parameters: {'C': 1.0196283639257155, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:46,627] Trial 24 finished with value: 0.9068639210982272 and
parameters: {'C': 141.31950206971078, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:48,771] Trial 25 finished with value: 0.8087098493935759 and
parameters: {'C': 890.3184318610129, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:50,957] Trial 26 finished with value: 0.9060242569638811 and
parameters: {'C': 73.73589124970229, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:53,107] Trial 27 finished with value: 0.9060375849660135 and
parameters: {'C': 13.154650988700915, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:55,473] Trial 28 finished with value: 0.8881114220978276 and
parameters: {'C': 264.11149060899714, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:33:57,511] Trial 29 finished with value: 0.8316939890710383 and
parameters: {'C': 1.6969762167759592, 'kernel': 'linear', 'gamma': 'auto'}. Best
is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:34:00,258] Trial 30 finished with value: 0.9076702652272426 and
parameters: {'C': 31.983897166047864, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:34:02,386] Trial 31 finished with value: 0.9060375849660135 and
parameters: {'C': 11.197293967172929, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:34:04,578] Trial 32 finished with value: 0.9002998800479807 and
parameters: {'C': 5.439621302631296, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:34:07,403] Trial 33 finished with value: 0.90848993735839 and
parameters: {'C': 34.541560814586795, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:34:09,588] Trial 34 finished with value: 0.9068639210982272 and
parameters: {'C': 143.27217753017425, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:34:13,577] Trial 35 finished with value: 0.90848993735839 and
```

parameters: {'C': 45.356687897905594, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:34:17,290] Trial 36 finished with value: 0.9052179128348661 and parameters: {'C': 9.186787907095612, 'kernel': 'poly', 'gamma': 'scale'}. Best is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:34:20,350] Trial 37 finished with value: 0.8316939890710383 and parameters: {'C': 362.40513892974155, 'kernel': 'linear', 'gamma': 'auto'}. Best is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:34:22,617] Trial 38 finished with value: 0.8970345195255233 and parameters: {'C': 3.247698230352215, 'kernel': 'poly', 'gamma': 'scale'}. Best is trial 11 with value: 0.90848993735839.
[I 2025-05-02 12:34:24,898] Trial 39 finished with value: 0.9101292816206851 and parameters: {'C': 118.34889837729717, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:28,099] Trial 40 finished with value: 0.8316939890710383 and parameters: {'C': 122.67409261160931, 'kernel': 'linear', 'gamma': 'scale'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:30,246] Trial 41 finished with value: 0.9076636012261762 and parameters: {'C': 20.596920063793227, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:32,403] Trial 42 finished with value: 0.9060242569638811 and parameters: {'C': 81.72570249005695, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:34,555] Trial 43 finished with value: 0.9060375849660135 and parameters: {'C': 11.462158009893995, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:37,191] Trial 44 finished with value: 0.5197920831667333 and parameters: {'C': 0.04557544921019752, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:40,881] Trial 45 finished with value: 0.4878315340530455 and parameters: {'C': 0.0010646805221667343, 'kernel': 'rbf', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:43,327] Trial 46 finished with value: 0.8716913234706117 and parameters: {'C': 519.3838843256264, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:45,473] Trial 47 finished with value: 0.9044182327069172 and parameters: {'C': 211.55192270816605, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:47,604] Trial 48 finished with value: 0.90848993735839 and parameters: {'C': 42.3486197096076, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:49,827] Trial 49 finished with value: 0.9002998800479807 and parameters: {'C': 5.576468464090218, 'kernel': 'poly', 'gamma': 'scale'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:52,264] Trial 50 finished with value: 0.8881114220978276 and parameters: {'C': 94.0380015213761, 'kernel': 'rbf', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:55,065] Trial 51 finished with value: 0.9076636012261762 and

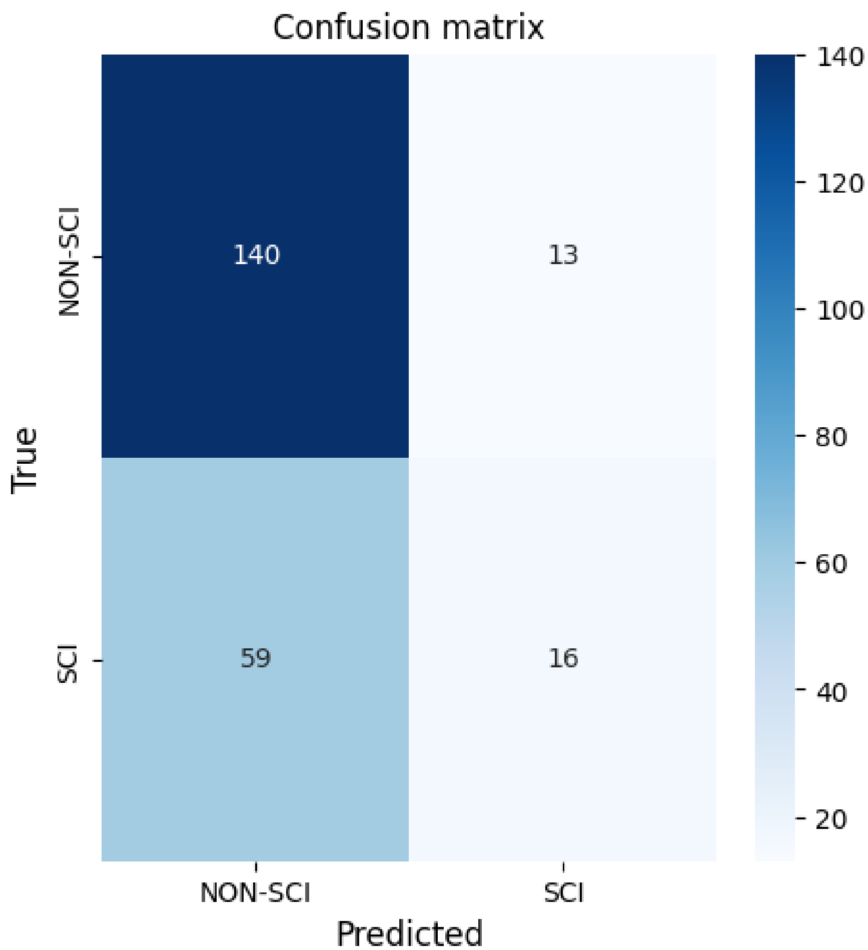
```
parameters: {'C': 61.55358715741307, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:57,448] Trial 52 finished with value: 0.90848993735839 and
parameters: {'C': 48.87354998985955, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:34:59,580] Trial 53 finished with value: 0.9076636012261762 and
parameters: {'C': 17.64564788582093, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:01,714] Trial 54 finished with value: 0.9076702652272426 and
parameters: {'C': 31.358959776462417, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:03,840] Trial 55 finished with value: 0.90848993735839 and
parameters: {'C': 41.632888749758166, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:06,467] Trial 56 finished with value: 0.8316939890710383 and
parameters: {'C': 197.58687514003822, 'kernel': 'linear', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:09,323] Trial 57 finished with value: 0.8978475276556045 and
parameters: {'C': 3.152445502063453, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:11,613] Trial 58 finished with value: 0.8724976675996269 and
parameters: {'C': 496.85965271575697, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:13,784] Trial 59 finished with value: 0.9076636012261762 and
parameters: {'C': 18.7654480233591, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:16,189] Trial 60 finished with value: 0.8938357990137279 and
parameters: {'C': 7.70930016207193, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:18,313] Trial 61 finished with value: 0.90848993735839 and
parameters: {'C': 40.39397980693631, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:20,666] Trial 62 finished with value: 0.9076636012261762 and
parameters: {'C': 86.14277811396363, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:23,492] Trial 63 finished with value: 0.90848993735839 and
parameters: {'C': 41.5545657023575, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:25,715] Trial 64 finished with value: 0.9052379048380648 and
parameters: {'C': 150.2458994413215, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:27,832] Trial 65 finished with value: 0.8962281753965081 and
parameters: {'C': 335.48698579927554, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:29,988] Trial 66 finished with value: 0.9076636012261762 and
parameters: {'C': 61.34142710469814, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:32,133] Trial 67 finished with value: 0.9076636012261762 and
```

parameters: {'C': 16.05061747333946, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:34,548] Trial 68 finished with value: 0.8316939890710383 and parameters: {'C': 120.16677549962424, 'kernel': 'linear', 'gamma': 'scale'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:37,403] Trial 69 finished with value: 0.9068439290950285 and parameters: {'C': 23.629867538862534, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:39,599] Trial 70 finished with value: 0.9019392243102757 and parameters: {'C': 7.217313018260566, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:41,768] Trial 71 finished with value: 0.906850593096095 and parameters: {'C': 57.60561274361565, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:43,932] Trial 72 finished with value: 0.90848993735839 and parameters: {'C': 39.55540813397541, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:46,095] Trial 73 finished with value: 0.906850593096095 and parameters: {'C': 104.99786140143189, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:48,449] Trial 74 finished with value: 0.90848993735839 and parameters: {'C': 50.50565927207005, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:51,223] Trial 75 finished with value: 0.9076702652272426 and parameters: {'C': 28.835426489668862, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:53,464] Trial 76 finished with value: 0.9044182327069172 and parameters: {'C': 212.0277276008264, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:56,114] Trial 77 finished with value: 0.6166733306677329 and parameters: {'C': 0.10207991440179218, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:35:58,673] Trial 78 finished with value: 0.8938224710115954 and parameters: {'C': 1.7818239225907821, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:01,222] Trial 79 finished with value: 0.8365787018525923 and parameters: {'C': 0.5047042326125218, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:03,962] Trial 80 finished with value: 0.8495868319338931 and parameters: {'C': 658.6978459624349, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:06,431] Trial 81 finished with value: 0.90848993735839 and parameters: {'C': 43.66453044203747, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:09,071] Trial 82 finished with value: 0.467339730774357 and parameters: {'C': 0.008236733830609968, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:11,216] Trial 83 finished with value: 0.906850593096095 and

```
parameters: {'C': 13.71627152515108, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:13,377] Trial 84 finished with value: 0.9068439290950285 and
parameters: {'C': 24.39673801008171, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:15,406] Trial 85 finished with value: 0.8316939890710383 and
parameters: {'C': 89.09777829623121, 'kernel': 'linear', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:18,189] Trial 86 finished with value: 0.9044182327069171 and
parameters: {'C': 164.13517065821867, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:20,627] Trial 87 finished with value: 0.8986871917899506 and
parameters: {'C': 311.2002301610543, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:22,802] Trial 88 finished with value: 0.9060242569638811 and
parameters: {'C': 69.47632088979937, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:24,934] Trial 89 finished with value: 0.9076702652272426 and
parameters: {'C': 30.533392784679645, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:27,123] Trial 90 finished with value: 0.9068572570971611 and
parameters: {'C': 10.317594919599733, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:29,263] Trial 91 finished with value: 0.90848993735839 and
parameters: {'C': 36.4265563186796, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:32,022] Trial 92 finished with value: 0.90848993735839 and
parameters: {'C': 44.171708138756145, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:34,469] Trial 93 finished with value: 0.9076636012261762 and
parameters: {'C': 18.494246558068415, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:36,621] Trial 94 finished with value: 0.9060242569638811 and
parameters: {'C': 79.47887663984784, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:39,013] Trial 95 finished with value: 0.8881114220978276 and
parameters: {'C': 53.98375863929295, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:41,172] Trial 96 finished with value: 0.9093096094895374 and
parameters: {'C': 122.89852074139925, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:43,400] Trial 97 finished with value: 0.9101292816206851 and
parameters: {'C': 119.79683120670187, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:46,182] Trial 98 finished with value: 0.9101292816206851 and
parameters: {'C': 120.53999348612868, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
[I 2025-05-02 12:36:48,605] Trial 99 finished with value: 0.9101292816206851 and
```

```
parameters: {'C': 116.82359003740609, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 39 with value: 0.9101292816206851.
```

```
Best hyperparameters: {'C': 118.34889837729717, 'kernel': 'poly', 'gamma':
'auto'}
Best accuracy: 0.9101292816206851
              precision    recall  f1-score   support

           0       0.70      0.92      0.80       153
           1       0.55      0.21      0.31        75

    accuracy                           0.68       228
   macro avg       0.63      0.56      0.55       228
weighted avg       0.65      0.68      0.64       228
```



Si on compare avec optuna et sans, on passe d'une accuracy de 0.8899 à 0.9101 (soit une augmentation de 2.02 %)

###Niveau 2 : SVC

Pour le niveeau 2, c'était SVC le plus pertinent. On va donc le comparer avec optuna

[ ]:
```
import optuna
from sklearn.model_selection import cross_val_score, KFold
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MaxAbsScaler, StandardScaler
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
from sklearn.utils import resample
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# ... (Your existing code for data loading and preprocessing) ...


# Assuming data_lvl2, y, X_text, X_train, X_test, y_train, y_test are defined␣
↪from previous code
# Copie des données niveau 2 (sur 0111)
data_lvl2 = dataPrepared["0111"].copy()
# 2. Création de la colonne cible (claim or reference)
data_lvl2['claim_or_ref'] = data_lvl2.apply(lambda row: 1 if␣
↪row['scientific_claim'] == 1 or row['scientific_reference'] == 1 else 0,␣
↪axis=1)
y = data_lvl2['claim_or_ref']
X_text = data_lvl2['text']
# 3. Vectorisation TF-IDF + MaxAbsScaler
vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=5, max_df=0.9)
X_vectorized = vectorizer.fit_transform(X_text)
scaler = MaxAbsScaler()
X_scaled = scaler.fit_transform(X_vectorized)
# 4. Split train/test
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42, stratify=y)

# Upsampling (adjust to your resampling method)
X_df = pd.DataFrame(X_train.toarray())
X_df['label'] = y_train.values
df_majority = X_df[X_df['label'] == 0]
df_minority = X_df[X_df['label'] == 1]
df_minority_upsampled = resample(
    df_minority,
```

```python
    replace=True,
    n_samples=len(df_majority),
    random_state=42
)
df_upsampled = pd.concat([df_majority, df_minority_upsampled])
X_train_balanced = df_upsampled.drop('label', axis=1).values
y_train_balanced = df_upsampled['label'].values

# StandardScaler for SVC
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_balanced)
X_test_scaled = scaler.transform(X_test.toarray())


def objective(trial):
    C = trial.suggest_float("C", 1e-3, 1e3, log=True)
    kernel = trial.suggest_categorical("kernel", ["linear", "rbf", "poly"])
    gamma = trial.suggest_categorical("gamma", ["scale", "auto"])
    svc = SVC(C=C, kernel=kernel, gamma=gamma, random_state=42)
    scores = cross_val_score(svc, X_train_scaled, y_train_balanced,␣
 ↪cv=KFold(n_splits=10, shuffle=True, random_state=42), scoring='accuracy')
    return scores.mean()

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

print("Best hyperparameters:", study.best_params)
print("Best accuracy:", study.best_value)

best_svc = SVC(**study.best_params, random_state=42)
best_svc.fit(X_train_scaled, y_train_balanced)

y_pred = best_svc.predict(X_test_scaled)
print(classification_report(y_test, y_pred))

conf_matrix = confusion_matrix(y_test, y_pred)
plot_curves_confusion(conf_matrix, ['CONTEXT', 'CLAIM/REF'])
```

```
[I 2025-05-02 12:40:12,908] A new study created in memory with name: no-
name-b13f592e-2f29-4b64-8864-08606e53b75f
[I 2025-05-02 12:40:15,308] Trial 0 finished with value: 0.8660433070866143 and
parameters: {'C': 25.49309532171721, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 0 with value: 0.8660433070866143.
[I 2025-05-02 12:40:18,852] Trial 1 finished with value: 0.47098302165354333 and
parameters: {'C': 0.008556985367697648, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.8660433070866143.
[I 2025-05-02 12:40:22,286] Trial 2 finished with value: 0.9145915354330709 and
```

```
parameters: {'C': 41.89877361696616, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 2 with value: 0.9145915354330709.
[I 2025-05-02 12:40:24,006] Trial 3 finished with value: 0.8518823818897638 and
parameters: {'C': 1.4403756885847705, 'kernel': 'linear', 'gamma': 'auto'}. Best
is trial 2 with value: 0.9145915354330709.
[I 2025-05-02 12:40:26,347] Trial 4 finished with value: 0.9028235728346458 and
parameters: {'C': 62.72036194196265, 'kernel': 'poly', 'gamma': 'auto'}. Best is
trial 2 with value: 0.9145915354330709.
[I 2025-05-02 12:40:29,228] Trial 5 finished with value: 0.47098302165354333 and
parameters: {'C': 0.012089171613909024, 'kernel': 'poly', 'gamma': 'scale'}.
Best is trial 2 with value: 0.9145915354330709.
[I 2025-05-02 12:40:31,171] Trial 6 finished with value: 0.8283833661417324 and
parameters: {'C': 0.003318887144538003, 'kernel': 'linear', 'gamma': 'scale'}.
Best is trial 2 with value: 0.9145915354330709.
[I 2025-05-02 12:40:34,272] Trial 7 finished with value: 0.9169475885826772 and
parameters: {'C': 15.656279884435927, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:40:37,413] Trial 8 finished with value: 0.9153850885826772 and
parameters: {'C': 18.345193147408896, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:40:40,025] Trial 9 finished with value: 0.9145915354330709 and
parameters: {'C': 628.2321019406495, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:40:43,073] Trial 10 finished with value: 0.800166092519685 and
parameters: {'C': 0.23911358693866294, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:40:45,751] Trial 11 finished with value: 0.9098855807086614 and
parameters: {'C': 2.365868066710831, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:40:49,039] Trial 12 finished with value: 0.916160187007874 and
parameters: {'C': 7.619215690255559, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:40:52,001] Trial 13 finished with value: 0.9145915354330709 and
parameters: {'C': 739.0950457267106, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:40:55,094] Trial 14 finished with value: 0.7711552657480315 and
parameters: {'C': 0.15460626974821204, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:40:56,844] Trial 15 finished with value: 0.8518823818897638 and
parameters: {'C': 10.751981158096793, 'kernel': 'linear', 'gamma': 'scale'}.
Best is trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:40:59,450] Trial 16 finished with value: 0.9145915354330709 and
parameters: {'C': 160.97428495634117, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:41:02,732] Trial 17 finished with value: 0.913810285433071 and
parameters: {'C': 3.9879045391456374, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:41:05,912] Trial 18 finished with value: 0.6357898622047244 and
```

```
parameters: {'C': 0.2960663295769313, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:41:07,673] Trial 19 finished with value: 0.8518823818897638 and
parameters: {'C': 6.007165680405432, 'kernel': 'linear', 'gamma': 'scale'}. Best
is trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:41:10,306] Trial 20 finished with value: 0.9145915354330709 and
parameters: {'C': 131.6201547951517, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 7 with value: 0.9169475885826772.
[I 2025-05-02 12:41:12,926] Trial 21 finished with value: 0.9169537401574803 and
parameters: {'C': 14.209806689680384, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:16,273] Trial 22 finished with value: 0.8879552165354332 and
parameters: {'C': 0.6970071523640228, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:20,073] Trial 23 finished with value: 0.7186454232283463 and
parameters: {'C': 0.051388098464955234, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:22,702] Trial 24 finished with value: 0.916160187007874 and
parameters: {'C': 8.017689725454188, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:25,350] Trial 25 finished with value: 0.9145915354330709 and
parameters: {'C': 114.56654767964861, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:28,328] Trial 26 finished with value: 0.899710875984252 and
parameters: {'C': 0.9409628843647136, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:31,743] Trial 27 finished with value: 0.9145915354330709 and
parameters: {'C': 257.2406397730541, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:34,207] Trial 28 finished with value: 0.8605807086614174 and
parameters: {'C': 22.31566727284858, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:35,938] Trial 29 finished with value: 0.8518823818897638 and
parameters: {'C': 42.80827095524998, 'kernel': 'linear', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:38,459] Trial 30 finished with value: 0.7657664862204725 and
parameters: {'C': 3.3780831244736853, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:41,091] Trial 31 finished with value: 0.9161663385826773 and
parameters: {'C': 9.474430977036029, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:44,272] Trial 32 finished with value: 0.9161663385826773 and
parameters: {'C': 13.307866440955257, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:47,325] Trial 33 finished with value: 0.9161663385826773 and
parameters: {'C': 17.7244897764254, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:50,706] Trial 34 finished with value: 0.9145915354330709 and
```

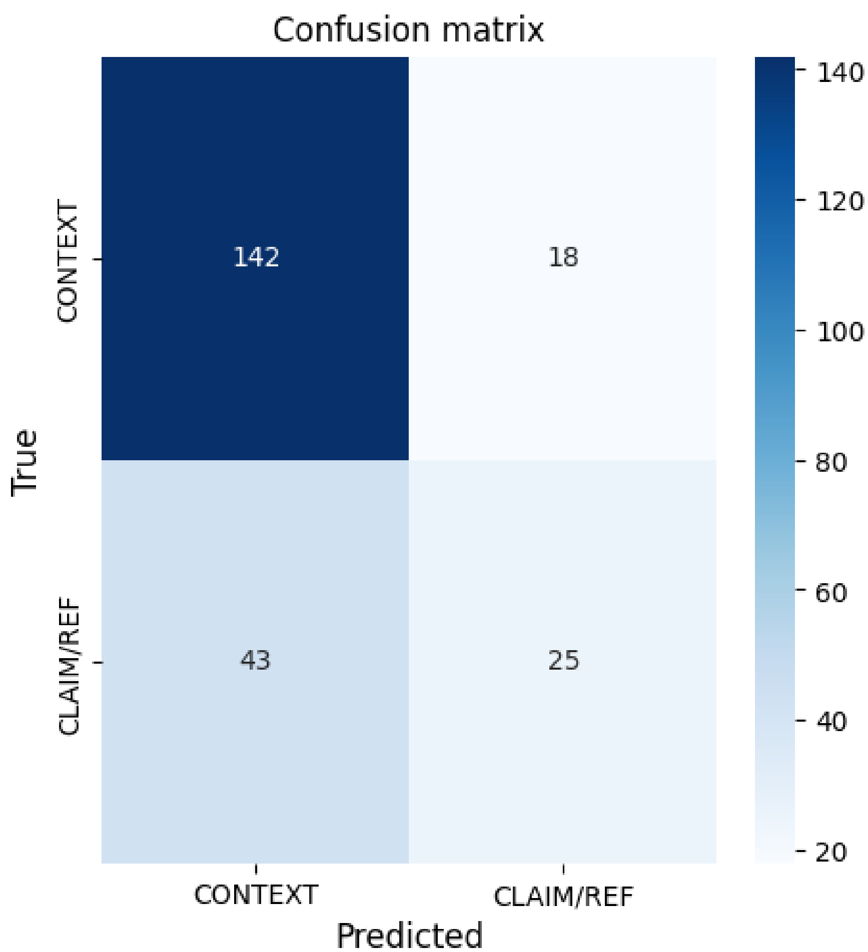
parameters: {'C': 36.650449311022314, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:53,479] Trial 35 finished with value: 0.9075479822834647 and parameters: {'C': 1.6502235379019816, 'kernel': 'rbf', 'gamma': 'auto'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:56,113] Trial 36 finished with value: 0.9145915354330709 and parameters: {'C': 69.25100557678182, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:41:59,560] Trial 37 finished with value: 0.9153850885826772 and parameters: {'C': 18.95078497792104, 'kernel': 'rbf', 'gamma': 'auto'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:01,399] Trial 38 finished with value: 0.8518823818897638 and parameters: {'C': 294.72338132623554, 'kernel': 'linear', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:03,739] Trial 39 finished with value: 0.9036048228346457 and parameters: {'C': 66.23000513116672, 'kernel': 'poly', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:06,376] Trial 40 finished with value: 0.9161663385826773 and parameters: {'C': 9.774534287406299, 'kernel': 'rbf', 'gamma': 'auto'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:09,008] Trial 41 finished with value: 0.9161663385826773 and parameters: {'C': 16.943640484408412, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:12,219] Trial 42 finished with value: 0.914597687007874 and parameters: {'C': 3.644370715222789, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:15,280] Trial 43 finished with value: 0.914597687007874 and parameters: {'C': 28.353670795696164, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:17,956] Trial 44 finished with value: 0.9161663385826773 and parameters: {'C': 13.035176443063767, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:20,650] Trial 45 finished with value: 0.9098855807086614 and parameters: {'C': 2.051703836330096, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:23,592] Trial 46 finished with value: 0.875406003937008 and parameters: {'C': 0.5554113373040696, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:27,000] Trial 47 finished with value: 0.9145915354330709 and parameters: {'C': 5.030857914459224, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:28,973] Trial 48 finished with value: 0.8518823818897638 and parameters: {'C': 42.46088059399646, 'kernel': 'linear', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:32,414] Trial 49 finished with value: 0.47098302165354333 and parameters: {'C': 0.001310682431303666, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:34,760] Trial 50 finished with value: 0.9020361712598426 and

```
parameters: {'C': 80.64553489198269, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:37,371] Trial 51 finished with value: 0.916160187007874 and
parameters: {'C': 8.406585899872807, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:40,700] Trial 52 finished with value: 0.9161663385826773 and
parameters: {'C': 13.514395910959234, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:43,644] Trial 53 finished with value: 0.9114603838582678 and
parameters: {'C': 2.566593168270304, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:46,293] Trial 54 finished with value: 0.9153727854330709 and
parameters: {'C': 6.23996752462987, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:48,896] Trial 55 finished with value: 0.9153789370078741 and
parameters: {'C': 31.743687989256514, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:51,505] Trial 56 finished with value: 0.9161663385826773 and
parameters: {'C': 9.440074353768619, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:54,900] Trial 57 finished with value: 0.9153850885826772 and
parameters: {'C': 22.270544658806763, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:42:58,027] Trial 58 finished with value: 0.9044229822834646 and
parameters: {'C': 1.2117474335150564, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:00,622] Trial 59 finished with value: 0.9145915354330709 and
parameters: {'C': 351.3510144458915, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:02,390] Trial 60 finished with value: 0.8518823818897638 and
parameters: {'C': 4.654776460552101, 'kernel': 'linear', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:05,037] Trial 61 finished with value: 0.9161663385826773 and
parameters: {'C': 14.540170461087001, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:08,281] Trial 62 finished with value: 0.9161663385826773 and
parameters: {'C': 16.47848008680961, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:11,281] Trial 63 finished with value: 0.9145915354330709 and
parameters: {'C': 49.74961495265019, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:13,924] Trial 64 finished with value: 0.9145915354330709 and
parameters: {'C': 112.2675877380602, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:16,582] Trial 65 finished with value: 0.9161663385826773 and
parameters: {'C': 10.103957200839705, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:20,193] Trial 66 finished with value: 0.6566867618110235 and
```

parameters: {'C': 0.02834193903451357, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:23,518] Trial 67 finished with value: 0.7547920767716535 and parameters: {'C': 2.7765479508667212, 'kernel': 'poly', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:26,149] Trial 68 finished with value: 0.914597687007874 and parameters: {'C': 27.11700160239311, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:28,799] Trial 69 finished with value: 0.916160187007874 and parameters: {'C': 6.456253237034848, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:31,427] Trial 70 finished with value: 0.9153850885826772 and parameters: {'C': 19.394654346480475, 'kernel': 'rbf', 'gamma': 'auto'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:34,212] Trial 71 finished with value: 0.9161663385826773 and parameters: {'C': 13.035888157716942, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:38,841] Trial 72 finished with value: 0.9161663385826773 and parameters: {'C': 10.771090578692661, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:41,459] Trial 73 finished with value: 0.9145915354330709 and parameters: {'C': 187.94542952931099, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:44,102] Trial 74 finished with value: 0.9145915354330709 and parameters: {'C': 4.618340109392907, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:46,762] Trial 75 finished with value: 0.9145915354330709 and parameters: {'C': 55.579484544480074, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:48,809] Trial 76 finished with value: 0.8518823818897638 and parameters: {'C': 31.804054552170765, 'kernel': 'linear', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:52,143] Trial 77 finished with value: 0.916160187007874 and parameters: {'C': 7.108488925048039, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:55,642] Trial 78 finished with value: 0.9091104822834646 and parameters: {'C': 1.7852744774897473, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:43:57,959] Trial 79 finished with value: 0.8989111712598425 and parameters: {'C': 91.0430605907598, 'kernel': 'poly', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:00,580] Trial 80 finished with value: 0.9161663385826773 and parameters: {'C': 17.304029619219794, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:03,700] Trial 81 finished with value: 0.9161663385826773 and parameters: {'C': 11.494628568854466, 'kernel': 'rbf', 'gamma': 'auto'}. Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:06,793] Trial 82 finished with value: 0.9161663385826773 and

```
parameters: {'C': 13.048087153147154, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:09,411] Trial 83 finished with value: 0.9153850885826772 and
parameters: {'C': 23.251851632583477, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:12,023] Trial 84 finished with value: 0.9145915354330709 and
parameters: {'C': 39.10604835641706, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:14,688] Trial 85 finished with value: 0.9153789370078741 and
parameters: {'C': 3.5284957967742305, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:17,883] Trial 86 finished with value: 0.916160187007874 and
parameters: {'C': 7.562258775009655, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:20,903] Trial 87 finished with value: 0.9161663385826773 and
parameters: {'C': 16.897736751414115, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:23,502] Trial 88 finished with value: 0.9145915354330709 and
parameters: {'C': 5.440915928608363, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:25,252] Trial 89 finished with value: 0.8518823818897638 and
parameters: {'C': 48.302286284803856, 'kernel': 'linear', 'gamma': 'scale'}.
Best is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:28,221] Trial 90 finished with value: 0.8573695866141733 and
parameters: {'C': 0.4676400882154844, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:31,343] Trial 91 finished with value: 0.9161663385826773 and
parameters: {'C': 9.971592204942468, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:34,427] Trial 92 finished with value: 0.914597687007874 and
parameters: {'C': 26.82910551819299, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:37,021] Trial 93 finished with value: 0.916160187007874 and
parameters: {'C': 8.064810702377095, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:39,633] Trial 94 finished with value: 0.9161663385826773 and
parameters: {'C': 13.69888908728389, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:42,285] Trial 95 finished with value: 0.9153789370078741 and
parameters: {'C': 3.3012425679078783, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:45,444] Trial 96 finished with value: 0.9153850885826772 and
parameters: {'C': 21.865067340886785, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:48,518] Trial 97 finished with value: 0.916160187007874 and
parameters: {'C': 8.559313492890949, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:50,862] Trial 98 finished with value: 0.8730745570866143 and
```

```
parameters: {'C': 33.713492308553334, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 21 with value: 0.9169537401574803.
[I 2025-05-02 12:44:53,537] Trial 99 finished with value: 0.9145915354330709 and
parameters: {'C': 4.863697921777832, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 21 with value: 0.9169537401574803.

Best hyperparameters: {'C': 14.209806689680384, 'kernel': 'rbf', 'gamma':
'scale'}
Best accuracy: 0.9169537401574803
              precision    recall  f1-score   support

           0       0.77      0.89      0.82       160
           1       0.58      0.37      0.45        68

    accuracy                           0.73       228
   macro avg       0.67      0.63      0.64       228
weighted avg       0.71      0.73      0.71       228
```



On passe d'une accuracy de 0.9023 à 0.9169 (une augmentation de 1.46%)

6.0.2 Niveau 3 : SVC

Pour notre niveau 3, c'était SVC le plus performant :

[ ]:
```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import RandomOverSampler
import optuna
import pandas as pd
import numpy as np
from collections import Counter
from sklearn.utils import resample


# Assuming data_lvl3, y, X_text, X_train, X_test, y_train, y_test are defined␣
 ↪from previous code
# Copie des données niveau 3 (sur 0111)
data_lvl3 = dataPrepared["0111"].copy()

# Fonction pour assigner le label du niveau 3
def get_level3_label(row):
    if row['scientific_claim'] == 1:
        return 'CLAIM'
    elif row['scientific_reference'] == 1:
        return 'REF'
    elif row['scientific_context'] == 1:
        return 'CONTEXT'
    else:
        return 'NON-SCI'

# Application de la fonction et suppression de 'NON-SCI'
data_lvl3['level3_label'] = data_lvl3.apply(get_level3_label, axis=1)
data_lvl3 = data_lvl3[data_lvl3['level3_label'] != 'NON-SCI']
y = data_lvl3['level3_label']
X_text = data_lvl3['text']

# Vectorisation TF-IDF + MaxAbsScaler
vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=5, max_df=0.9)
X_vectorized = vectorizer.fit_transform(X_text)
scaler = MaxAbsScaler()
X_scaled = scaler.fit_transform(X_vectorized)

# Split train/test
```

```python
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42, stratify=y)

# Upsampling avec RandomOverSampler (adjust to your resampling method)
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)


# StandardScaler for SVC
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_resampled.toarray()) # Fit and␣
↪transform on the training data
X_test_scaled = scaler.transform(X_test.toarray()) # Transform the testing data


def objective(trial):
    C = trial.suggest_float("C", 1e-3, 1e3, log=True)
    kernel = trial.suggest_categorical("kernel", ["linear", "rbf", "poly"])
    gamma = trial.suggest_categorical("gamma", ["scale", "auto"])
    svc = SVC(C=C, kernel=kernel, gamma=gamma, random_state=42)
    scores = cross_val_score(svc, X_train_scaled, y_resampled,␣
↪cv=KFold(n_splits=10, shuffle=True, random_state=42), scoring='accuracy')
    return scores.mean()

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

print("Best hyperparameters:", study.best_params)
print("Best accuracy:", study.best_value)

best_svc = SVC(**study.best_params, random_state=42)
best_svc.fit(X_train_scaled, y_resampled)

y_pred = best_svc.predict(X_test_scaled)
print(classification_report(y_test, y_pred))

conf_matrix = confusion_matrix(y_test, y_pred)
plot_curves_confusion(conf_matrix, ['CONTEXT', 'CLAIM', 'REF'])
```

```
[I 2025-05-02 12:48:34,243] A new study created in memory with name: no-name-beb55fc3-1940-493a-b77f-343ca194712a
[I 2025-05-02 12:48:34,646] Trial 0 finished with value: 0.9523809523809523 and parameters: {'C': 420.8077194143475, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:35,117] Trial 1 finished with value: 0.7238095238095238 and parameters: {'C': 5.595899491680863, 'kernel': 'poly', 'gamma': 'auto'}. Best is trial 0 with value: 0.9523809523809523.
```

```
[I 2025-05-02 12:48:35,479] Trial 2 finished with value: 0.8761904761904763 and
parameters: {'C': 0.827985155469654, 'kernel': 'linear', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:35,968] Trial 3 finished with value: 0.6841269841269841 and
parameters: {'C': 1.1213060925927063, 'kernel': 'poly', 'gamma': 'auto'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:36,248] Trial 4 finished with value: 0.880952380952381 and
parameters: {'C': 0.19600540915409276, 'kernel': 'linear', 'gamma': 'auto'}.
Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:37,227] Trial 5 finished with value: 0.29523809523809524 and
parameters: {'C': 0.0013152999112212632, 'kernel': 'rbf', 'gamma': 'auto'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:37,823] Trial 6 finished with value: 0.8682539682539682 and
parameters: {'C': 0.02028455533203402, 'kernel': 'linear', 'gamma': 'scale'}.
Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:38,851] Trial 7 finished with value: 0.29523809523809524 and
parameters: {'C': 0.004310259218194081, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:39,131] Trial 8 finished with value: 0.8746031746031747 and
parameters: {'C': 1.0457336825751642, 'kernel': 'linear', 'gamma': 'scale'}.
Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:39,498] Trial 9 finished with value: 0.9507936507936507 and
parameters: {'C': 62.11309781620548, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:39,871] Trial 10 finished with value: 0.9523809523809523 and
parameters: {'C': 901.4166381518937, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:40,264] Trial 11 finished with value: 0.9523809523809523 and
parameters: {'C': 839.3121374369335, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:40,650] Trial 12 finished with value: 0.9523809523809523 and
parameters: {'C': 998.3585882553211, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:40,967] Trial 13 finished with value: 0.9523809523809523 and
parameters: {'C': 170.01439911641984, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:41,225] Trial 14 finished with value: 0.9476190476190476 and
parameters: {'C': 24.198492623872074, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:41,472] Trial 15 finished with value: 0.9523809523809523 and
parameters: {'C': 188.7896028734391, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:41,725] Trial 16 finished with value: 0.7317460317460318 and
parameters: {'C': 16.416857760580605, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:41,972] Trial 17 finished with value: 0.9523809523809523 and
parameters: {'C': 250.7865882876263, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
```

```
[I 2025-05-02 12:48:42,232] Trial 18 finished with value: 0.9476190476190476 and
parameters: {'C': 10.531021196321111, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:42,481] Trial 19 finished with value: 0.7365079365079364 and
parameters: {'C': 62.74059536125279, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:42,892] Trial 20 finished with value: 0.6888888888888888 and
parameters: {'C': 0.06896524085564543, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:43,153] Trial 21 finished with value: 0.9523809523809523 and
parameters: {'C': 718.1783672621494, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:43,403] Trial 22 finished with value: 0.9523809523809523 and
parameters: {'C': 446.2719909632593, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:43,648] Trial 23 finished with value: 0.9523809523809523 and
parameters: {'C': 93.30281619263991, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:43,895] Trial 24 finished with value: 0.9523809523809523 and
parameters: {'C': 966.9652719827114, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:44,152] Trial 25 finished with value: 0.9476190476190476 and
parameters: {'C': 35.31637337696693, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:44,395] Trial 26 finished with value: 0.9523809523809523 and
parameters: {'C': 235.7175581215852, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:44,632] Trial 27 finished with value: 0.946031746031746 and
parameters: {'C': 3.320716628864399, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:44,877] Trial 28 finished with value: 0.7444444444444445 and
parameters: {'C': 350.11634365252576, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:45,062] Trial 29 finished with value: 0.8730158730158731 and
parameters: {'C': 4.309565633449775, 'kernel': 'linear', 'gamma': 'auto'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:45,340] Trial 30 finished with value: 0.7396825396825397 and
parameters: {'C': 105.93523261899276, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:45,589] Trial 31 finished with value: 0.9523809523809523 and
parameters: {'C': 951.6357738724118, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:45,837] Trial 32 finished with value: 0.9523809523809523 and
parameters: {'C': 512.3215178102223, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:46,085] Trial 33 finished with value: 0.9523809523809523 and
parameters: {'C': 398.5675856795162, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
```

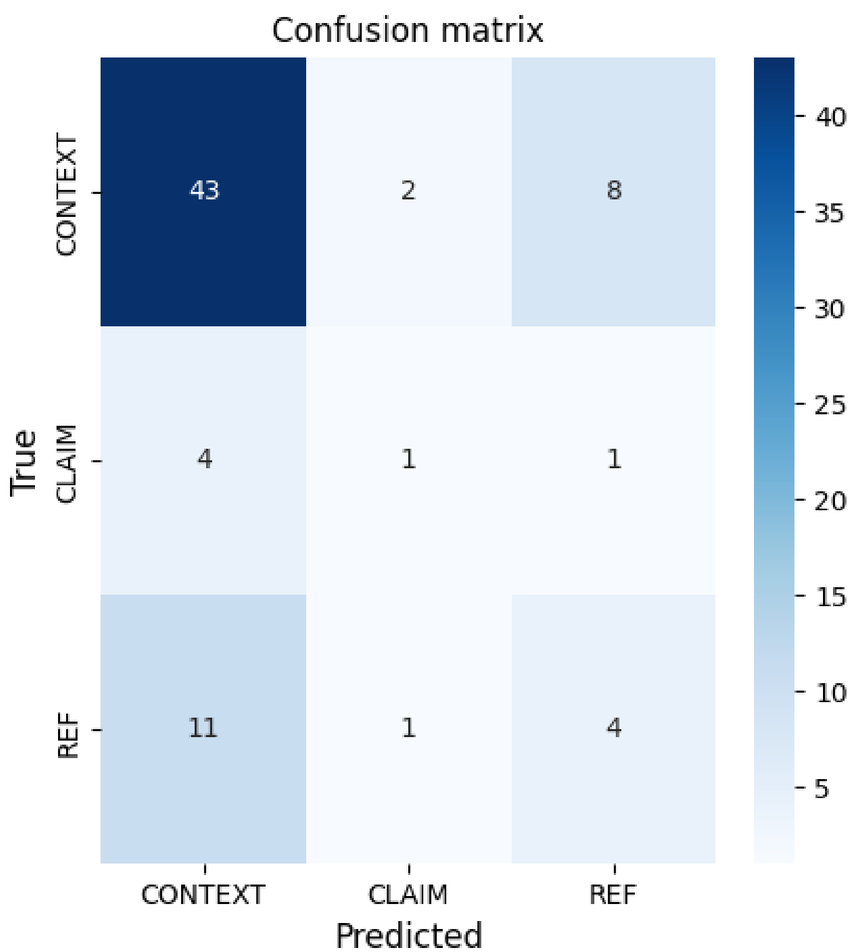
```
[I 2025-05-02 12:48:46,345] Trial 34 finished with value: 0.9523809523809523 and
parameters: {'C': 905.3328872430178, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:46,608] Trial 35 finished with value: 0.8730158730158731 and
parameters: {'C': 44.54671247160636, 'kernel': 'linear', 'gamma': 'auto'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:46,857] Trial 36 finished with value: 0.9523809523809523 and
parameters: {'C': 132.7181390007551, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:47,105] Trial 37 finished with value: 0.9476190476190476 and
parameters: {'C': 7.950294739695595, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:47,909] Trial 38 finished with value: 0.8730158730158731 and
parameters: {'C': 350.459499565562, 'kernel': 'linear', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:48,238] Trial 39 finished with value: 0.5809523809523809 and
parameters: {'C': 0.2746818280210692, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:48,497] Trial 40 finished with value: 0.9523809523809523 and
parameters: {'C': 117.39445920278551, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:48,745] Trial 41 finished with value: 0.9523809523809523 and
parameters: {'C': 226.25347811858538, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:48,992] Trial 42 finished with value: 0.9523809523809523 and
parameters: {'C': 559.4091557308503, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:49,241] Trial 43 finished with value: 0.9523809523809523 and
parameters: {'C': 166.1230333587406, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:49,504] Trial 44 finished with value: 0.946031746031746 and
parameters: {'C': 2.2042867897569955, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:49,753] Trial 45 finished with value: 0.9476190476190476 and
parameters: {'C': 22.64590465248902, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:50,060] Trial 46 finished with value: 0.8730158730158731 and
parameters: {'C': 69.43611634582997, 'kernel': 'linear', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:50,510] Trial 47 finished with value: 0.4587301587301587 and
parameters: {'C': 0.02440818408596911, 'kernel': 'rbf', 'gamma': 'auto'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:50,759] Trial 48 finished with value: 0.9523809523809523 and
parameters: {'C': 275.39240240437795, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:51,101] Trial 49 finished with value: 0.9523809523809523 and
parameters: {'C': 630.3524257340938, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
```

[I 2025-05-02 12:48:51,495] Trial 50 finished with value: 0.7444444444444445 and parameters: {'C': 179.00614799554464, 'kernel': 'poly', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:51,866] Trial 51 finished with value: 0.9523809523809523 and parameters: {'C': 967.9169696231719, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:52,247] Trial 52 finished with value: 0.9523809523809523 and parameters: {'C': 461.33545064885266, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:52,903] Trial 53 finished with value: 0.29523809523809524 and parameters: {'C': 0.0025464623038578, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:53,279] Trial 54 finished with value: 0.9476190476190476 and parameters: {'C': 51.03737484609628, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:53,665] Trial 55 finished with value: 0.9523809523809523 and parameters: {'C': 296.08320221459417, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:54,050] Trial 56 finished with value: 0.9507936507936507 and parameters: {'C': 84.74432525060247, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:54,536] Trial 57 finished with value: 0.888888888888889 and parameters: {'C': 0.43112211581636933, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:54,855] Trial 58 finished with value: 0.9476190476190476 and parameters: {'C': 31.359143159599395, 'kernel': 'rbf', 'gamma': 'auto'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:55,061] Trial 59 finished with value: 0.8730158730158731 and parameters: {'C': 13.005621709850367, 'kernel': 'linear', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:55,311] Trial 60 finished with value: 0.9523809523809523 and parameters: {'C': 142.10308127827074, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:55,558] Trial 61 finished with value: 0.9523809523809523 and parameters: {'C': 231.82624779791186, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:55,820] Trial 62 finished with value: 0.9523809523809523 and parameters: {'C': 601.8351827855037, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:56,068] Trial 63 finished with value: 0.9523809523809523 and parameters: {'C': 393.37087444189467, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:56,318] Trial 64 finished with value: 0.9523809523809523 and parameters: {'C': 663.4061749273176, 'kernel': 'rbf', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:56,561] Trial 65 finished with value: 0.7444444444444445 and parameters: {'C': 192.331161653785, 'kernel': 'poly', 'gamma': 'scale'}. Best is trial 0 with value: 0.9523809523809523.

```
[I 2025-05-02 12:48:56,824] Trial 66 finished with value: 0.9523809523809523 and
parameters: {'C': 93.40513425783263, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:57,069] Trial 67 finished with value: 0.9523809523809523 and
parameters: {'C': 962.6448344084713, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:57,317] Trial 68 finished with value: 0.9523809523809523 and
parameters: {'C': 316.99656005643703, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:57,563] Trial 69 finished with value: 0.9523809523809523 and
parameters: {'C': 580.5027788202076, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:58,035] Trial 70 finished with value: 0.8730158730158731 and
parameters: {'C': 154.8072281494415, 'kernel': 'linear', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:58,281] Trial 71 finished with value: 0.9523809523809523 and
parameters: {'C': 674.7579945341059, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:58,527] Trial 72 finished with value: 0.9523809523809523 and
parameters: {'C': 408.41802107184276, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:58,786] Trial 73 finished with value: 0.9523809523809523 and
parameters: {'C': 224.7545419701397, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:59,193] Trial 74 finished with value: 0.7793650793650793 and
parameters: {'C': 0.10215945051805178, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:59,440] Trial 75 finished with value: 0.9523809523809523 and
parameters: {'C': 823.4214111252065, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:59,681] Trial 76 finished with value: 0.7444444444444445 and
parameters: {'C': 442.93687520998577, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:48:59,945] Trial 77 finished with value: 0.9523809523809523 and
parameters: {'C': 116.81217445502749, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:00,189] Trial 78 finished with value: 0.9523809523809523 and
parameters: {'C': 303.44275818880556, 'kernel': 'rbf', 'gamma': 'auto'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:00,432] Trial 79 finished with value: 0.9523809523809523 and
parameters: {'C': 504.72912086066407, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:00,681] Trial 80 finished with value: 0.9507936507936507 and
parameters: {'C': 57.29861002063873, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:00,949] Trial 81 finished with value: 0.9523809523809523 and
parameters: {'C': 669.6118609778363, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
```

```
[I 2025-05-02 12:49:01,194] Trial 82 finished with value: 0.9523809523809523 and
parameters: {'C': 268.3550532926017, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:01,439] Trial 83 finished with value: 0.9523809523809523 and
parameters: {'C': 408.9992435315794, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:01,685] Trial 84 finished with value: 0.9523809523809523 and
parameters: {'C': 923.0533072897144, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:01,950] Trial 85 finished with value: 0.9523809523809523 and
parameters: {'C': 157.85524331970427, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:02,200] Trial 86 finished with value: 0.946031746031746 and
parameters: {'C': 1.8439514171282854, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:02,520] Trial 87 finished with value: 0.8730158730158731 and
parameters: {'C': 77.14891749306938, 'kernel': 'linear', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:02,768] Trial 88 finished with value: 0.9523809523809523 and
parameters: {'C': 734.7405166227168, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:03,027] Trial 89 finished with value: 0.7444444444444445 and
parameters: {'C': 320.6173305274107, 'kernel': 'poly', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:03,467] Trial 90 finished with value: 0.29523809523809524
and parameters: {'C': 0.009287912040448817, 'kernel': 'rbf', 'gamma': 'auto'}.
Best is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:03,716] Trial 91 finished with value: 0.9523809523809523 and
parameters: {'C': 208.29981293392711, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:03,976] Trial 92 finished with value: 0.9523809523809523 and
parameters: {'C': 111.49493455424432, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:04,223] Trial 93 finished with value: 0.9476190476190476 and
parameters: {'C': 40.21671979813042, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:04,471] Trial 94 finished with value: 0.9523809523809523 and
parameters: {'C': 398.5723232052009, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:04,719] Trial 95 finished with value: 0.9523809523809523 and
parameters: {'C': 540.9991390737517, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:05,128] Trial 96 finished with value: 0.9523809523809523 and
parameters: {'C': 978.8093660127504, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:05,509] Trial 97 finished with value: 0.9523809523809523 and
parameters: {'C': 240.33280743393698, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
```

```
[I 2025-05-02 12:49:05,895] Trial 98 finished with value: 0.9523809523809523 and
parameters: {'C': 160.97009741474628, 'kernel': 'rbf', 'gamma': 'scale'}. Best
is trial 0 with value: 0.9523809523809523.
[I 2025-05-02 12:49:06,298] Trial 99 finished with value: 0.9523809523809523 and
parameters: {'C': 519.8700388086686, 'kernel': 'rbf', 'gamma': 'scale'}. Best is
trial 0 with value: 0.9523809523809523.

Best hyperparameters: {'C': 420.8077194143475, 'kernel': 'rbf', 'gamma':
'scale'}
Best accuracy: 0.9523809523809523
              precision    recall  f1-score   support

       CLAIM       0.74      0.81      0.77        53
     CONTEXT       0.25      0.17      0.20         6
         REF       0.31      0.25      0.28        16

    accuracy                           0.64        75
   macro avg       0.43      0.41      0.42        75
weighted avg       0.61      0.64      0.62        75
```



On passe cette fois de 0.9023 à 0.9523 soit une aumgnetation de 5% au vue de la matrice de confusion, cela ne semble pas être le meilleure classfiieur.