جامعة محمد الأول بوجدة
UNIVERSITE MOHAMMED PREMIER OUJDA
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵎⵓⵃⴰⵎⵎⴷ ⴰⵏⴰⵎⵓⵔ ⵏ ⵡⵓⵊⴷⴰ

Université Mohammed Premier

Faculté des Sciences d'Oujda

Département d'Informatique

Master M2I Deuxième Année

◇ Oujda−Morocco ◇

كلية العلوم
Faculté des Sciences
ⵜⴰⵖⵔⵉⵎⵜ ⵏ ⵜⵣⵔⴼⵜ ⴷ ⵜⵉⵙⵙⵏⵙⵉⵡⵉⵏ

MINI PROJET

Ingénierie Linguistique

MASTER Spécialisé

INGENIERIE INFORMATIQUE

RÉALISATION D'UN SYSTÈME PERMETTANT DE MESURER LA LISIBILITÉ DES TEXTES ARABES BASÉ SUR LES SVM

Etudiantes

Qassimi Ikhlas

Ferhat Oumaima

Prof.MAZROUI Azzeddine FSO, Université Mohammed Premier, Oujda

Année universitaire : 2023−2024

Table des matières

Table des figures

Introduction générale

L'évaluation de la lisibilité des textes revêt une importance capitale dans divers contextes, influençant la compréhension et l'accessibilité des informations présentées. Dans le cadre de ce projet dédié à la création d'un système d'évaluation de la lisibilité pour les textes en langue arabe, nous avons bénéficié de l'ensemble de données fourni par notre professeur, comprenant une gamme de caractéristiques cruciales pour cette tâche.

Les fichiers textes fournis ont été le point de départ de notre travail. Ces fichiers, une fois traités, ont fourni un ensemble de caractéristiques pertinentes pour évaluer la lisibilité des textes en arabe. Parmi ces caractéristiques, nous avons utilisé :

SL1 (Nombre de phrases dans un texte) WL1 (Nombre de mots dans un texte) WL4 (Nombre de stems dans un texte) WL5 (Nombre de caractères dans un texte) AMb1 (Nombre de lemmes ambigus) VL1 (Nombre de lemmes distincts) VL2 (Nombre de lemmes fréquentes) La préparation et le traitement de ces données ont constitué une étape cruciale de notre projet. En exploitant ces caractéristiques, nous avons cherché à établir des liens entre la structure linguistique des textes arabes et leur niveau de lisibilité.

Ce rapport détaillera donc les méthodes de prétraitement des données fournies par notre professeur, ainsi que la manière dont nous avons utilisé ces caractéristiques pour entraîner et évaluer un modèle basé sur les Machines à Vecteurs de Support ou Support Vector Machine (SVM). Ce modèle vise à prédire avec précision le niveau de lisibilité des textes arabes, offrant ainsi des perspectives d'application significatives dans des domaines tels que l'éducation, la communication et le traitement automatique des langues.

Remerciements

Nous débutons nos remerciements en exprimant notre profonde gratitude envers Dieu le Tout-Puissant pour nous avoir accordé la force, la persévérance et la guidance nécessaires pour mener à bien ce projet.

Nous tenons à adresser nos remerciements les plus sincères à notre professeur, Monsieur **Azzeddine Mazroui**, pour son encadrement, ses conseils avisés et son soutien tout au long de cette étude. Sa disponibilité et son expertise ont grandement enrichi notre travail.

Nous exprimons également notre reconnaissance à nos amis et à toutes les personnes qui ont apporté leur précieuse aide et leur soutien, contribuant ainsi à la réussite de ce projet.

Oumaima FERHAT
Ikhlas QASSIMI

Chapitre 1

SVM pour classification

Introduction

— Les SVM, initialement développés pour la classification binaire, consistent à trouver une règle de décision reposant sur la séparation optimale des données par un hyperplan à marge maximale.

— Cette approche découle des travaux pionniers de Vapnik et Chervonenkis dans le domaine de l'apprentissage.

— Les SVM ont rapidement gagné en popularité en raison de leur capacité à traiter efficacement des ensembles de données volumineux, de leur fondement théorique solide et de leurs performances robustes dans des applications pratiques diverses.

— Leur adoption s'est étendue à une multitude de domaines, incluant la bio-informatique, la recherche d'information, la finance, le traitement d'images, et bien d'autres.

— On distingue deux types de données :

  ▷ Les données linéairement séparables

  ▷ Les données non linéairement séparables

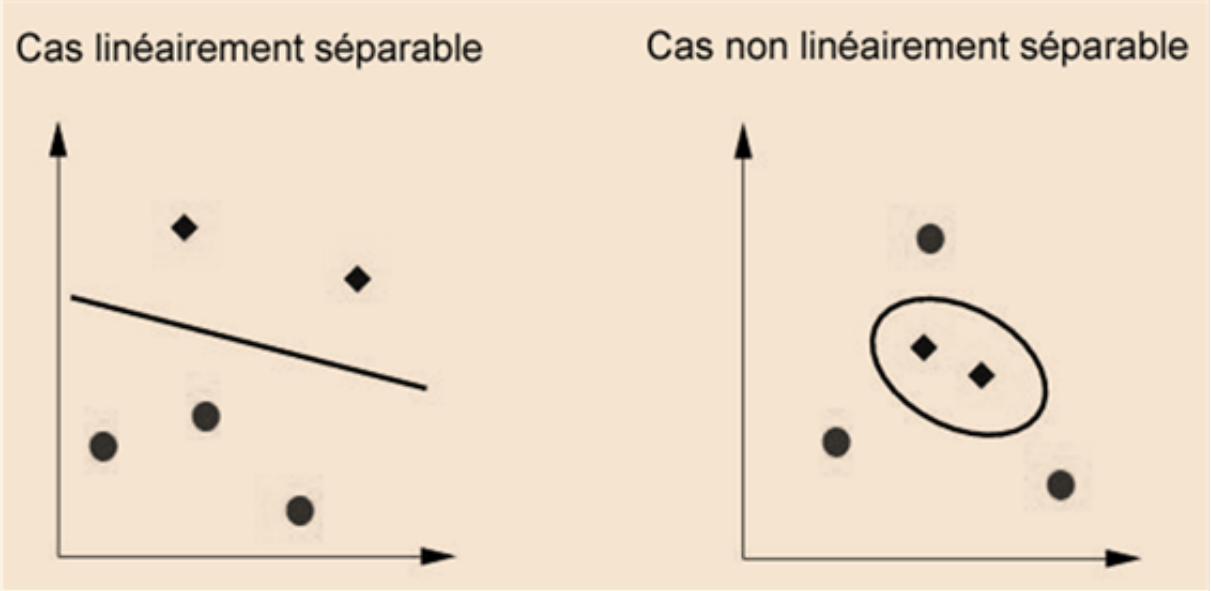

Figure 1.1 – les modèles linéairement séparable et non linéairement séparable.

1.1 Principes fondamentaux des SVM

SVM reposent sur deux principes fondamentaux :

— **La notion de marge maximale :** La marge représente la distance entre la frontière de séparation et les exemples les plus proches de l'ensemble d'apprentissage, appelés vecteurs supports. L'objectif des SVM est de choisir la frontière de séparation de manière à maximiser cette marge. Cette approche est justifiée par la théorie développée par Vapnik et Chervonenkis, qui met l'accent sur la généralisation et la capacité de prédiction du modèle.

— **Transformation d'espace pour les données non linéairement séparables :** Lorsque les données ne peuvent pas être séparées de manière linéaire, les SVM utilisent une technique de transformation de l'espace de représentation des données d'entrée vers un espace de dimension supérieure, parfois même vers un espace de dimension infinie. Cette transformation permet de rendre probable l'existence d'une séparation linéaire dans ce nouvel espace. Cette transformation est réalisée grâce à l'utilisation d'une fonction noyau (kernel) qui possède l'avantage de ne pas exiger la connaissance explicite de la transformation à appliquer pour le changement d'espace. Les noyaux, tels que les noyaux linéaires, polynomiaux ou gaussiens (RBF - Radial Basis Function), permettent d'effectuer cette transformation d'espace de manière efficace et de trouver des frontières de décision non linéaires dans des espaces de dimensions supérieures.

1.1.1 SVM linéaires

L'objectif de la classification linéaire est de chercher le meilleur hyperplan (un espace de dimension (m-1)) qui sépare les observations Xi appartenant à la classe c de ceux appartenant à la classe -c lorsque cela est possible.

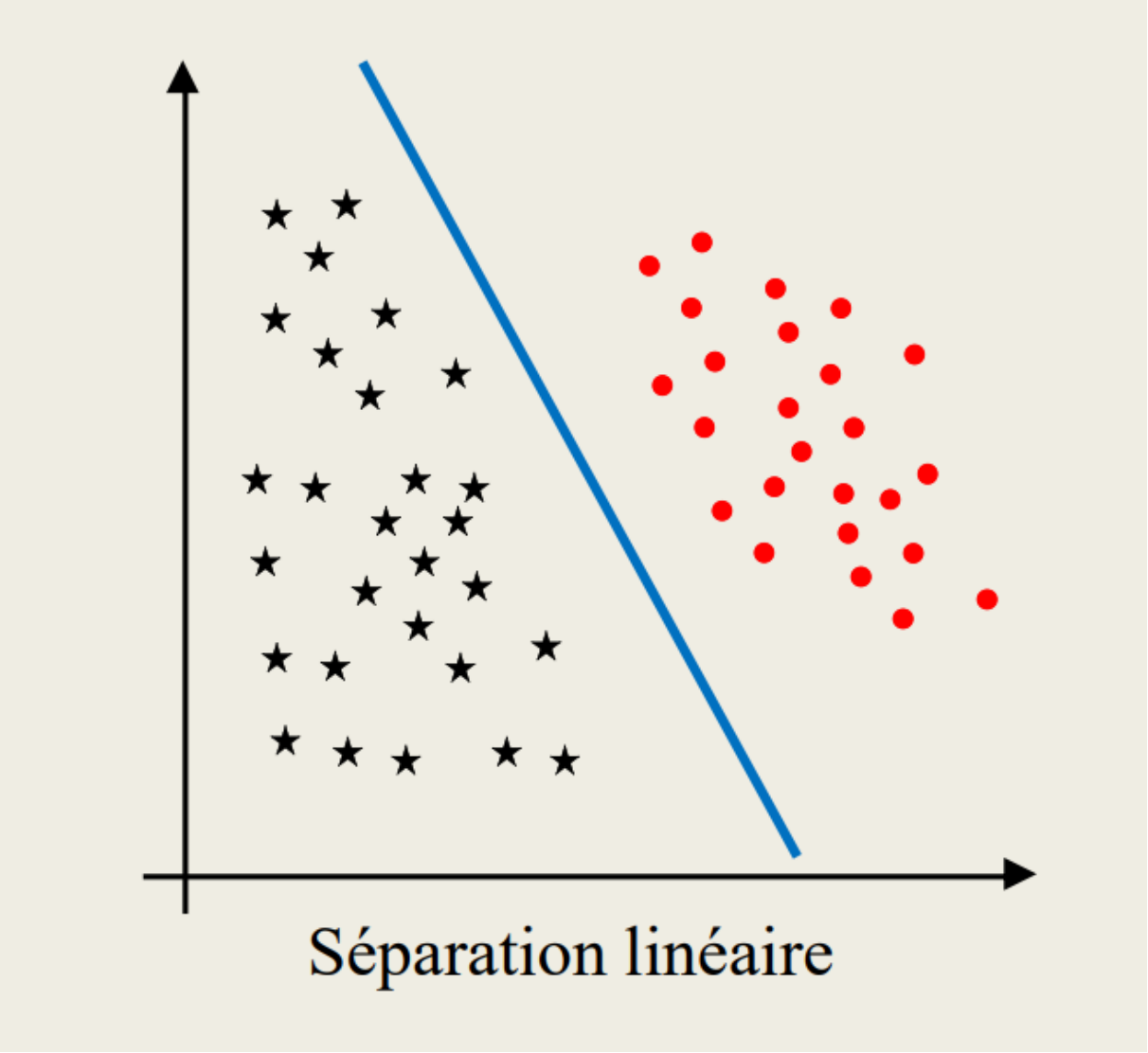

Figure 1.2 – Séparation linéaire

→**Hyperplan séparateur :** La marge d'un hyperplan est la distance entre l'hyperplan et la donnée (ou les données) la plus proche. Les vecteurs supports sont les données (observations) les plus proches. L'hyperplan optimal (en bleu) est celui qui est à la même distance des données les plus proches des deux classes, c.à.d. celui ayant la marge maximale.

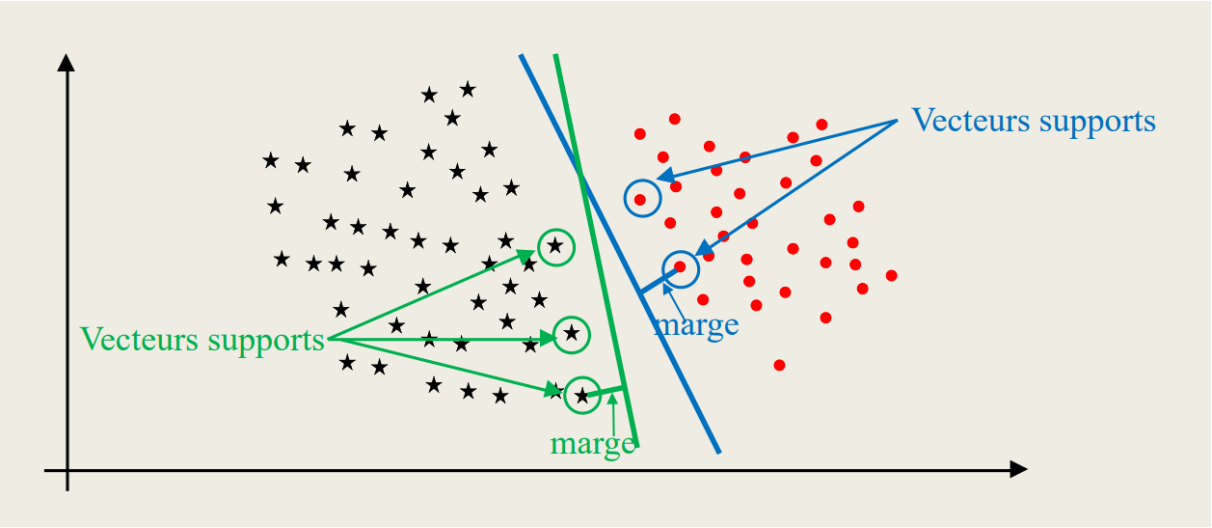

Figure 1.3 – Hyperplan séparateur

1.1.2 SVM non linéaires

La notion de marge maximale et la procédure de recherche de l'hyperplan séparateur telles que présentées pour l'instant ne permettent de résoudre que des problèmes de discrimination linéairement séparables. C'est une limitation sévère qui condamne à ne pouvoir résoudre que des problèmes jouets, ou très particuliers. Afin de remédier au problème de l'absence de séparateur linéaire, l'idée de l'astuce du noyau (en anglais kernel trick) est de reconsidérer le problème dans un espace de dimension supérieure, éventuellement de dimension infinie. Dans ce nouvel espace, il est alors probable qu'il existe une séparation linéaire. Changement de dimension de l'espace à fin de trouver le plan séparateur des deux exemples. Grâce à l'astuce du noyau.

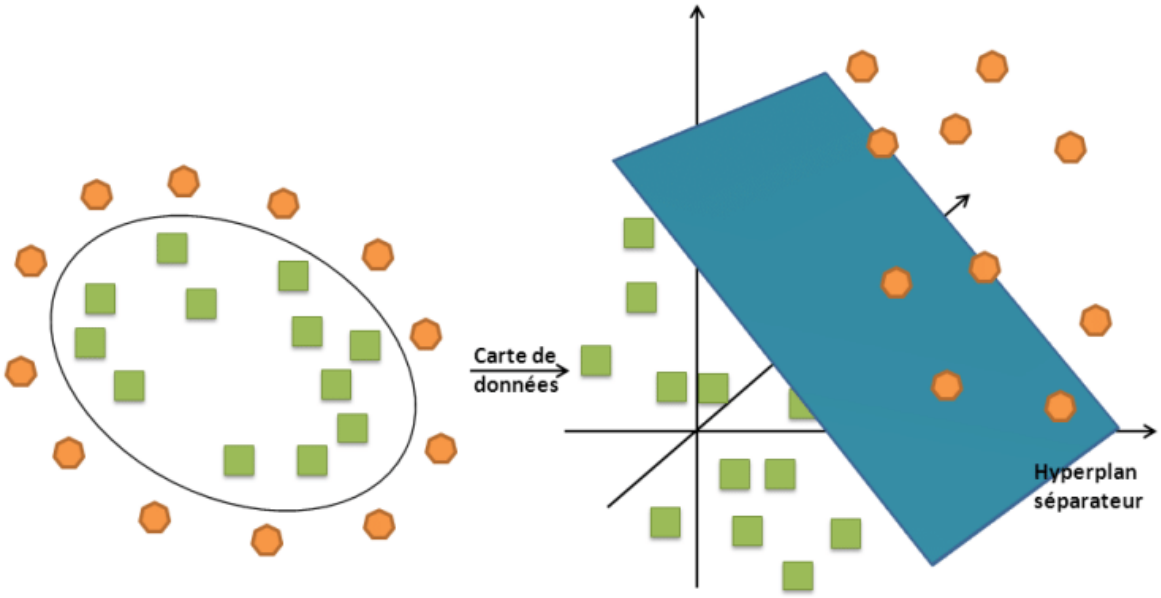

Figure 1.4 – SVM non linéaires

1.2 Approches pour les SVM en classification multi-classe

1.2.1 One-vs-Rest (OvR) ou One-vs-All (OvA)

Cette approche consiste à former un classifieur SVM pour chaque classe par rapport au reste des classes. Lors de la prédiction, les résultats de chaque classifieur sont évalués, et la classe avec la décision la plus confiante est choisie.

1.2.2 One-vs-One (OvO)

Cette méthode consiste à former un classifieur SVM pour chaque paire de classes possibles. Lors de la prédiction, chaque classifieur "vote" pour sa classe correspondante, et la classe avec le plus grand nombre de votes est sélectionnée.

1.2.3 Hierarchical SVM (HSVM)

Cette approche hiérarchique divise les classes en une structure d'arborescence. À chaque nœud de l'arbre, un SVM est entraîné pour discriminer les sous-classes associées.

1.3 Applications des SVM

1.3.1 Classification

- ✓ **Classification de documents :** Les SVM sont utilisés pour classer des documents dans différentes catégories telles que les articles de presse, les e-mails (spam vs non-spam), etc.
- ✓ **Détection de fraudes financières :** Les SVM peuvent être appliqués pour détecter les transactions frauduleuses dans les services financiers en utilisant des caractéristiques des transactions pour distinguer les transactions légitimes des frauduleuses.
- ✓ **Diagnostic médical :** Les SVM peuvent être utilisés pour classer les données médicales, par exemple pour la détection de maladies basée sur des données biomédicales comme l'imagerie médicale, les dossiers médicaux, etc.
- ✓ **Reconnaissance faciale :** Les SVM sont utilisés dans les systèmes de reconnaissance faciale pour identifier et classifier les visages parmi différents individus.

1.3.2 Régression

- ✓ **Prévision de prix de l'immobilier :** Les SVM peuvent être utilisés pour prédire les prix des biens immobiliers en utilisant des caractéristiques telles que la localisation, la taille, le type de propriété, etc.
- ✓ **Prévision de la consommation d'énergie :** Les SVM peuvent être utilisés pour prédire la consommation d'énergie en utilisant des données historiques telles que la météo, les habitudes de consommation, etc.
- ✓ **Évaluation de crédit :** Les SVM peuvent être utilisés pour évaluer le risque de crédit en prédisant la probabilité de défaut de paiement des emprunteurs en se basant sur diverses caractéristiques financières et personnelles.
- ✓ **Prévision de la demande :** Les SVM peuvent être utilisés pour prédire la demande de produits en utilisant des données historiques de vente, de saisonnalité, etc.

1.4 Avantages des SVM

- ✓ **Efficacité en haute dimension :** Les SVM excellent lorsque le nombre de caractéristiques est élevé par rapport au nombre d'échantillons. Ils sont efficaces dans des espaces de grande dimension comme la classification d'images.
- ✓ **Bonne généralisation :** Ils ont une bonne capacité à généraliser à partir d'un ensemble de données limité, réduisant le risque de surajustement (overfitting).
- ✓ **Capacité à gérer les marges non-linéaires :** Grâce à l'utilisation de noyaux (kernels), les SVM peuvent apprendre des frontières de décision complexes et non linéaires.
- ✓ **Gestion efficace des espaces de grande taille :** Ils ne dépendent que d'un sous-ensemble des échantillons d'entraînement, appelés vecteurs supports, ce qui rend leur entraînement efficace dans les ensembles de données volumineux.

1.5 Inconvénients des SVM

- ✓ **Sensibilité aux paramètres et au choix du noyau :** Les performances des SVM dépendent fortement du choix du noyau et de la régularisation, ce qui peut être difficile à déterminer.

- ✓ **Temps de calcul plus élevé :** Pour de grands ensembles de données, l'entraînement d'un SVM peut être coûteux en termes de temps de calcul, surtout lorsque la taille des données augmente.
- ✓ **Pas directement adaptés aux données très bruitées :** Les SVM peuvent être sensibles aux données bruitées ou à des étiquettes incorrectes, ce qui peut affecter leur performance.
- ✓ **Difficulté à interpréter les résultats :** La nature de la fonction de décision des SVM peut rendre difficile l'interprétation des raisons pour lesquelles une certaine décision est prise.

Conclusion

En conclusion, les SVM sont des modèles d'apprentissage automatique puissants et polyvalents, offrant plusieurs avantages tels que leur efficacité en haute dimension, leur capacité à généraliser à partir de petits ensembles de données et leur aptitude à gérer les marges non-linéaires grâce à l'utilisation de noyaux.

Cependant, ils présentent également des inconvénients, notamment leur sensibilité aux paramètres, le temps de calcul plus élevé pour de grands ensembles de données, ainsi que la difficulté à interpréter leurs résultats dans certains cas.

Le choix d'utiliser les SVM dépend du contexte spécifique du problème, des données disponibles et des objectifs de la modélisation. Comprendre les avantages et les inconvénients des SVM est crucial pour une utilisation efficace de cet algorithme dans diverses applications d'apprentissage automatique.

Chapitre 2

Conception

Introduction

Dans ce chapitre, nous décrirons la conception du projet visant à utiliser les SVM pour mesurer la lisibilité des textes arabes. La conception du projet comprend plusieurs étapes, notamment :

2.1 Collecte et Préparation des Données

Nous avons collecté les données textuelles arabes fournies par notre professeur. Ces données ont ensuite été prétraitées et nettoyées pour éliminer les éventuelles incohérences, les caractères spéciaux indésirables et les doublons.

2.2 Choix et Configuration des SVM

Nous avons choisi les SVM comme algorithme principal pour notre modèle. Nous avons également décidé des paramètres spécifiques tels que le type de noyau (linéaire, gaussien, polynomial) et les valeurs de régularisation après une série de tests et de validations croisées.

2.3 Extraction des Caractéristiques

Les caractéristiques de lisibilité ont été extraites des textes arabes prétraités, ce qui inclut le nombre de phrases, de mots, de caractères, ainsi que les lemmes, les stems et d'autres caractéristiques linguistiques.

2.4 Entraînement du Modèle

Nous avons divisé nos données en ensembles d'entraînement et de test, puis entraîné notre modèle SVM sur l'ensemble d'entraînement en utilisant les caractéristiques extraites.

2.5 Évaluation et Validation

Pour évaluer les performances du modèle, nous avons utilisé des mesures telle que l'exactitude.

2.6 Intégration et Déploiement

Enfin, nous avons intégré le modèle dans une application ou un système plus large pour évaluer la lisibilité des textes arabes. Cette étape peut inclure le déploiement du modèle dans un environnement en production.

2.7 Prétraitement des données

Pour calculer le nombre de phrases, mots et caractéres.

```python
def analyze_arabic_text(text):
    # Nombre de phrases dans le texte
    sentences = text.count('.') + text.count('؟') + text.count('!')

    # Nombre de mots dans le texte
    words = len(text.split())

    # Nombre de caractères dans le texte
    characters = len(text)

    return sentences, words, characters

def analyze_arabic_file(file_path):
    # Lecture du contenu du fichier
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            file_content = file.read() # lire tout le contenu du fichier et stocker le dans la variable file_content

        # Analyse du texte du fichier en utilisant la fonction analyze_arabic_text
        sentences_count, words_count, characters_count = analyze_arabic_text(file_content)

        # Affichage des résultats
        #print("Le nombre de phrases dans le fichier est : ", sentences_count)
        #print("Le nombre de mots dans le fichier est : ", words_count)
        #print("Le nombre de caractères dans le fichier est : ", characters_count)
        return sentences_count, words_count, characters_count
```

Figure 2.1 – le nombre de phrases, mots et caractéres

Pour extraire les lemmes et les stems.

```python
from jpype import startJVM, shutdownJVM, java
import os
import sys
from pathlib import Path
import jpype
from jpype import *

def extraire_colonne_secondaire(texte):
    mots = texte.split('{{')
    colonne_secondaire = [mot.split(':')[1][:-2] if '}}' in mot else '' for mot in mots]
    colonne_secondaire = [mot.strip('}').strip() for mot in colonne_secondaire if mot]
    return colonne_secondaire


def lsp(chemin_fichier):
    #print(jpype.getDefaultJVMPath())
    path1='c:\\tmp'

    if not jpype.isJVMStarted():
        jpype.startJVM(jpype.getDefaultJVMPath() ,'-ea', classpath=[path1],convertStrings=True )

    path2= 'C:\\Users\\mimif\\OneDrive\\Bureau\\S3\maz\\Ressources lisibilité\\PosTaggerAlKhalil.jar'
    jpype.addClassPath(path2)

    path3= 'C:\\Users\\mimif\\OneDrive\\Bureau\\S3\maz\\Ressources lisibilité\\ADAT-Analyzer.jar'
```

Figure 2.2 – liste de lemmes et stems

```python
    #Import the Java class
    LemmaStemma = jpype.JClass("net.oujda_nlp_team.ADATAnalyzer")
    POStag = jpype.JClass("net.nlp.parser.TestParserClient")

    # Chemin vers le fichier
    #chemin_fichier = r"C:\Users\mimif\OneDrive\Bureau\S3\maz\Ressources lisibilité\niveaux\1\استجار شقة.txt"
    #chemin_fichier = r"C:\Users\mimif\OneDrive\Bureau\S3\maz\Ressources lisibilité\niveaux\test.txt"

    with open(chemin_fichier, encoding="utf8") as file:
        file_content = file.read()

    #Lemmatizer, type de retour String
    lemmes = LemmaStemma.getInstance().processADATLemmatizerString(file_content)

    #Stemmer, type de retour String
    stems = LemmaStemma.getInstance().processADATStemmerString(file_content)

    #$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
    lemmes = extraire_colonne_secondaire(lemmes)
    stems = extraire_colonne_secondaire(stems)

     # Retirer les marqueurs "{{" et "}}" de chaque élément avant de les fusionner
    lemmes = [lemme.replace('{{', '').replace('}}', '') for lemme in lemmes]
    stems = [stem.replace('{{', '').replace('}}', '') for stem in stems]
    # Concaténer les lemmes et les stems en une seule chaîne de caractères
    lemme_string = ' '.join(lemmes)
```

Figure 2.3 – liste de lemmes et stems

Pour extraire les lemmes ambigus (lemmes qui obtiennent plusieurs tags PoS distincts).

```
from jpype import startJVM, shutdownJVM, java
import os
import sys
from pathlib import Path
import jpype
from jpype import *

def write_lemmas_to_file(lemmas_list):
    # Chemin vers le fichier temporaire
    file_path = r"C:\Users\mimif\OneDrive\Bureau\S3\maz\Ressources lisibilité\tmp.txt"  # Adapter le chemin si nécessaire

    # Écriture des lemmes dans le fichier temporaire
    with open(file_path, 'w', encoding='utf-8') as file:
        #for lemma in lemmas_list:
            file.write(lemmas_list)#file.write(lemma + '\n')

    return file_path

def extraire_tags_POS(fichier_POS):
    # Lire le contenu du fichier de sortie POS
    with open(fichier_POS, 'r', encoding='utf-8') as file:
        contenu = file.read()

    # Extraire les tags PoS de chaque {{ }} dans le fichier de sortie POS
    tags_POS = []
    elements = contenu.split('{{')
```

Figure 2.4 – les lemmes ambigus

```
    for element in elements:
        if '|' in element and '+' in element:
            tag = element.split('+')[1].split('|')[0]
            tags_POS.append(tag)

    # Construire une chaîne avec des espaces entre chaque tag POS
    tags_string = ' '.join(tags_POS)

    return tags_string

def POS(chemin_fichier):

    #chemin_fichier = write_lemmas_to_file(lems)
    #print(jpype.getDefaultJVMPath())
    path1='c:\\tmp'

    if not jpype.isJVMStarted():
        jpype.startJVM(jpype.getDefaultJVMPath() ,'-ea', classpath=[path1],convertStrings=True )

    path3= 'C:\\Users\\mimif\\OneDrive\\Bureau\\S3\maz\\Ressources lisibilité\\ADAT-Analyzer.jar'
    jpype.addClassPath(path3)

    #Import the Java class
    POSclass = jpype.JClass("net.oujda_nlp_team.ADATAnalyzer")

    with open(chemin_fichier, encoding="utf8") as file:
        file_content = file.read()
```

Figure 2.5 – les lemmes ambigus

Pour enlever les ponctuations de la langue arabe.

```python
def enlever_ponctuation_arabe(fichier):
    # Liste des symboles de ponctuation arabes
    ponctuation_arabe = '﴾﴿:!،؛؟«»ـ٠١٢٣٤٥٦٧٨٩٪٬٫؍ـّـُـِـٌـٍـْـٓـٔـٕـٖـٗـ٘ـٙـٚـٛـٜـٝـٞـٟـٰـٱـٲـٳـٴـٵـٶـٷـٸـٹـٺـٻـټـٽـپـٿـڀـځـڂـڃـڄـڅـچـڇـڈـډـڊـڋـڌـڍـڎـڏـڐـڑـڒـړـڔـڕـږـڗـژـڙـښـڛـڜـڝـڞـڟـڠـڡـڢـڣـڤـڥـڦـڧـڨـکـڪـګـڬـڭـڮـگـڰـڱـڲـڳـڴـڵـڶـڷـڸـڹـںـڻـڼـڽـھـڿـۀـہـۂـۃـۄـۅـۆـۇـۈـۉـۊـۋـیـۍـێـۏـېـۑـےـۓـەـۮـۯـ۰۱۲۳۴۵۶۷۸۹ۺـۻـۼـ۽ـ۾ـۿـ‘’‚‛“”„‟•…:'

    # Lecture du fichier
    with open(fichier, 'r', encoding='utf-8') as file:
        contenu = file.read()

    # Suppression de la ponctuation arabe
    contenu_sans_ponctuation = ''.join(caractere for caractere in contenu if caractere not in ponctuation_arabe)

    # Écriture du contenu sans ponctuation dans le même fichier
    with open(fichier, 'w', encoding='utf-8') as file:
        file.write(contenu_sans_ponctuation)
```

Figure 2.6 – enlever les ponctuations de la langue arabe

Pour enlever les ponctuations du language Latin.

```python
import string

def enlever_ponctuation(fichier):
    # Liste des symboles de ponctuation
    ponctuation = string.punctuation

    # Lecture du fichier
    with open(fichier, 'r', encoding='utf-8') as file:
        contenu = file.read()

    # Suppression de la ponctuation
    '''When using join() in Python, the string specified before the join() method acts as a separator between the elements being joined.
     indicates that all the characters that pass the condition (i.e.,
     those not in ponctuation) will be concatenated directly without any spaces or separators between them, forming a single string.
    '''
    contenu_sans_ponctuation = ''.join(caractere for caractere in contenu if caractere not in ponctuation) # '' emptycaract not space caract

    # Écriture du contenu sans ponctuation dans le même fichier
    with open(fichier, 'w', encoding='utf-8') as file:
        file.write(contenu_sans_ponctuation)
```

Figure 2.7 – enlever les ponctuations du language Latin

Pour compter le nombre de lemmes et stems.

```python
def count_element(lemmes, stems):

    # Calcul du nombre de lemmes générés
    nombre_lemmes = len(lemmes.split())

    # Calcul du nombre de stems
    nombre_stems = len(stems.split())

    # Création d'un ensemble (set) pour obtenir les lemmes distincts
    #Lorsque nous convertissons une liste (ou toute autre structure itérable) en un ensemble en utilisant set(),
        #Python élimine automatiquement les doublons, ne laissant que les éléments uniques.
    lemmes_distincts = set(lemmes.split())

    # Calcul du nombre de lemmes distincts
    nombre_lemmes_distincts = len(lemmes_distincts)

    # Calcul du nombre de lemmes fréquents (apparaissant plus d'une fois)
    lemmes_freq = {}
    for lemme in lemmes:
        if lemme in lemmes_freq:
            lemmes_freq[lemme] += 1
        else:
            lemmes_freq[lemme] = 1

    nombre_lemmes_freq = sum(1 for lemme, freq in lemmes_freq.items() if freq > 1)

    return nombre_lemmes, nombre_lemmes_distincts, nombre_lemmes_freq, nombre_stems
```

Figure 2.8 – le nombre de lemmes et stems

pour genérer le fichier du dataset avec tous les caracteristiques demandées.

```python
import os
import xlsxwriter

# Chemin du répertoire principal que vous souhaitez parcourir
chemin_repertoire = r"C:\Users\mimif\OneDrive\Bureau\S3\maz\Ressources lisibilité\niveaux"

# Chemin du fichier xlsx où vous souhaitez stocker les données
chemin_fichier = r"C:\Users\mimif\OneDrive\Bureau\S3\maz\Ressources lisibilité\res.xlsx"
workbook = xlsxwriter.Workbook(chemin_fichier)

# The workbook object is then used to add new
# worksheet via the add_worksheet() method.
worksheet = workbook.add_worksheet()

# Écriture des en-têtes
worksheet.write('A1', 'Nom du fichier')
worksheet.write('B1', 'Contenu')
worksheet.write('C1', 'sentences_count')
worksheet.write('D1', 'words_count')
worksheet.write('E1', 'characters_count')
worksheet.write('F1', 'lemmes')
worksheet.write('G1', 'lemmes_ambigus')
worksheet.write('H1', 'nombre_lemmes')
worksheet.write('I1', 'nombre_lemmes_distincts')
worksheet.write('J1', 'nombre_lemmes_ambigus')
```

Figure 2.9 – dataset

```
worksheet.write('O1', 'classe')
ligne = 1  # Pour commencer à écrire à partir de la deuxième ligne (après les en-têtes)
# Parcours récursif des fichiers et sous-répertoires
for repertoire_actuel, sous_repertoires, fichiers in os.walk(chemin_repertoire):
    # Pour chaque fichier trouvé dans le répertoire actuel
    for nom_fichier in fichiers:
        # Afficher le nom du sous-répertoire et le nom du fichier
        #os.path.basename(repertoire_actuel) pour avoir seulement le nom du sous rep sans le chemin absolu
        #print(f"Sous-répertoire : {os.path.basename(repertoire_actuel)}, Fichier : {nom_fichier}")
        classe = os.path.basename(repertoire_actuel)
        path = repertoire_actuel + "\\" + nom_fichier

        # Calcul nbr de phrases, mots et caracts
        sentences_count, words_count, characters_count = analyze_arabic_file(path)
        # Enlever la ponctuation
        enlever_ponctuation_arabe(path)
        enlever_ponctuation(path)
        # Extraction de lemmes et de stems
        lemmes, stems = lsp(path)
        # calcul nombre de lemme, lemmeDinstinct, lemmeFreq, et nombre de stem
        nombre_lemmes, nombre_lemmes_distincts, nombre_lemmes_freq, nombre_stems = count_element(lemmes, stems)
        #les Lemmes ambigus
        lem_amb = []
        #extractions des tags PoS
        for item in lemmes:
            tmp_path = write_lemmas_to_file(item)
            pos = POS(tmp_path)
```

Figure 2.10 – dataset

```
        lem_amb = ' '.join(lem_amb)
        #Le nombre de lemmes ambigus
        nombre_lemmes_ambigus = len(lem_amb.split())

        #print(path)
        with open(path, encoding="utf8") as file:
            contenu = file.read()
        # Écrire les données dans le fichier Excel
        worksheet.write(ligne, 0, nom_fichier)  # Colonne A
        worksheet.write(ligne, 1, contenu)
        worksheet.write(ligne, 2, sentences_count)
        worksheet.write(ligne, 3, words_count)
        worksheet.write(ligne, 4, characters_count)
        worksheet.write(ligne, 5, lemmes)
        worksheet.write(ligne, 6, lem_amb)
        worksheet.write(ligne, 7, nombre_lemmes)
        worksheet.write(ligne, 8, nombre_lemmes_ambigus)
        worksheet.write(ligne, 9, nombre_lemmes_distincts)
        worksheet.write(ligne, 10, nombre_lemmes_freq)
        worksheet.write(ligne, 11, stems)
        worksheet.write(ligne, 12, nombre_stems)
        worksheet.write(ligne, 13, pos)
        worksheet.write(ligne, 14, classe)  # Colonne L
        ligne += 1  # Passage à la ligne suivante

# Fermeture du classeur
workbook.close()
```

Figure 2.11 – dataset

2.8 Extrait du dataset

```
        #Le nombre de lemmes ambigus
        nombre_lemmes_ambigus = len(lem_amb.split())

        #print(path)
        with open(path, encoding="utf8") as file:
            contenu = file.read()
        # Écrire les données dans le fichier Excel
        worksheet.write(ligne, 0, nom_fichier)  # Colonne A
        worksheet.write(ligne, 1, contenu)
        worksheet.write(ligne, 2, sentences_count)
        worksheet.write(ligne, 3, words_count)
        worksheet.write(ligne, 4, characters_count)
        worksheet.write(ligne, 5, lemmes)
        worksheet.write(ligne, 6, lem_amb)
        worksheet.write(ligne, 7, nombre_lemmes)
        worksheet.write(ligne, 8, nombre_lemmes_ambigus)
        worksheet.write(ligne, 9, nombre_lemmes_distincts)
        worksheet.write(ligne, 10, nombre_lemmes_freq)
        worksheet.write(ligne, 11, stems)
        worksheet.write(ligne, 12, nombre_stems)
        worksheet.write(ligne, 13, pos)
        worksheet.write(ligne, 14, classe)  # Colonne L
        ligne += 1  # Passage à la ligne suivante

# Fermeture du classeur
workbook.close()
```

Figure 2.12 – dataset

Le dataset se compose de 272 instances, les principales caractéristiques (Features) sont :

SL1 = Le nombre de phrases dans un texte

WL1 = Le nombre de mots dans un texte

WL4 = Le nombre de stems dans un texte

WL5 = Le nombre de caractères dans un texte

AMb1 = Le nombre de lemmes ambigus (lemmes qui obtiennent plusieurs tags PoS distincts)

VL1 = Le nombre de lemmes distincts dans un texte

VL2 = Le nombre de lemmes fréquentes (lemmes apparaissant plus d'une fois) dans un texte

2.9 TF-IDF

Le TF-IDF (Term Frequency-Inverse Document Frequency) est une technique utilisée en traitement automatique du langage naturel (TALN) pour évaluer l'importance relative d'un terme (mot) dans un document par rapport à une collection de documents.

Voici une explication détaillée de chaque terme :

- **TF (Term Frequency)** mesure la fréquence d'un terme dans un document spécifique. Elle est calculée en comptant le nombre de fois où un terme apparaît dans un document et en le divisant par le nombre total de termes dans ce document. La formule de TF est donnée par :

$$TF = \frac{Nombre de fois o le terme apparat dans un document}{Nombre total de termes dans le document}$$

La logique derrière TF est que les termes importants apparaissent souvent dans un document.

- **IDF (Inverse Document Frequency)** est une mesure de l'importance d'un terme dans l'ensemble du corpus (ensemble de documents). IDF est calculé en prenant le logarithme inverse du nombre de documents dans le corpus divisé par le nombre de documents contenant le terme spécifique. La formule de IDF est donnée par :

$$IDF = \log_e \left(\frac{Nombretotaldedocuments}{Nombrededocumentscontenantleterme} \right)$$

Les termes qui apparaissent dans un petit nombre de documents auront une valeur IDF plus élevée, soulignant leur importance relative.

La formule du TF-IDF est obtenue en multipliant TF par IDF :

$$TF - IDF = TF \times IDF$$

2.10 Partie SVM

Le code suivant implémente un modèle SVM pour classifier des données textuelles en utilisant des caractéristiques TF-IDF et d'autres aspects spécifiques du texte .

Il commence par extraire les caractéristiques textuelles TF-IDF à l'aide du TfidfVectorizer. En parallèle, il collecte d'autres attributs comme le nombre de mots, de caractères, de lemmes, etc., pour enrichir les données textuelles.

Ensuite, le code utilise GridSearchCV pour optimiser le modèle SVM. Cette technique explore différentes combinaisons d'hyperparamètres afin de sélectionner les meilleurs paramètres pour améliorer les performances du modèle SVM dans la classification des données textuelles.

```
xtrain_tfidf = tfidf_vect.transform(train_x)
xvalid_tfidf = tfidf_vect.transform(valid_x)'''
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from scipy.sparse import hstack# Importez vos fonctions personnalisées

# Lecture des données à partir du fichier CSV
dataFrame = pd.read_csv('res.csv')

# Prétraitement du contenu
dataFrame['Contenu'] = dataFrame['Contenu'].apply(lambda Text: remStopWords(str(Text)))
dataFrame['Contenu'] = dataFrame['Contenu'].apply(lambda Text: clean(Text))
dataFrame['Contenu'] = dataFrame['Contenu'].apply(lambda Text: lemWords(Text))

# Suppression des doublons
dataFrame.drop_duplicates(subset="Contenu", keep='first', inplace=True)

# Division en ensembles d'entraînement et de validation
train_data, valid_data = train_test_split(dataFrame, test_size=0.2, random_state=50)

# Encodage des étiquettes de classe
encoder = LabelEncoder()
train_data['classe'] = encoder.fit_transform(train_data['classe'])
valid_data['classe'] = encoder.transform(valid_data['classe'])

# Extraction des caractéristiques TF-IDF
tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', max_features=50000)
tfidf_vect.fit(dataFrame['Contenu'])

xtrain_tfidf = tfidf_vect.transform(train_data['Contenu'])
xvalid_tfidf = tfidf_vect.transform(valid_data['Contenu'])

# Concaténation des nouvelles caractéristiques avec les caractéristiques TF-IDF
nouvelles_caracteristiques_train = train_data[['sentences_count', 'words_count', 'characters_count', 'nombre_lemm
nouvelles_caracteristiques_valid = valid_data[['sentences_count', 'words_count', 'characters_count', 'nombre_lemm

#xtrain_combined = hstack((xtrain_tfidf, nouvelles_caracteristiques_train))
#xvalid_combined = hstack((xvalid_tfidf, nouvelles_caracteristiques_valid))
```

Model Training

In [6]:
```
def train_model(classifier, feature_vector_train, label,feature_vector_valid, is_neural_net=False):
    # fit the training dataset on the classifier
    classifier.fit(feature_vector_train, label)
    # predict the labels on validation dataset
    predictions = classifier.predict(feature_vector_valid)
    return metrics.accuracy_score(predictions, valid_y),predictions
```

In [7]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
train_y = train_data['classe']
valid_y = valid_data['classe']

# Paramètres à rechercher
params_grid = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4, 1e-5, 'auto'],
                'C': [1, 10, 100, 1000, 10000]},
               {'kernel': ['linear'], 'C': [1, 10, 100, 1000, 10000],
                'gamma': [1e-3, 1e-4, 1e-5, 'auto']}
              ]

clf_svm = GridSearchCV(SVC(), params_grid, cv=5)
```

In [*]:
```
accuracy, predictions = train_model(clf_svm,nouvelles_caracteristiques_train, train_y, nouvelles_caracteristiques

print("accuracy: ",accuracy*100)
```

Figure 2.13 – Modèle SVM pour classifier des données textuelles en utilisant des aspects spécifiques du texte

```
78
```

In [37]:

```
#read data
dataFrame = pd.read_csv('res.csv')

dataFrame['Contenu']=dataFrame['Contenu'].apply(lambda Text: remStopWords(str(Text)))
dataFrame['Contenu'] = dataFrame['Contenu'].apply(lambda Text : clean(Text))
dataFrame['Contenu']=dataFrame['Contenu'].apply(lambda Text: lemWords(Text))

dataFrame.drop_duplicates(subset ="Contenu",keep = 'first', inplace = True)

train_x, valid_x, train_y, valid_y = train_test_split(dataFrame['Contenu'], dataFrame['classe'],test_size=0.2,ran

before_encode_valid_y = dataFrame['classe'].unique()

encoder = preprocessing.LabelEncoder()

train_y = encoder.fit_transform(train_y)
valid_y = encoder.fit_transform(valid_y)

tfidf_vect = TfidfVectorizer(analyzer='word',token_pattern=r'\w{1,}', max_features=50000)

tfidf_vect.fit(dataFrame['Contenu'])

xtrain_tfidf = tfidf_vect.transform(train_x)
xvalid_tfidf = tfidf_vect.transform(valid_x)
```

Model Training

In [38]:

```
def train_model(classifier, feature_vector_train, label,feature_vector_valid, is_neural_net=False):
    # fit the training dataset on the classifier
    classifier.fit(feature_vector_train, label)
    # predict the labels on validation dataset
    predictions = classifier.predict(feature_vector_valid)
    return metrics.accuracy_score(predictions, valid_y),predictions
```

In [39]:

```
params_grid = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4,1e-5,'auto'],
                    'C': [1, 10, 100, 1000,10000]},
                   {'kernel': ['linear'], 'C': [1, 10, 100, 1000,10000],'gamma': [1e-3, 1e-4,1e-5,'auto']},
                              {'kernel': ['poly'], 'degree':[1,2,3,4,5],'C': [1, 10, 100, 1000,10000],'gamma': [
              ]


clf_svm = GridSearchCV(SVC(), params_grid, cv=5)
```

In [40]:

```
accuracy, predictions = train_model(clf_svm,xtrain_tfidf, train_y, xvalid_tfidf)

print("accuracy: ",accuracy*100)
```

```
accuracy:  50.0
```

Figure 2.14 – Modèle SVM pour classifier des données textuelles en utilisant des caractéristiques TF-IDF

```
xvalid_tfidf = tfidf_vect.transform(valid_x)'''
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from scipy.sparse import hstack# Importez vos fonctions personnalisées

# Lecture des données à partir du fichier CSV
dataFrame = pd.read_csv('res.csv')

# Prétraitement du contenu
dataFrame['Contenu'] = dataFrame['Contenu'].apply(lambda Text: remStopWords(str(Text)))
dataFrame['Contenu'] = dataFrame['Contenu'].apply(lambda Text: clean(Text))
dataFrame['Contenu'] = dataFrame['Contenu'].apply(lambda Text: lemWords(Text))

# Suppression des doublons
dataFrame.drop_duplicates(subset="Contenu", keep='first', inplace=True)

# Division en ensembles d'entraînement et de validation
train_data, valid_data = train_test_split(dataFrame, test_size=0.2, random_state=50)

# Encodage des étiquettes de classe
encoder = LabelEncoder()
train_data['classe'] = encoder.fit_transform(train_data['classe'])
valid_data['classe'] = encoder.transform(valid_data['classe'])

# Extraction des caractéristiques TF-IDF
tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', max_features=50000)
tfidf_vect.fit(dataFrame['Contenu'])

xtrain_tfidf = tfidf_vect.transform(train_data['Contenu'])
xvalid_tfidf = tfidf_vect.transform(valid_data['Contenu'])

# Concaténation des nouvelles caractéristiques avec les caractéristiques TF-IDF
nouvelles_caracteristiques_train = train_data[['sentences_count', 'words_count', 'characters_count', 'nombre_lemm
nouvelles_caracteristiques_valid = valid_data[['sentences_count', 'words_count', 'characters_count', 'nombre_lemm

xtrain_combined = hstack((xtrain_tfidf, nouvelles_caracteristiques_train))
xvalid_combined = hstack((xvalid_tfidf, nouvelles_caracteristiques_valid))
```

Model Training

In [6]:
```
def train_model(classifier, feature_vector_train, label,feature_vector_valid, is_neural_net=False):
    # fit the training dataset on the classifier
    classifier.fit(feature_vector_train, label)
    # predict the labels on validation dataset
    predictions = classifier.predict(feature_vector_valid)
    return metrics.accuracy_score(predictions, valid_y),predictions
```

In [7]:
```
from sklearn.svm import SVC
from sklearn.multiclass import OneVsOneClassifier, OneVsRestClassifier
from sklearn.model_selection import GridSearchCV
train_y = train_data['classe']
valid_y = valid_data['classe']

params_grid = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4, 1e-5, 'auto'],
                'C': [1, 10, 100, 1000, 10000]},
               {'kernel': ['linear'], 'C': [1, 10, 100, 1000, 10000],
                'gamma': [1e-3, 1e-4, 1e-5, 'auto']}
              ]

clf_svm = GridSearchCV(SVC(), params_grid, cv=5)
```

In [*]:
```
accuracy, predictions = train_model(clf_svm,xtrain_combined, train_y, xvalid_combined)

print("accuracy: ",accuracy*100)
```

Model To Pickle

Figure 2.15 – Modèle SVM pour classifier des données textuelles en utilisant des caractéristiques TF-IDF et d'autres aspects spécifiques du texte

```
accuracy:  50.0

Rapport de classification :
Tableau de classification pour le noyau :
              precision    recall  f1-score   support

           0       0.67      0.44      0.53         9
           1       0.20      0.25      0.22         4
           2       0.67      0.60      0.63        20
           3       0.38      0.50      0.43        12
           4       0.44      0.44      0.44         9

    accuracy                           0.50        54
```

Figure 2.16 – En se basant uniquement sur les lemmes et en appliquant la méthode TF-IDF.

Nous avons obtenu une précision de 0,5.

Tableau de classification pour le noyau 'linear':

	precision	recall	f1-score	support
1	1.00	0.75	0.86	8
1+	0.67	1.00	0.80	4
2	0.68	0.90	0.78	21
2+	0.20	0.08	0.11	13
3	0.50	0.56	0.53	9
accuracy			0.64	55

Tableau de classification pour le noyau 'rbf':

	precision	recall	f1-score	support
1	0.40	0.25	0.31	8
1+	1.00	0.00	0.00	4
2	0.45	0.95	0.62	21
2+	1.00	0.00	0.00	13
3	0.33	0.22	0.27	9
accuracy			0.44	55

Tableau de classification pour le noyau 'poly':

	precision	recall	f1-score	support
1	0.45	0.62	0.53	8
1+	1.00	0.00	0.00	4
2	0.45	0.90	0.60	21
2+	1.00	0.00	0.00	13
3	0.50	0.11	0.18	9
accuracy			0.45	55

Tableau de classification pour le noyau 'sigmoid':

	precision	recall	f1-score	support
1	0.00	0.00	1.00	8
1+	1.00	0.00	0.00	4
2	0.55	0.76	0.64	21
2+	1.00	0.00	0.00	13
3	0.00	0.00	1.00	9
accuracy			0.29	55

Figure 2.17 – En se basant uniquement sur les caractéristiques du texte

En utilisant uniquement les caractéristiques suivantes du texte : SL1 (le nombre de phrases), WL1 (le nombre de mots), WL4 (le nombre de stems), WL5 (le nombre de caractères), AMb1 (le nombre de lemmes ambigus), VL1 (le nombre de lemmes distincts) et VL2 (le nombre de lemmes fréquentes), Nous avons obtenu une précision de 0,64.

Tableau de classification pour le noyau 'linear':

	precision	recall	f1-score	support
0	1.00	0.62	0.77	8
1	0.50	1.00	0.67	4
2	0.84	0.80	0.82	20
3	0.67	0.62	0.64	13
4	0.50	0.56	0.53	9
accuracy			0.70	54

Tableau de classification pour le noyau 'rbf':

	precision	recall	f1-score	support
0	0.50	0.50	0.50	8
1	1.00	0.00	0.00	4
2	0.45	0.95	0.61	20
3	1.00	0.00	0.00	13
4	0.75	0.33	0.46	9
accuracy			0.48	54

Tableau de classification pour le noyau 'poly':

	precision	recall	f1-score	support
0	0.50	0.88	0.64	8
1	1.00	0.00	0.00	4
2	0.46	0.85	0.60	20
3	1.00	0.00	0.00	13
4	0.67	0.22	0.33	9
accuracy			0.48	54

Tableau de classification pour le noyau 'sigmoid':

	precision	recall	f1-score	support
0	0.00	0.00	1.00	8
1	1.00	0.00	0.00	4
2	0.40	0.95	0.56	20
3	1.00	0.00	0.00	13
4	1.00	0.00	0.00	9
accuracy			0.35	54

Figure 2.18 – Optimisation de la Prédiction de la Lisibilité des Textes Arabes : Combinaison Caractéristiques Textuelles et TF-IDF

En combinant méthode et caractéristiques précédentes, une précision de 0.7 est obtenue.

2.11 Démonstration

2.11.1 Interface de saisie

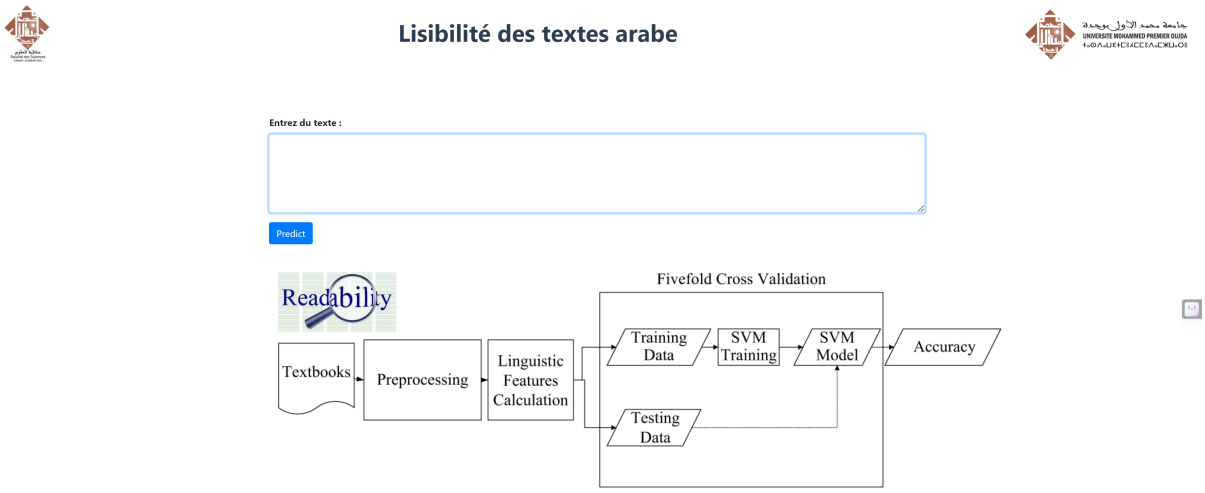

Figure 2.19 – Entrer un texte

Cette interface permet à l'utilisateur de saisir un texte afin de prédire son niveau de lisibilité.

Figure 2.20 – Prédiction de résultat

Cette interface offre à l'utilisateur un accès au niveau de lisibilité ainsi qu'aux informations textuelles suivantes :

Caractéristiques :

— SL1 : Le nombre de phrases dans un texte

— WL1 : Le nombre de mots dans un texte

— WL4 : Le nombre de stems dans un texte

— WL5 : Le nombre de caractères dans un texte

— AMb1 : Le nombre de lemmes ambigus (lemmes qui obtiennent plusieurs tags PoS distincts)

— VL1 : Le nombre de lemmes distincts dans un texte

— VL2 : Le nombre de lemmes fréquentes (lemmes apparaissant plus d'une fois) dans un texte

Conclusion

Ce chapitre détaille la conception de notre projet visant à utiliser les SVM pour évaluer la lisibilité des textes arabes en suivant ces différentes étapes.

Conclusion Générale

La réalisation de ce projet visant à évaluer la lisibilité des textes arabes à l'aide des SVM a été une expérience enrichissante et instructive.

Au cours de ce projet, nous avons exploré les différentes phases de conception, de collecte et de préparation des données jusqu'à l'évaluation et à la validation du modèle SVM. Nous avons appris l'importance de choisir les bonnes caractéristiques, de paramétrer les SVM de manière appropriée, ainsi que d'évaluer correctement la performance du modèle.

Les SVM se sont avérés être des outils puissants pour la classification et l'évaluation de la lisibilité des textes arabes. Leur capacité à gérer les données de grande dimension et à apprendre des frontières de décision complexes en fait des candidats pertinents pour une variété d'applications dans le domaine de l'analyse de texte.

Cependant, nous avons également rencontré des défis, notamment la sensibilité aux paramètres, le temps de calcul élevé pour de grands ensembles de données et la nécessité d'une préparation minutieuse des caractéristiques pour obtenir des performances optimales.

En conclusion, ce projet nous a permis d'acquérir une compréhension approfondie du language PYTHON, des SVM et de leur application dans l'analyse de la lisibilité des textes arabes. Il reste encore des opportunités d'amélioration et de recherche future pour perfectionner l'efficacité de ces modèles dans des contextes plus larges.