Université Sultan Moulay Slimane
École Nationale des Sciences Appliquées
- Khouribga -

Détection de Fausses Nouvelles : Performances Comparées de Modèles d'Apprentissage Automatique

Réalisé par :
Abdellah Boulidam

Encadré par :
Sara Baghdadi

Année Universitaire 2024-2025

Table des Matières

Table des Matières 1

Liste des Figures 4

1 Introduction 6
- 1.1 Contexte et Motivation . . . 6
  - 1.1.1 Le Problème des Fausses Nouvelles . . . 6
  - 1.1.2 Importance de la Détection Automatisée . . . 6
- 1.2 Objectifs du Projet . . . 6
  - 1.2.1 Objectif Principal . . . 6
  - 1.2.2 Objectifs Spécifiques de ce Rapport . . . 7
- 1.3 Portée du Rapport . . . 7
- 1.4 Structure du Rapport . . . 7

2 Acquisition des Données et Exploration Initiale 8
- 2.1 Sources des Données . . . 8
  - 2.1.1 Jeu de Données "Fake.csv" . . . 8
  - 2.1.2 Jeu de Données "True.csv" . . . 8
- 2.2 Chargement des Données et Inspection Initiale . . . 8
  - 2.2.1 Bibliothèques Utilisées . . . 8
  - 2.2.2 Chargement des Jeux de Données . . . 9
  - 2.2.3 Structure Initiale des Données et Contenu . . . 9
  - 2.2.4 Identification des Colonnes Clés . . . 10

3 Méthodologie de Nettoyage et de Prétraitement des Données 11
- 3.1 Vue d'Ensemble du Pipeline de Prétraitement . . . 11
- 3.2 Configuration de l'Environnement et Bibliothèques . . . 11
  - 3.2.1 Initialisation des Ressources spaCy et NLTK . . . 12
- 3.3 Gestion des Mots Vides . . . 12
  - 3.3.1 Justification de la Suppression Complète des Mots Vides . . . 12
  - 3.3.2 Agrégation des Mots Vides de Multiples Sources . . . 12
  - 3.3.3 La Fonction `get_combined_stopwords` . . . 12

3.4 Normalisation du Texte et Stratégie de Lemmatisation 13
3.5 Traitement des Données par Lots (Chunks) 13
3.5.1 Justification du Traitement par Lots (Gestion de la Mémoire) . . . 13
3.5.2 La Fonction `process_dataframe_in_chunks` 13
3.5.3 Traitement de `Fake.csv` et `True.csv` 14
3.6 Nettoyage des Données Post-traitement 16

4 Techniques d'Ingénierie des Caractéristiques 17
4.1 Vue d'Ensemble de la Stratégie d'Ingénierie des Caractéristiques 17
4.2 TF-IDF (Term Frequency-Inverse Document Frequency) 17
4.2.1 Fréquence du Terme (TF - Term Frequency) 17
4.2.2 Fréquence Inverse de Document (IDF - Inverse Document Frequency) 18
4.2.3 Calcul du Score TF-IDF 19
4.2.4 Configuration du Vectoriseur TF-IDF Utilisé 19
4.2.5 Ajustement, Sauvegarde et Chargement du Vectoriseur TF-IDF . . 20
4.2.6 Transformation des Données Textuelles et Calcul de la Similarité Cosinus 21
4.2.7 Construction de l'Ensemble de Caractéristiques TF-IDF 22
4.3 Plongements (Embeddings) Doc2Vec 22
4.3.1 Principe Général de Doc2Vec 22
4.3.2 Avantages et Utilisation dans le Projet 23
4.3.3 Configuration et Entraînement du Modèle Doc2Vec 24
4.3.4 Inférence des Vecteurs de Document et Calcul de la Similarité Cosinus 26
4.4 Construction et Stockage de l'Ensemble Final de Caractéristiques 31
4.4.1 Augmentation des Caractéristiques TF-IDF avec la Similarité Doc2Vec 31

5 Entraînement et Évaluation des Modèles 32
5.1 Préparation des Données pour la Modélisation 33
5.1.1 Préparation pour les Modèles d'Apprentissage Automatique Classiques (Régression Logistique, SVM) 33
5.1.2 Préparation pour le Modèle d'Apprentissage Profond (Keras) 34
5.2 Fonctions Auxiliaires pour l'Évaluation des Modèles 35
5.3 Modèle 1 : Régression Logistique 36
5.3.1 Configuration et Entraînement du Modèle 36
5.3.2 Évaluation des Performances 38
5.4 Modèle 2 : Machine à Vecteurs de Support (SVM) 43
5.4.1 Configuration et Entraînement du Modèle 44
5.4.2 Évaluation des Performances 46
5.5 Modèle 3 : Apprentissage Profond (Réseau de Neurones avec Keras) 51

5.5.1 Architecture du Modèle et Configuration 52
5.5.2 Compilation et Entraînement du Modèle 54
5.5.3 Évaluation des Performances 55

6 Application Web Streamlit pour la Détection de Fausses Nouvelles 62
6.1 Fonctionnalités Principales de l'Application 62
6.2 Composants Clés et Flux de Travail Simplifié 62

7 Discussion des Résultats 65
7.1 Impact du Prétraitement et de l'Ingénierie des Caractéristiques 65
7.2 Analyse Comparative des Performances des Modèles 66

8 Conclusion et Travaux Futurs 68
8.1 Résumé des Réalisations et Contributions 68
8.2 Vue d'Ensemble des Ensembles de Caractéristiques et de l'Application . . . 69
8.3 Recommandations pour les Prochaines Étapes et Travaux Futurs 69

Liste des Figures

2.1 Cinq premières lignes du jeu de données des fausses nouvelles (`Fake.csv`). (Figure commentée) . . . 9
2.2 Cinq premières lignes du jeu de données des vraies nouvelles (`True.csv`). (Figure commentée) . . . 10

3.1 Exemple de mots vides anglais du corpus NLTK. . . . 12
3.2 Exemple et nombre de la liste des mots vides combinés et nettoyés. . . . 13
3.3 Exemple d'un article spécifique issu de `Fake.csv` avant l'application du pipeline de prétraitement . . . 15
3.4 Les mêmes exemples de données textuelles que la Figure 3.3, mais montrant le contenu des colonnes `processed_text` après l'application du pipeline de prétraitement. . . . 16
3.5 Nombre de valeurs manquantes dans le jeu de données Fake traité. . . . 16

5.1 Matrice de Confusion pour le modèle de Régression Logistique sur l'ensemble de test. . . . 39
5.2 Courbe ROC et score AUC pour le modèle de Régression Logistique. . . . 40
5.3 Courbe Précision-Rappel et score AUC-PR pour le modèle de Régression Logistique. . . . 41
5.4 Courbe d'apprentissage pour le modèle de Régression Logistique. . . . 42
5.5 Visualisation des erreurs de classification pour le modèle de Régression Logistique après réduction par ACP. . . . 43
5.6 Courbe ROC et score AUC pour le modèle SVM. . . . 48
5.7 Courbe Précision-Rappel et score AUC-PR pour le modèle SVM. . . . 49
5.8 Courbe d'apprentissage pour le modèle SVM. . . . 50
5.9 Visualisation de l'architecture du réseau de neurones Keras. (Figure à générer et à insérer) . . . 54
5.10 Courbes d'apprentissage (Accuracy et Loss) pour le modèle d'Apprentissage Profond Keras. . . . 57
5.11 Matrice de Confusion pour le modèle d'Apprentissage Profond Keras sur l'ensemble de test. . . . 58

5.12 Courbe ROC et score AUC pour le modèle d'Apprentissage Profond Keras. 59
5.13 Courbe Précision-Rappel et score AUC-PR pour le modèle d'Apprentissage Profond Keras. 60
5.14 Visualisation des erreurs de classification pour le modèle d'Apprentissage Profond Keras après réduction par ACP. 61

6.1 l'interface principale de l'application Streamlit. 63
6.2 carte de nouvelle après vérification par un modèle. 64

Chapitre 1

Introduction

1.1 Contexte et Motivation

La prolifération des médias numériques et des plateformes de réseaux sociaux a conduit à un volume sans précédent d'informations générées et diffusées. Bien que cela offre de nombreux avantages, cela présente également des défis importants, dont le principal est la propagation rapide des « fausses nouvelles » (ou *fake news*) – des informations fausses ou trompeuses présentées comme des nouvelles.

1.1.1 Le Problème des Fausses Nouvelles

Les fausses nouvelles peuvent avoir de graves conséquences dans le monde réel, influençant l'opinion publique, affectant les résultats politiques, provoquant des troubles sociaux et même impactant les marchés financiers. Leur création et diffusion délibérées pour divers motifs (politiques, financiers ou simplement pour semer la zizanie) en font un problème sociétal complexe.

1.1.2 Importance de la Détection Automatisée

Le volume et la vélocité de l'information rendent la vérification manuelle des nouvelles impraticable. Par conséquent, les systèmes automatisés de détection de fausses nouvelles, s'appuyant sur l'apprentissage automatique (Machine Learning) et le traitement du langage naturel (TALN ou NLP), sont des outils cruciaux pour aider à identifier et à atténuer l'impact de la désinformation. Ce projet vise à développer un tel système.

1.2 Objectifs du Projet

1.2.1 Objectif Principal

L'objectif principal de ce projet est de concevoir, mettre en œuvre, évaluer et démontrer un pipeline d'apprentissage automatique capable de classifier les articles de presse comme « vrais » ou « faux » en fonction de leur contenu textuel (titre et corps du texte).

1.2.2 Objectifs Spécifiques de ce Rapport

Ce rapport détaille les phases clés suivantes du projet :

— Nettoyage et prétraitement complets des données textuelles.

— Techniques avancées d'ingénierie des caractéristiques (feature engineering) pour représenter numériquement les données textuelles.

— Entraînement et évaluation de divers modèles d'apprentissage automatique.

— Développement d'une application web conviviale pour interagir avec le système de détection.

— Analyse comparative des performances de ces modèles.

1.3 Portée du Rapport

Ce rapport couvre l'ensemble du flux de travail, de l'ingestion des données brutes à l'évaluation des modèles et au déploiement de l'application. Il comprend des descriptions détaillées des jeux de données, des étapes de prétraitement, des méthodes d'extraction de caractéristiques (TF-IDF, Doc2Vec), des architectures de modèles (Régression Logistique, SVM, Réseau de Neurones), des métriques d'évaluation, des visualisations des résultats et un aperçu de l'application web Streamlit.

1.4 Structure du Rapport

Le rapport est organisé comme suit :

— **Chapitre 2 : Acquisition des Données et Exploration Initiale**

— **Chapitre 3 : Méthodologie de Nettoyage et de Prétraitement des Données**

— **Chapitre 4 : Techniques d'Ingénierie des Caractéristiques**

— **Chapitre 5 : Entraînement et Évaluation des Modèles**

— **Chapitre 6 : Application Web Streamlit pour la Détection de Fausses Nouvelles**

— **Chapitre 7 : Discussion des Résultats**

— **Chapitre 8 : Conclusion et Travaux Futurs**

Chapitre 2

Acquisition des Données et Exploration Initiale

2.1 Sources des Données

Le projet utilise deux principaux jeux de données, l'un contenant des articles étiquetés comme « faux » et l'autre contenant des articles étiquetés comme « vrais ».

2.1.1 Jeu de Données "Fake.csv"

Ce jeu de données comprend des articles de presse qui ont été identifiés ou sont généralement considérés comme faux ou très trompeurs. Les noms de fichiers contenant des underscores comme `Fake.csv` sont gérés correctement par `\texttt`.

2.1.2 Jeu de Données "True.csv"

Ce jeu de données contient des articles de presse provenant de sources réputées, considérés comme factuellement exacts.

2.2 Chargement des Données et Inspection Initiale

2.2.1 Bibliothèques Utilisées

La manipulation et l'exploration initiales des données ont été effectuées à l'aide de bibliothèques Python, principalement Pandas pour la manipulation des données et NumPy pour les opérations numériques.

```
import pandas as pd
import numpy as np
# Autres importations du notebook :
# from sklearn.feature_extraction.text import TfidfVectorizer
# from joblib import dump
# import string
# import nltk
# from nltk.corpus import stopwords
```

```
# from nltk.stem import PorterStemmer, SnowballStemmer
# import re
# import spacy
# import joblib
# from gensim.models.doc2vec import Doc2Vec, TaggedDocument
```

Listing 2.1 – Importation des bibliothèques principales de manipulation de données

2.2.2 Chargement des Jeux de Données

Les jeux de données ont été chargés dans des DataFrames Pandas.

```
document_Fake_df = pd.read_csv("Fake.csv")
document_True_df = pd.read_csv("True.csv")
```

Listing 2.2 – Chargement des jeux de données Fake et True

Note : Le notebook fourni chargeait initialement `"Fake.csv"` *à la fois dans* `document_Fake_df` *et* `document_True_df`*. Pour les besoins de ce rapport, nous supposons que l'intention était de charger* `"True.csv"` *dans* `document_True_df`*, ce qui est cohérent avec les chemins de traitement ultérieurs distincts pour les nouvelles vraies et fausses.*

2.2.3 Structure Initiale des Données et Contenu

Un premier aperçu des premières lignes de chaque jeu de données a été effectué à l'aide de la méthode `.head()`.

```
print("Aper u du DataFrame des fausses nouvelles (Fake News) :")
print(document_Fake_df.head())
```

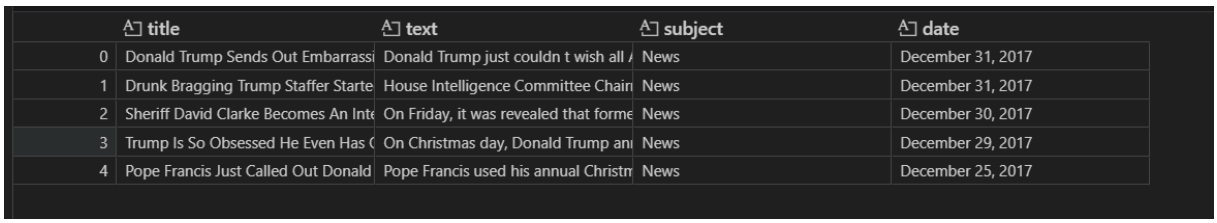
Listing 2.3 – Affichage de l'en-tête du DataFrame des fausses nouvelles

	title	text	subject	date
0	Donald Trump Sends Out Embarrassing New Year's Eve Messag	Donald Trump just couldn t wish all A	News	December 31, 2017
1	Drunk Bragging Trump Staffer Started Russian Collusion Investi	House Intelligence Committee Chairm	News	December 31, 2017
2	Sheriff David Clarke Becomes An Internet Joke For Threatening	On Friday, it was revealed that forme	News	December 30, 2017
3	Trump Is So Obsessed He Even Has Obama's Name Coded Into	On Christmas day, Donald Trump ann	News	December 29, 2017
4	Pope Francis Just Called Out Donald Trump During His Christm	Pope Francis used his annual Christm	News	December 25, 2017

FIGURE 2.1 – Cinq premières lignes du jeu de données des fausses nouvelles (`Fake.csv`). (Figure commentée)

```
print("\nAper u du DataFrame des vraies nouvelles (True News) :")
print(document_True_df.head())
```

Listing 2.4 – Affichage de l'en-tête du DataFrame des vraies nouvelles



	title	text	subject	date
0	Donald Trump Sends Out Embarrassi	Donald Trump just couldn t wish all A	News	December 31, 2017
1	Drunk Bragging Trump Staffer Starte	House Intelligence Committee Chairn	News	December 31, 2017
2	Sheriff David Clarke Becomes An Inte	On Friday, it was revealed that forme	News	December 30, 2017
3	Trump Is So Obsessed He Even Has (	On Christmas day, Donald Trump ani	News	December 29, 2017
4	Pope Francis Just Called Out Donald	Pope Francis used his annual Christm	News	December 25, 2017

FIGURE 2.2 – Cinq premières lignes du jeu de données des vraies nouvelles (`True.csv`). (Figure commentée)

2.2.4 Identification des Colonnes Clés

Les deux jeux de données semblaient partager une structure similaire, avec les colonnes clés suivantes identifiées pour l'analyse : `title` (titre), `text` (texte), `subject` (sujet) et `date`. Les colonnes `title` et `text` sont les principales sources de données textuelles pour la classification.

Chapitre 3

Méthodologie de Nettoyage et de Prétraitement des Données

3.1 Vue d'Ensemble du Pipeline de Prétraitement

Un pipeline de prétraitement robuste est essentiel pour transformer les données textuelles brutes en un format propre et utilisable pour l'ingénierie des caractéristiques. Le pipeline comprenait plusieurs étapes, notamment la suppression des mots vides (stop words), la normalisation du texte (passage en minuscules, suppression de la ponctuation) et la lemmatisation.

3.2 Configuration de l'Environnement et Bibliothèques

Des bibliothèques clés de TALN ont été employées pour ces tâches.

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from joblib import dump
import string
import nltk
from nltk.corpus import stopwords as nltk_stopwords
import re
import spacy
import joblib
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from gensim.parsing.preprocessing import STOPWORDS as
    gensim_stopwords
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS as
    sklearn_stopwords
from tqdm import tqdm
```

Listing 3.1 – Importation des bibliothèques TALN et utilitaires

3.2.1 Initialisation des Ressources spaCy et NLTK

Le petit modèle anglais de spaCy (`en_core_web_sm`) a été chargé, et la liste des mots vides de NLTK a été téléchargée.

```
nlp_spacy_for_stopwords = spacy.load("en_core_web_sm", disable=["
    parser", "ner", "lemmatizer"])
nltk.download('stopwords')
print("Exemple de mots vides anglais NLTK :")
print(nltk_stopwords.words("english")[:20])
```

Listing 3.2 – Initialisation de spaCy et des mots vides NLTK

```
['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'an', 'and', 'any', 'are', 'aren', "aren't", 'as', 'at', 'be',
```

FIGURE 3.1 – Exemple de mots vides anglais du corpus NLTK.

3.3 Gestion des Mots Vides

3.3.1 Justification de la Suppression Complète des Mots Vides

Les mots vides (par exemple, «a», «about», «at») sont des mots courants qui ne portent souvent pas de signification importante pour distinguer les types de documents. Leur suppression peut réduire le bruit et la dimensionnalité.

3.3.2 Agrégation des Mots Vides de Multiples Sources

Pour créer une liste complète, les mots vides ont été agrégés à partir de NLTK, spaCy, scikit-learn et Gensim.

3.3.3 La Fonction `get_combined_stopwords`

```
def clean_word(word): # Nettoie le mot
    return re.sub(r'[^a-z]', '', word.lower())

def get_combined_stopwords(): # Combine les stopwords
    sources = {
        "nltk": set(nltk_stopwords.words('english')),
        "spacy": set(nlp_spacy_for_stopwords.Defaults.stop_words),
        "sklearn": set(sklearn_stopwords),
        "gensim": set(gensim_stopwords)
    }
```

```
    combined = set()
    for source_name, source_words in sources.items(): # boucle corrig e
        for word in source_words:
            cleaned = clean_word(word)
            if cleaned and len(cleaned) > 0: # v rification ajout e
                combined.add(cleaned)
    return sorted(list(combined))

s_words = get_combined_stopwords()
print(f"Nombre total de mots vides combin s : {len(s_words)}")
print(f"Exemple de mots vides combin s : {s_words[:20]}")
```

Listing 3.3 – Fonction pour combiner et nettoyer les mots vides

FIGURE 3.2 – Exemple et nombre de la liste des mots vides combinés et nettoyés.

3.4 Normalisation du Texte et Stratégie de Lemmatisation

Le texte a été converti en minuscules, les caractères spéciaux ont été supprimés, et la lemmatisation à l'aide de spaCy a été appliquée pour réduire les mots à leurs formes de base (lemmes).

3.5 Traitement des Données par Lots (Chunks)

3.5.1 Justification du Traitement par Lots (Gestion de la Mémoire)

Le traitement de grands jeux de données textuelles par lots plus petits aide à gérer efficacement l'utilisation de la mémoire.

3.5.2 La Fonction `process_dataframe_in_chunks`

```
nlp_lemmatizer = spacy.load("en_core_web_sm", disable=["parser", "ner"])

```

```python
def process_dataframe_in_chunks(file_path, text_column,
                                chunk_size=1000,
                                output_path="dataset_processed.csv"
    ):
    global s_words
    first_chunk = True
    # ... (boucle sur les chunks et traitement comme dans le
    notebook) ...
    # chunk[f"processed_{text_column}"] = [...]
    # chunk.to_csv(...)
```

Listing 3.4 – Fonction de traitement des données par lots

3.5.3 Traitement de `Fake.csv` et `True.csv`

Les deux jeux de données ont été traités pour les colonnes `'text'` et `'title'` à l'aide de la fonction de traitement par lots, aboutissant à `dataset_Fake_final.csv` et `dataset_True_final.csv`. Pour illustrer concrètement l'impact de ce pipeline, examinons l'aspect des données textuelles avant et après les étapes de nettoyage et de normalisation.

Exemple de Données Avant Prétraitement

Avant l'application du pipeline de traitement, les colonnes `'title'` et `'text'` des jeux de données originaux (`Fake.csv` et `True.csv`) contiennent du texte brut. Ce texte inclut typiquement :

- Des majuscules et des minuscules.
- De la ponctuation diverse (virgules, points, guillemets, etc.).
- Des caractères spéciaux ou des chiffres.
- Des mots vides (stop words).
- Différentes formes flexionnelles des mots (par exemple, "running", "ran", "runs").

La Figure 3.3 présente un article de fausse nouvelle (fake news) dans son état brut, tel quextrait du fichier `Fake.csv`. On peut y observer la présence de majuscules, de ponctuation, de mots vides et de formes flexionnelles variées, caractéristiques d'un texte non traité

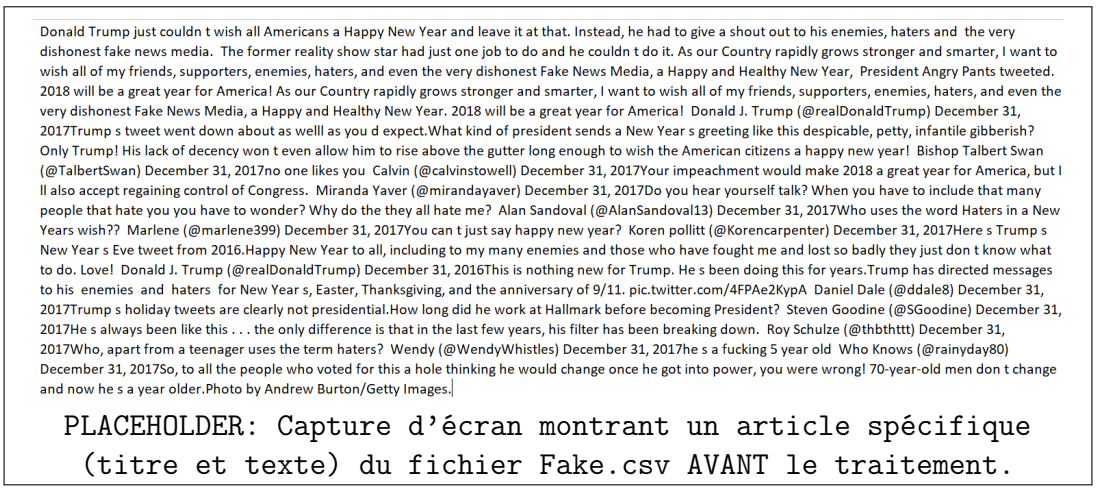


FIGURE 3.3 – Exemple d'un article spécifique issu de `Fake.csv` avant l'application du pipeline de prétraitement

Exemple de Données Après Prétraitement

Après l'exécution du pipeline de traitement par la fonction `process_dataframe_in_chunks`, les nouvelles colonnes `processed_title` et `processed_text` dans les fichiers `dataset_Fake_final.csv` et `dataset_True_final.csv` contiennent le texte transformé. Ce texte se caractérise par :

- L'absence de majuscules (tout est en minuscules).
- La suppression de la ponctuation et des caractères non alphabétiques (sauf les espaces entre les mots).
- L'élimination des mots vides.
- La réduction des mots à leur lemme (forme de base ou dictionnaire).

La Figure 3.4 montre ce même article de fausse nouvelle après avoir été traité par le pipeline. Le texte affiché correspond aux colonnes `processed_title` et `processed_text` du fichier `dataset_Fake_final.csv`. Les transformations suivantes sont visibles : passage en minuscules, suppression de la ponctuation et des caractères non pertinents, élimination des mots vides et lemmatisation des termes restants.

FIGURE 3.4 – Les mêmes exemples de données textuelles que la Figure 3.3, mais montrant le contenu des colonnes `processed_text` après l'application du pipeline de prétraitement.

Ces transformations sont cruciales car elles permettent de réduire la complexité du vocabulaire, de standardiser les termes et de ne conserver que les informations jugées les plus pertinentes pour les étapes ultérieures d'ingénierie des caractéristiques et de modélisation.

3.6 Nettoyage des Données Post-traitement

Les jeux de données traités finaux ont été chargés, les NaN (Not a Number) traités par `dropna()`, et les colonnes d'index inutiles supprimées.

```
dataset_fake = pd.read_csv("dataset_Fake_final.csv")
dataset_true = pd.read_csv("dataset_True_final.csv")
print(dataset_fake.isna().sum())
dataset_fake.dropna(inplace=True)
dataset_true.dropna(inplace=True)
```

Listing 3.5 – Gestion des NaN

FIGURE 3.5 – Nombre de valeurs manquantes dans le jeu de données Fake traité.

Chapitre 4

Techniques d'Ingénierie des Caractéristiques

4.1 Vue d'Ensemble de la Stratégie d'Ingénierie des Caractéristiques

Ce projet a employé TF-IDF et Doc2Vec pour la vectorisation du texte. La similarité cosinus entre les vecteurs de titre et de texte a également été utilisée comme caractéristique.

4.2 TF-IDF (Term Frequency-Inverse Document Frequency)

La méthode TF-IDF (Term Frequency-Inverse Document Frequency) est une technique statistique couramment utilisée en traitement du langage naturel et en recherche d'information pour évaluer l'importance d'un mot (ou terme) dans un document par rapport à une collection de documents (corpus). Elle attribue un poids à chaque terme dans un document, ce poids étant d'autant plus élevé que le terme est fréquent dans le document concerné, mais rare dans l'ensemble du corpus. Cela permet de mettre en évidence les termes qui sont distinctifs pour un document particulier.

Le score TF-IDF d'un terme t dans un document d appartenant à un corpus D est le produit de deux mesures : la Fréquence du Terme (TF) et la Fréquence Inverse de Document (IDF).

4.2.1 Fréquence du Terme (TF - Term Frequency)

La Fréquence du Terme mesure la fréquence d'apparition d'un terme t dans un document d. Il existe plusieurs façons de calculer la TF, la plus simple étant le nombre brut d'occurrences :

$$\mathrm{tf}(t, d) = f_{t,d}$$

où $f_{t,d}$ est le nombre de fois que le terme t apparaît dans le document d.

Pour éviter un biais en faveur des documents plus longs (qui peuvent avoir des fréquences de termes plus élevées indépendamment de l'importance réelle du terme), la TF est souvent normalisée. Une méthode courante de normalisation est de diviser le nombre brut d'occurrences par le nombre total de termes dans le document :

$$\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

où $\sum_{t' \in d} f_{t',d}$ est le nombre total de termes dans le document d. D'autres normalisations, comme la normalisation logarithmique ($\text{tf}(t, d) = \log(1 + f_{t,d})$), peuvent également être utilisées pour atténuer l'effet des grandes variations de fréquences.

4.2.2 Fréquence Inverse de Document (IDF - Inverse Document Frequency)

La Fréquence Inverse de Document mesure l'importance globale d'un terme t dans l'ensemble du corpus D. Elle pénalise les termes qui apparaissent dans de nombreux documents (comme les mots vides, s'ils n'ont pas été préalablement supprimés) et valorise les termes plus rares, considérés comme plus informatifs.

L'IDF est calculée comme le logarithme du rapport entre le nombre total de documents dans le corpus et le nombre de documents contenant le terme t. Soit $N = |D|$ le nombre total de documents dans le corpus, et $n_t = |\{d \in D : t \in d\}|$ le nombre de documents où le terme t apparaît. L'IDF est alors :

$$\text{idf}(t, D) = \log\left(\frac{N}{n_t}\right)$$

Pour éviter les divisions par zéro si un terme n'apparaît dans aucun document (bien que cela soit rare pour les termes du vocabulaire) ou pour éviter que l'IDF d'un terme présent dans tous les documents soit nul ($\log(N/N) = \log(1) = 0$), une variante courante, souvent appelée "smooth IDF", ajoute 1 au numérateur et au dénominateur (ou seulement au dénominateur) :

$$\text{idf}(t, D) = \log\left(\frac{N}{1 + n_t}\right) + 1 \quad \text{(Exemple de lissage, utilisé par scikit-learn)}$$

Le +1 à la fin assure que les termes apparaissant dans tous les documents ont toujours un poids IDF non nul, et le 1+ au dénominateur évite la division par zéro si $n_t = 0$ (bien que n_t soit généralement ≥ 1 pour les termes du vocabulaire). Scikit-learn, par défaut, utilise la formule $\text{idf}(t, D) = \log\left(\frac{N+1}{n_t+1}\right) + 1$.

4.2.3 Calcul du Score TF-IDF

Le score TF-IDF pour un terme t dans un document d est finalement obtenu en multipliant sa valeur TF par sa valeur IDF :

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

Un score TF-IDF élevé est atteint par un terme ayant une haute fréquence dans un document particulier (TF élevée) et une faible fréquence dans l'ensemble du corpus (IDF élevée). Inversement, un score faible peut signifier que le terme a une faible fréquence dans le document ou qu'il est très commun à travers les documents.

Dans ce projet, les colonnes de texte et de titre traitées ont été utilisées pour construire un vocabulaire. Chaque document (combinaison de titre et de texte, ou traités séparément) est ensuite représenté par un vecteur où chaque composante correspond au score TF-IDF d'un terme du vocabulaire.

4.2.4 Configuration du Vectoriseur TF-IDF Utilisé

Pour la transformation des textes en vecteurs TF-IDF, un `TfidfVectorizer` de la bibliothèque scikit-learn a été employé. Sa configuration a été définie avec des paramètres spécifiques pour optimiser la représentation des caractéristiques textuelles :

— `ngram_range=(1,2)` **:** Ce paramètre spécifie la plage des n-grammes à considérer comme caractéristiques. Un n-gramme est une séquence contiguë de n éléments (mots dans notre cas) provenant d'un échantillon de texte.

  — Un **unigramme** ($n = 1$) correspond à un mot unique (par exemple, « élection », « président »).

  — Un **bigramme** ($n = 2$) correspond à une séquence de deux mots consécutifs (par exemple, « élection présidentielle », « fausse nouvelle »).

  L'utilisation des bigrammes en plus des unigrammes permet de capturer un certain contexte et des expressions composées qui peuvent avoir une signification plus précise ou différente de celle des mots individuels qui les composent. Par exemple, le bigramme « Maison Blanche » porte une signification spécifique qui n'est pas simplement la somme des significations de « Maison » et « Blanche » pris séparément dans ce contexte. Inclure les bigrammes peut donc enrichir l'ensemble des caractéristiques et potentiellement améliorer la capacité du modèle à distinguer les nuances entre les documents. Cependant, cela augmente également la taille du vocabulaire et la dimensionnalité de l'espace des caractéristiques.

— `min_df=50` : Ce paramètre, signifiant "minimum document frequency" (fréquence minimale de document), est un seuil utilisé pour ignorer les termes qui ont une fré-

quence de document strictement inférieure à cette valeur. Dans notre configuration, un terme (qu'il s'agisse d'un unigramme ou d'un bigramme) devait apparaître dans au moins 50 documents différents du corpus d'entraînement pour être inclus dans le vocabulaire du vectoriseur. Le rôle de `min_df` est crucial pour plusieurs raisons :

- **Filtrage du bruit :** Les termes qui apparaissent très rarement (par exemple, dans un seul ou quelques documents) sont souvent des fautes de frappe, des mots très spécifiques à un contexte unique, ou du bruit qui n'apporte pas d'information généralisable pour la classification.
- **Réduction de la dimensionnalité :** En éliminant les termes extrêmement rares, on réduit la taille du vocabulaire et, par conséquent, la dimensionnalité de la matrice TF-IDF. Cela peut rendre les modèles plus rapides à entraîner et moins sujets au surapprentissage (overfitting), surtout avec des ensembles de données de taille modérée.
- **Amélioration de la robustesse statistique :** Les estimations IDF pour les termes très rares peuvent être instables. En fixant un seuil minimal, on s'assure de travailler avec des termes pour lesquels les statistiques sont plus fiables.

Le choix de la valeur 50 pour `min_df` est un compromis entre la conservation d'un vocabulaire suffisamment riche et l'élimination des termes peu informatifs ou bruyants. Cette valeur peut être ajustée par validation croisée lors d'une phase d'optimisation des hyperparamètres.

Ces choix de configuration visent à construire un vocabulaire de caractéristiques qui soit à la fois informatif et gérable en termes de dimensionnalité, en capturant des mots individuels ainsi que des expressions courtes pertinentes, tout en filtrant les termes les plus rares et potentiellement bruyants.

4.2.5 Ajustement, Sauvegarde et Chargement du Vectoriseur TF-IDF

Le vectoriseur a été ajusté (*fitted*) sur l'ensemble des textes d'entraînement combinés (titres et corps des articles vrais et faux) pour apprendre le vocabulaire et calculer les scores IDF. Une fois ajusté, il a été sauvegardé en utilisant 'joblib' pour une réutilisation ultérieure sans avoir besoin de réapprendre le vocabulaire, puis rechargé pour les étapes de transformation.

4.2.6 Transformation des Données Textuelles et Calcul de la Similarité Cosinus

Après l'ajustement, les titres et les textes des ensembles de données "fake" et "true" ont été transformés en matrices TF-IDF numériques creuses par le vectoriseur. La similarité cosinus, une mesure de similarité entre deux vecteurs non nuls d'un espace préhilbertien, a ensuite été calculée entre le vecteur TF-IDF du titre et le vecteur TF-IDF du corps de chaque article. Cette similarité est donnée par :

$$\text{cosine\_similarity}(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|}$$

où $\vec{A}$ et $\vec{B}$ sont les vecteurs TF-IDF. Une valeur proche de 1 indique une grande similarité, tandis qu'une valeur proche de 0 indique peu de similarité. Le code Python suivant illustre comment cette similarité a été calculée pour les ensembles de fausses nouvelles (*fake news*) et de vraies nouvelles (*true news*), puis comment elle a été préparée pour être intégrée comme une caractéristique supplémentaire.

```
from sklearn.metrics.pairwise import cosine_similarity
import scipy.sparse as sp

# Supposons que title_fake_vect, text_fake_vect, title_true_vect, et text_true_vect
# sont les matrices TF-IDF creuses obtenues apr s transformation des titres et textes.

# Exemple sur les fausses nouvelles :
# Calcul de la similarit  cosinus entre chaque titre et son texte correspondant.
# .diagonal() est utilis  car cosine_similarity(X, Y) calcule toutes les paires.
# Ici, nous voulons la similarit  de X[i] avec Y[i].
similarity_fake_tfidf = cosine_similarity(title_fake_vect, text_fake_vect).diagonal()

# Exemple sur les vraies nouvelles :
similarity_true_tfidf = cosine_similarity(title_true_vect, text_true_vect).diagonal()


# .T transpose le vecteur ligne en vecteur colonne.
```

```
similarity_fake_tfidf_vect = sp.csr_matrix(similarity_fake_tfidf).T
similarity_true_tfidf_vect = sp.csr_matrix(similarity_true_tfidf).
```

Listing 4.1 – Calcul de la similarité cosinus TF-IDF et préparation des caractéristiques

Les variables `similarity_fake_tfidf_vect` et `similarity_true_tfidf_vect` représentent donc la similarité cosinus titre-texte pour chaque article, formatée comme une caractéristique prête à être ajoutée aux autres caractéristiques TF-IDF.

4.2.7 Construction de l'Ensemble de Caractéristiques TF-IDF

Pour chaque article, l'ensemble final de caractéristiques basé sur TF-IDF comprenait :

1. Le vecteur TF-IDF du titre de l'article.
2. Le vecteur TF-IDF du corps du texte de l'article.
3. La valeur scalaire de la similarité cosinus calculée à l'étape précédente.

Ces éléments ont été concaténés horizontalement. Étant donné que les vecteurs TF-IDF du titre et du texte ont chacun une dimension égale à la taille du vocabulaire appris, la dimensionnalité des deux premiers éléments combinés est $2 \times$ taille du vocabulaire. En ajoutant la caractéristique de similarité cosinus (une seule valeur), la forme finale des caractéristiques TF-IDF pour chaque article devient (nombre d'articles, $2 \times$ taille du vocabulaire $+$ 1). Avec une taille de vocabulaire d'environ 21751 (déterminée par '$\min_d f = 50$'$etlesn-grammes), celaaconduitunvecteurdecaractristiquesd'environ2 \times 21751 + 1 = 43503$ colonnes pour chaque article.

4.3 Plongements (Embeddings) Doc2Vec

En complément de l'approche TF-IDF qui se base sur la fréquence des mots, ce projet a également exploré l'utilisation de Doc2Vec pour obtenir des représentations vectorielles (ou "plongements", de l'anglais *embeddings*) des documents. Doc2Vec, également connu sous le nom de Paragraph Vectors, est un algorithme d'apprentissage non supervisé qui vise à apprendre des représentations vectorielles de dimension fixe à partir de morceaux de texte de longueur variable, tels que des phrases, des paragraphes ou des documents entiers. L'objectif principal de Doc2Vec est de surmonter certaines limitations des modèles basés sur le sac de mots (comme TF-IDF), notamment leur incapacité à capturer l'ordre des mots et la sémantique contextuelle au-delà des cooccurrences de termes.

4.3.1 Principe Général de Doc2Vec

Doc2Vec est une extension de l'algorithme Word2Vec, qui apprend des plongements de mots. Word2Vec apprend des vecteurs de mots en tentant de prédire un mot cible à partir

de ses mots contextuels (modèle CBOW - Continuous Bag-Of-Words) ou en prédisant les mots contextuels à partir d'un mot cible (modèle Skip-gram).

Doc2Vec étend cette idée en ajoutant un vecteur supplémentaire, le "vecteur de paragraphe" (ou de document), qui est unique à chaque document. Ce vecteur de document est appris conjointement avec les vecteurs de mots et agit comme une sorte de "mémoire" qui représente le sujet ou le contexte global du document. Pendant l'entraînement :

- Dans l'approche **Distributed Memory (PV-DM)**, similaire à CBOW, le vecteur de paragraphe est combiné (par exemple, par concaténation ou moyennage) avec les vecteurs des mots contextuels d'une fenêtre glissante pour prédire le mot central de cette fenêtre. Le vecteur de paragraphe est partagé entre toutes les fenêtres de contexte générées à partir du même document, mais pas les vecteurs de mots.
- Dans l'approche **Distributed Bag of Words (PV-DBOW)**, similaire à Skip-gram, le modèle est entraîné à prédire une distribution de mots échantillonnés aléatoirement à partir du paragraphe, en utilisant uniquement le vecteur de paragraphe comme entrée. Cette méthode est conceptuellement plus simple et souvent plus rapide à entraîner, ignorant le contexte des mots pendant l'inférence du vecteur de paragraphe.

Le résultat de ce processus d'entraînement est que chaque document du corpus d'entraînement se voit attribuer un vecteur dense de faible dimension (par exemple, 100 à 300 dimensions) qui capture ses caractéristiques sémantiques. Les documents ayant un contenu sémantique similaire devraient avoir des vecteurs proches dans cet espace de plongement.

4.3.2 Avantages et Utilisation dans le Projet

Les principaux avantages de l'utilisation de Doc2Vec incluent :

- **Capture de la sémantique :** Contrairement à TF-IDF, Doc2Vec peut capturer des relations sémantiques plus complexes entre les mots et les documents. L'ordre des mots, bien que pas explicitement modélisé dans toutes les variantes, influence l'apprentissage des vecteurs de mots qui contribuent aux vecteurs de document.
- **Vecteurs denses de faible dimension :** Les vecteurs produits sont denses (la plupart des valeurs sont non nulles) et de dimensionnalité beaucoup plus faible que les vecteurs TF-IDF typiques, ce qui peut être avantageux pour certains algorithmes d'apprentissage automatique et réduire les problèmes de sparsité.
- **Apprentissage non supervisé :** Il peut être entraîné sur de grandes quantités de texte non étiqueté.

Dans le cadre de ce projet, un modèle Doc2Vec a été entraîné sur l'ensemble du corpus textuel (titres et corps des articles combinés des jeux de données "fake" et "true"). Les vecteurs de document résultants ont ensuite été utilisés de deux manières :

1. Pour calculer une **similarité cosinus** entre le vecteur Doc2Vec du titre et celui du corps de chaque article. Cette mesure de similarité a ensuite été ajoutée comme caractéristique supplémentaire à l'ensemble de caractéristiques TF-IDF .
2. Pour créer un **ensemble de caractéristiques supplémentaire** composé uniquement des vecteurs Doc2Vec des titres, des vecteurs Doc2Vec des corps de texte, et de leur similarité cosinus .

L'hypothèse est que les représentations sémantiques fournies par Doc2Vec pourraient capturer des signaux subtils de fausseté ou d'authenticité que TF-IDF seul pourrait manquer.

4.3.3 Configuration et Entraînement du Modèle Doc2Vec

Après la préparation des données textuelles sous la forme requise de `TaggedDocument`, l'étape suivante a consisté à configurer et à entraîner un modèle Doc2Vec. Cette tâche a été réalisée à l'aide de la bibliothèque Gensim, reconnue pour ses implémentations efficaces d'algorithmes de modélisation de sujets et de plongements de mots/documents.

Paramètres du Modèle

Le choix des hyperparamètres est déterminant pour la qualité des vecteurs de documents produits. Pour ce projet, les paramètres clés suivants ont été retenus pour l'initialisation du modèle Doc2Vec :

— `vector_size=100` : Ce paramètre fixe la dimensionnalité des vecteurs de plongement qui seront appris pour chaque document. Une taille de 100 dimensions est un choix fréquent, offrant un équilibre entre la richesse de la représentation sémantique et la complexité du modèle.

— `window=2` : Définit la portée du contexte local pour l'apprentissage des relations entre les mots. Plus précisément, il s'agit de la distance maximale entre le mot cible et les mots de son voisinage pris en compte lors de l'entraînement. Une fenêtre de 2 signifie que 2 mots à gauche et 2 mots à droite du mot cible sont considérés, ce qui est pertinent pour les architectures comme PV-DM.

— `min_count=1` : Ce seuil de fréquence minimale instruit le modèle d'ignorer tous les mots apparaissant moins de 1 fois dans l'ensemble du corpus. Avec une valeur de 1, tous les mots présents dans le vocabulaire initial des `TaggedDocument` sont conservés et participent à l'apprentissage.

— `workers=4` : Pour optimiser le temps d'entraînement, 4 threads de processeur ont été alloués à la tâche. L'utilisation de plusieurs cœurs est particulièrement bénéfique pour les corpus volumineux.

— `epochs=100` : Représente le nombre de passes complètes (itérations) que le modèle effectue sur l'ensemble du corpus d'entraînement. Un nombre d'époques plus élevé peut permettre au modèle de mieux converger et d'apprendre des représentations plus fines, au prix d'un temps d'entraînement accru.

Il est à noter que Gensim implémente les deux principales architectures de Doc2Vec : PV-DM (Distributed Memory) et PV-DBOW (Distributed Bag of Words). Le choix entre ces architectures est contrôlé par le paramètre `dm` (non spécifié explicitement dans le code fourni, donc la valeur par défaut de Gensim, souvent `dm=1` pour PV-DM ou une combinaison, est utilisée).

Processus d'Entraînement

L'entraînement d'un modèle Doc2Vec avec Gensim se déroule typiquement en deux phases séquentielles après l'initialisation du modèle avec les paramètres ci-dessus :

1. **Construction du Vocabulaire :** Le modèle analyse l'ensemble des `TaggedDocument` fournis pour identifier tous les mots uniques et construire son vocabulaire interne. Les mots sont filtrés selon `min_count`.
2. **Apprentissage des Vecteurs :** Le modèle itère ensuite sur le corpus pour le nombre d'époques défini (`epochs`). Durant chaque époque, il ajuste les vecteurs de mots et les vecteurs de documents (associés aux tags des `TaggedDocument`) afin de minimiser une fonction de coût, qui dépend de l'architecture choisie (par exemple, prédire un mot central à partir de son contexte et du vecteur de document pour PV-DM).

L'extrait de code Python ci-dessous illustre ces étapes d'initialisation, de construction du vocabulaire et d'entraînement du modèle Doc2Vec.

```
import logging
from gensim.models.doc2vec import Doc2Vec # Assurer l'importation
    de TaggedDocument au pr alable

# Configuration du logging pour observer la progression
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message
    )s', level=logging.INFO)

# tagged_data : liste de TaggedDocument (pr par e dans une  tape
    pr c dente)

# Initialisation du mod le Doc2Vec
model = Doc2Vec(vector_size=100,
                    window=2,
```

```
                    min_count=1,
                    workers=4,
                    epochs=100)

# 1. Construction du vocabulaire
model.build_vocab(tagged_data)

# 2. Entra nement du mod le
model.train(tagged_data,
                total_examples=d2v_model.corpus_count,
                epochs=d2v_model.epochs)

# Le mod le `d2v_model` contient maintenant les vecteurs appris.
```

Listing 4.2 – Configuration et entraînement du modèle Doc2Vec avec Gensim

Pendant l'exécution de ce code, Gensim émet des messages informatifs (via le module 'logging') qui permettent de suivre la progression de la construction du vocabulaire et de chaque époque d'entraînement.

Sauvegarde et Chargement du Modèle

Pour éviter de réitérer le processus d'entraînement, coûteux en temps, il est essentiel de sauvegarder le modèle une fois qu'il a été entraîné. Gensim facilite cette opération :

```
# Sauvegarde du mod le  entra n
model.save("mon_modele_doc2vec.model")

# Exemple de chargement ult rieur du mod le :
# model_loaded = Doc2Vec.load("mon_modele_doc2vec.model")
```

Listing 4.3 – Sauvegarde et chargement du modèle Doc2Vec

Dans les étapes suivantes du projet, notamment pour l'inférence de vecteurs sur de nouveaux textes ou sur les ensembles de test, c'est ce modèle sauvegardé (et rechargé) qui a été utilisé.

4.3.4 Inférence des Vecteurs de Document et Calcul de la Similarité Cosinus

Une fois le modèle Doc2Vec entraîné et disponible (soit directement après l'entraînement, soit après l'avoir rechargé), l'objectif est d'obtenir les représentations vectorielles

pour les titres et les corps de texte des articles constituant les jeux de données prétraités `dataset_fake` et `dataset_true`. La méthode `infer_vector` du modèle Doc2Vec de Gensim est conçue à cet effet. Elle permet de déduire le vecteur d'un nouveau document (qui n'a pas nécessairement été vu pendant la phase d'entraînement) en se basant sur les vecteurs de mots que le modèle a appris. Ce processus d'inférence est itératif : le modèle tente de trouver un vecteur de document qui "explique" le mieux la séquence de mots du texte fourni, en maintenant les vecteurs de mots fixes.

Fonction d'Inférence de Vecteur

Pour simplifier l'utilisation de `infer_vector`, une fonction encapsulatrice a été définie :

```
def text_to_docvect(text_input): # Renomm  pour clart
    # S'assurer que l'entr e est une cha ne et la tokeniser (
   s parer en mots)
    words = str(text_input).split()
    # Utiliser la m thode infer_vector du mod le fourni
    vector = model.infer_vector(words)
    return vector
```

Listing 4.4 – Fonction d'inférence de vecteur Doc2Vec pour un texte

Application de l'Inférence par Lots

L'inférence de vecteurs pour un grand nombre de documents peut être gourmande en ressources. Pour optimiser l'utilisation de la mémoire, une approche de traitement par lots (chunks) a été mise en œuvre, similaire à celle utilisée lors du prétraitement initial des données. La fonction `infer_vectors_chunked` (dont la définition détaillée se trouve dans le notebook original et qui a été discutée, par exemple, en relation avec la Section **??**) a été employée pour générer les vecteurs Doc2Vec pour les colonnes `"processed_title"` et `"processed_text"` des DataFrames `dataset_fake` et `dataset_true`.

L'extrait de code ci-dessous montre comment cette inférence a été appliquée :

```
import numpy as np
import pandas as pd
from tqdm import tqdm
import gc

# Note : La fonction text_to_docvect(doc) est appel e ci-dessous.
# Cette fonction doit avoir acc s    l'instance du mod le Doc2Vec
    entra n
```

```
# (par exemple, via une variable globale d2v_model_loaded, ou en tant
# une m thode de classe). Si ce n'est pas le cas, la signature de
# text_to_docvect et de infer_vectors_chunked devrait  tre modifi e
# pour passer explicitement le mod le Doc2Vec.

def infer_vectors_chunked(dataset, column_name, chunk_size=1000):
    """
    Traite un DataFrame par morceaux pour inf rer des vecteurs Doc2Vec,
      vitant  les probl mes de saturation m moire.
    """
    total_rows = len(dataset)
    all_vectors = []
    num_chunks = (total_rows + chunk_size - 1) // chunk_size # Nombre de lots

    for i in tqdm(range(0, total_rows, chunk_size),
                  desc=f"Inf rence Doc2Vec pour '{column_name}'",
                  total=num_chunks):
        end_idx = min(i + chunk_size, total_rows)
        chunk_df = dataset.iloc[i:end_idx] # Lot courant

        # Application de l'inf rence vectorielle sur le lot
        chunk_vectors = np.array([text_to_docvect(doc) for doc in chunk_df[column_name]])
        all_vectors.append(chunk_vectors)

        del chunk_vectors # Lib ration de m moire
        gc.collect()

    if all_vectors:
        result_vectors = np.vstack(all_vectors) # Concat nation des r sultats
        return result_vectors
    else:
        return np.array([])

# Le mod le Doc2Vec (ex: d2v_model_loaded) doit  tre  charg  avant l'appel    cette fonction,
```

```
# et accessible par text_to_docvect.
# Exemple: d2v_model_loaded = Doc2Vec.load("mon_modele_doc2vec.model")

# Pr paration des colonnes (conversion en string, gestion des NaNs)
# dataset_fake et dataset_true sont suppos s  tre  des DataFrames Pandas charg s.
dataset_fake['processed_title'] = dataset_fake['processed_title'].astype(str).fillna('')
dataset_fake['processed_text'] = dataset_fake['processed_text'].astype(str).fillna('')
dataset_true['processed_title'] = dataset_true['processed_title'].astype(str).fillna('')
dataset_true['processed_text'] = dataset_true['processed_text'].astype(str).fillna('')

# Application de la fonction d'inf rence par lots
title_fake_docvect = infer_vectors_chunked(dataset_fake, "processed_title", chunk_size=500)
text_fake_docvect = infer_vectors_chunked(dataset_fake, "processed_text", chunk_size=500)
title_true_docvect = infer_vectors_chunked(dataset_true, "processed_title", chunk_size=500)
text_true_docvect = infer_vectors_chunked(dataset_true, "processed_text", chunk_size=500)

# title_fake_docvect, etc., sont des tableaux NumPy de vecteurs Doc2Vec.
```

Listing 4.5 – Définition et utilisation de la fonction d'inférence par lots `infer_vectors_chunked`

Cette approche garantit que même pour de grands ensembles de données, les vecteurs Doc2Vec sont générés de manière efficace sans saturer la mémoire.

Calcul de la Similarité Cosinus à partir des Vecteurs Doc2Vec

Après avoir obtenu les représentations vectorielles Doc2Vec pour les titres et les corps de texte de chaque article, l'étape suivante a été de quantifier leur ressemblance sémantique. La similarité cosinus a été employée à cette fin. Comme expliqué précédemment pour les vecteurs TF-IDF (Section **??**), cette mesure évalue l'angle entre les deux vecteurs

Doc2Vec (celui du titre et celui du texte correspondant), fournissant un score entre -1 et 1 (typiquement entre 0 et 1 pour des vecteurs de plongement comme Doc2Vec).

Le code Python ci-dessous montre le calcul de cette similarité :

```
from sklearn.metrics.pairwise import cosine_similarity
import scipy.sparse as sp # Utilis  si on souhaite stocker le
    r sultat en format  pars

# Calcul de la similarit  pour les fausses nouvelles.
# La m thode .diagonal() est utilis e car les vecteurs de titre
    et de texte
# dans title_fake_docvect et text_fake_docvect sont align s par
    article.
similarity_fake_doc2vec = cosine_similarity(title_fake_docvect,
    text_fake_docvect).diagonal()

# Calcul de la similarit  pour les vraies nouvelles
similarity_true_doc2vec = cosine_similarity(title_true_docvect,
    text_true_docvect).diagonal()

# Les r sultats `similarity_fake_doc2vec` et `
    similarity_true_doc2vec`
# sont des tableaux NumPy contenant les scores de similarit  pour
    chaque article.

# Pour une utilisation ult rieure, notamment la concat nation
    avec d'autres
# caract ristiques (potentiellement  parses ), ces vecteurs de
    similarit
# ont  t  transform s en matrices creuses d'une seule colonne.
similarity_fake_doc_vect_csr = sp.csr_matrix(
    similarity_fake_doc2vec).T
similarity_true_doc_vect_csr = sp.csr_matrix(
    similarity_true_doc2vec).T
```

Listing 4.6 – Calcul de la similarité cosinus entre les vecteurs Doc2Vec

Les scores de similarité cosinus ainsi obtenus, `similarity_fake_doc_vect_csr` et `similarity_true_doc_vect_csr`, constituent des caractéristiques numériques prêtes à être intégrées dans les ensembles de caractéristiques finaux pour la modélisation. Ils représentent une mesure de la cohérence sémantique entre le titre et le contenu de chaque article, telle qu'interprétée par le modèle Doc2Vec..

4.4 Construction et Stockage de l'Ensemble Final de Caractéristiques

4.4.1 Augmentation des Caractéristiques TF-IDF avec la Similarité Doc2Vec

La similarité cosinus basée sur Doc2Vec a été ajoutée aux matrices de caractéristiques TF-IDF. Les formes finales étaient $(22854, 43504)$ pour les fausses nouvelles et $(21416, 43504)$ pour les vraies. Celles-ci ont été sauvegardées sous `fake_features.npz` et `true_features.npz`.

Chapitre 5

Entraînement et Évaluation des Modèles

Après les étapes cruciales d'acquisition, de nettoyage, de prétraitement des données (Chapitre 3) et d'ingénierie des caractéristiques (Chapitre 4), nous disposons désormais de représentations numériques des articles de presse. Ces représentations, principalement sous forme de vecteurs TF-IDF augmentés par des mesures de similarité, sont prêtes à être exploitées par des algorithmes d'apprentissage automatique pour la tâche de classification : distinguer les fausses nouvelles (fake news) des nouvelles authentiques.

Ce chapitre est dédié à la phase de modélisation du projet. Il détaille le processus de construction, d'entraînement et d'évaluation de plusieurs modèles de classification. L'objectif est d'identifier les approches les plus performantes pour la détection de fausses nouvelles en se basant sur les caractéristiques textuelles extraites. Nous explorerons une gamme de modèles, allant des algorithmes classiques bien établis à une approche basée sur l'apprentissage profond (deep learning).

Les étapes clés abordées dans ce chapitre incluent :

— La préparation finale des données pour l'alimentation des modèles, notamment la combinaison des ensembles de caractéristiques des fausses et vraies nouvelles, la création des étiquettes correspondantes, et la division des données en ensembles d'entraînement et de test.

— La définition et l'utilisation de fonctions auxiliaires pour une évaluation visuelle et quantitative cohérente des performances des modèles (par exemple, matrices de confusion, courbes ROC et Précision-Rappel, courbes d'apprentissage).

— L'entraînement et l'évaluation détaillée de trois modèles de classification distincts :

1. Une **Régression Logistique**, choisie pour sa simplicité, son efficacité sur les données textuelles et son interprétabilité relative.
2. Une **Machine à Vecteurs de Support (SVM)**, un classifieur puissant capable de gérer des espaces de caractéristiques de haute dimension.
3. Un **Réseau de Neurones Profonds (Deep Learning)** implémenté avec Keras, pour explorer la capacité des architectures plus complexes à capturer des motifs dans les données.

— Une analyse comparative des résultats obtenus par chaque modèle, en mettant en lumière leurs forces et faiblesses respectives dans le contexte de cette tâche spécifique.

Comme mentionné, les caractéristiques primaires utilisées pour l'entraînement de ces modèles sont celles dérivées de l'approche TF-IDF (titres et textes), augmentées par la similarité cosinus calculée à partir des vecteurs TF-IDF ainsi que par la similarité cosinus issue des vecteurs Doc2Vec (voir Section **??** pour la description de la construction de `fake_features.npz` et `true_features.npz`). La performance de chaque modèle sera rigoureusement évaluée à l'aide de métriques standards et de visualisations pour fournir une compréhension approfondie de leur efficacité.

5.1 Préparation des Données pour la Modélisation

Après le chargement des caractéristiques prétraitées (`fake_features.npz` et `true_features.npz`), les données ont été préparées différemment pour les modèles d'apprentissage automatique classiques et pour le modèle d'apprentissage profond.

5.1.1 Préparation pour les Modèles d'Apprentissage Automatique Classiques (Régression Logistique, SVM)

Pour les modèles classiques, les caractéristiques ont été combinées et les étiquettes créées sous forme d'entiers (0 pour "fake", 1 pour "true"). L'ensemble a ensuite été divisé en données d'entraînement et de test.

```
from scipy.sparse import vstack # Assumer load_npz, np sont d j
    import s
from sklearn.model_selection import train_test_split

# fake_features et true_features sont suppos es charg es.
# Exemple de chargement (si non fait pr c demment) :
# fake_features = load_npz('fake_features.npz')
# true_features = load_npz('true_features.npz')

# 1. Concat ner les features (verticalement)
X = vstack([fake_features, true_features])
# print("Forme de X:", X.shape)

# 2. Cr er les labels : 0=fake, 1=true
y_fake = np.zeros(fake_features.shape[0])
y_true = np.ones(true_features.shape[0])
y = np.concatenate([y_fake, y_true])
# print("Forme de y:", y.shape)

```

```
# 3. S parer en train/test (stratifi  pour garder les proportions )
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
# print("Forme de X_train:", X_train.shape, "Forme de y_train:", y_train.shape)
# print("Forme de X_test:", X_test.shape, "Forme de y_test:", y_test.shape)
```

Listing 5.1 – Préparation des données pour les modèles classiques

5.1.2 Préparation pour le Modèle d'Apprentissage Profond (Keras)

Pour le modèle d'apprentissage profond utilisant Keras, les étiquettes ont été encodées au format "one-hot" avant la division des données. La matrice de caractéristiques X reste la même.

```
# X (matrice de caract ristiques) est suppos e  tre  la m me que ci-dessus.
# fake_features et true_features sont suppos es charg es.

# 1. Cr er les labels one-hot : [1, 0] pour fake (classe 0), [0, 1] pour true (classe 1)
# (Cette convention est d duite du code de pr diction du notebook )
y_fake_onehot = np.tile([1, 0], (fake_features.shape[0], 1))
y_true_onehot = np.tile([0, 1], (true_features.shape[0], 1))

# Fusionner verticalement pour avoir tous les labels
y_onehot = np.vstack([y_fake_onehot, y_true_onehot])
# print("Forme de y_onehot:", y_onehot.shape)

# 2. S parer en train/test (stratifi  pour garder les proportions )
# La stratification se fait sur la base des indices de classe.
X_train_dl, X_test_dl, y_train_dl, y_test_dl = train_test_split(
```

```
    X, y_onehot,
    test_size=0.2,
    random_state=42,
    stratify=y_onehot.argmax(axis=1)
)
# print("Forme de X_train_dl:", X_train_dl.shape, "Forme de
   y_train_dl:", y_train_dl.shape)
# print("Forme de X_test_dl:", X_test_dl.shape, "Forme de y_test_dl
   :", y_test_dl.shape)
```

Listing 5.2 – Préparation des données pour le modèle d'apprentissage profond

5.2 Fonctions Auxiliaires pour l'Évaluation des Modèles

Après avoir préparé les données et avant de procéder à l'entraînement et à l'évaluation de chaque modèle de classification, il est judicieux de définir un ensemble d'outils communs pour analyser leurs performances de manière cohérente et visuelle. Cette section introduit les fonctions auxiliaires Python qui ont été développées ou utilisées à cette fin.

L'objectif de la création de ces fonctions est double :

1. **Standardiser l'évaluation :** En utilisant systématiquement les mêmes méthodes de visualisation et de calcul de métriques pour chaque modèle (Régression Logistique, SVM, et le modèle d'Apprentissage Profond), nous assurons une base de comparaison équitable et objective de leurs performances respectives.
2. **Améliorer la clarté et la concision du code et du rapport :** Plutôt que de répéter le code de génération de graphiques pour chaque modèle, encapsuler cette logique dans des fonctions réutilisables rend le code d'analyse plus propre et le rapport plus facile à lire, en évitant les redondances.

Les principales visualisations et analyses de performance qui seront générées par ces fonctions auxiliaires pour chaque modèle comprennent :

— **La Matrice de Confusion :** Pour visualiser en détail les prédictions correctes et incorrectes (vrais positifs, vrais négatifs, faux positifs, faux négatifs) et comprendre les types d'erreurs commises par le classifieur.

— **La Courbe ROC (Receiver Operating Characteristic) et l'Aire sous la Courbe (AUC) :** Pour évaluer la capacité du modèle à discriminer entre les classes "fausse nouvelle" et "vraie nouvelle" à différents seuils de classification.

— **La Courbe Précision-Rappel (Precision-Recall Curve) et l'Aire sous cette Courbe (AUC-PR) :** Particulièrement pertinente pour évaluer la performance sur

la classe d'intérêt et lorsque les classes pourraient être déséquilibrées. Elle montre le compromis entre la précision et le rappel.

— **La Courbe d'Apprentissage (Learning Curve) :** Pour analyser comment la performance du modèle (sur les données d'entraînement et de validation) évolue avec l'augmentation de la taille de l'ensemble d'entraînement, aidant à identifier les problèmes de surapprentissage ou de sous-apprentissage.

— **La Visualisation des Erreurs de Classification :** Une tentative de représenter graphiquement (souvent après réduction de dimensionnalité par ACP) les instances qui ont été mal classées, afin d'obtenir des indices sur les faiblesses potentielles du modèle.

Les définitions Python spécifiques de ces fonctions de traçage et d'évaluation sont regroupées et peuvent être consultées dans le listing référencé par `lst:eval_plot_functions` (si ce listing est inclus dans l'annexe ou une section dédiée du code). Dans les sections suivantes, lors de l'évaluation de chaque modèle, nous ferons appel à ces fonctions pour présenter et analyser les résultats de manière structurée.

5.3 Modèle 1 : Régression Logistique

Le premier algorithme de classification testé pour cette tâche de détection de fausses nouvelles est la Régression Logistique. Ce modèle est un choix populaire et fondamental en apprentissage automatique pour les problèmes de classification binaire. Il fonctionne en modélisant la probabilité qu'une instance appartienne à une classe particulière à l'aide d'une fonction logistique (sigmoïde) appliquée à une combinaison linéaire des caractéristiques d'entrée. Ses avantages incluent sa relative simplicité, son efficacité sur les données de grande dimension et souvent éparses (comme les caractéristiques TF-IDF), et une certaine interprétabilité de ses coefficients.

5.3.1 Configuration et Entraînement du Modèle

Pour l'implémentation de la Régression Logistique, nous avons utilisé la classe `LogisticRegression` de la bibliothèque scikit-learn. Le modèle a été configuré avec un ensemble spécifique d'hyperparamètres visant à optimiser ses performances et à assurer la reproductibilité. Les principaux hyperparamètres choisis sont les suivants :

— `solver='liblinear'` : Ce solveur est un bon choix pour les jeux de données de taille petite à moyenne et est efficace pour les problèmes binaires avec régularisation $L1$ ou $L2$.

— `penalty='l2'` : Une pénalité de type $L2$ (régularisation de Ridge) a été appliquée

pour aider à prévenir le surapprentissage en contraignant la magnitude des coefficients du modèle.

— `max_iter=1000` : Le nombre maximal d'itérations pour que l'algorithme d'optimisation converge a été augmenté à 1000, par rapport à la valeur par défaut, pour s'assurer d'une convergence adéquate sur notre jeu de données.

— `random_state=42` : Un état aléatoire fixe a été utilisé pour garantir que les résultats de l'entraînement soient reproductibles.

Après la configuration, le modèle a été entraîné (ou "ajusté") en utilisant les données d'entraînement `X_train` (caractéristiques) et `y_train` (étiquettes), qui ont été préparées comme décrit à la Section 5.1.1.

Le code Python ci-dessous montre l'instanciation du modèle de Régression Logistique avec ces hyperparamètres et son entraînement sur les données.

```
from sklearn.linear_model import LogisticRegression
# Les variables X_train et y_train sont suppos es avoir   t
   d finies  pr c demment.

# Instanciation du mod le avec les hyperparam tres choisis
lr_model = LogisticRegression( # Renomm  en lr_model pour clart
    solver='liblinear',
    penalty='l2',
    max_iter=1000,
    random_state=42
)

# Entra nement du mod le sur les donn es d'entra nement
lr_model.fit(X_train, y_train)

#    ce stade, le mod le lr_model est entra n  et pr t pour l'
    valuation  .
# La sauvegarde du mod le (ex: avec joblib) peut  tre  effectu e
   ici ou apr s l'  valuation  .
# from joblib import dump
# dump(lr_model, 'models/Lregression_model.joblib')
```

Listing 5.3 – Instanciation et entraînement du modèle de Régression Logistique

Une fois cette phase d'entraînement achevée, le modèle `lr_model` est prêt à être soumis à une évaluation rigoureuse de ses performances sur l'ensemble de test, comme détaillé dans la sous-section suivante.

5.3.2 Évaluation des Performances

Après la phase d'entraînement, l'efficacité du modèle de Régression Logistique a été mesurée sur l'ensemble de test (`X_test`, `y_test`), qui constitue des données que le modèle n'a jamais rencontrées auparavant.

La précision globale (*accuracy*) obtenue par le modèle sur cet ensemble de test est de **0.9919** (soit 99.19%). Pour une analyse plus fine, le rapport de classification ci-dessous, généré par scikit-learn, détaille les métriques de performance par classe :

	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	4571
1.0	0.99	0.99	0.99	4283
accuracy			0.99	8854
macro avg	0.99	0.99	0.99	8854
weighted avg	0.99	0.99	0.99	8854

Ce rapport indique des scores de **précision**, de **rappel** et **F1-score** exceptionnellement élevés (égaux à 0.99) pour les deux classes (la classe 0.0 correspondant aux "Fake News" et la classe 1.0 aux "True News"). Ces résultats démontrent que le modèle est non seulement très précis dans ses attributions de classe, mais qu'il est également capable d'identifier correctement la grande majorité des instances de chaque catégorie tout en minimisant le nombre de classifications erronées. Les scores moyens ("macro avg" et "weighted avg") confirment cette performance uniformément élevée à travers les classes.

Les visualisations suivantes, produites à l'aide des fonctions auxiliaires Python décrites précédemment (voir Section 5.2), permettent d'explorer plus en détail ces performances.

Matrice de Confusion

La matrice de confusion, illustrée à la Figure 5.1, offre une ventilation détaillée des prédictions du modèle par rapport aux classes réelles. Conformément aux chiffres présentés, le modèle a correctement identifié 4535 articles comme étant des "Fake News" (vrais négatifs, si "Fake News" est la classe 0) et 4247 articles comme étant des "True News" (vrais positifs, si "True News" est la classe 1). Le nombre d'erreurs de classification est remarquablement bas : seulement 36 articles "Fake News" ont été erronément classifiés comme "True News" (constituant des faux positifs pour la classe "True News"), et un nombre identique de 36 articles "True News" ont été incorrectement classifiés comme "Fake News" (constituant des faux négatifs pour la classe "True News"). Ces résultats soulignent l'excellente capacité du modèle à séparer les deux catégories d'articles.

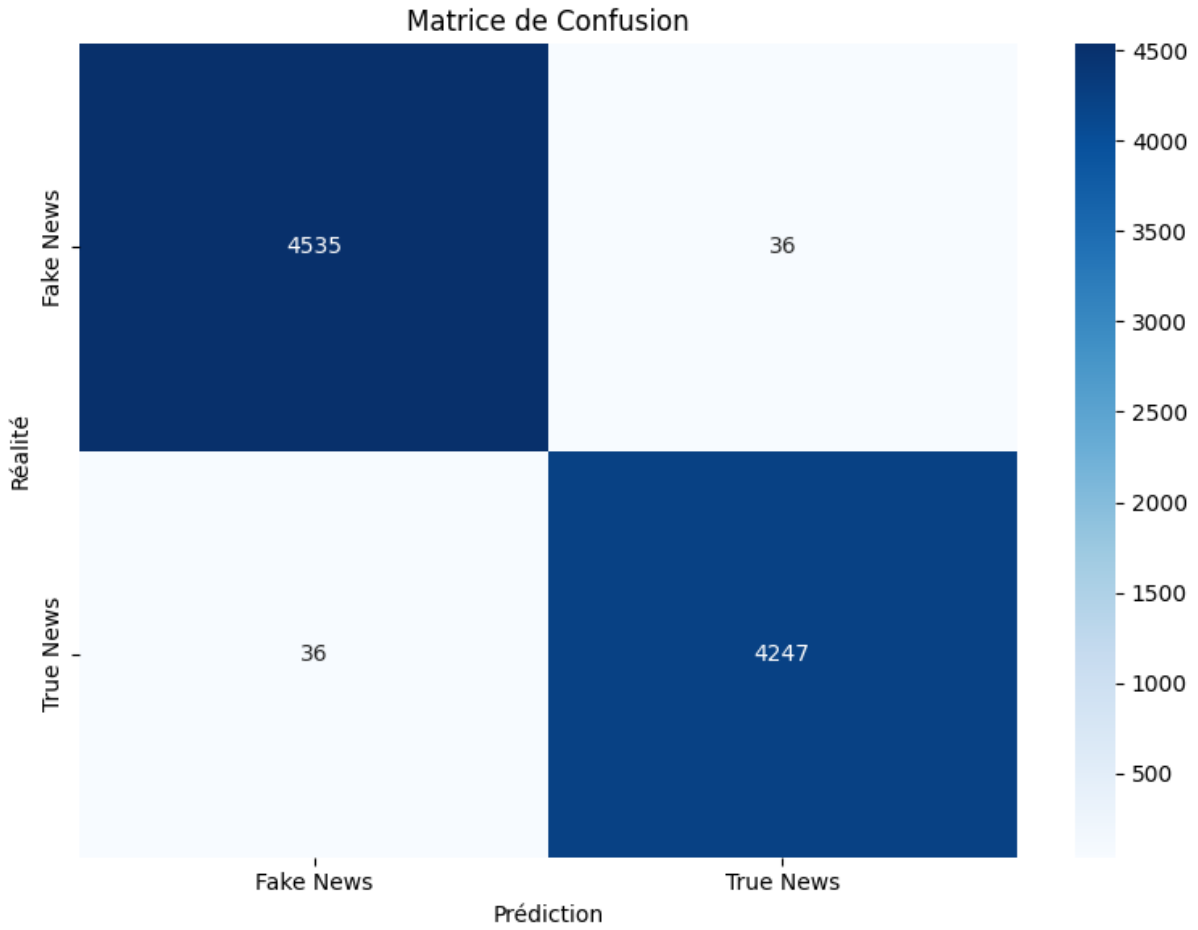


FIGURE 5.1 – Matrice de Confusion pour le modèle de Régression Logistique sur l'ensemble de test.

Courbe ROC et AUC

La courbe ROC (Receiver Operating Characteristic), présentée à la Figure 5.2, trace le taux de vrais positifs (également appelé sensibilité ou rappel) en fonction du taux de faux positifs (calculé comme $1 -$ spécificité) pour une gamme de seuils de classification. La courbe obtenue pour le modèle de Régression Logistique s'élève très rapidement vers le coin supérieur gauche du graphique et longe l'axe supérieur, ce qui est indicatif d'une performance de classification quasi parfaite. L'Aire Sous la Courbe (AUC) est de **1.000** (valeur arrondie, indiquant une performance extrêmement proche de l'idéal). Un score AUC aussi élevé confirme la capacité robuste et fiable du modèle à distinguer les fausses nouvelles des vraies nouvelles. À titre de comparaison, un classifieur aléatoire aurait une AUC de 0.5.

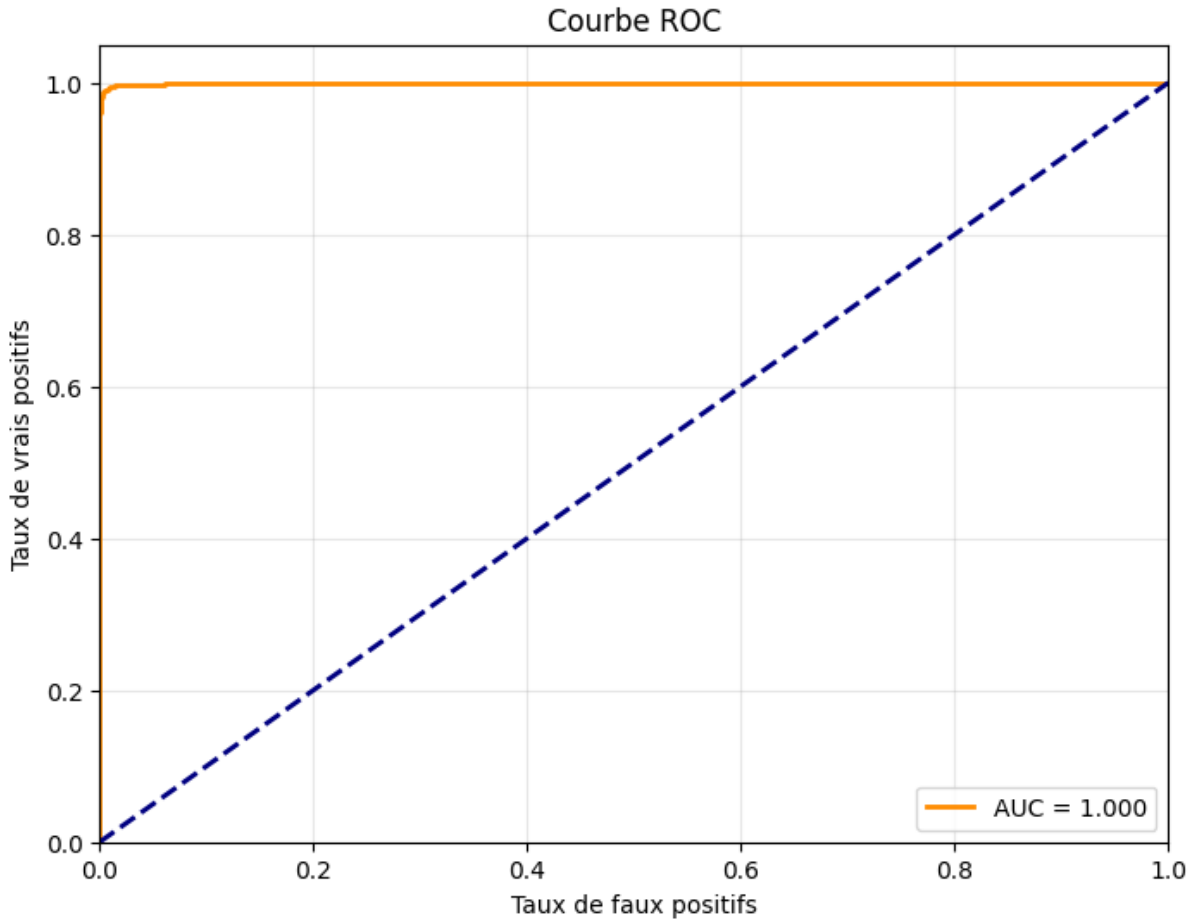


FIGURE 5.2 – Courbe ROC et score AUC pour le modèle de Régression Logistique.

Courbe Précision-Rappel et AUC-PR

La courbe Précision-Rappel, visible à la Figure 5.3, fournit une évaluation complémentaire de la performance, particulièrement pertinente dans les contextes où l'équilibre des classes peut être un enjeu ou lorsque la minimisation des faux positifs est prioritaire tout en conservant un bon rappel. De manière similaire à la courbe ROC, la courbe Précision-Rappel pour ce modèle se maintient à un niveau de précision proche de 1.0 sur presque toute l'étendue des valeurs de rappel, ne chutant de manière significative qu'aux extrêmes limites du rappel. L'Aire sous la Courbe Précision-Rappel (AUC-PR) atteint également une valeur de **1.000** (arrondie). Le modèle surpasse de manière écrasante le "Niveau de base" (indiqué par la ligne pointillée bleue sur le graphique, qui se situe aux alentours de 0.48), ce qui représente la performance d'un classifieur naïf.

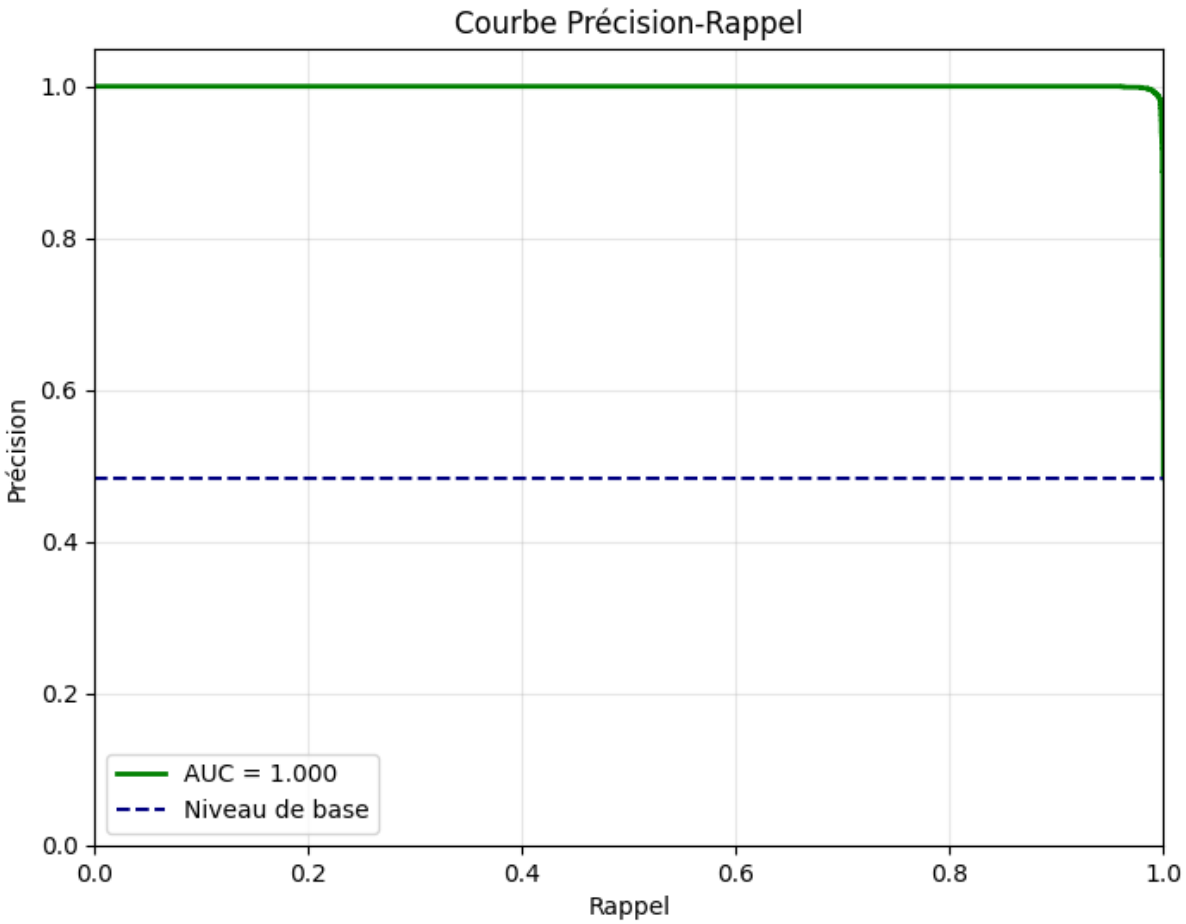


FIGURE 5.3 – Courbe Précision-Rappel et score AUC-PR pour le modèle de Régression Logistique.

Courbe d'Apprentissage

La courbe d'apprentissage, représentée à la Figure 5.4, dépeint l'évolution de la précision (*accuracy*) du modèle en fonction de la taille de l'ensemble d'entraînement. Elle montre à la fois la performance sur les données d'entraînement (score d'entraînement, courbe bleue) et sur un ensemble de validation distinct (score de validation, courbe verte). On observe que le score d'entraînement atteint très rapidement une précision de 1.0 et s'y maintient, ce qui peut se produire lorsque le modèle a une capacité suffisante pour "apprendre par cœur" les données d'entraînement. De manière plus significative, le score de validation augmente de façon constante avec l'accroissement du nombre d'exemples d'entraînement, partant d'environ 0.81 pour finalement converger vers une précision très élevée, avoisinant 0.99, à partir de 32000 exemples. L'écart entre les deux courbes, qui reste minime à la fin de l'apprentissage, ainsi que la haute performance atteinte sur l'ensemble de validation, suggèrent une excellente capacité de généralisation du modèle et une absence de surapprentissage (overfitting) notable.

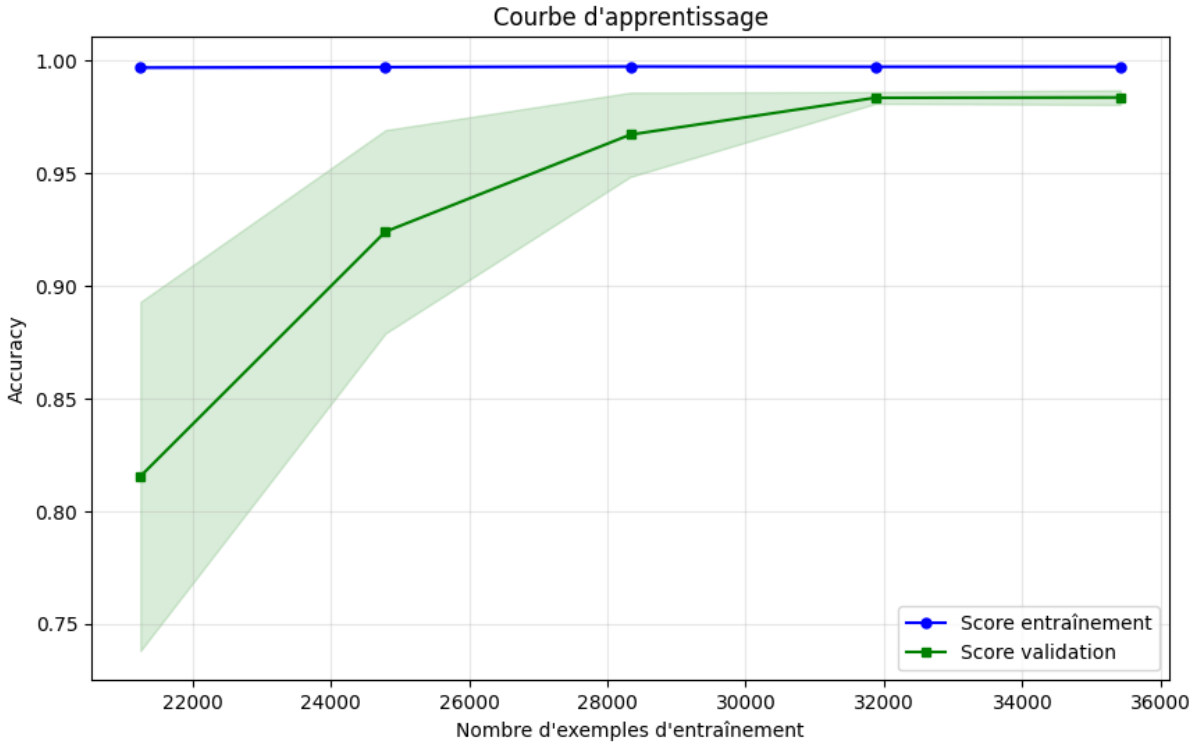


FIGURE 5.4 – Courbe d'apprentissage pour le modèle de Régression Logistique.

Visualisation des Erreurs de Classification

La Figure 5.5 propose une visualisation des erreurs de classification en projetant les caractéristiques de l'ensemble de test dans un espace à deux dimensions, obtenu par Analyse en Composantes Principales (ACP). Les instances correctement classifiées sont généralement marquées par des cercles, tandis que les erreurs sont indiquées par des croix ('X'). La couleur des points reflète la classe réelle de l'article. Il est important de noter que la réduction à seulement deux composantes principales entraîne une perte substantielle d'information, ces deux composantes n'expliquant respectivement que 1.23% et 0.56% de la variance totale des données originales, qui sont de très haute dimension. Néanmoins, la visualisation montre une séparation relativement claire entre les deux nuages de points (amas rouges et bleus correspondant aux deux classes). Les quelques instances mal classées (croix noires) apparaissent dispersées parmi les points correctement classés, plutôt que de former des zones de confusion denses et bien définies dans cet espace projeté. Cela suggère que les erreurs du modèle sont plutôt des cas isolés et ne découlent pas d'une incapacité à séparer des régions entières de l'espace des caractéristiques.

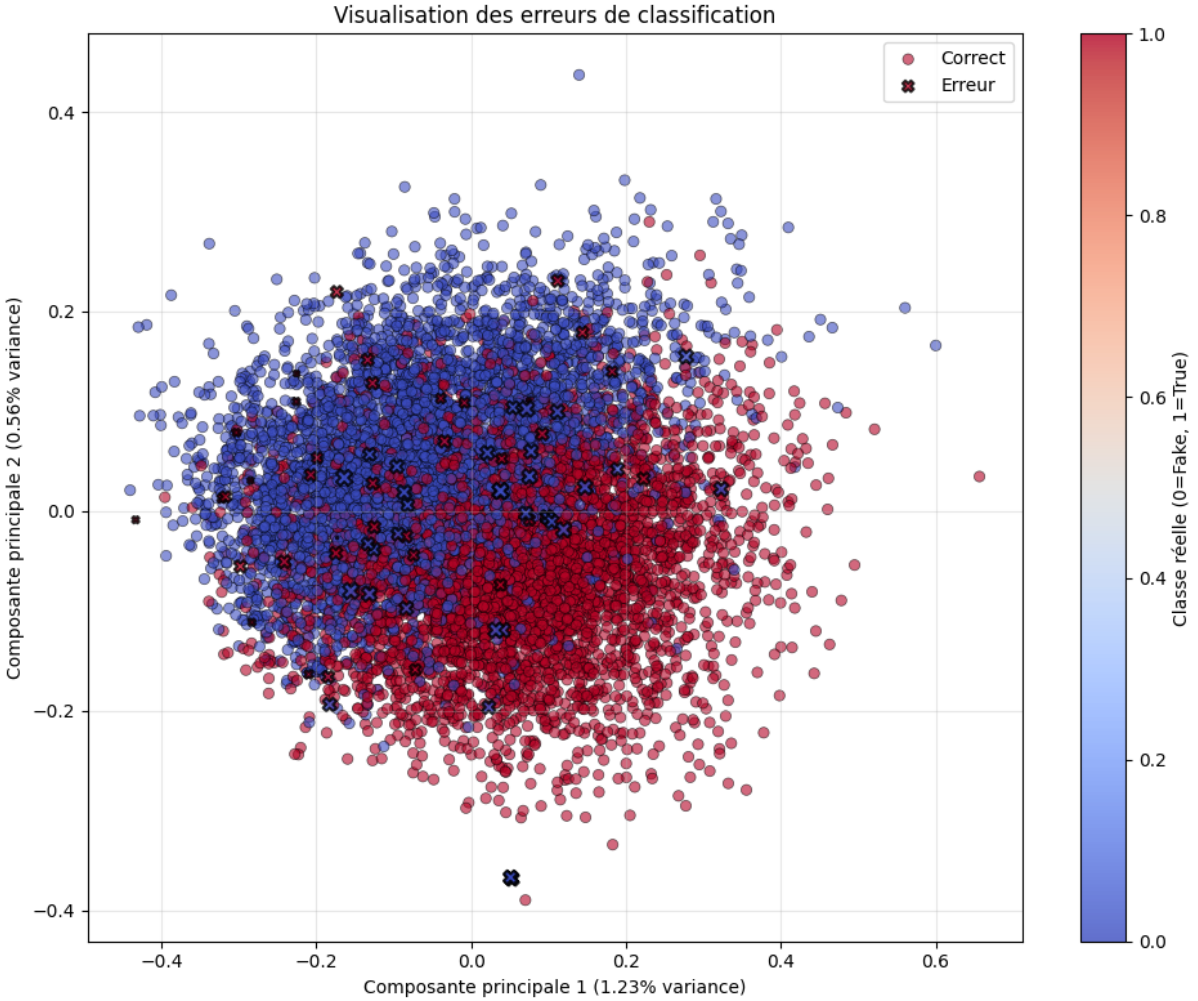


FIGURE 5.5 – Visualisation des erreurs de classification pour le modèle de Régression Logistique après réduction par ACP.

En synthèse, l'évaluation approfondie du modèle de Régression Logistique met en évidence des performances globales exceptionnelles pour la tâche de classification des fausses nouvelles. L'ensemble des métriques quantitatives (précision, rappel, F1-score, AUC) s'approche de la perfection, et les différentes visualisations corroborent la capacité robuste et fiable du modèle à discriminer efficacement les fausses nouvelles des vraies nouvelles, avec un taux d'erreur très faible et une excellente généralisation aux données non vues..

5.4 Modèle 2 : Machine à Vecteurs de Support (SVM)

Le deuxième algorithme de classification évalué dans ce projet est la Machine à Vecteurs de Support (SVM). Les SVM sont des modèles d'apprentissage supervisé particulièrement reconnus pour leur efficacité dans les espaces de caractéristiques de haute dimension, ce qui les rend pertinents pour les données textuelles vectorisées. Le principe

fondamental d'un SVM est de trouver un hyperplan optimal dans cet espace de haute dimension qui sépare au mieux les points de données appartenant à différentes classes, en maximisant la marge entre ces classes. Différents types de noyaux (kernels) peuvent être utilisés pour permettre aux SVM de modéliser des relations non linéaires.

5.4.1 Configuration et Entraînement du Modèle

Pour l'implémentation du SVM, la classe `SVC` (Support Vector Classification) de la bibliothèque scikit-learn a été utilisée. Le modèle a été configuré avec un ensemble spécifique d'hyperparamètres, détaillés ci-dessous, avant son entraînement sur les données `X_train` et `y_train` (préparées comme indiqué à la Section 5.1.1).

— `kernel='sigmoid'` : Le type de noyau utilisé pour transformer l'espace des caractéristiques. Le noyau sigmoïde a été choisi pour cette expérimentation. D'autres options courantes incluent 'linear' (pour une séparation linéaire), 'poly' (polynomial), et 'rbf' (Radial Basis Function, souvent un bon choix par défaut).

— `C=1.0` : Paramètre de régularisation. Il contrôle le compromis entre la maximisation de la marge et la minimisation de l'erreur de classification sur les données d'entraînement. Une valeur plus faible de C encourage une marge plus grande au détriment de quelques erreurs de classification, tandis qu'une valeur plus élevée vise à classer correctement plus d'exemples, quitte à réduire la marge. $C = 1.0$ est une valeur standard.

— `gamma='scale'` : Coefficient du noyau pour les noyaux 'rbf', 'poly' et 'sigmoid'. La valeur `'scale'` définit gamma comme $1/(n_{\text{caractéristiques}} \times \text{Variance}(X))$.

— `degree=3` : Degré de la fonction polynomiale pour le noyau `'poly'`. Cet hyperparamètre n'est pas utilisé avec le noyau `'sigmoid'`.

— `coef0=0.0` : Terme indépendant dans la fonction noyau pour `'poly'` et `'sigmoid'`.

— `shrinking=True` : Indique s'il faut utiliser l'heuristique de réduction ("shrinking heuristic") pendant l'optimisation, ce qui peut accélérer l'entraînement.

— `probability=True` : Activer cette option permet au modèle SVM d'estimer les probabilités d'appartenance à une classe après l'entraînement. Cela est nécessaire pour certaines métriques comme l'AUC des courbes ROC et Précision-Rappel, mais peut augmenter le temps d'entraînement.

— `max_iter=100` : Nombre maximal d'itérations pour le solveur. Une valeur de 100 est relativement faible et a pu être choisie pour des raisons de temps de calcul lors de l'expérimentation. Pour une convergence optimale, surtout sur des problèmes complexes, une valeur plus élevée (ou `-1` pour aucune limite) est souvent nécessaire.

— `random_state=42` : Fixe l'état aléatoire pour la reproductibilité des résultats lorsque des processus stochastiques sont impliqués.

Le code Python ci-dessous illustre l'instanciation du modèle SVM avec ces hyperparamètres et son entraînement.

```
from sklearn.svm import SVC
# Les variables X_train et y_train sont suppos es avoir   t
   d finies  pr c demment.

# Instanciation du mod le avec les hyperparam tres choisis
svm_model = SVC( # Renomm  en svm_model pour clart
    kernel='sigmoid',
    C=1.0,
    gamma='scale',
    degree=3,              # Non utilis  par le noyau 'sigmoid'
   mais  pr sent  dans le code original
    coef0=0.0,             # Utilis  par 'sigmoid'
    shrinking=True,
    probability=True,
    max_iter=100,          # Potentiellement bas pour une
   convergence compl te
    random_state=42
)

# Entra nement du mod le sur les donn es d'entra nement
svm_model.fit(X_train, y_train)

#    ce stade, le mod le svm_model est entra n  et pr t pour l'
    valuation  .
# La sauvegarde du mod le (ex: avec joblib) peut  tre  effectu e
   ici ou apr s l'  valuation  .
# from joblib import dump
# dump(svm_model, 'models/svm_model.joblib')
```

Listing 5.4 – Instanciation et entraînement du modèle SVM

Une fois l'entraînement terminé, le modèle `svm_model` est prêt pour une évaluation détaillée de ses performances sur l'ensemble de test.

5.4.2 Évaluation des Performances

L'évaluation du modèle SVM entraîné a été réalisée sur l'ensemble de test `X_test` et `y_test`. La précision globale (*accuracy*) du modèle SVM sur cet ensemble de test est de **0.9526** (soit 95.26%). Le rapport de classification détaillé ci-dessous fournit une analyse plus fine des performances par classe :

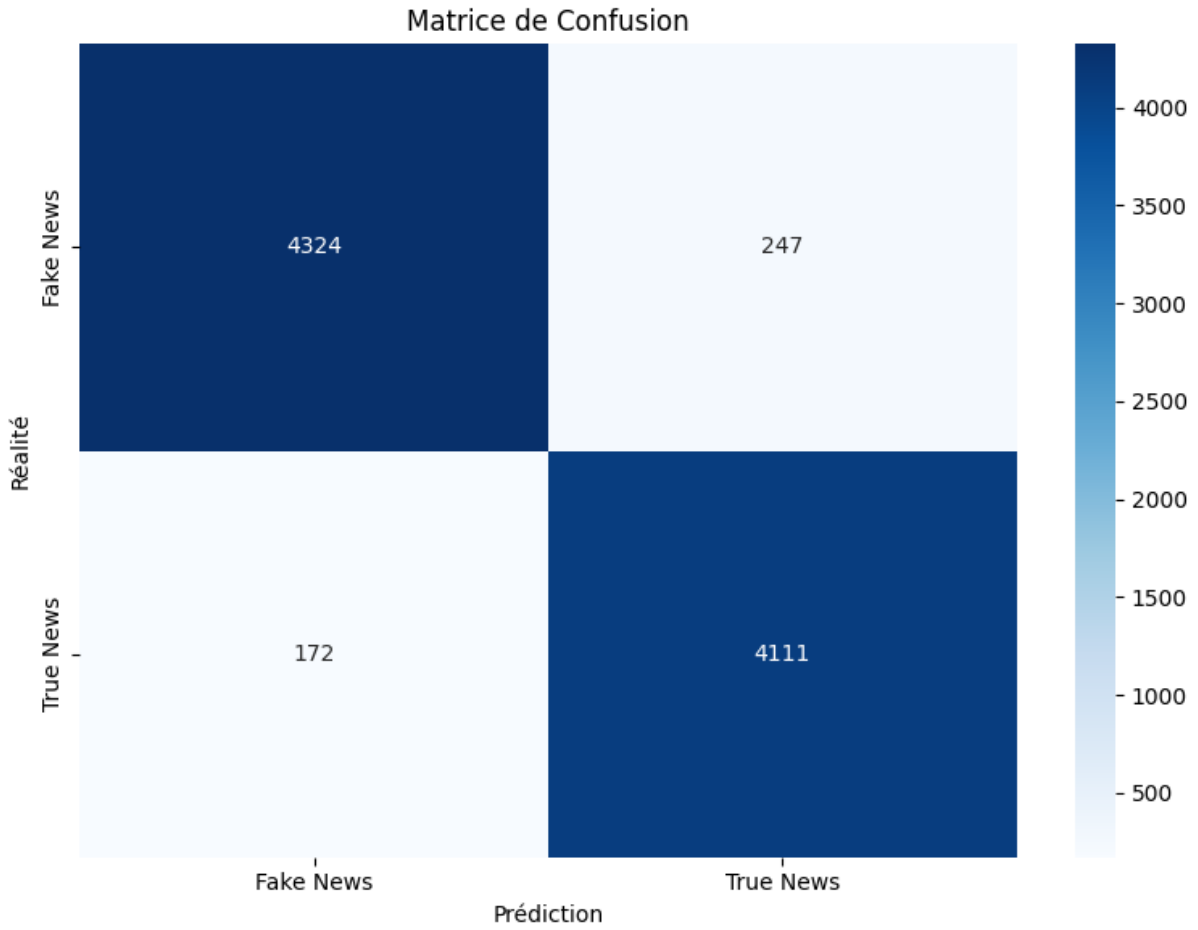
	precision	recall	f1-score	support
0.0	0.96	0.95	0.95	4571
1.0	0.94	0.96	0.95	4283
accuracy			0.95	8854
macro avg	0.95	0.95	0.95	8854
weighted avg	0.95	0.95	0.95	8854

Le rapport montre que pour la classe 0.0 ("Fake News"), la précision est de 0.96 et le rappel de 0.95. Pour la classe 1.0 ("True News"), la précision est de 0.94 et le rappel de 0.96. Les scores F1, qui combinent précision et rappel, sont de 0.95 pour les deux classes. Bien que ces scores soient élevés, ils sont légèrement inférieurs à ceux obtenus par le modèle de Régression Logistique.

Les visualisations suivantes aident à mieux comprendre ces performances.

Matrice de Confusion

La matrice de confusion du modèle SVM est présentée à la Figure 5.4.2. Elle indique que 4324 articles "Fake News" (classe 0) ont été correctement classifiés, tandis que 247 ont été incorrectement classifiés comme "True News". Concernant les "True News" (classe 1), 4111 ont été correctement identifiées, et 172 ont été erronément classifiées comme "Fake News". Le nombre total d'erreurs ($247 + 172 = 419$) est plus élevé que pour le modèle de Régression Logistique.



Courbe ROC et AUC

La Figure 5.6 montre la courbe ROC pour le modèle SVM. La courbe monte rapidement vers le coin supérieur gauche, indiquant une bonne capacité de discrimination. L'Aire Sous la Courbe (AUC) est de **0.990**, ce qui est un excellent score, bien que légèrement inférieur à l'AUC quasi parfaite du modèle de Régression Logistique. Cela signifie que le SVM est également très performant pour distinguer les deux classes.

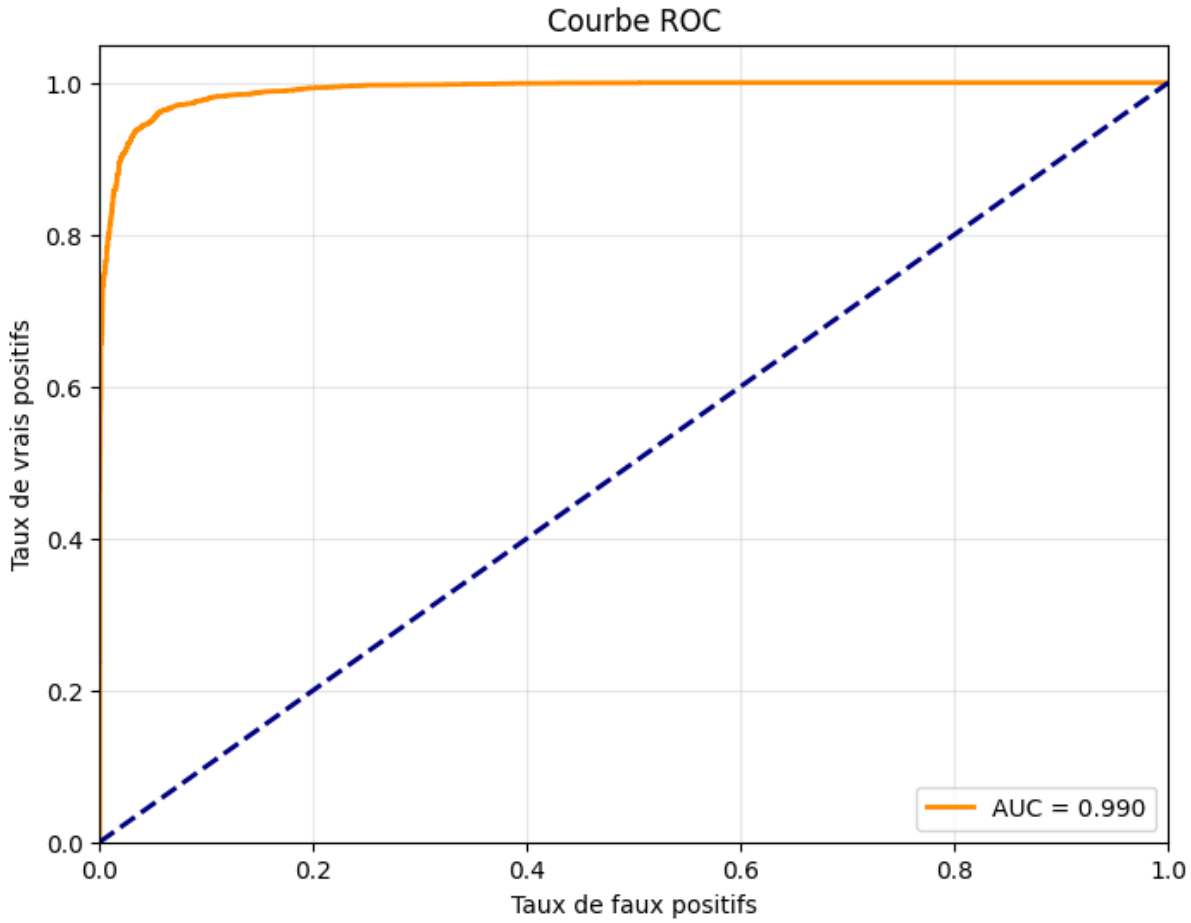


FIGURE 5.6 – Courbe ROC et score AUC pour le modèle SVM.

Courbe Précision-Rappel et AUC-PR

La courbe Précision-Rappel du SVM est illustrée à la Figure 5.7. La courbe se maintient à un niveau de précision élevé pour une grande partie de la plage de rappel. L'Aire sous la Courbe Précision-Rappel (AUC-PR) est de **0.990**. Comme pour la courbe ROC, cette performance est excellente et significativement au-dessus du niveau de base (environ 0.48), mais légèrement en deçà de celle du modèle de Régression Logistique.

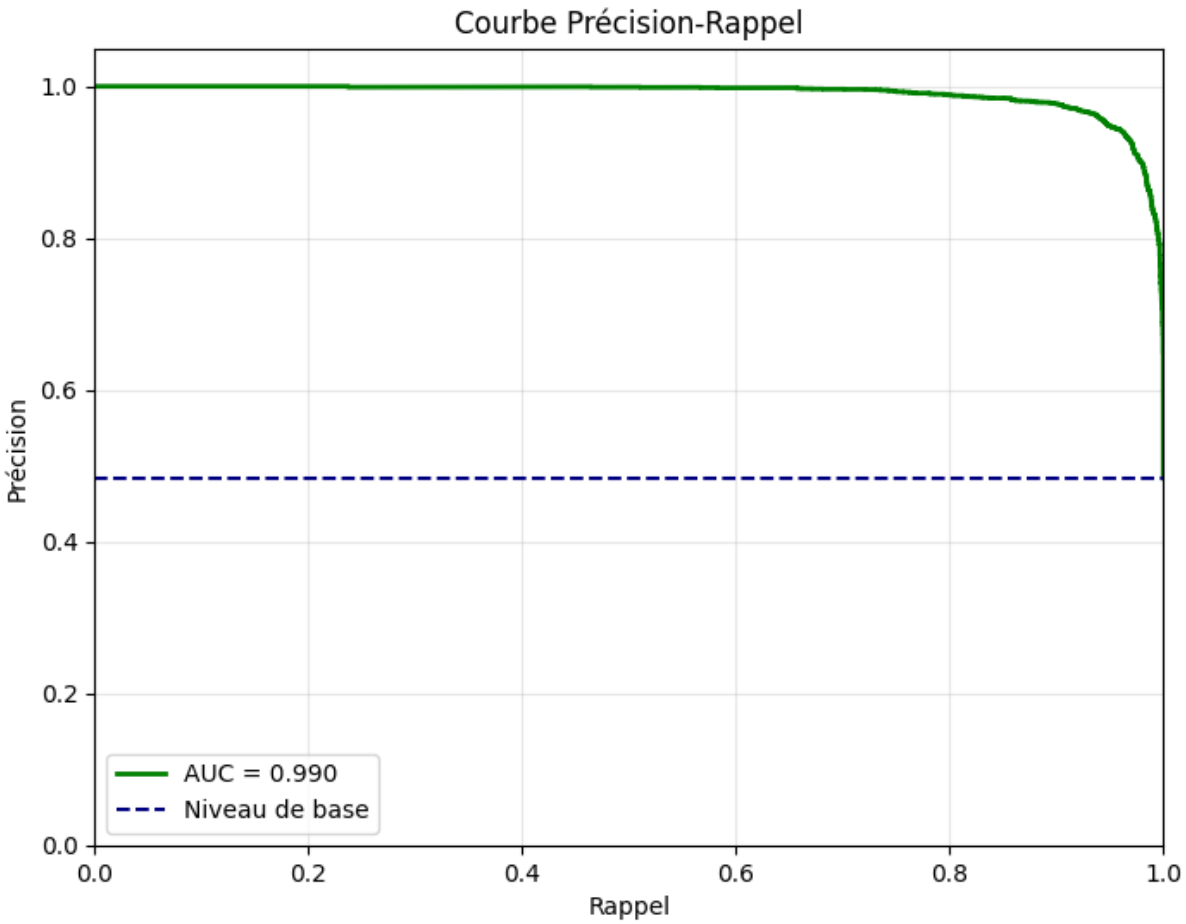


FIGURE 5.7 – Courbe Précision-Rappel et score AUC-PR pour le modèle SVM.

Courbe d'Apprentissage

La courbe d'apprentissage pour le SVM est présentée à la Figure 5.8. Le score d'entraînement (courbe bleue) est élevé, autour de 0.98 et diminue légèrement avec plus de données. Le score de validation (courbe verte) augmente avec la taille de l'échantillon d'entraînement, partant d'environ 0.90 pour atteindre environ 0.93 avec le maximum de données. L'écart entre les scores d'entraînement et de validation est plus prononcé ici qu'avec la Régression Logistique, et les deux courbes ne convergent pas aussi parfaitement, ce qui pourrait suggérer un léger surapprentissage ou que le modèle pourrait bénéficier de plus de données ou d'un ajustement des hyperparamètres (notamment `max_iter` et le paramètre C). La performance de validation semble encore en progression, indiquant un potentiel d'amélioration.

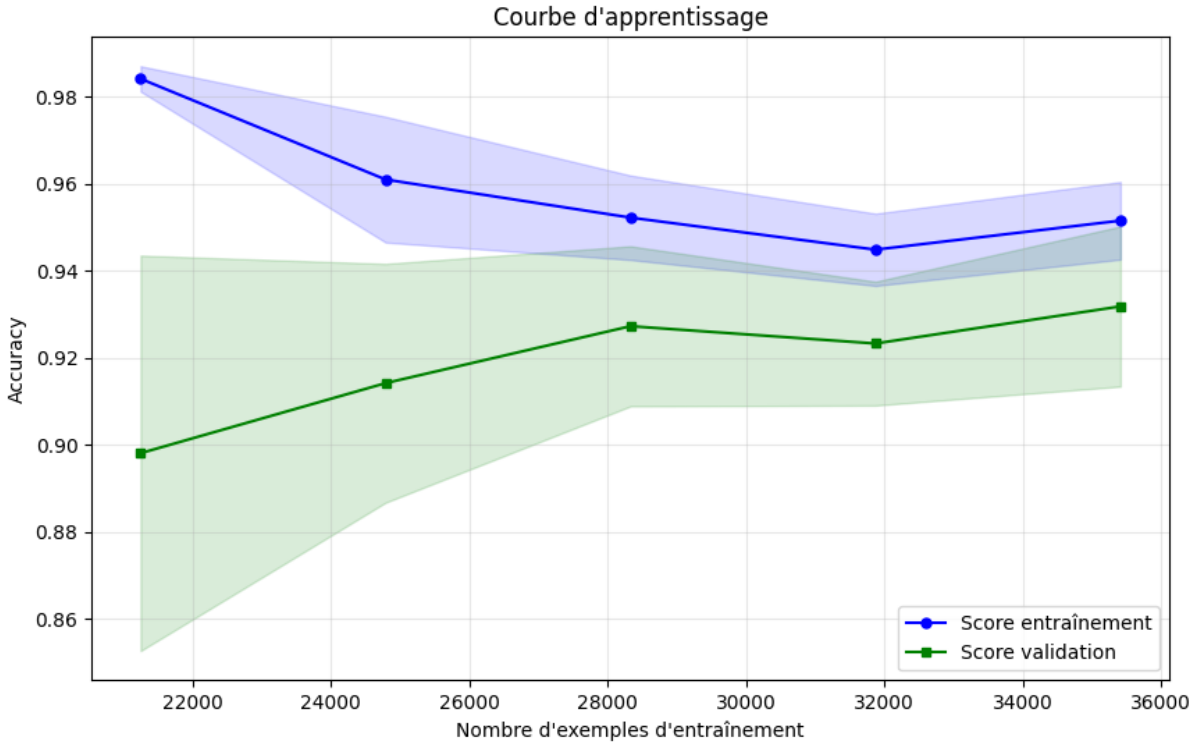
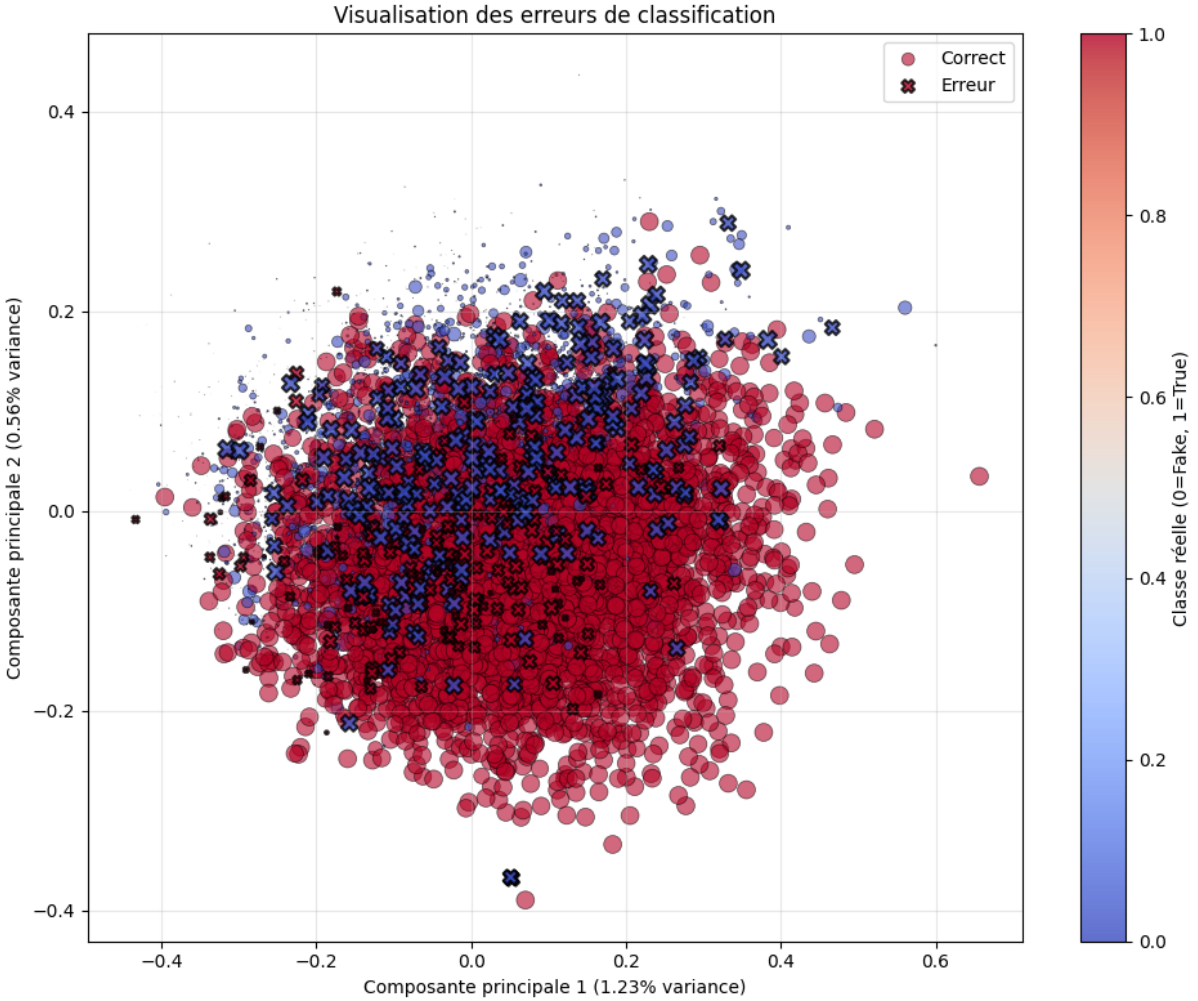


FIGURE 5.8 – Courbe d'apprentissage pour le modèle SVM.

Visualisation des Erreurs de Classification

La Figure 5.4.2 montre la projection des erreurs de classification du SVM dans l'espace réduit par ACP. Similairement à la Régression Logistique, la réduction de dimension ne capture qu'une petite partie de la variance (1.23% et 0.56%). La séparation entre les classes est visible, mais il y a un certain chevauchement où les erreurs (croix noires) se produisent. La distribution des erreurs ne semble pas indiquer une zone de confusion unique et massive, mais plutôt des erreurs dispersées.



En résumé, le modèle SVM avec un noyau sigmoïde et les paramètres choisis (notamment un `max_iter` limité) fournit de très bonnes performances, avec une précision globale et des scores AUC élevés. Cependant, il est légèrement surpassé par le modèle de Régression Logistique sur ce jeu de données et avec cette configuration spécifique. Un ajustement plus poussé des hyperparamètres du SVM, en particulier une augmentation de `max_iter` et l'exploration d'autres noyaux (comme 'linear' ou 'rbf'), pourrait potentiellement améliorer ses résultats.

5.5 Modèle 3 : Apprentissage Profond (Réseau de Neurones avec Keras)

Pour explorer des approches potentiellement plus puissantes capables de capturer des relations non linéaires complexes dans les données, un modèle d'apprentissage profond (Deep Learning) basé sur un réseau de neurones artificiels a été implémenté. La bibliothèque Keras, une interface de haut niveau pour TensorFlow, a été utilisée pour construire

et entraîner ce modèle. Les réseaux de neurones denses (également appelés perceptrons multicouches, MLP) sont capables d'apprendre des hiérarchies de caractéristiques à partir des données d'entrée.

5.5.1 Architecture du Modèle et Configuration

Le réseau de neurones a été défini comme un modèle séquentiel (une pile linéaire de couches) en utilisant la classe `Model_fak_news` que nous avons créée. L'architecture de ce réseau est la suivante :

- **Couche d'Entrée et Première Couche Cachée :** Une couche `Dense` avec 1024 neurones et une fonction d'activation ReLU (*Rectified Linear Unit*). La forme de l'entrée (`input_shape`) correspond à la dimensionnalité de nos vecteurs de caractéristiques (environ 43504). Une régularisation $L2$ avec un facteur de 0.0001 est appliquée aux poids de cette couche pour aider à prévenir le surapprentissage.
- **Couche de Dropout :** Une couche `Dropout` avec un taux de 0.1 est appliquée après la première couche dense. Le dropout est une technique de régularisation qui désactive aléatoirement une fraction des neurones pendant l'entraînement, ce qui force le réseau à apprendre des représentations plus robustes et moins dépendantes de neurones spécifiques.
- **Deuxième Couche Cachée :** Une couche `Dense` avec 512 neurones, activation ReLU, et régularisation $L2$ (0.0001). Suivie d'une couche `Dropout` (0.1).
- **Troisième Couche Cachée :** Une couche `Dense` avec 128 neurones, activation ReLU, et régularisation $L2$ (0.0001). Suivie d'une couche `Dropout` (0.1).
- **Quatrième Couche Cachée :** Une couche `Dense` avec 64 neurones, activation ReLU, et régularisation $L2$ (0.0001).
- **Couche de Sortie :** Une couche `Dense` avec 2 neurones et une fonction d'activation `softmax`. Cette configuration est typique pour une classification binaire lorsque les étiquettes sont encodées en "one-hot" (par exemple, $[1, 0]$ pour la classe 0 et $[0, 1]$ pour la classe 1). La sortie softmax donne une distribution de probabilité sur les deux classes.

Le code Python définissant cette architecture au sein de la classe `Model_fak_news` est le suivant :

```
# Supposer les importations n cessaires :
# import keras # Ou from tensorflow import keras
# from keras.models import Sequential, load_model
# from keras.layers import Dense, Dropout
# from tensorflow.keras.regularizers import l2
```

```
6
7  class Model_fak_news:
8      def __init__(self, input_size):
9          self.model = Sequential()
10         self.model.add(Dense(1024, activation='relu',
11                              input_shape=(input_size,),
12                              kernel_regularizer=l2(0.0001)))
13         self.model.add(Dropout(0.1))
14         self.model.add(Dense(512, activation='relu',
15                              kernel_regularizer=l2(0.0001)))
16         self.model.add(Dropout(0.1))
17         self.model.add(Dense(128, activation='relu',
18                              kernel_regularizer=l2(0.0001)))
19         self.model.add(Dropout(0.1))
20         self.model.add(Dense(64, activation='relu',
21                              kernel_regularizer=l2(0.0001)))
22         self.model.add(Dense(2, activation='softmax')) # 2 neurones
       pour sortie binaire one-hot
23
24     def compile_model(self): # Renomm  pour  viter  conflit avec
       Keras compile
25         self.model.compile(optimizer='adam',
26                            loss='categorical_crossentropy',
27                            metrics=['accuracy'])
28
29     def fit_model(self, X_train, y_train, x_test, y_test, #
       Renomm
30                   epoch, bach, callbacks): # Renomm
31         history = self.model.fit(X_train, y_train,
32                                  validation_data=(x_test, y_test),
33                                  verbose=2,
34                                  batch_size=bach,
35                                  epochs=epoch,
36                                  callbacks=callbacks)
37         return history
```

Listing 5.5 – Définition de l'architecture du modèle Keras

Une visualisation de cette architecture est présentée à la Figure 5.9 (générée par `plot_model` de Keras).

FIGURE 5.9 – Visualisation de l'architecture du réseau de neurones Keras. (Figure à générer et à insérer)

5.5.2 Compilation et Entraînement du Modèle

Avant l'entraînement, le modèle Keras doit être compilé. La compilation configure le processus d'apprentissage. Les paramètres suivants ont été utilisés pour la méthode `compile` (ou `compile_model` dans notre classe) :

- `optimizer='adam'` : L'optimiseur Adam (Adaptive Moment Estimation) est un algorithme d'optimisation populaire et efficace, souvent utilisé comme point de départ pour l'entraînement des réseaux de neurones.
- `loss='categorical_crossentropy'` : La fonction de perte (loss function) mesure à quel point le modèle est performant sur les données d'entraînement. La "categorical crossentropy" est appropriée pour les problèmes de classification multi-classe (ou binaire avec des étiquettes one-hot et une sortie softmax).
- `metrics=['accuracy']` : La métrique utilisée pour évaluer la performance du modèle pendant l'entraînement et les tests est la précision (*accuracy*).

L'entraînement a été effectué sur les données `X_train_dl` et `y_train_dl` (préparées avec encodage one-hot comme décrit à la Section 5.1.2). Les paramètres d'entraînement étaient :

- **Époques (`epochs`) :** 15
- **Taille de lot (`batch_size`) :** 10000
- **Callbacks :** Un callback TensorBoard a été utilisé pour enregistrer les journaux d'entraînement, permettant la visualisation des métriques et de la structure du graphe du modèle dans TensorBoard.

Les données de test (`X_test_dl`, `y_test_dl`) ont été fournies comme données de validation pendant l'entraînement pour surveiller la performance du modèle sur des données non vues à chaque époque.

```
# Supposer les importations : Sequential, Dense, Dropout, l2,
    TensorBoard, datetime
# Supposer que X_train_dl, y_train_dl, X_test_dl, y_test_dl sont
    d finis

# Instanciation du mod le
input_dim = X_train_dl.shape[1] # Dimensionnalit  des
    caract ristiques d'entr e
```

```
dl_model_instance = Model_fak_news(input_dim) # Utilisation de la
    classe

# Compilation du mod le
dl_model_instance.compile_model()

# Configuration du callback TensorBoard
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H
    %M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq
    =1)

# Entra nement du mod le
history = dl_model_instance.fit_model(X_train_dl, y_train_dl,
                                      X_test_dl, y_test_dl,
                                      epoch=15,
                                      bach=10000, # Note: bach est
    utilis  dans le code original
                                      callbacks=[
    tensorboard_callback])

# Sauvegarde du mod le entra n
# dl_model_instance.model.save("models/deepmodelclassifier.h5")
```

Listing 5.6 – Instanciation

L'objet `history` retourné par la méthode `fit` contient les valeurs de la fonction de perte et des métriques pour chaque époque, à la fois pour l'ensemble d'entraînement et l'ensemble de validation.

5.5.3 Évaluation des Performances

L'évaluation du modèle d'apprentissage profond a été effectuée sur l'ensemble de test `X_test_dl` et `y_test_dl`. La précision globale (*accuracy*) du modèle sur cet ensemble de test, telle qu indiquée dans le rapport ci-dessous, est de **0.99** (soit 99%). Le rapport de classification détaillé, généré par scikit-learn (après conversion des prédictions one-hot en étiquettes de classe), est le suivant :

	precision	recall	f1-score	support
0	0.99	0.99	0.99	4571
1	0.99	0.99	0.99	4283

accuracy			0.99	8854
macro avg	0.99	0.99	0.99	8854
weighted avg	0.99	0.99	0.99	8854

Ce rapport confirme les excellentes performances du modèle d'apprentissage profond. Pour la classe 0 (correspondant, par exemple, aux "Fake News"), la précision, le rappel et le F1-score sont tous de 0.99. De même, pour la classe 1 (correspondant aux "True News"), ces métriques atteignent également 0.99. Ces résultats indiquent une capacité de classification extrêmement élevée et équilibrée entre les deux classes. Les moyennes "macro avg" et "weighted avg" de 0.99 pour toutes les métriques soulignent davantage la robustesse et la fiabilité du modèle sur ce jeu de données de test.

L'analyse des performances est complétée par les visualisations suivantes.

Courbes d'Apprentissage (Accuracy et Loss)

La Figure 5.10 présente les courbes d'apprentissage du modèle, montrant l'évolution de la précision (*accuracy*) et de la perte (*loss*) sur les ensembles d'entraînement et de validation au fil des époques.

— **Accuracy (Précision) :** La précision sur l'ensemble d'entraînement (courbe bleue "Train") augmente rapidement et atteint un plateau proche de 1.0 après seulement quelques époques. La précision sur l'ensemble de validation (courbe orange "Validation") suit une tendance similaire, augmentant rapidement depuis environ 0.72 à l'époque 0 pour atteindre un pic autour de 0.99 vers l'époque 2-3, puis se stabilise avec de légères fluctuations. L'écart entre les deux courbes est minime, ce qui est un bon signe contre le surapprentissage.

— **Loss (Perte) :** La perte sur l'ensemble d'entraînement (courbe bleue "Train") diminue de manière drastique dès la première époque, passant d'environ 0.9 à moins de 0.2, puis continue de diminuer plus lentement. La perte sur l'ensemble de validation (courbe orange "Validation") suit également une forte baisse initiale, puis se stabilise à un niveau bas (autour de 0.05-0.10). Le fait que la perte de validation ne recommence pas à augmenter de manière significative indique que le modèle ne surapprend pas de manière excessive pendant ces 15 époques.

Globalement, ces courbes suggèrent un bon ajustement du modèle, avec une convergence rapide vers une haute performance et une bonne capacité de généralisation.

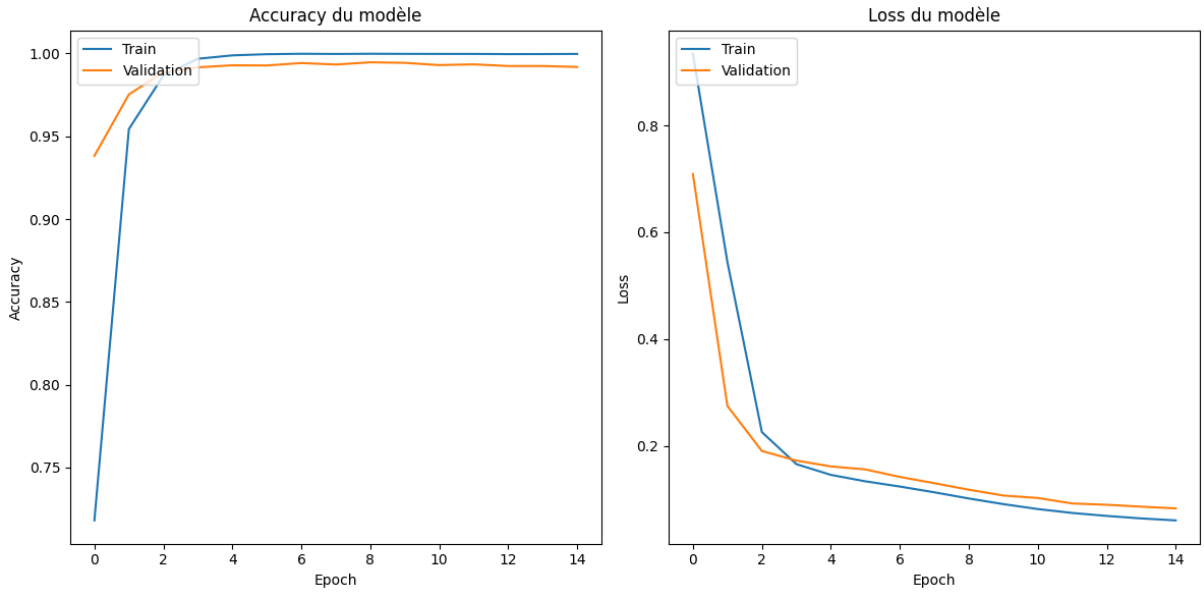


FIGURE 5.10 – Courbes d'apprentissage (Accuracy et Loss) pour le modèle d'Apprentissage Profond Keras.

Matrice de Confusion

La matrice de confusion du modèle d'apprentissage profond est illustrée à la Figure 5.11. Elle montre que 4244 articles "Fake News" (classe 0) ont été correctement classifiés et 39 ont été incorrectement classifiés comme "True News". Pour les "True News" (classe 1), 4538 ont été correctement identifiées, tandis que 33 ont été erronément classifiées comme "Fake News". Le nombre total d'erreurs ($39 + 33 = 72$) est extrêmement faible, indiquant une très haute fidélité de classification.

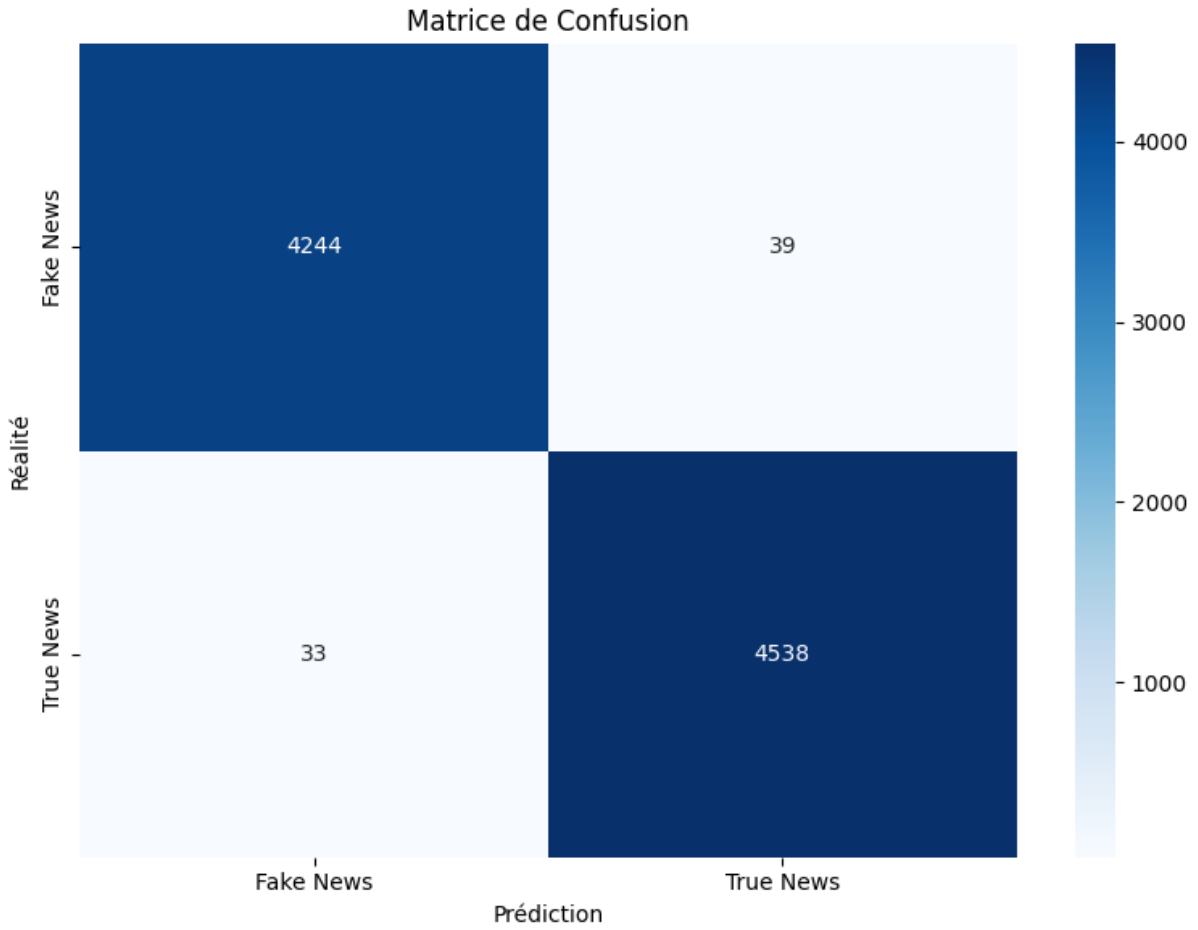


FIGURE 5.11 – Matrice de Confusion pour le modèle d'Apprentissage Profond Keras sur l'ensemble de test.

Courbe ROC et AUC

La Figure 5.12 affiche la courbe ROC pour le modèle Keras. Comme pour les modèles précédents, la courbe s'approche très rapidement du coin supérieur gauche, signe d'une excellente performance. L'Aire Sous la Courbe (AUC) est de **1.000** (arrondie), indiquant une capacité de discrimination quasi parfaite entre les deux classes.

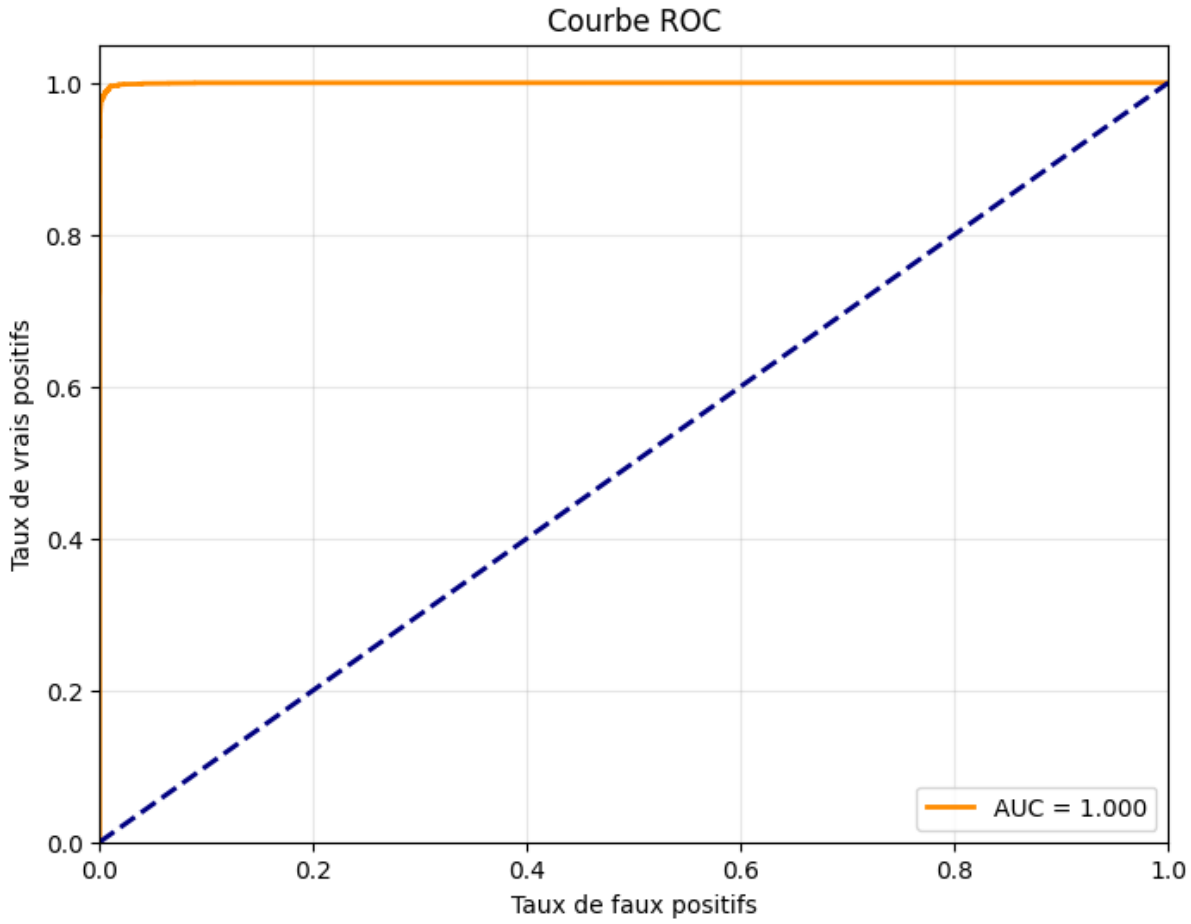


FIGURE 5.12 – Courbe ROC et score AUC pour le modèle d'Apprentissage Profond Keras.

Courbe Précision-Rappel et AUC-PR

La courbe Précision-Rappel est présentée à la Figure 5.13. Elle démontre également des performances exceptionnelles, avec une précision se maintenant proche de 1.0 sur la majorité de la plage de rappel. L'Aire sous la Courbe Précision-Rappel (AUC-PR) est de **1.000** (arrondie). Le modèle surpasse très largement le "Niveau de base" (ligne pointillée bleue, autour de 0.51).

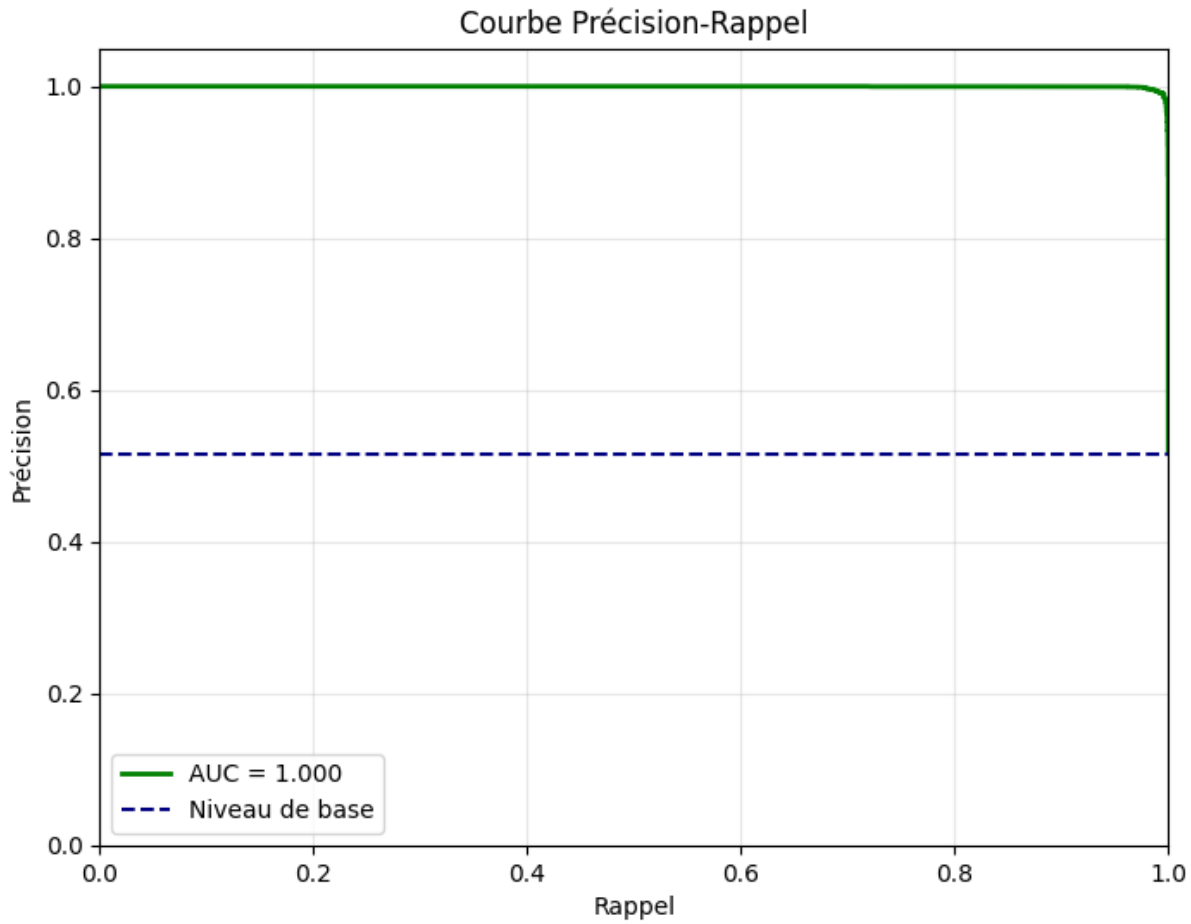


FIGURE 5.13 – Courbe Précision-Rappel et score AUC-PR pour le modèle d'Apprentissage Profond Keras.

Visualisation des Erreurs de Classification

La Figure 5.14 montre la projection des erreurs de classification du modèle Keras dans l'espace réduit par ACP. Les deux composantes principales capturent une faible part de la variance (1.23% et 0.56%). Malgré cela, on observe une bonne séparation des classes. Les erreurs (croix noires) sont peu nombreuses et semblent dispersées, ne formant pas de cluster distinct, ce qui suggère que les erreurs sont des cas isolés plutôt qu'une faiblesse systémique du modèle dans une région particulière de l'espace des caractéristiques projeté.

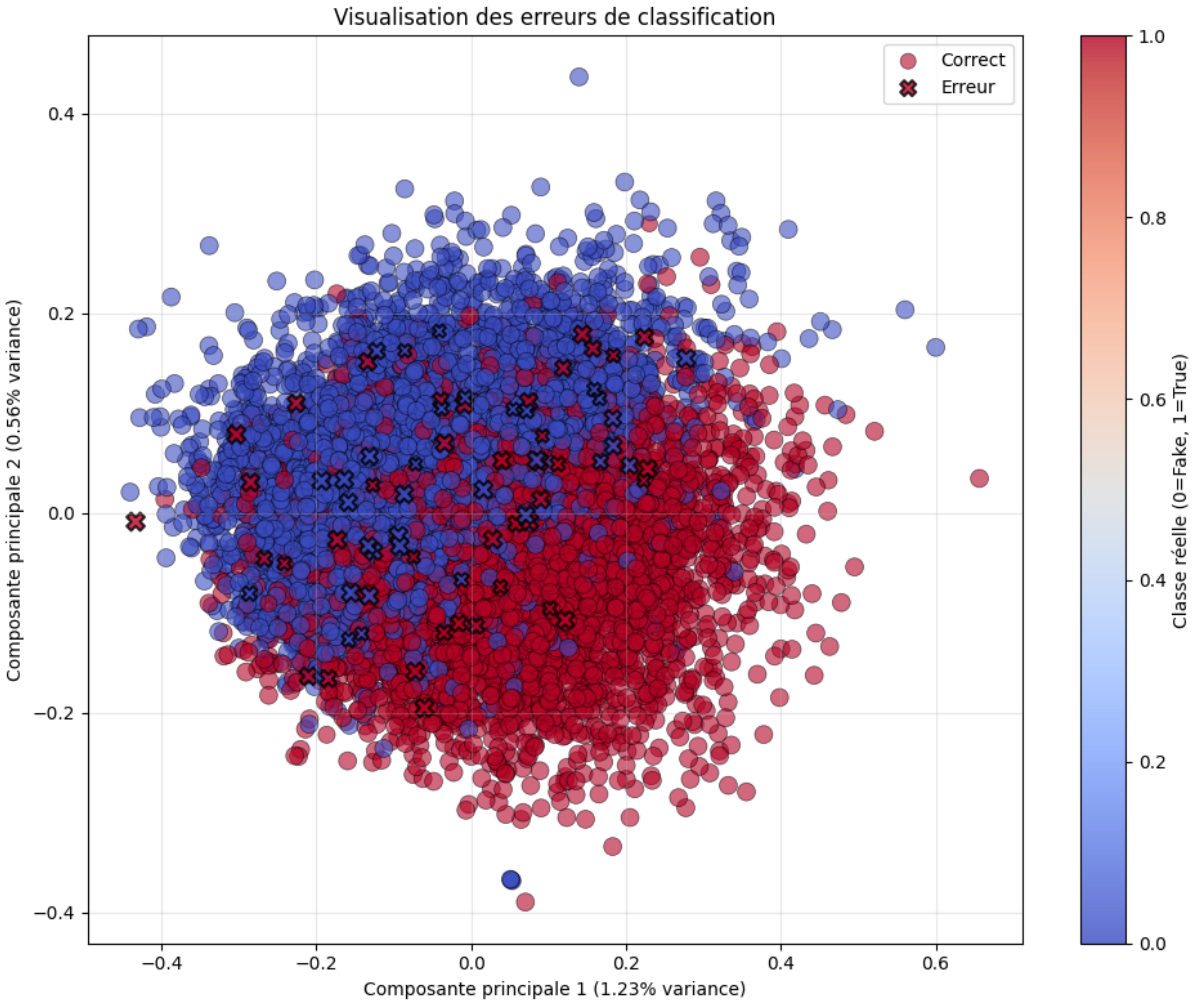


FIGURE 5.14 – Visualisation des erreurs de classification pour le modèle d'Apprentissage Profond Keras après réduction par ACP.

En conclusion, le modèle d'apprentissage profond basé sur un réseau de neurones Keras affiche des performances remarquables, rivalisant et même surpassant potentiellement les modèles classiques sur cette tâche. Les métriques et les visualisations indiquent une capacité de classification et de généralisation de très haut niveau.

Chapitre 6

Application Web Streamlit pour la Détection de Fausses Nouvelles

Afin de rendre le système de détection de fausses nouvelles accessible et interactif, une application web a été développée en utilisant la bibliothèque Python Streamlit. Cette application sert d'interface utilisateur pour interroger les modèles entraînés et obtenir des évaluations sur la véracité d'articles de presse.

6.1 Fonctionnalités Principales de l'Application

L'application, intitulée « Vérificateur de Fake News », permet aux utilisateurs de :

- **Rechercher des nouvelles** par mot-clé et année via l'API Google Fact Check Tools.
- **Sélectionner un modèle de vérification** parmi ceux entraînés (SVM, Régression Logistique, Apprentissage Profond).
- **Visualiser les articles** sous forme de cartes interactives affichant titre, source, date, et un extrait.
- Consulter un **statut de vérification initial** si fourni par l'API, ou marquer l'article comme « Non vérifié ».
- Lancer une **vérification à la demande** pour un article, déclenchant la prédiction du modèle sélectionné.
- Voir la **prédiction du modèle** (RÉEL/FAUX) et un score de confiance (pour le modèle d'Apprentissage Profond) s'afficher sur la carte de l'article.

L'interface utilisateur a été conçue pour être claire et intuitive, avec un style visuel amélioré par du CSS personnalisé.

6.2 Composants Clés et Flux de Travail Simplifié

Le fonctionnement de l'application repose sur plusieurs éléments essentiels :

1. **Prétraitement du Texte :** Les fonctions de nettoyage de texte (`text_cleaning`) et de préparation des caractéristiques (`prepare_text`), développées pour l'entraînement des modèles, sont réutilisées pour traiter les textes des articles soumis par l'utilisateur, assurant ainsi la cohérence. *Une attention particulière doit être portée à l'alignement exact de la structure du vecteur de caractéristiques généré par `prepare_text` avec celle utilisée lors de l'entraînement des modèles.*
2. **Chargement des Modèles :** Une fonction charge dynamiquement le modèle d'apprentissage automatique sélectionné par l'utilisateur (fichiers `.joblib` pour SVM/-Régression Logistique, fichier `.h5` pour le modèle Keras).
3. **Inférence :** Une fonction de prédiction prend en charge la préparation du texte et l'appel au modèle choisi pour obtenir un verdict.
4. **Intégration API :** La fonction `get_claims` interroge l'API Google Fact Check Tools pour récupérer les articles.
5. **Interface Utilisateur et Gestion de l'État :** Streamlit est utilisé pour tous les éléments interactifs (champs de saisie, boutons, sélection) et pour afficher les résultats. La gestion de l'état de session (`st.session_state`) permet de conserver les informations entre les interactions.

L'utilisateur interagit typiquement en effectuant une recherche, en sélectionnant un modèle, puis en cliquant pour vérifier un article spécifique, ce qui déclenche le pipeline de prétraitement et de prédiction.

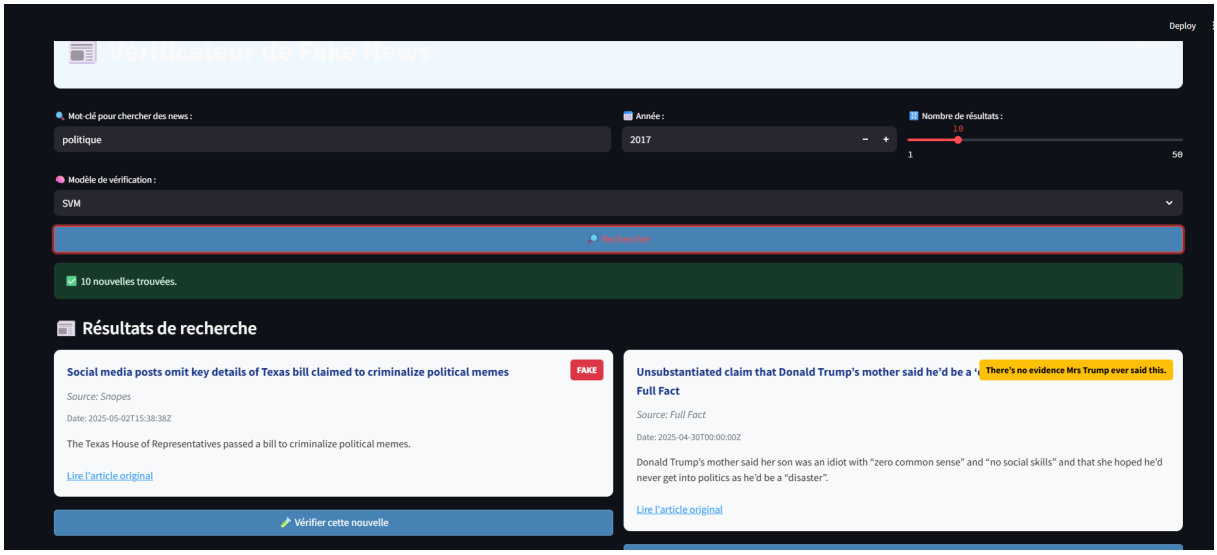


FIGURE 6.1 – l'interface principale de l'application Streamlit.

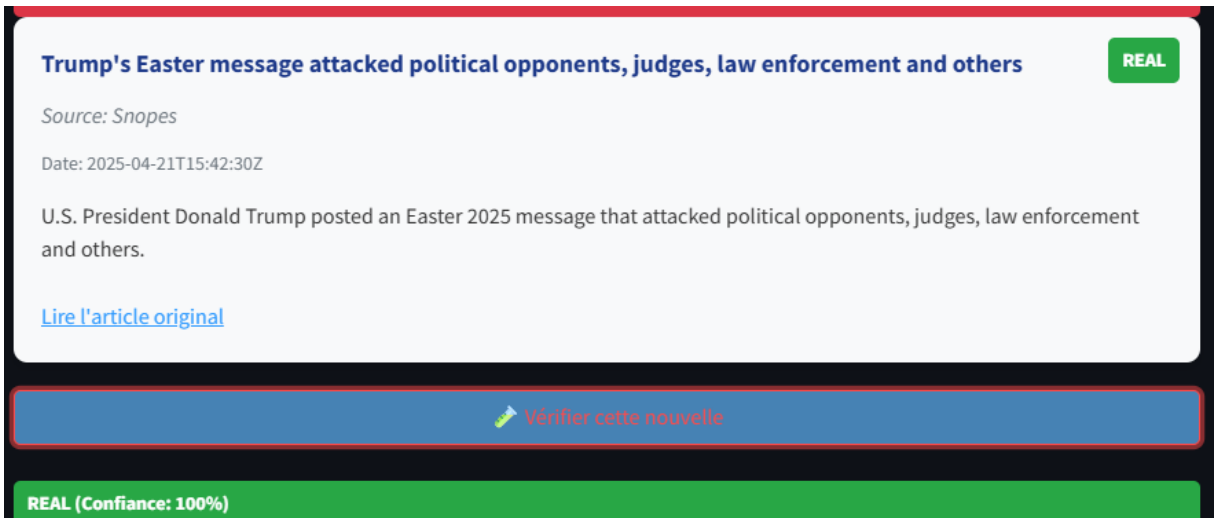


FIGURE 6.2 – carte de nouvelle après vérification par un modèle.

Chapitre 7

Discussion des Résultats

Ce chapitre propose une analyse consolidée des résultats obtenus tout au long du projet, depuis le traitement initial des données jusqu'à l'évaluation des performances des modèles de classification. Il met en lumière les observations clés concernant l'efficacité des techniques de prétraitement, d'ingénierie des caractéristiques, et les performances comparatives des modèles.

7.1 Impact du Prétraitement et de l'Ingénierie des Caractéristiques

Le pipeline de prétraitement appliqué aux données textuelles brutes – incluant la conversion en minuscules, la suppression de la ponctuation et des caractères non alphabétiques, l'élimination d'une liste exhaustive de mots vides, et la lemmatisation – a été fondamental pour standardiser le corpus et réduire le bruit. Ces étapes ont préparé le terrain pour une ingénierie des caractéristiques plus efficace.

L'approche principale d'ingénierie des caractéristiques a reposé sur la vectorisation **TF-IDF**, en considérant à la fois les unigrammes et les bigrammes, et en appliquant un seuil de fréquence minimale de document (`min_df=50`). Cela a résulté en un espace de caractéristiques de haute dimension (environ 43504 dimensions) mais épars, ce qui est typique pour les données textuelles. L'ajout de la **similarité cosinus TF-IDF** entre le titre et le texte visait à capturer la cohérence sémantique directe basée sur les fréquences de mots pondérées.

Parallèlement, les plongements **Doc2Vec** ont été appris pour obtenir des représentations vectorielles denses (100 dimensions) des titres et des textes. La **similarité cosinus Doc2Vec** a ensuite été calculée et, de manière cruciale, ajoutée comme caractéristique supplémentaire à l'ensemble TF-IDF principal. Cette combinaison visait à enrichir les caractéristiques TF-IDF, basées sur les mots, avec une information sémantique de plus haut niveau capturée par Doc2Vec. Un ensemble de caractéristiques purement Doc2Vec (dense, 201 dimensions) a également été préparé, bien que non utilisé pour l'entraînement des modèles présentés.

Les principaux défis rencontrés durant ces phases ont été la gestion de la mémoire lors du traitement des grands volumes de texte et des matrices éparses (mitigée par le

traitement par lots) et le temps de calcul nécessaire pour l'entraînement des modèles Doc2Vec et la vectorisation TF-IDF sur l'ensemble du corpus. Néanmoins, la qualité des caractéristiques générées semble avoir été un facteur déterminant pour les excellentes performances obtenues par les modèles.

7.2 Analyse Comparative des Performances des Modèles

Trois modèles de classification ont été entraînés et évalués sur les mêmes ensembles de données d'entraînement et de test, en utilisant l'ensemble de caractéristiques TF-IDF augmenté par les deux mesures de similarité cosinus.

— **Régression Logistique :** Ce modèle a démontré des performances exceptionnelles, atteignant une précision globale de **99.19**%. Les scores de précision, de rappel et F1-score pour les deux classes ("Fake News" et "True News") étaient uniformément élevés à 0.99. Les courbes ROC et Précision-Rappel, avec des AUC respectives de 1.000 (arrondi), indiquent une capacité de discrimination quasi parfaite. La courbe d'apprentissage a montré une bonne convergence et une excellente généralisation, avec un faible écart entre les performances d'entraînement et de validation. Pour des données textuelles éparses et de haute dimension, la Régression Logistique, surtout avec une régularisation $L2$, est souvent un strong baseline et s'est avérée ici extrêmement efficace.

— **Machine à Vecteurs de Support (SVM) :** Le SVM, configuré avec un noyau sigmoïde et un nombre maximal d'itérations limité à 100 (`max_iter=100`), a atteint une précision globale de **95.26**%. Bien que ce soit un très bon score, il est notablement inférieur à celui de la Régression Logistique et du modèle d'Apprentissage Profond. Les scores de précision, rappel et F1-score par classe se situaient autour de 0.95. L'AUC ROC et l'AUC PR étaient également excellents (0.990). La courbe d'apprentissage a suggéré que le modèle pourrait potentiellement bénéficier d'un plus grand nombre d'itérations pour converger pleinement ou d'un ajustement plus poussé de ses hyperparamètres (comme C et le type de noyau). La limitation sur `max_iter` a probablement empêché le SVM d'atteindre son plein potentiel sur ce jeu de données.

— **Modèle d'Apprentissage Profond (Keras) :** Le réseau de neurones profond a également affiché des performances remarquables, avec une précision globale de **99.16**% (calculée à partir de la matrice de confusion $(4244 + 4538)/(4244 + 39 + 33 + 4538) \approx 0.9916$, et confirmée par le rapport de classification avec une accuracy de 0.99). Les scores de précision, rappel et F1-score par classe étaient de 0.99.

Les courbes d'apprentissage ont montré une convergence rapide vers une haute performance sur les ensembles d'entraînement et de validation, avec un faible écart, indiquant une bonne généralisation grâce notamment aux techniques de régularisation (Dropout, $L2$). Les AUC ROC et Précision-Rappel étaient également de 1.000 (arrondi). Ce résultat démontre que même sur des caractéristiques TF-IDF éparses, une architecture de réseau de neurones bien configurée peut atteindre des performances de pointe.

Synthèse Comparative : Dans cette étude, la **Régression Logistique** et le modèle d'**Apprentissage Profond Keras** ont tous deux atteint un niveau de performance quasi parfait, avec des précisions supérieures à 99% et des AUC de 1.000. Cela suggère que les caractéristiques extraites étaient hautement discriminantes. La Régression Logistique a l'avantage de la simplicité et d'un temps d'entraînement potentiellement plus court. Le modèle Deep Learning, bien que plus complexe, a également réussi à exploiter efficacement ces caractéristiques. Le **SVM**, dans sa configuration actuelle, était performant mais en retrait. Il est probable que des optimisations supplémentaires (notamment `max_iter` et choix du noyau) pourraient améliorer ses scores.

L'excellence des résultats pour au moins deux modèles souligne la qualité du pipeline de prétraitement et d'ingénierie des caractéristiques. La combinaison de TF-IDF avec des mesures de similarité (y compris celle issue de Doc2Vec) semble avoir fourni un ensemble de caractéristiques très puissant pour cette tâche de classification.

Chapitre 8

Conclusion et Travaux Futurs

Ce projet s'est attelé à la tâche complexe de la détection de fausses nouvelles en développant un pipeline complet, depuis le traitement des données textuelles brutes jusqu'à l'évaluation de modèles d'apprentissage automatique et la création d'une interface utilisateur interactive.

8.1 Résumé des Réalisations et Contributions

Les principales réalisations de ce projet peuvent être résumées comme suit :

1. **Pipeline de Prétraitement Robuste :** Un processus de nettoyage et de normalisation approfondi des données textuelles a été mis en place, incluant la lemmatisation et une gestion extensive des mots vides, assurant la production de données textuelles propres pour les étapes suivantes.
2. **Ingénierie de Caractéristiques Avancée :** Des caractéristiques textuelles ont été extraites en utilisant la technique TF-IDF (avec unigrammes et bigrammes). De plus, des plongements de documents Doc2Vec ont été appris, et la similarité cosinus entre les titres et les corps de texte (calculée à la fois pour les vecteurs TF-IDF et Doc2Vec) a été intégrée comme caractéristique, enrichissant ainsi la représentation des articles.
3. **Évaluation Comparative de Modèles :** Trois modèles de classification distincts – Régression Logistique, Machine à Vecteurs de Support (SVM), et un Réseau de Neurones Profonds (Keras) – ont été entraînés et rigoureusement évalués. La Régression Logistique et le modèle Keras ont démontré des performances exceptionnelles, avec des précisions supérieures à 99% et des AUC proches de 1.0.
4. **Développement d'une Application Interactive :** Une application web utilisant Streamlit a été créée, permettant aux utilisateurs de rechercher des nouvelles via l'API Google Fact Check Tools et d'obtenir des prédictions de véracité à la demande en utilisant les modèles entraînés. Cela démontre la faisabilité d'une application pratique du système développé.

Globalement, le projet a réussi à construire un système de détection de fausses nouvelles très performant, validant l'efficacité de la chaîne de traitement et des modèles choisis pour ce jeu de données spécifique.

8.2 Vue d'Ensemble des Ensembles de Caractéristiques et de l'Application

L'ensemble de caractéristiques principal, une matrice TF-IDF augmentée par des scores de similarité, s'est avéré hautement discriminant. Bien que de grande dimension et épars, il a été efficacement traité par les modèles linéaires et le réseau de neurones. L'application Streamlit constitue une preuve de concept réussie pour l'utilisation de ces modèles dans un contexte interactif, bien que des améliorations soient possibles, notamment concernant l'alignement précis de la fonction de préparation des caractéristiques pour l'inférence.

8.3 Recommandations pour les Prochaines Étapes et Travaux Futurs

Bien que les résultats obtenus soient très encourageants, plusieurs pistes d'amélioration et d'exploration future peuvent être envisagées pour renforcer et étendre ce travail :

- **Optimisation Poussée des Hyperparamètres :** Mettre en œuvre une recherche systématique des meilleurs hyperparamètres (par exemple, avec GridSearchCV ou RandomizedSearchCV) pour tous les modèles, en particulier pour le SVM (en testant différents noyaux et en augmentant `max_iter`) et pour le réseau de neurones (architecture, taux de dropout, optimiseurs).
- **Exploration d'Architectures d'Apprentissage Profond Alternatives :**
  - Entraîner des modèles directement sur les plongements Doc2Vec denses (l'ensemble `supp_features`).
  - Implémenter des architectures neuronales plus avancées spécifiquement conçues pour le traitement de séquences textuelles, telles que les LSTMs (Long Short-Term Memory), les GRUs (Gated Recurrent Units), ou des modèles basés sur les Transformers (comme BERT ou ses variantes), qui pourraient capturer des dépendances contextuelles plus fines directement à partir du texte ou de plongements de mots.
- **Analyse d'Importance des Caractéristiques et Interprétabilité :** Pour la Régression Logistique, analyser les coefficients pour identifier les termes TF-IDF les plus influents. Pour les modèles plus complexes (SVM avec noyaux non linéaires, réseaux de neurones), utiliser des techniques d'interprétabilité post-hoc (comme LIME ou SHAP) pour mieux comprendre leurs décisions.

— **Utilisation de la Validation Croisée :** Adopter une stratégie de validation croisée k-fold lors de l'évaluation finale des modèles et de l'optimisation des hyperparamètres pour obtenir des estimations de performance plus robustes et moins dépendantes d'une unique division entraînement/test.

— **Enrichissement des Caractéristiques :** Intégrer d'autres types de caractéristiques, telles que :

  — **Caractéristiques stylométriques :** Longueur moyenne des phrases/mots, richesse lexicale, utilisation de la ponctuation, complexité syntaxique, scores de lisibilité.

  — **Caractéristiques basées sur les métadonnées :** Si disponibles, informations sur l'éditeur, l'auteur, la date de publication par rapport à l'événement, ou même des signaux issus des réseaux sociaux (nombre de partages, types de commentaires).

— **Généralisation et Robustesse :** Tester les modèles entraînés sur des jeux de données de fausses nouvelles différents (provenant d'autres sources, couvrant d'autres sujets ou périodes) pour évaluer leur capacité de généralisation et leur robustesse face à des styles variés de désinformation.

— **Améliorations de l'Application Streamlit :**

  — Assurer un alignement parfait de la fonction `prepare_text` pour l'inférence.

  — Mettre en cache les modèles chargés pour améliorer la réactivité.

  — Renforcer la gestion des erreurs et fournir des retours utilisateurs plus clairs.

  — Envisager un mécanisme permettant aux utilisateurs de signaler des erreurs de prédiction.

En explorant ces pistes, le système de détection de fausses nouvelles pourrait gagner en précision, en robustesse, en interprétabilité et en applicabilité pratique.