Université Paris Nanterre
Université Sorbonne nouvelle
Institut national des langues et civilisations orientales
pluriTAL
Master Traitement Automatique des Langues
http://plurital.org

Année académique 22/23

Classification des micro-blogs hashtagués #SéismeTurque par l'intention des utilisateurs

Etudiant	Xinhao ZHANG Yingzi LIU
Enseignant responsable	Yoann Dupont
Cours Suivi	LZST001-P-Fouille de Textes Université Sorbonne Nouvelle
Date	8 Mai 2023

twitter api v2

i. Table des matières

Table des matières

1. Introduction

En TAL, les textes en langue naturelle sont considérés comme des données non-structurées ou, à tout le moins, leur structure n'est pas directement mise à disposition pour des applications informatiques. S'agissant donc de déduire des données structurées à partir de données dites non structurées, la fouille de textes est une tâche importante visant à extraire l'information contenue dans les textes afin de la rendre accessible à d'autres applications.

Notre projet s'intéresse plus particulièrement à la classification du texte, une branche fondamentale de la fouille de textes. La classification de texte consiste à ranger des documents dans des catégories pré-établies dans une taxinomie, ou découvertes préalablement avec un outil de clustering. Cependant, l'analyse des intentions en est une approche qui nous prête attention pour notre devoir. C'est pourquoi nous choisirions d'effectuer la classification en fonction des objectifs ou des motivations sous-jacents derrière les microblogs écrits par les utilisateurs. Cela nous permettrait donc d'analyser les besoins, les préférences ou les opinions des utilisateurs.

Pour le choix du texte, nous nous focaliserions sur les microblogs, les messages courts postés sur les réseaux sociaux, tels que tweets. À l'ère mobile, le nombre d'utilisateurs de microblogs augmente de manière considérable, et les microblogs jouent un rôle de plus en plus important dans la vie quotidienne. Alors la classification des intentions des microblogs est progressivement devenue un champ de recherche intéressant.

Parallèlement, les tremblements de terre sont l'une des principales catastrophes naturelles sur Terre et, il y a trois mois, le tremblement de terre qui a frappé la Turquie et la Syrie a gravement dévasté une zone de plus de 20 000km^2. "la pire catastrophe naturelle en un siècle en Europe", selon l'Organisation mondiale de la Santé, laisse deux pays traumatisés. Malheureusement, les tremblements de terre sont des catastrophes imprévisibles et, même aujourd'hui, les technologies en plein essor ne permettent pas de prédire efficacement quand et où ils se produiront. Avec nos devoirs, nous espérons que la classification des microblogs à propos des séimes turques permettrait d'apporter une aide efficace aux secours et à la reconstruction du pays pour la période post-séisme.

En employant le logiciel libre de data-mining Weka, nous testerons différents programmes d'apprentissage automatique. Nous nous sommes surtout intéressé aux miscroblogs tweets accompagné avec le hashtag *#sésimeturque*. Nous avons constitué un corpus composé de trois classes en fonction de l'intention des auteurs : ***Aide***(appel à l'aide), ***Deuil***(prière et souhait), ***News***(actualité). Pour chacune des différentes classes, nous avons respectivement récolté cent posts sur Twitter ainsi que sur Facebook. Nous avons donc obtenu un corpus complet composé de trois cent micro-blogs. Puis, nous avons procédé à des expérimentations dans l'objectif de définir la meilleure combinaison possible entre l'une des représentations des textes et l'un des classifieurs.

2. Méthodologie du travail

Notre travail se divise en deux parties principales : l'acquisition de données de microblogs relatives aux séismes turcs et le prétraitement de texte, ainsi que la classification de texte combinant plusieurs modèles d'algorithmes de classification à l'aide du logiciel Weka et du module Python sklearn.

Tout d'abord, nous avons collecté à la main les textes de microblogs relatifs aux séisme turque entre février et avril cette année. Afin de rendre le classificateur final plus largement applicable, nous avons également récupéré des informations concernées à partir de forums et de sites web, en plus des données de microblogages sur Twitter et Facebook. Ensuite, nous avons obtenu les données de texte nécessaires par annotation et classification manuelles, puis avons effectué une segmentation de texte et le prétraitement spécifique et avons possédé au finale les données d'expériences en convertissant les textes en format *arff* reconnaissable par Weka.

Enfin, nous avons utilisé plusieurs algorithmes de classification classiques pour classifier l'intention des microblogs par le biais de deux plateformes différentes Weka et Sklearn et avons comparé les résultats expérimentaux pour essayer de sélectionner l'algorithme de classification optimal. Le schéma du système est présenté dans la Figure 1.

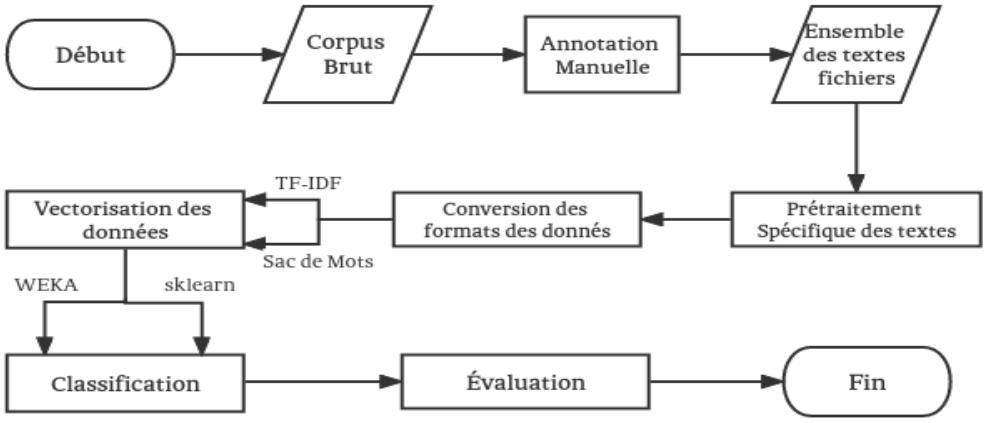

FIGURE 1 – schéma du processus du projet

3. Collecte et Prétraitement des données

3.1. Constitution du corpus brut

Ce sont les textes des microblogs contenant le hashtag *#sésimeturque* qui constituent notre corpus. Dans le cadre de la collecte de données, notre première approche a été de développer un programme pour extraire automatiquement les tweets liés au séisme en Turquie sur Twitter. Cependant, en testant plusieurs modules en Python, tels que tweepy, scweet, nous avons rencontré beaucoup de difficultés à nous procurer l'API officielle pour scraper les données. Cette méthode s'est révélée ainsi inefficace en raison des restrictions de confidentialité des données sur Twitter et des difficultés techniques rencontrées par notre programme. Face à cette problématique, nous avons opté pour une approche manuelle en collectant les données pertinentes sur Twitter et Facebook. Dans ce cas, nous avons heureusement découvert une fonctionnalité assez utile sur Twitter - ***Recherche Avancée***, qui nous permet de rechercher plus facilement des tweets filtrés en utilisant des critères de recherche spécifiques, tels que des mots-clés, des hashtags, des comptes d'utilisateurs, des dates et des emplacements géographiques.

Bien que cette nouvelle approche ait pris davantage de temps et d'efforts, elle nous a permis tout de même de rassembler une base de données complète pour notre projet. Nous avons été rigoureux dans notre sélection en nous assurant que les données que

nous avons collectées étaient représentatives et variées pour garantir que notre corpus soit exhaustif et reflète de manière précise la diversité des contenus relatifs à notre sujet d'étude.

Après avoir collecté et organisé les données pertinentes, nous avons d'abord classé les données en trois catégories selon leur contenu : ***Aide***(appel à l'aide), ***Deuil***(prière et souhait), ***News***(actualité). Des exemples de données sont présentés dans la Figure 2 :

Classe	Exemple	Quantité
Aide	AGISSONS ! #Smileyounited a mis en place une #cagnotte. Plus d'informations allez directement sur le site smile younited ou scanner le Qr code. Merci de partager cette information. #seisme#Aide #hmeinclusive	100
Deuil	Une pensée au peuple Turque victime d'un séisme de magnitude. Qu'Allah leur vienne en aide.	100
news	Le bilan du #séisme survenu le 6 février en Turquie et en Syrie dépasse désormais 50 000 morts après l'annonce vendredi par la #Turquie du décès de 44 218 personnes sur son territoire.	100

FIGURE 2 – exemples des données classifiées

3.2. Prétraitement des textes

Pour prétraiter des données, nous avons d'abord lancé le script vectorisation.py fourni par le professeur avec le fichier mots_vides.txt par défaut pour la première fois afin d'opérer un essai, et nous n'avons que pu avoir un résultat peu décisif dans un premier temps. Il a donc été reconnu que nous devrions personnaliser nos propres stopwords au fur et à mesure des expériences, de sorte que le fichier mots_vides.txt soit le plus adapté possible à notre corpus et ensuite permette de supprimer tous les mots jugés non pertinents pour la classficiation. C'est pourquoi nous n'avons pas pris en considération l'ensemble de stopwords en français fourni par le module nltk en Python. Entre temps, nous avons obtenue un fichier ARFF (Attribute-Relation File Format), le format de fichier utilisé pour représenter des données dans Weka. En observant les attributs du

fichiers, nous avons alors pu identifier plusieurs particularités stimulantes et intéressantes de notre corpus qui nous a semblé néanmoins nuisible à l'entraînement ainsi que le résultat. Ces problèmes présente en effet des caractéristiques distinctes, c'est pourquoi nous avons pris la décision de les régler de manière différentes. Voici nos remarques par rapport à notre corpus brut :

1. En raison de la diversité des utilisateurs, il existe différentes polices de texte dans notre corpus, qui apparaissent dans leur intégralité dans les résultats et sont segmentés comme attributs différents. Globalement, les polices *mathematical bold* et *mathematical bold en gras* sont fréquemment présentées.

2. Le corpus contient un certain nombre d'emojis. Par exemple, lorsqu'il s'agit de prières, de nombreux utilisateurs utilisent l'emoji des mains jointes, ou lorsqu'il s'agit de tweets relatifs au séisme en Turquie, l'emoji du drapeau turc apparaît. Ces emojis ne sont pas reconnus par le script, et certains emojis n'ont pas de séparation claire avec les mots environnants, ce qui rend difficile leur segmentation précise.

3. Dans la catégorie Aide, de nombreux utilisateurs partagent des liens vers des sites web de dons afin de collecter des fonds pour les opérations de secours, mais ces liens ne sont pas correctement identifiés et traîtés.

4. Des symboles tels que "€", "$" sont présents dans le corpus, notamment dans les tweets de la catégorie "Aide". Ces symboles sont généralement précédés d'un nombre sans espace et ne sont pas correctement identifiés.

5. Le corpus contient des signes de ponctuation spéciaux, tels que « • »,« / », « + », qui ne nous semblent pas nécessaires pour notre étude de classification.

Nous avons pris en compte les éléments problématiques mentionnés précédemment et avons effectué un prétraitement de notre corpus par le programme conçu pour améliorer la qualité du corpus et obtenir des résultats d'analyse plus précis. Voici les méthodes de traitement que nous avons appliquées dans notre script Prétraitement_du_corpus.py :

1. Pour résoudre le problème d'incohérence des polices de caractères, notre script permet de normaliser les caractères spéciaux en caractères normaux par l'intermédiaire de la fonction unicodedata.normalize.

2. Pour gérer les emojis, nous avons estimé qu'ils pouvaient être utiles dans le cadre de notre étude de classification, et nous les avons donc convertis en texte à l'aide de la fonction *emojize* du module emoji. Nous avons également ajouté des espaces avant et après le texte de chaque emoji pour faciliter leur segmentation.

3. En ce qui concerne les liens, nous avons les identifié à l'aide d'une expressions régulière,

```
texte = re.sub(r'(https?://\S+)|(www.\S+)', 'URL:', texte)
```

et les substitué par le préfixe "URL :", car ils peuvent tout à fait servir des critères de classification pour notre corpus. Ensuite, nous avons également ajouté un espace avant et après chaque lien pour délimiter correctement le lien dans le texte.

4. Comme le symbole "€", "$" est pertinent pour notre corpus dans la catégorie "Aide" étant donné son lien avec l'argent, nous l'avons remplacé par le contenu textuel correspondant "euros" et "dollars" dans notre script.

5. Pour les ponctuations inutiles comme "•", nous les remplaçons directement par des espaces tout en ajoutant une ligne de code dans la fonction nettoyage du script vectorisation.py.

En tout cas, il est bien prouvé à la fin que ces pré-traitements ont considérablement amélioré la qualité et la pertinence de notre corpus.

4. Description des expériences

Nous avons donc procédé à des expérimentations pour guider la phase d'apprentissage des classifieurs. En effet, nous avons exploré deux outils puissants dans le cadre du machine learning, le logiciel de fouille de textes - Weka ainsi que la bibliothèque open-source en Python - Sklearn(ou scikit-learn), où une grande variété d'algorithmes de classification sont disponibles. En vue de trouver le classifieur le plus performant possible pour notre ensemble de données, nous avons donc décidé d'entraîner et tester respectivement les modèles de classification avec les deux dispositifs susmentionnés. Nous avons ainsi

comparé les performances de plusieurs algorithmes de classification disponibles depuis ces outils et avons constaté que certains algorithmes étaient plus efficaces que d'autres en fonction de la nature des données et du problème de classification spécifique. Afin d'éviter le surapprentissage des modèles, nous avons donc eu recours à la validation croisée, et nous avons pu évaluer les performances de chaque algorithme de classification et les comparer les uns aux autres.

Pour les critères d'évaluation, nous avons mesuré la précision, le rappel et la F-mesure(moyenne harmonique). La précision fait référence à la proportion de textes classés comme vrais parmi le nombre total de textes classés comme vrais. Le rappel fait référence à la proportion de textes classés comme vrais parmi le nombre total de textes réellement vrais. La F-mesure est considérée comme une mesure globale de la précision P et du rappel R. Plus F-mesure est élevée, plus la précision de la classification expérimentale est élevée. De plus, Avec la tentative en Sklearn, nous avons spécifiquement importé le module cohen_kappa_score pour calculer le coefficient kappa pendant l'évaluation des classifieurs aussi. Le score kappa peut mesurer ici l'accord entre les prédictions du classificateur et les étiquettes de test réelles. Et il prend des valeurs entre -1 et 1, où une valeur de 1 indique un accord parfait entre le modèle et les résultats réels, une valeur de 0 indique un accord aléatoire et une valeur négative indique un accord pire que celui qui serait obtenu par hasard.

De plus, ce qui nous semble véritablement intéressant, c'est de vouloir savoir quel impact les approches différentes de vectorisation des mots vont avoir sur la performance des classifieurs. En effet, il existe deux approches fameuses permettant l'extraction des features du texte Sac de Mots ou bien TF-IDF, qui est considéré comme une étape indispensable quant au prétraitement des données textes dans machine learning. Vu que le professeur a utilisé bag of words dans le script de référence vectorisation.py, nous avons donc simplement essayé de vectoriser les mots en Python en employant l'approche TF-IDF lors de l'expérimentation avec Sklearn.

4.1. Expérimentations sur WEKA

4.1.1. ComplementNaiveBayes

Le classificateur de Naïf Bayes est un algorithme de classification basé sur le simple théorème de Bayes, dont l'équation est comme ceci :

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} = \frac{P(c)IIP(t|c)}{P(d)}, d = t_1, t_2, ..., t_n$$

La probabilité d'une catégorie donnée dans un ensemble de données est appelée "probabilité postérieure" et peut être calculée en prenant en compte la probabilité initiale de cette catégorie dans l'ensemble de données et la probabilité conditionnelle de l'apparition d'un document appartenant à cette catégorie. On suppose que les différentes caractéristiques d'un document sont indépendantes les unes des autres. En général, P(t|c) représente la fréquence des documents de la catégorie c dans lesquels le mot t apparaît, mais pour prendre en compte la fréquence d'apparition du mot dans le document, le modèle polynomial utilise P(t|c) comme la proportion de la fréquence des mots dans lesquels le mot t apparaît dans la catégorie c. Cette équation peut être résumée de la manière suivante :

$$C_d = max_i P(c_i|d) \infty max_i P(c_i) \Pi P(t|c)$$

L'outil Weka propose une variété d'algorithmes basés sur le modèle de Bayes pour la classification et la prédiction. Parmi ces algorithmes, on trouve BayesNet, NaiveBayesMultinomial, NaiveBayesMultinomialUpdateable et d'autres encore.

Après avoir exécuté ces algorithmes sur notre ensemble de données, nous avons comparé les résultats de classification pour chaque algorithme. Nous avons constaté que l'algorithme NaiveBayesMultinomial a donné les meilleurs résultats en termes de précision. NaiveBayesMultinomial est un algorithme qui suppose que toutes les variables sont indépendantes les unes des autres et suit une distribution multinomiale. Les tableaux et figures suivantes (TABLE 1, Figure 3, Figure 4) présentent les résultats de classification obtenus avec l'algorithme NaiveBayesMultinomial :

Précision	0.872
Rappel	0.870
F-mesure	0.869
Temps de calcul	0.01s
Kappa Statistic	0.805

TABLE 1 – Résultat du classifieur NaiveBayesMultinominal

MATRICE DE CONFUSION			
a	b	c	< – classified as
90	3	7	a = aide
4	94	2	b = deuil
8	15	77	c = news

FIGURE 3 – Matrice de confusion du classifieur NaiveBayesMultinominal

Les résultats indiquent que l'algorithme NaiveBayesMultionmial a une précision élevée pour toutes les classes. Les taux de vrais positifs sont également élevés pour les classes "aide" et "deuil". Les taux de faux positifs sont relativement faibles pour toutes les classes, ce qui suggère que l'algorithme a réussi à classifier les données avec une grande exactitude. Enfin, la moyenne pondérée des scores de précision, de rappel et de F-mesure est également élevée, ce qui suggère que l'algorithme est bien adapté à la classification de l'ensemble de nos données relatives à la séisme en Turquie.

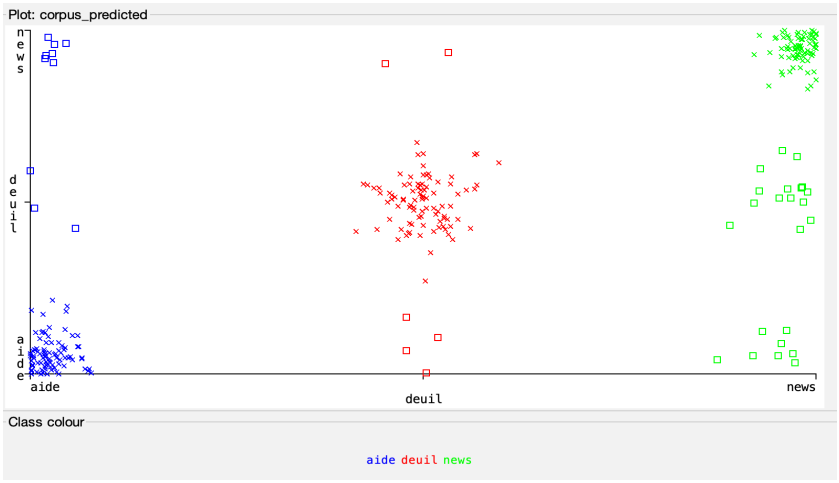

FIGURE 4 – Visualisation des erreurs du classifieur NaiveBayesMultinominal

4.1.2. SMO

Les SVM (Support Vector Machines) sont une famille d'algorithmes d'apprentissage automatique utilisés pour la classification et la régression. Ils sont des classificateurs de texte efficaces pour les petits ensembles de données. Ils se basent sur des concepts statistiques et utilisent un algorithme de mappage non linéaire pour transformer les données en données de haute dimension afin de mieux distinguer les différentes classes et maximiser l'intervalle de classification. SMO est un algorithme spécifique utilisé pour entraîner les SVM à marge douce (soft-margin SVMs) qui permettent une certaine erreur de classification dans les données d'apprentissage.

Nous avons mené des expériences en utilisant différentes valeurs pour le paramètre "Cross-validation" de l'algorithme SMO dans Weka. Nous avons constaté que la précision de classification était la plus élevée lorsque le nombre de "Folds" pour la "Cross-validation" était égal à 8. Les résultats de classification obtenus avec l'algorithme SMO sont présentés dans le Tableau 2 et la Figure 5 ci-dessous.

Précision	0.853
Rappel	0.843
F-mesure	0.843
Temps de calcul	0.21s
Kappa Statistic	0.765

TABLE 2 – Résultat du classifieur SMO

MATRICE DE CONFUSION			
a	b	c	<– classified as
78	12	10	a = aide
2	96	2	b = deuil
5	16	79	c = news

FIGURE 5 – Matrice de confusion du classifieur SMO

Les résultats montrent une précision élevée pour toutes les classes, ce qui est similaire à l'algorithme NaiveBayesMultionmial. Cependant, il convient de noter que SMO présente un taux de vrais positifs élevé pour la classe "deuil" mais un taux de vrais positifs relativement faible pour la classe "aide" par rapport aux deux autres classes. La matrice de confusion indique que la plupart des erreurs de classification sont dues à la confusion entre les classes "aide" et "deuil". La matrice de confusion indique que la plupart des erreurs de classification sont dues à la confusion entre les classes "aide" et "deuil".

4.1.3. JRip

JRip est en effet une version améliorée de l'algorithme de classification par règle RIPPER. Il utilise une méthode de classification en deux étapes : d'abord, il génère un ensemble de règles simples en utilisant l'algorithme RIPPER, puis il élague les règles redondantes ou inutiles à l'aide d'un processus de pruning.

Nous avons également testé l'algorithme JRip sur notre ensemble de données en utilisant Weka. Voici un exemple de règles générées par JRip pour cet ensemble de données spécifique :

```
JRIP rules:
======
(victimes <= 0) and (pensee <= 0) and (vous <= 0)
and (turque <= 0) => xClasse=news (137.0/48.0)
(pensee >= 1) => xClasse=deuil (60.0/2.0)
(allah >= 1) => xClasse=deuil (7.0/0.0)
(peuples >= 1) => xClasse=deuil (3.0/0.0)
(condoleance >= 1) => xClasse=deuil (3.0/0.0)
(condoleances >= 1) => xClasse=deuil (5.0/1.0)
=> xClasse=aide (85.0/14.0)
```

Selon les règles établies, le classifieur a donc obtenu le résultat suivant :

Précision	0.753
Rappel	0.750
F-mesure	0.751
Temps de calcul	1.08s
Kappa Statistic	0.625

TABLE 3 – Résultat du classifieur JRip

Les résultats suggèrent que l'algorithme JRip est capable de fournir des résultats de classification précis, bien que quelques erreurs de classification aient été commises entre les classes "aide" et "deuil". Les règles générées par l'algorithme JRip peuvent également être utiles pour mieux comprendre les facteurs qui contribuent à la classification de chaque instance dans une classe particulière.

MATRICE DE CONFUSION			
a	b	c	<– classified as
75	8	17	a = aide
5	77	18	b = deuil
14	13	73	c = news

FIGURE 6 – Matrice de confusion du classifieur JRip

4.1.4. PART

PART (Partial Decision Trees) est un algorithme d'apprentissage automatique utilisé pour la classification et la construction d'arbres de décision. PART utilise une approche de réduction de la complexité pour éviter le sur-apprentissage. Cette approche consiste à diviser les données en sous-ensembles plus petits plutôt que de créer des arbres complets. Cela permet de construire des arbres plus simples et plus facilement interprétables. Les résultats de classification obtenus avec l'algorithme PART sont présentés dans le tableau 4 et la figure 7 ci-dessous.

Précision	0.821
Rappel	0.820
F-mesure	0.820
Temps de calcul	0.5s
Kappa Statistic	0.73

TABLE 4 – Résultat du classifieur PART

Dans les résultats, on peut voir que la classe "aide" a la précision la plus élevée, suivie de près par la classe "deuil" et la classe "news". La matrice de confusion montre également que le nombre d'erreurs de classification est relativement faible, ce qui indique que l'algorithme est capable de bien distinguer entre les différentes classes.

MATRICE DE CONFUSION			
a	b	c	<– classified as
79	10	11	a = aide
8	83	9	b = deuil
7	9	84	c = news

FIGURE 7 – Matrice de confusion du classifieur PART

4.1.5. J48

J48 est une méthode d'apprentissage automatique qui combine l'algorithme J48 avec la technique de transformation de Fourier (FT). L'idée principale de J48 est d'appliquer une transformation de Fourier discrète (DFT) aux données d'entrée avant d'appliquer l'algorithme J48. La DFT convertit les données dans un espace de fréquence, ce qui permet de mieux distinguer les différentes caractéristiques des données. Une fois les données transformées, l'algorithme J48 est appliqué pour construire un arbre de décision. L'arbre de décision est construit en utilisant des règles de décision déduites de la distribution de fréquence des données. En utilisant ces règles de décision, l'arbre peut classer efficacement de nouvelles données en fonction de leurs caractéristiques. Le tableau 5 et la figure 8 présentent les résultats de classification obtenus avec l'algorithme J48, tandis que la figure 9 montre sa visualisation de l'arbre de décisions.

Précision	0.827
Rappel	0.820
F-mesure	0.821
Temps de calcul	0.36s
Kappa Statistic	0.73

TABLE 5 – Résultat du classifieur J48

MATRICE DE CONFUSION			
a	b	c	<– classified as
84	3	13	a = aide
7	76	17	b = deuil
6	8	86	c = news

FIGURE 8 – Matrice de confusion du classifieur J48

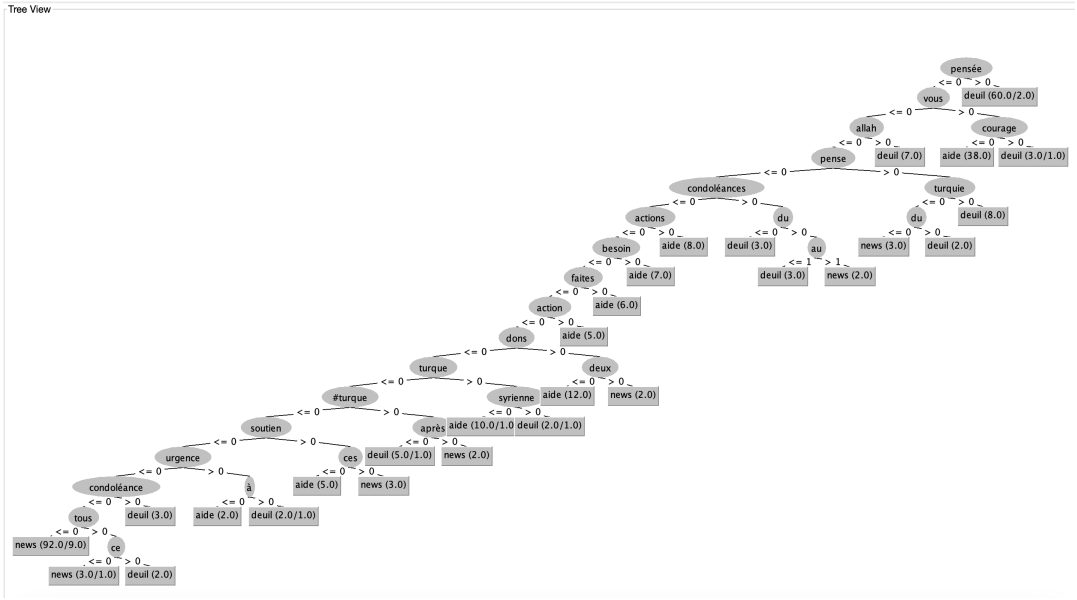

FIGURE 9 – Visualisation de l'arbre de décisions du classifieur J48

Les résultats de classification obtenus avec l'algorithme J48 sont globalement similaires à ceux obtenus avec l'algorithme PART, avec une précision élevée pour toutes les classes. Comme pour l'algorithme PART, la classe "aide" a la précision la plus élevée, suivie de près par la classe "deuil" et la classe "news".

4.1.6. Synthèse

En comparant les résultats des différents classificateurs, on peut conclure que le classificateur NaiveBayesMultionmial est le meilleur avec une F-mesure moyenne de 0,869 pour la classification des trois catégories de données. Le tableau Figure 10 ci-dessus montre les résultats du classificateur NaiveBayesMultionmial pour chaque catégorie (Aide, deuil, News) séparément.

	Aide	**Deuil**	**News**
Précision	0.882	0.839	0.895
Rappel	0.900	0.940	0.770
F-mesure	0.891	0.887	0.828

FIGURE 10 – Les résultats du classificateur NaiveBayesMultionmial pour chaque catégorie

On peut également constater que les algorithmes de classification ont des performances variables pour les différentes classes. Le classificateur NaiveBayesMultionmial obtient des résultats précis pour la classe "aide", mais des résultats moins bons pour les autres classes. Le classificateur SMO obtient des résultats similaires pour toutes les classes, mais avec une précision inférieure à celle de NaiveBayesMultionmial pour la classe "aide". Le classificateur JRip a des performances assez homogènes pour toutes les classes, avec une précision légèrement inférieure à celle de NaiveBayesMultionmial pour la classe "aide". Enfin, le classificateur J48 obtient les meilleurs résultats pour la classe "deuil" et des performances similaires à celles de JRip pour les autres classes. En général, tous les classifieurs ont des résultats similaires pour les classes "aide" et "deuil", tandis que la classe "news" est plus difficile à classer avec précision.

4.2. Expérimentations sur Sklearn

Ayant installé avec succès le module Sklearn en Python, nous avons tout d'abord créé une fonction particulière pour vectoriser les textes avec TF-IDF avec le module TfidfVectorizer au lieu de Sac de Mots. Nous avons ensuite entraîné le modèle de NaiveBayesmultinomial, SVC, DecisionTree, Kneighbors comme classifieur automatique, le dernier qui n'existe pas sur Weka. Le script rassemblant toutes les manipulations des fonctions et la présenation des résultats se bien trouve dans le script sklearn_test.ipynb ci-joint.

Dans mon script, il divise l'ensemble de données en un ensemble d'entraînement et un ensemble de test. Pendant notre expérimention ici, nous avons déterminé 20 % de données à utiliser pour l'ensemble de test en ajustant l'argument test_size à 0.2. Ensuite, il entraîne le classificateur en utilisant l'ensemble d'entraînement et fait des prédictions sur l'ensemble de test en utilisant la méthode predict.Le calcul de précision, rappel, f-mesure et kappa score est utilisé également pour évaluer la performance du classificateur en tenant compte de la possibilité de prédictions aléatoires.

4.2.1. NaiveBayesmultinomial

Comme sur Weka, la bibliothèque scikit-learn fournit également une implémentation facile à utiliser du Naive Bayes multinomial via la classe MultinomialNB.

Voici les tableaux et figures (TABLE 6, Figure 11) présentant les résultats de cet algorithme :

Précision	0.82
Rappel	0.82
F-mesure	0.82
Kappa Score	0.722

TABLE 6 – Résultat du classifieur NaiveBayesmultinomial sur Sklearn

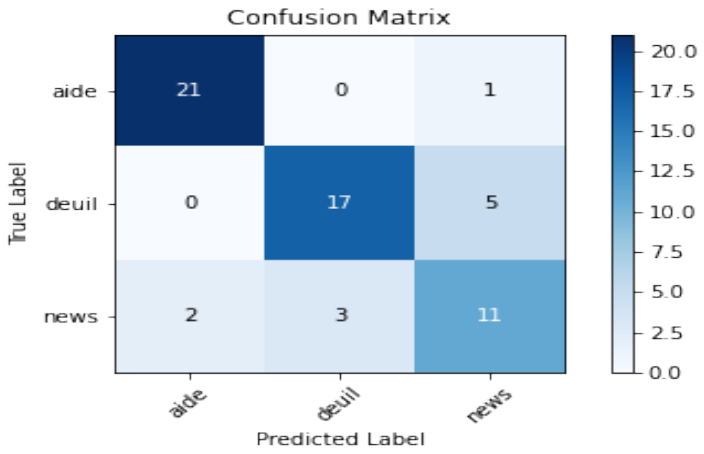

FIGURE 11 – Matrice de confusion du classifieur Bayes depuis sklearn

Comme le résultat le montre, il est indiqué que le classifieur bayésien a bien fonctionné pour la plupart des classes, avec seulement quelques erreurs. En particulier, il a bien classé la plupart des instances de la classe "aide" et "deuil", mais a eu un peu plus de difficulté avec la classe "news". Le score Kappa est de 0,72, ce qui indique une bonne fiabilité de la classification.

4.2.2. SVC

SVC (Support Vector Classification) est un type de modèle de classification basé sur SVM (Support Vector Machine). Cet algorithme de classification binaire et multiclasse populaire est bien implémenté dans le module scikit-learn de Python. Il est capable de construire un hyperplan dans un espace de grande dimension pour séparer efficacement des classes de données et atteindre une classification précise sur les données d'entraînement et de test.

D'après le résultat, le classifieur SVC semble avoir eu du mal à distinguer les classes "aide" et "deuil", ce qui est bien visible dans la Figure 12. Tout comme le classifieur Bayes, la classe "news" sont diffcilement classifée en particuler. Le rapport du résultat de SVC peut se trouver ci-dessous dans la Table 7. Globalement, les performances pour chaque classe sont bonnes, avec des précisions, des rappels et des scores F1 plutôt élevés, mais avec des variations entre les classes quand même.

Précision	0.81
Rappel	0.80
F-mesure	0.80
Kappa Score	0.699

TABLE 7 – Résultat du classifieur SVC sur Sklearn

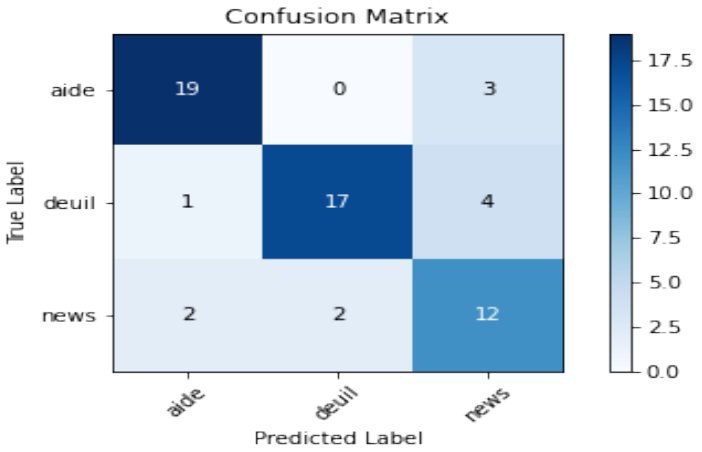

FIGURE 12 – Matrice de confusion du classifieur SVC depuis sklearn

4.2.3. DecisionTree

En sklearn, l'arbre de décision constitue aussi un classifieur vraiment important. Il est construit en choisissant la caractéristique qui divise le mieux les données en fonction de l'entropie ou du critère de Gini. Le critère de Gini mesure l'impureté d'un ensemble de données, tandis que l'entropie mesure l'incertitude dans un ensemble de données. L'algorithme continue à diviser l'ensemble de données en nœuds jusqu'à ce que chaque nœud contienne des données d'une seule classe. Une fois l'arbre de décision construit, il peut être utilisé pour classifier de nouvelles données en suivant le chemin dans l'arbre correspondant aux valeurs des attributs de la nouvelle donnée.

Par souci du sur-apprentissage, nous avons essayé d'utiliser cette fois-ci la méthode de GridSearchCV pour trouver les meilleurs hyperparamètres et choisir la combinaison qui donne la meilleure performance en utilisant une validation croisée de 5-fold.

Précision	0.75
Rappel	0.75
F-mesure	0.75
Kappa Score	0.621

TABLE 8 – Résultat du classifieur Decisiontree sur Sklearn

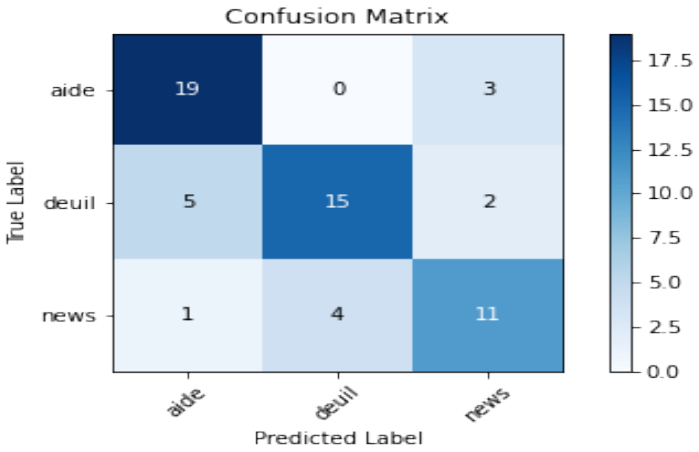

FIGURE 13 – Matrice de confusion du classifieur decisiontree depuis sklearn

En comparaison avec les deux classifieurs mentionnés précédemment, l'arbre de décision a obtenu des performances inférieures, avec une précision, un rappel et un score F1 plus faibles pour chaque classe. De plus, il a eu plus de difficulté à distinguer les classes "aide" et "deuil". Cependant, le score Kappa est comparable à celui des autres classifieurs, ce qui indique une fiabilité similaire de la classification. Les résultats ont été montré ci-dessous dans la Table 8 et Figure 13.

Nous vous avons mis également dans la Figure 14 une visualisation d'arbre généré à l'aide du module graphviz, qui ne peut que se trouver dans la page suivante.

4.2.4. KNeighbors

Le modèle k-NN est un algorithme de classification supervisée qui prédit la classe d'un nouvel exemple en cherchant les k exemples les plus proches dans l'ensemble de

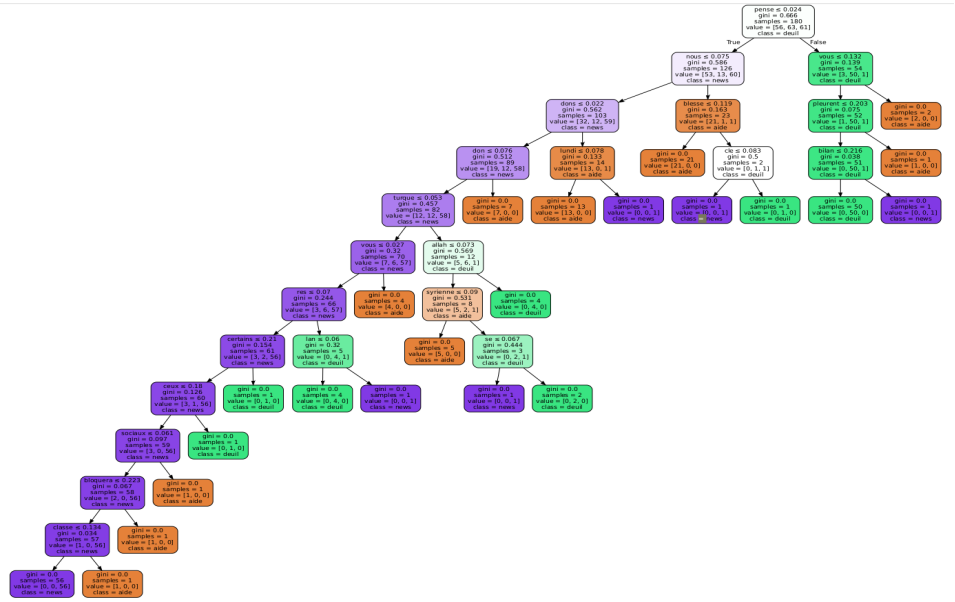

FIGURE 14 – Aperçu global d'arbre de décision depuis Sklearn

données d'entraînement et en déterminant la classe majoritaire. La distance entre deux exemples est mesurée en calculant la distance euclidienne ou de Manhattan. En scikit-learn, l'implémentation de k-NN est fournie par la classe KNeighborsClassifieur. Avec GridSearchCV, on peut également effectuer une recherche de grille afin de nous permettre d'obtenir le résultat le plus satisfaisant tout en prédisant les paramètres pertinents.

Par l'intermédiaire de la fonction GridSearchCV, nous avons donc pu trouver les meilleurs hyperparamètres : n_neighbors=14 et weights="distance". Les résultats dans la Table 9 et la Figure 15 indique que ce classifieur a encore plus de mal à distinguer la classe "deuil" des autres. Le score Kappa à 0,624 représente une fiabilité modérée de la classification.

Précision	0.76
Rappel	0.75
F-mesure	0.75
Kappa Score	0.624

TABLE 9 – Résultat du classifieur k-nearest Neighbors sur Sklearn

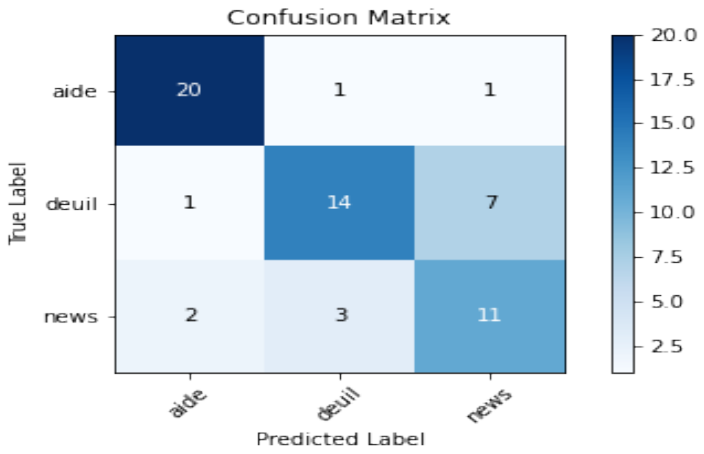

FIGURE 15 – Matrice de confusion du classifieur KNN depuis sklearn

4.2.5. Remarque

Sur sklearn, nous nous sommes servis de la méthode TF-IDF pour la vectorisation autrement que comme nous l'avons appliqué dans les expériences sur Weka. Cependant, les résultats obtenus avec cette méthode diffèrent selon les classifieurs utilisés. En comparant avec les résultats de Weka, nous avons constaté que les classifieurs étaient tout moins performants. Nous pouvons supposer que notre corpus n'est pas approprié pour le prétraitement de vectorisation par TF-IDF. Il est probablement plus pertinent d'utiliser le bag of words pour l'extraction des features de texte lors de la classification de textes courts car il est plus facile de capturer l'essence du texte en se concentrant sur les mots individuels plutôt que sur la fréquence d'un mot quelconque dans un document.

En outre, il est intéressant de remarqer que même dans les expériences menées sur Sklearn, les deux premiers algorithmes (naivesbayesnomiales et classifieur des vecteurs) ont obtenu d'excellents résultats par rapport à l'arbre de décision ou à d'autres classifieurs, ce qui est similaire à ce que nous avons observé dans Weka. En fin de compte, nos expériences avec Sklearn permettent de servir à confirmer notre conclusion obtenue avec Weka dans ce cas-là.

5. Conclusion

Dans le domaine de l'apprentissage automatique, il est bien connu que chaque algorithme de classification a ses avantages et ses limites. Les algorithmes de classification tant sur Weka que sur Sklearn utilisent différents critères pour classer les données, tels que la distance euclidienne, la corrélation, la similarité cosinus, etc. Cependant, chaque algorithme est basé sur un ensemble de suppositions et d'hypothèses, et chaque ensemble de données peut avoir des caractéristiques qui ne correspondent pas à ces hypothèses. Par conséquent, il est difficile de dire quel algorithme fonctionnera le mieux pour un ensemble de données spécifique.

Dans le cas de notre corpus, les résultats obtenus avec l'algorithme de Bayes sont satisfaisants, mais cela ne signifie pas nécessairement que Bayes est le meilleur choix pour notre classification, car il existe de nombreuses options de prétraitement du corpus qui peuvent influencer les résultats. De plus, notre corpus est relativement limité en termes de volume de données, et notre méthode de prétraitement des données peut être améliorée en utilisant des outils de traitement linguistique plus sophistiqués tels que Spacy, NLTK ou CoreNLP. Tous ces facteurs ont un impact significatif sur les résultats de classification obtenus avec différents algorithmes. À partir de nos expériences, nous avons conclu que le prétraitement avec TF-IDF ne convient pas très bien à notre corpus, et que l'utilisation de la modélisation BOW pour extraire les caractéristiques du texte est plus appropriée, car elle permet de se concentrer sur les mots individuels plutôt que sur leur fréquence dans le document, ce qui facilite la capture de l'essence du texte. En fin de compte, la sélection de l'algorithme de classification optimal dépendra des caractéristiques spécifiques de notre corpus et des choix de prétraitement des données que nous avons faits.

En outre, la qualité des résultats de classification dépend des paramètres d'entrée spécifiés pour chaque algorithme. Les valeurs optimales des paramètres dépendent aussi des caractéristiques du corpus, ce qui rend la sélection de l'algorithme de classification approprié encore plus complexe. Par conséquent, il est essentiel de trouver la meilleure combinaison entre la représentation des textes et les classifieurs pour notre projet.

6. Ressource

– scikit-learn : https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

– emoji : https://pypi.org/project/emoji/

– Unicodedata : https://docs.python.org/fr/3/library/unicodedata.html

– RE : https://docs.python.org/fr/3/library/re.html

– Graphviz : https://pypi.org/project/graphviz/

– Twitter API : https://developer.twitter.com/en/docs/twitter-api

– Twitter : https://twitter.com/lang=fr

– Facebook : https://www.facebook.com/home.php