Projet de Machine Learning

Fake News Detection

Mise en place

Lien avec Gdrive

```
import sys
from google.colab import drive
drive.mount('/content/gdrive')
my_local_drive='/content/gdrive/MyDrive/M1/Projet/donnees'

%cd $my_local_drive
%pwd
%ls
```

```
Mounted at /content/gdrive
/content/gdrive/MyDrive/M1/Projet/donnees
HAI817_Projet_test.csv   Models/            Projet_HAI817I.ipynb
HAI817_Projet_train.csv  MyNLPUtilities.py  __pycache__/
```

Importation des différentes librairies requises

```
# Librairie a installer
# """ Commenter cette ligne pour installer
!python -m pip install --upgrade pip
!pip install langdetect
!pip install contractions
!pip install wordcloud==1.8.2.2
!pip install spacy
!pip install nltk

!python -m spacy download en_core_web_lg
!python -m spacy download en_core_web_sm
!python -m spacy download fr_core_news_sm
!python -m spacy download es_core_news_sm
!pip install spacy-language-detection
!pip install spacy_langdetect==0.1.1
!python -m spacy download fr_core_news_md


!pip install iso639
# """
```

```
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->fr-
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!=1.8.1,<
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->s
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spac
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.6.
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.10/dist-packages (from thinc<8.2.0,>=8.1.8->spacy<3.
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.10/dist-packages (from thinc<8.2.0,>=8.1.8->sp
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.10/dist-packages (from typer<0.8.0,>=0.3.0->spacy<3
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->spacy<3.6.0,>=3.5.0->fr-
Installing collected packages: fr-core-news-md
Successfully installed fr-core-news-md-3.5.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manag
✓ Download and installation successful
You can now load the package via spacy.load('fr_core_news_md')
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting iso639
  Downloading iso639-0.1.4.tar.gz (11 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: iso639
  Building wheel for iso639 (setup.py) ... done
  Created wheel for iso639: filename=iso639-0.1.4-py3-none-any.whl size=11188 sha256=f4924c1c6a6b84bc27012798068804e2c147a15be42
  Stored in directory: /root/.cache/pip/wheels/a5/b9/38/a15d4cd0f6d25de2d8731726b1d6ec9efcd3e04db912dd4e7f
Successfully built iso639
```

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

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from sklearn import metrics
from sklearn.pipeline import Pipeline
from sklearn.base import TransformerMixin
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer


# librairies des classifiers utilisés
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import GridSearchCV
from sklearn.utils import resample
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_fscore_support
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
from MyNLPUtilities import *

# librairies générales
import pickle
from scipy.stats import randint
import string
import time
import base64
import sys
import re
import os

import contractions

# librairie BeautifulSoup
from bs4 import BeautifulSoup
```

```
# librairie affichage
import wordcloud

## detection de language
import langdetect
from langdetect import detect

import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('tagsets')
nltk.download("stopwords")
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download("stopwords")
from nltk import sent_tokenize
from nltk import RegexpParser
from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk.corpus import wordnet as wn

import spacy
from spacy.tokens import Span
from spacy.lang.fr import French
from spacy_langdetect import LanguageDetector
from spacy.tokens import Doc
from spacy.language import Language
from spacy_language_detection import LanguageDetector

import iso639

import en_core_web_sm
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package tagsets to /root/nltk_data...
[nltk_data]   Unzipping help/tagsets.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Récupération du dataset et formatage selon la tâche de classification

```
df_train = pd.read_csv('HAI817_Projet_train.csv')
df_test = pd.read_csv('HAI817_Projet_test.csv')

# Rennomage des classes pour plus de lisibilité
df_train["our rating"] = df_train["our rating"].replace({"true": "VRAI",
                                                         "false": "FAUX",
                                                         "mixture": "MIXTE",
                                                         "other": "AUTRE"})

df_test["our rating"] = df_test["our rating"].replace({"true": "VRAI",
                                                       "false": "FAUX",
                                                       "mixture": "MIXTE",
                                                       "other": "AUTRE"})

def tacheDeClassification(df, num):
    if num == 1:
        # Sélectionner que les lignes avec "true" ou "false" dans la colonne "our rating"
        df = df[(df['our rating'] == "VRAI") | (df['our rating'] == "FAUX")]

    elif num == 2:
        df["our rating"] = df["our rating"].replace({"VRAI": "VRAIouFAUX",
                                                     "FAUX": "VRAIouFAUX"})

        df = df[(df['our rating'] == "VRAIouFAUX") | (df['our rating'] == "AUTRE")]

    return df.reset_index(drop=True)

tache = 1
# 1 : VRAI vs FAUX
```

```
# 2 : VRAIouFAUX vs AUTRE
# 3 : VRAI vs FAUX vs AUTRE vs MIXTURE
dfFakeNews = tacheDeClassification(df_train, tache)
dfFakeNewsTest = tacheDeClassification(df_test, tache)

print(dfFakeNews['our rating'].value_counts())
print(dfFakeNewsTest['our rating'].value_counts())
```

```
FAUX    578
VRAI    211
Name: our rating, dtype: int64
FAUX    315
VRAI    210
Name: our rating, dtype: int64
```

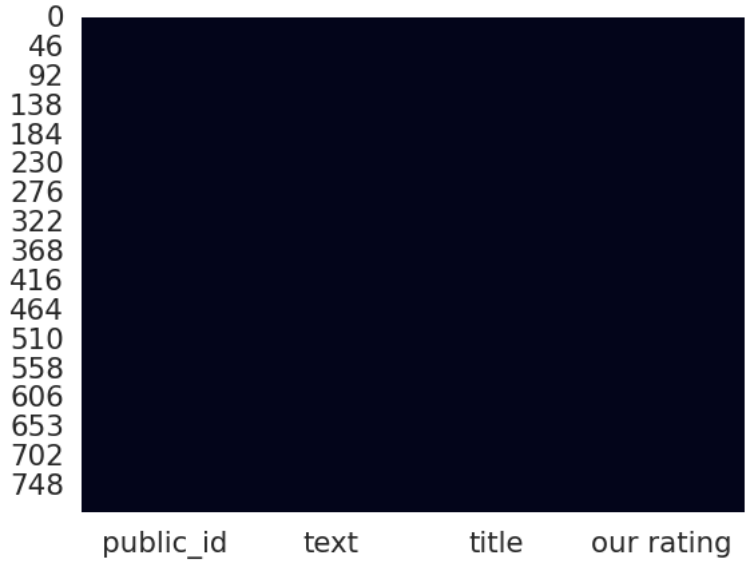
Pré-traitements

Visualisation des valeurs manquantes

On supprime ici les lignes avec au moins une colonne vide pour s'assurer qu'il n'y ai pas de données manquantes

```
dfFakeNews = dfFakeNews.dropna()
dfFakeNewsTest = dfFakeNewsTest.dropna()

sns.heatmap(dfFakeNews.isnull(), cbar=False)
```

```
<Axes: >
```



Rééquilibrage des classes

Fonction de rééquilibrage des classes

```
# Ajout de valeurs pour équilibrage
def upSampling(df):
    # Détection du nombre de classes
    classes = df['our rating'].unique()
    num_classes = len(classes)

    if num_classes < 2:
        raise ValueError("Le nombre de classes doit être au moins 2")

    # Détermination de la taille de la classe majoritaire
    counts = df['our rating'].value_counts().sort_values()
    majority_size = counts.iloc[-1]

    # Upsampling des classes minoritaires
    df_upsampled = []
    for c in classes:
        if counts[c] == majority_size:
            df_c = df[df['our rating'] == c]
        else:
```

```
            df_c = resample(df[df['our rating'] == c], replace=True, n_samples=majority_size, random_state=42)
        df_upsampled.append(df_c)

    # Concaténation des classes
    df_upsampled = pd.concat(df_upsampled)
    df_upsampled = df_upsampled.reset_index(drop=True)

    return df_upsampled




# Suppression de valeurs pour équilibrage
# Ajout de valeurs pour équilibrage
def downSampling(df):
    # Détection du nombre de classes
    classes = df['our rating'].unique()
    num_classes = len(classes)

    if num_classes < 2:
        raise ValueError("Le nombre de classes doit être au moins 2")

    # Détermination de la taille de la classe minoritaire
    counts = df['our rating'].value_counts().sort_values()
    minority_size = counts.iloc[0]

    # Downsampling des classes majoritaires
    df_downsampled = []
    for c in classes:
        if counts[c] == minority_size:
            df_c = df[df['our rating'] == c]
        else:
            df_c = resample(df[df['our rating'] == c], replace=False, n_samples=minority_size, random_state=42)
        df_downsampled.append(df_c)

    # Concaténation des classes
    df_downsampled = pd.concat(df_downsampled)
    df_downsampled = df_downsampled.reset_index(drop=True)

    return df_downsampled
```

Rééquilibrage des classes

```
# Copie du dataFrame
dfPret = dfFakeNews.copy()

print ("Nombre d'occurrences par classe :")
print(dfPret['our rating'].value_counts())
print ("Shape: " + str(dfPret.shape))

# Application du downSampling sur le dataFrame
dfPret = downSampling(dfPret)
# dfPret = upSampling(dfPret)

print ("Après downsampling :")
print(dfPret['our rating'].value_counts())
```

```
Nombre d'occurrences par classe :
FAUX    571
VRAI    206
Name: our rating, dtype: int64
Shape: (777, 4)
Après downsampling :
FAUX    206
VRAI    206
Name: our rating, dtype: int64
```

Automatisation des pré-traitements

Définition des fonctions

```
nlp = spacy.load("en_core_web_sm")

# Chargement de la liste de stopwords en anglais
the_stopwords = set(stopwords.words("english"))
```

```
# 0. Supprimer les textes qui se sont pas en anglais
# (à appliquer seulement quand on concatène les titres et les textes ensembles)
def supp_not_english_words(textes):
    tab_langue_textes = []
    # Parcours des textes dans le tableau donnée
    for txt in textes:
        tab_langue_textes.append(langdetect.detect(txt)) # detection de la langue
    # Parcours des langues
    for i in range(len(tab_langue_textes)):
        if tab_langue_textes[i] != 'en': # si la langue n'est pas anglaise
            textes.pop(i) # on supprime le texte dans le tableau d'origine
    return textes


# 1. Generation de wordclouds pour affichage
def generate_wordcloud(texts):
    for text in texts:
        # Detection de la langue
        lang = detect(text)

        # Affichage des word clouds
        wc = wordcloud.WordCloud(
            background_color="black", max_words=100, max_font_size=35
        )
        wc = wc.generate(str(text))
        fig = plt.figure(num=1)
        # Affichage du graphique
        plt.axis("off")
        plt.imshow(wc, cmap=None)
        plt.title(f"Langue: {lang}")
        plt.show()


# 2. Mettre sous forme de token
def tokenize_titles(texts):
    word_tokens_list_texts = []
    # Parcours des textes
    for txt in texts:
        # tokenisation de chaque texte
        word_tokens = word_tokenize(txt)
        word_tokens_list_texts.append(word_tokens)  # ajout à la liste des tokens
    return word_tokens_list_texts


# 3. Donner la categorie des mots
def get_pos_tags(word_tokens_list):
    cate_word_tokens = []
    # Parcours du tableau de token
    for sentence in word_tokens_list:
        sentence_str = " ".join(sentence) # formation de la phrase courante
        doc = nlp(sentence_str) #
        pos_tags = [
            (token.text, token.pos_) # mot + categorie
            for token in doc
        ] # formation du tableau de chaque mot avec sa catégorie
        cate_word_tokens.append(pos_tags) # ajout dans le tableau de tous les textes
    return cate_word_tokens


# 4. Filtrer les mots en focntion de leurs types selective_pos
def filter_by_pos(text, selective_pos):
    """
    - text (list): une liste de phrases où chaque phrase est une liste de mots.
    - selective_pos (list): une liste de catégories grammaticales à conserver.
            Exemple : ['PROPN', 'NOUN', 'VERB', 'ADJ', 'ADV', 'PRON']
    """
    filter_cate_word_text = []
    for phrase in text:
        tab_courant = []
        for word in phrase:
            if word[1] in selective_pos:
                tab_courant.append(word)
        filter_cate_word_text.append(tab_courant)
    return filter_cate_word_text


# 5. Suppression des majuscule sur les mots qui ne sont pas des noms propres sauf si on veut tout en minuscule (is_all_min)
def lowercase_filter(filter_cate_word_text, lower, is_all_min):
    min_filter_cate_text = []
    for phrase in filter_cate_word_text:
        mots_minuscules = []
        for word in phrase:
```

```
            if lower and (is_all_min or word[1] != "PROPN"):
                mots_minuscules.append(word[0].lower())  # transformer
            else:
                mots_minuscules.append(word[0])
        min_filter_cate_text.append(mots_minuscules)
    return min_filter_cate_text



# 6. Supprimer les stop words
def remove_stopwords(text):
    # Tokenisation des phrases
    word_tokens_list = []
    for sentence in text:
        word_tokens_list.append(word_tokenize(" ".join(sentence)))
    # Suppression des stopwords et join des tokens
    final_list_sentence = []
    for word_tokens in word_tokens_list:
        tokens = [word for word in word_tokens if word not in the_stopwords]
        # sentence = " ".join(tokens)
        final_list_sentence.append(tokens)
    return final_list_sentence


# 7. Lemmatisation des mots
def lemmatize_text(text):
    lemmatizer = WordNetLemmatizer()
    lemmas_final_word_text = []
    for phrase in text:
        doc_lemme = nlp(" ".join(phrase))
        lemmas_courant = []
        for token in doc_lemme:
            if token.text.endswith("-"):
                continue  # enleve les mots composé avec un tiret
            lemmas_courant.append(lemmatizer.lemmatize(token.text))  # ajout a la liste en lematisant le mot
        lemmas_final_word_text.append(lemmas_courant)  # ajout a la grande liste
    return lemmas_final_word_text


## Tests
# tok = tokenize_titles(dfPret["title"])
# a = get_pos_tags(tok[:2])
# b = filter_by_pos(a, ['PROPN', 'VERB'])
# c = lowercase_filter(b, False)
# d = remove_stopwords(c)
# e = lemmatize_text(d)
# print(e)
```

Fonction d'automatisation des pré-traitements

```
def preTraitement(
    texte,
    selective_pos=['PROPN', 'NOUN', 'VERB', 'ADJ', 'ADV', 'PRON'], # catégories grammaticales
    filter_category=False,       # filtre les categories de mots qu'on veut
    supp_all_maj=False,          # supprimer toutes les majuscules des textes
    supp_maj=False,              # supprimer les majuscules sur tous les mots sauf les noms propres
    supp_stopwords=False,        # supprimer les stopwords
    lemmatisation=False          # mettre les mots sous leur forme de base
    ):
    """
    Entrée : liste de liste de mots
    Sortie : liste de liste de mots
    """

    # generate_wordcloud(texte)
    tab_texte = tokenize_titles(texte)      # met sous forme de token (tableau de mots)

    if filter_category:
        tab_texte = get_pos_tags(tab_texte) # donne la categorie des mots (LONG)
        tab_texte = filter_by_pos(tab_texte, selective_pos) # ne garde que les selective_pos
        # suppression des MaJ ici car besoin des catégories des mots
        tab_texte = lowercase_filter(tab_texte, supp_maj, supp_all_maj)  # Mise en minuscule

    if supp_stopwords:
        tab_texte = remove_stopwords(tab_texte)  # supprime les stop-words

    if lemmatisation:
        tab_texte = lemmatize_text(tab_texte)   # met sous forme de lemme

    return [" ".join(txt) for txt in tab_texte]
```

Traitement

Transformation du jeu de données

Concaténer les titres aux textes pour pouvoir appliquer sur les deux

Classification

Pré-traitement des variables

```
Xtitle = preTraitement(dfPret["title"],
                       selective_pos=['NOUN', 'VERB', 'ADJ'],
                       filter_category=False,
                       supp_maj=True,
                       supp_all_maj=True,
                       supp_stopwords=True,
                       lemmatisation=True)

Xtext = preTraitement(dfPret["text"],
                       selective_pos=['NOUN', 'VERB', 'ADJ'],
                       filter_category=False,
                       supp_maj=True,
                       supp_all_maj=True,
                       supp_stopwords=True,
                       lemmatisation=True)
```

Concaténation du texte à traiter

```
X_title_pret = []
X_text_pret = []
X_full_pret = []
for titre, texte in zip(Xtitle, Xtext):
    X_title_pret.append(titre)
    X_text_pret.append(texte)
    X_full_pret.append(titre + " " + texte)

# Variable de prédiction
y_pret = dfPret["our rating"]
```

Création du jeu d'apprentissage

Traitement des données avec TF_IDF

```
def transformationTFIDF(X_pret, y_entree):
    # Transformation du texte en données utilisables par les classifieurs
    tf = TfidfVectorizer(max_features=None, # Mettre un cap sur la taille du tableau
                         max_df=0.8)         # limiter aux mots apparaissant dans moins de 80% des docs
    X_transformed = tf.fit_transform(X_pret).toarray()

    # Séparation du jeu de données
    trainsize = 0.8
    testsize = 1 - trainsize
    seed = 30

    X_train, X_test, y_train, y_test = train_test_split(X_transformed,
                                                        y_entree,
                                                        train_size=trainsize,
                                                        random_state=seed,
                                                        test_size=testsize)

    return X_transformed, X_train, X_test, y_train, y_test
```

Comparaison des classifieurs

```
models = []
models.append(('MultinomialNB',MultinomialNB()))
```

```
models.append(('LR', LogisticRegression(solver='lbfgs')))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('RandomForest', RandomForestClassifier()))
models.append(('SVM', SVC()))
seed = 7

def compareClassifiers(X_entree, y_entree):

    print("\n==================================")
    X_transformed, X_train, X_test, y_train, y_test = transformationTFIDF(X_entree, y_entree)

    for name, model in models:
        print("╔═══════"+len(name)*"="+"═══════╗")
        print("║       "+name+"       ║")
        print("╚═══════"+len(name)*"="+"═══════╝")
        start_time = time.time()
        clf = model
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        classes = np.unique(y_test)
        print(f"Accuracy : {round(accuracy_score(y_test,y_pred),3)}, temps : {round(time.time() - start_time,2)}s")
        # cnf_matrix = confusion_matrix(y_test,y_pred)
        # plot_confusion_matrix(cnf_matrix, classes)

    return X_transformed, X_train, X_test, y_train, y_test


X_full_transformed, X_train, X_test, y_train, y_test  = compareClassifiers(X_full_pret, y_pret)
```

```
==================================
╔═══════════════════════╗
║       MultinomialNB       ║
╚═══════════════════════╝
Accuracy : 0.675, temps : 0.09s
╔═══════════════╗
║       LR      ║
╚═══════════════╝
Accuracy : 0.747, temps : 0.56s
╔═══════════════╗
║       KNN     ║
╚═══════════════╝
Accuracy : 0.578, temps : 0.41s
╔═══════════════╗
║       CART    ║
╚═══════════════╝
Accuracy : 0.554, temps : 1.45s
╔═══════════════════════╗
║       RandomForest        ║
╚═══════════════════════╝
Accuracy : 0.675, temps : 1.49s
╔═══════════════╗
║       SVM     ║
╚═══════════════╝
Accuracy : 0.747, temps : 4.38s
```

Comparaison des classifieurs avec des K-fold

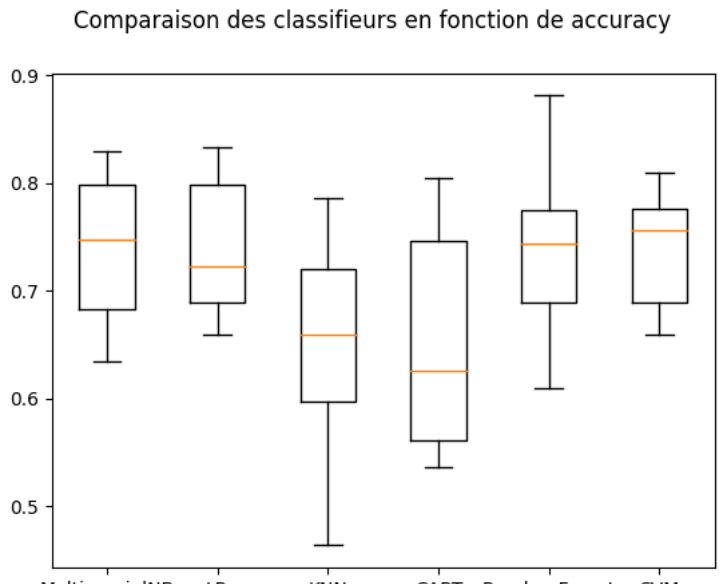
```
def compare_models(models, X_transformed, y, score):
    seed = 7

    allresults = []
    results = []
    names = []

    for name, model in models:
        print("==================================")
        print(f"Evaluation de {name}")
        start_time = time.time()

        # cross validation en 10 fois
        kfold = KFold(n_splits=10, random_state=seed, shuffle=True)

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

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

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

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

        # affichage résultats par score décroissant
        print("\n Résultats :")
        for result in allresults:
            print(f"{result.name} ({round(result.timespent,1)}s) :")
            print(f" - {score} : {round(result.scoremean,3)}")
            print(f" - equart-type : {round(result.stdresult,3)}")

        # affichage graphique
        fig = plt.figure()
        fig.suptitle(f"Comparaison des classifieurs en fonction de {score}")
        ax = fig.add_subplot(111)
        plt.boxplot(results)
        ax.set_xticklabels(names)
        ax.tick_params(axis='x', labelsize=10)

        return allresults[0].name
```

Comparaison des classifieurs avec un k-fold

```
best_accuracy = compare_models(models, X_full_transformed, y_pret, "accuracy")
```

```
Evaluation de SVM
              precision    recall  f1-score   support

        FAUX       0.69      0.86      0.77       206
        VRAI       0.82      0.62      0.71       206

    accuracy                           0.74       412
   macro avg       0.75      0.74      0.74       412
weighted avg       0.75      0.74      0.74       412


 Résultats :
SVM (58.8s) :
 - accuracy : 0.74
 - equart-type : 0.051
LR (13.7s) :
 - accuracy : 0.738
 - equart-type : 0.061
MultinomialNB (2.0s) :
 - accuracy : 0.738
 - equart-type : 0.066
RandomForest (18.6s) :
 - accuracy : 0.737
 - equart-type : 0.074
KNN (2.5s) :
 - accuracy : 0.655
 - equart-type : 0.091
CART (10.5s) :
 - accuracy : 0.653
 - equart-type : 0.101
```



Etude du meilleur classifieur

Définition des paramètres

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

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

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

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

  'SVM' : [{
      'C': [0.1,1, 10, 100],
```

```
        'gamma': [1,0.1,0.01,0.001],
        'kernel': ['rbf', 'poly', 'sigmoid']}
             ],

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

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

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

    'RandomForest':[{'n_estimators': [4, 6, 9],
           'max_features': ['log2', 'sqrt','auto'],
           'criterion': ['entropy', 'gini'],
           'max_depth': [10, 20],
           'min_samples_split': [2, 3, 5],
           'min_samples_leaf': [1,5,8]
           }],

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

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

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

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

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

Récupération des paramètres du meilleur classifieur

```
print(f"Meilleur classifieur pour l'accuracy : {best_accuracy}")

for name, model in models:
    if best_accuracy == name:
        best_model_name = name
        best_model = model
        best_params_lite = params_lite[name]
        best_params = params[name]  # Long 💀💀💀
        break
```

```
 Meilleur classifieur pour l'accuracy : SVM
```

Variation des hyperparamètres du meilleur classifieur

Fonction de test des hyperparamètres

```
def testDesHyperParams(model, params, score):
    gd_sr = GridSearchCV(estimator=model,
                         param_grid=params,
                         scoring=score,
                         cv=5,
                         n_jobs=-1,
                         return_train_score=True)

    gd_sr.fit(X_train, y_train)

    return gd_sr
```

Tests des hyperparamètres sur les différents scores

```
score = "accuracy"

gridSearch = testDesHyperParams(best_model, best_params_lite, score)
best_estimator = gridSearch.best_estimator_

print(f"meilleur score pour {score} : {round(gridSearch.best_score_, 3)}")
print(f"meilleurs paramètres : {gridSearch.best_params_}")
print(f"meilleur estimateur : {best_estimator}")
```

```
meilleur score pour accuracy : 0.784
meilleurs paramètres : {'C': 1, 'gamma': 0.001, 'kernel': 'linear'}
meilleur estimateur : SVC(C=1, gamma=0.001, kernel='linear')
```

Création du pipeline

Fonction pour la création du pipeline

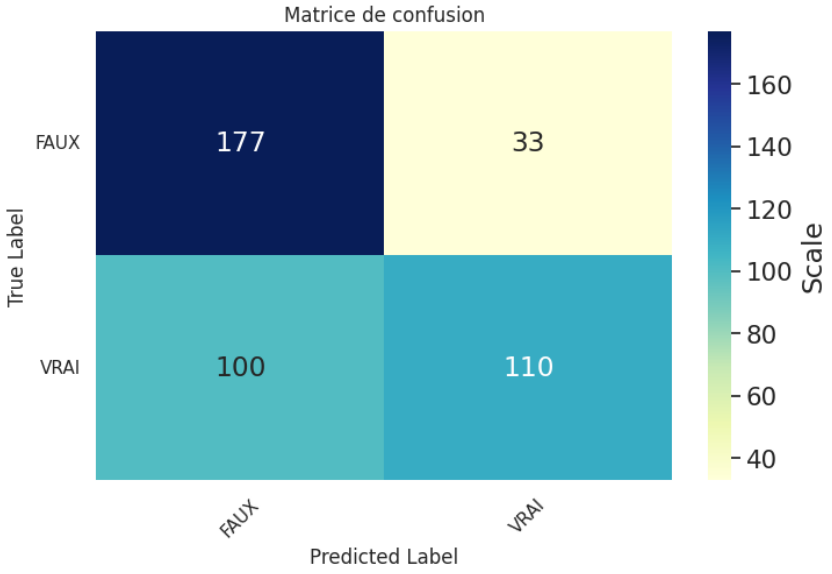
```
def grosPipeline(toTrain, toPredict, estimator):

    # Application du downSampling sur les dataFrames
    toTrain = downSampling(toTrain)
    toPredict = downSampling(toPredict)

    XtitleTrain = toTrain["title"]
    XtextTrain = toTrain["text"]
    XallTrain = []
    for titre, texte in zip(XtitleTrain, XtextTrain):
        XallTrain.append(titre + " " + texte)

    XtitleTest = toPredict["title"]
    XtextTest = toPredict["text"]
    XallTest = []
    for titre, texte in zip(XtitleTest, XtextTest):
        XallTest.append(titre + " " + texte)

    X_train = preTraitement(XallTrain,
                            selective_pos=['NOUN', 'VERB', 'ADJ'],
                            filter_category=False,
                            supp_maj=True,
                            supp_all_maj=True,
                            supp_stopwords=True,
                            lemmatisation=True)

    X_test = preTraitement(XallTest,
                           selective_pos=['NOUN', 'VERB', 'ADJ'],
                           filter_category=False,
                           supp_maj=True,
                           supp_all_maj=True,
                           supp_stopwords=True,
                           lemmatisation=True)


    y_train = toTrain["our rating"]
    y_test = toPredict["our rating"]

    # X_train, X_test, y_train, y_test = train_test_split(X_train,
    #                                                     y_train,
    #                                                     train_size=0.8,
    #                                                     random_state=40,
    #                                                     test_size=0.2)


    tf = TfidfVectorizer()
    X_train_transformed = tf.fit_transform(X_train).toarray()
    X_test_transformed = tf.transform(X_test).toarray()

    clf = RandomForestClassifier()

    # Entrainer le classifieur sur le jeu de données d'entraînement
    clf.fit(X_train_transformed, y_train)

    # Faire des prédictions sur le jeu de données de test
    y_pred = clf.predict(X_test_transformed)

    # Calculer les métriques d'évaluation
    class_names = toTrain["our rating"].unique().tolist()
    p, r, f, s = precision_recall_fscore_support(y_test, y_pred, labels=class_names)

    print(f"Performances du pipe : accuracy = {round(accuracy_score(y_test, y_pred),3)}")
```

```
# Affichage les résultats
print("==================================")
print("Classification report:")
print(classification_report(y_test, y_pred))
print("==================================")
print("Precision, recall, f1-score et support par classe :")
for i, (lbl, a, b, c, d) in enumerate(zip(class_names, p, r, f, s)):
  print(f"Classe {lbl} : {round(a,2)}, {round(b,2)}, {round(c,2)}, {d}")
print("==================================")
print("Matrice de confusion : ")
plot_confusion_matrix(confusion_matrix(y_test,y_pred), np.unique(y_test))
```

Utilisation du pipeline sur le jeu de test

```
dfTrain = dfFakeNews.copy()
dfTest = dfFakeNewsTest.copy()

grosPipeline(dfTrain, dfTest, best_estimator)
```

```
Performances du pipe : accuracy = 0.683
==================================
Classification report:
              precision    recall  f1-score   support

        FAUX       0.64      0.84      0.73       210
        VRAI       0.77      0.52      0.62       210

    accuracy                           0.68       420
   macro avg       0.70      0.68      0.68       420
weighted avg       0.70      0.68      0.68       420

==================================
Precision, recall, f1-score et support par classe :
Classe FAUX : 0.64, 0.84, 0.73, 210
Classe VRAI : 0.77, 0.52, 0.62, 210
==================================
Matrice de confusion :
```



Mise en production du modèle

Fonction pour la mise en production du modèle

```
def miseEnProduction(toPredict, filename, estimator):

    # Application du downSampling sur le dataFrame
    toPredict = downSampling(toPredict)

    Xtitle = toPredict["title"]
    Xtext = toPredict["text"]
    Xall = []
    for titre, texte in zip(Xtitle, Xtext):
```

```
    for titre, texte in zip(xtitle, xtext):
        Xall.append(titre + " " + texte)

    X_pret = preTraitement(Xall,
                           selective_pos=['NOUN', 'VERB', 'ADJ'],
                           filter_category=False,
                           supp_maj=True,
                           supp_all_maj=True,
                           supp_stopwords=True,
                           lemmatisation=True)


    y = toPredict["our rating"]

    tf = TfidfVectorizer()
    X_transformed = tf.fit_transform(X_pret).toarray()

    scaler = StandardScaler()
    scaler.fit(X_transformed)

    model = estimator
    model.fit(X_transformed, y)

    pickle.dump(model, open(f"{filename}-model.pkl", 'wb'))
    pickle.dump(tf, open(f"{filename}-vectorizer.pkl", 'wb'))
    pickle.dump(scaler, open(f"{filename}-scaler.pkl", 'wb'))

    print(f"Modèle sauvegardé dans : {filename}")
```

Fonction de prédiction utilisant le modèle enregistré

```
def predict_fake_news(article, filename):

    model = pickle.load(open(f"{filename}-model.pkl", 'rb'))
    vectorizer = pickle.load(open(f"{filename}-vectorizer.pkl", 'rb'))
    scaler = pickle.load(open(f"{filename}-scaler.pkl", 'rb'))

    X_entree = preTraitement(article,
                           selective_pos=['NOUN', 'VERB', 'ADJ'],
                           filter_category=False,
                           supp_maj=True,
                           supp_all_maj=True,
                           supp_stopwords=True,
                           lemmatisation=True)

    # tf = TfidfVectorizer()
    X_transformed = vectorizer.transform(X_entree).toarray()

    # Appliquer le scaler sur le vecteur de caractéristiques
    X_test_scaled = scaler.transform(X_transformed)

    # Prédire la classe du texte en entrée
    prediction = model.predict(X_test_scaled)

    # Afficher la prédiction
    for i in range(len(prediction)):
      print(f"L'article {i+1} est : {prediction[i]}")
```

Mise en production du modèle avec le jeu de données au complet

```
filename = "Models/FakeNewsTrue-False"
if not os.path.exists("Models"):
    os.makedirs("Models")

dfProd = dfFakeNews.copy()

miseEnProduction(dfProd, filename, best_estimator)
```

```
Modèle sauvegardé dans : Models/FakeNewsTrue-False
```

Prédiction d'articles avec le modèle

Prédiction d'un vrai article du New York Times

```
articles_a_tester = []
```

```
# Articles vrais (New York Time)
title1 = u"""Specter of Trump Loosens Tongues, if Not Purse Strings, in Silicon Valley"""
text1 = u"""After years of scorning the political process, Silicon Valley has leapt into the fray. The prospect of a President Donald J.
title2 = u"""Many in U.K. Greet King Charles’s Coronation With a ‘Take It or Leave It’ Shrug"""
text2 = u"""When King Charles III is crowned on Saturday, he will undergo a ritual so rare in modern British history that it last occurre
title3 = u"""Airman in Leaks Case Worked on a Global Network Essential to Drone Missions"""
text3 = u"""WASHINGTON — On an Air National Guard base in Cape Cod, Mass., more than 1,200 military service members and civilians maintai

# Articles faux (WELFake Dataset)
title4 = u"""Schumer calls on Trump to appoint official to oversee Puerto Rico relief"""
text4 = u"""WASHINGTON (Reuters) - Charles Schumer, the top Democrat in the U.S. Senate, called on President Donald Trump on Sunday to na
title5 = u"""No Change Expected for ESPN Political Agenda Despite Huge Subscriber Decline - Breitbart"""
text5 = u"""As more and more sports fans turn off ESPN to protest the networkâ€™s  social and political agenda, parent company Disneyâ€"
title6 = u"""U.N. seeks humanitarian pause in Sanaa where streets "battlegrounds"""
text6 = u"""GENEVA (Reuters) - The United Nations called on Monday for a humanitarian pause in the Yemeni capital of Sanaa on Tuesday to

articles_a_tester.append(title1 + text1)
articles_a_tester.append(title2 + text2)
articles_a_tester.append(title3 + text3)
articles_a_tester.append(title4 + text4)
articles_a_tester.append(title5 + text5)
articles_a_tester.append(title6 + text6)

predict_fake_news(articles_a_tester, filename)
```

```
L'article 1 est : VRAI
L'article 2 est : VRAI
L'article 3 est : VRAI
L'article 4 est : VRAI
L'article 5 est : FAUX
L'article 6 est : FAUX
```

✓ 0s completed at 8:03 PM