XSA - X SENTIMENT ANALYSIS

RAPPORT DE PROJET

Pierre LAGUE & Paul-Henri ICHER

Repository : https://github.com/Jakcrimson/pjeb_twitter_sentiment_analysis

PJE-C : Analyse de sentiments sur Twitter

Université de Lille

December 2023

Contents

1 Description générale du projet

1.1 Problématique

Peut-on construire un classifieur efficace pour détecter, au sein d'un ou plusieurs tweet, l'émotion qu'ils expriment ?

Au-delà de la compréhension théorique et de l'implémentation de différents algorithmes d'apprentissage supervisé (classification naïve sur la base d'un dictionnaire, KNN, et classification bayésienne), le but de ce projet est de concevoir une application dont l'usage intuitif et ergonomique permet de comparer ces différentes méthodes et d'apprécier les performances de chacune en fonction du jeu de données considéré.
Dans ce cadre, nous avons souhaité fournir dans notre application suffisament de flexibilité dans le traitement des données et leur préprocessing pour permettre à l'utilisateur de sélectionner différentes bases d'entraînement (et également les modifier si nécessaire) afin d'évaluer les différents classifieurs.

1.2 Architecture de l'application - Logiciel

Nous avons décidé d'opter pour un patron de conception similaire à celui nommé Modèle-Vue-Contrôleur en faisant appel au maximum à la programmation orientée objet. Dans notre cas, la vue et le controlleur sont reliées dans un dossier "gui" contenant les fichiers suivants :

- xsa.py : le point d'entrée de notre programme, l'exécutable. Ce script est très long et propose toutes les fonctionnalités d'entraînement, de test (sur un dataset ou sur un single input), de sélection des hyperparamètres, de la cross-validation etc. C'est un choix assumé. La documentation ainsi qu'une implémentation "propre" nous permet de nous y retrouver.
- csv_cleaner.py : le script pour la méthode de nettoyage de données lors de l'ouverture d'un fichier csv.
- csv_editor.py : le script pour l'implémentation de l'éditeur de csv intégré à notre application.

. Tous nos modèles se trouvent dans le dossier "algorithms", ce dernier est organisé de la façon suivante :

- knn.py : le classifieur des K plus proches voisins.

- naive_bayes.py : le classifieur de Bayes naïf.
- naive_classification.py : le classifieur naïf basé sur l'appartenance des mots d'un tweet à un dictionnaire de mots positifs ou négatifs.
- metrics.py : un script permettant de calculer le taux d'erreur de l'accuracy d'un modèle à partir d'un csv issue d'une classification.

1.3 Architecture de l'application - Interface Graphique

L'utilisation de l'application est compatible avec des datasets structurés de la même manière que ceux fournis dans le cadre du PJE, c'est-à-dire comprenant les colonnes "target", "ids", "date", "flag", "utilisateur", "texte".

A partir de cette structure de données, l'application est organisée en 3 grandes parties :

- Le CVS_editor (cf Figure 1). Ce panel permet d'ouvrir un fichier csv, de l'éditer, et de le sauvegarder. Cet outil est conçu pour permettre une modification manuelle des données. Il peut par exemple être utilisé pour les labeliser les données une par une.

Figure 1: Une fois le fichier csv ouvert, il est possible pour l'utilisateur de l'éditer en écrivant/supprimant des informations dans les cases.

- Le CSV_viewer (cf Figure 2). Ce panel permet de visualiser un fichier csv. C'est via cette commande que la fonction de pre-processing "csv_cleaner" est proposée. Notre application propose alors l'utilisateur de nettoyer les données comme indiqué plus haut puis d'afficher le résultat. C'est dans ce panel que l'utilisateur va pouvoir ouvrir et formater des données pour ensuite les utiliser pour entraîner ou tester les différents modèles.
- Le "Algorithm Panel" (cf Figure 3). Ce panel permet de choisir un algorithme de classification parmi ceux proposés et d'utiliser le dataset chargé via CSV_viewer pour différentes opérations (test, entraînement, affichage des statistiques, etc.).

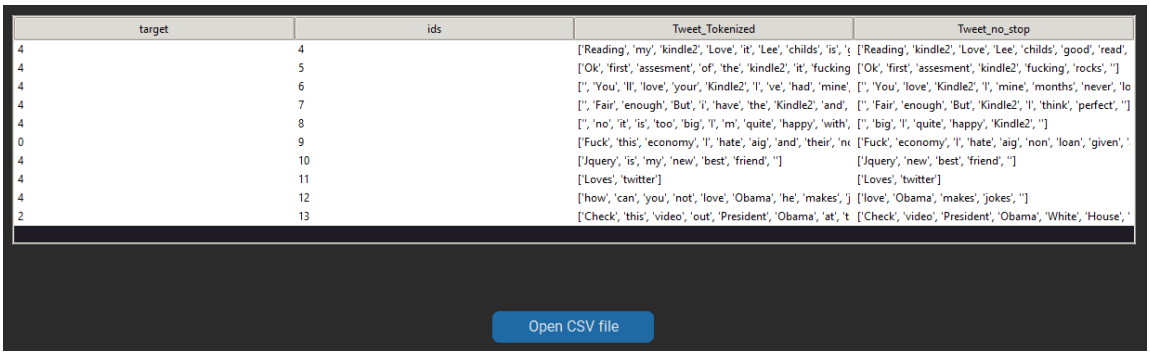


Figure 2: Un fichier d'entraînement est ouvert dans le viewer.

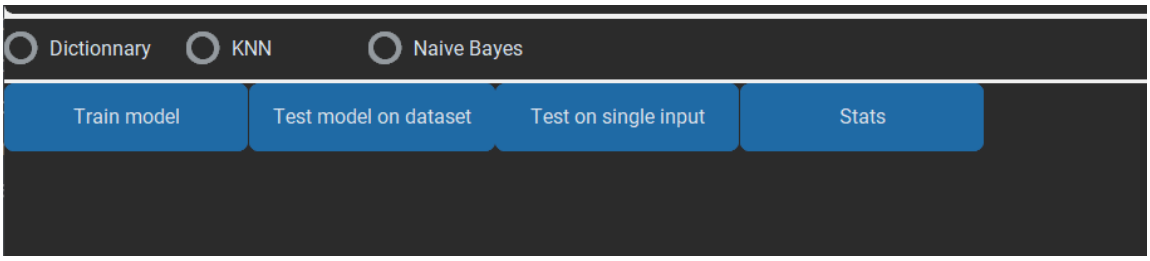


Figure 3: Classifieurs sélectionables via l'application et les options de leur utilisation

Si l'utilisateur ouvre l'application pour la première fois, il peut dans un premier temps ouvrir un fichier CSV avec le formatage requis ("target", "ids", "date", "flag", "utilisateur", "texte"). Avant cela, il pouvait également éditer manuellement le CSV et l'enregistrer si besoin.
Quand le dataset est ouvert et nettoyé, l'utilisateur peut l'utiliser pour entraîner un classifieur. Il ouvrira ensuite un second dataset qu'il sera possible d'utiliser pour tester les modèles entrainés par exemple.

1.4 Description de l'organisation au sein du binôme

Au fil du projet, notre binôme à su faire preuve de beaucoup d'équilibre, de communication et d'entre-aide. Paul-Henri s'est occupé de faire une pré-implémentation des algorithmes de classification en local ainsi qu'une batterie de tests pour assurer leur bon fonctionnement. Suite à cela, Pierre s'est occupé d'implémenter l'intégralité de l'application (interface graphique et intégration des scripts et fonctionnalités). De manière plus globale, tous les scripts dans "algorithms" ont été pris en charge par Paul-Henri et tous les scripts dans "gui" ont été pris en charge par Pierre. Nous avons utilisé le système de versionnage github pour mettre à jour notre application.

2 Détails des différents travaux réalisés

2.1 Préparation/nettoyage des données, base d'apprentissage

- **Étapes de traitement et fonctions utilisées**

La classe 'Csv_Cleaner' est conçue pour prétraiter et nettoyer des ensembles de données de tweets en vue de leur classification. Cette classe peut être utilisée soit pour nettoyer un fichier CSV entier contenant plusieurs tweets, soit pour nettoyer un seul tweet fourni en tant qu'entrée. L'initialisation de la classe prend en compte le type d'entrée, offrant ainsi une flexibilité pour les deux scénarios.

Voici le dataset de base sur lequel nous avons testé notre pré-processing :

	A	B	C	D	E	F
1	4	3	Mon May 11 03:17:40 UTC 2009	kindle2	tpryan	@stellargirl I loooooooovvvvveee my Kindle2. Not that the DX is cool, but the 2 is fantastic in its own right.
2	4	4	Mon May 11 03:18:03 UTC 2009	kindle2	vcu451	Reading my kindle2... Love it... Lee childs is good read.
3	4	5	Mon May 11 03:18:54 UTC 2009	kindle2	chadfu	Ok, first assesment of the #kindle2 ...it fucking rocks!!!
4	4	6	Mon May 11 03:19:04 UTC 2009	kindle2	SIX15	@kenburbary You'll love your Kindle2. I've had mine for a few months and never looked back. The new big one is huge! No need for remorse! :)
5	4	7	Mon May 11 03:21:41 UTC 2009	kindle2	yamarama	@mikefish Fair enough. But i have the Kindle2 and I think it's perfect :)
6	4	8	Mon May 11 03:22:00 UTC 2009	kindle2	GeorgeVHulme	@richardebaker no. it is too big. I'm quite happy with the Kindle2.
7	0	9	Mon May 11 03:22:30 UTC 2009	aig	Seth937	Fuck this economy. I hate aig and their non loan given asses.
8	4	10	Mon May 11 03:26:10 UTC 2009	jquery	dcostalis	Jquery is my new best friend.
9	4	11	Mon May 11 03:27:15 UTC 2009	twitter	PJ_King	Loves twitter
10	4	12	Mon May 11 03:29:20 UTC 2009	obama	mandanicole	how can you not love Obama? he makes jokes about himself.
11	2	13	Mon May 11 03:32:42 UTC 2009	obama	jpeb	Check this video out -- President Obama at the White House Correspondents' Dinner http://bit.ly/IMXUM
12	0	14	Mon May 11 03:32:48 UTC 2009	obama	kylesellers	@Karoli I firmly believe that Obama/Pelosi have ZERO desire to be civil. It's a charade and a slogan, but they want to destroy conservatism
13	4	15	Mon May 11 03:33:38 UTC 2009	obama	theviewfans	House Correspondents dinner was last night whoopi, barbara & sherri went, Obama got a standing ovation
14	4	16	Mon May 11 05:05:58 UTC 2009	nike	MumsFP	Watchin Espn..Jus seen this new Nike Commerical with a Puppet Lebron..sh*t was hilarious…LMAO!!!
15	0	17	Mon May 11 05:06:22 UTC 2009	nike	vincentx24x	dear nike, stop with the flywire. that shit is a waste of science. and ugly. love, @vincentx24x
16	4	18	Mon May 11 05:20:15 UTC 2009	lebron	cameronwylie	#lebron best athlete of our generation, if not all time (basketball related) I don't want to get into inter-sport debates about __1/2
17	0	19	Mon May 11 05:20:28 UTC 2009	lebron	luv8242	I was talking to this guy last night and he was telling me that he is a die hard Spurs fan. He also told me that he hates LeBron James.
18	4	20	Mon May 11 05:21:04 UTC 2009	lebron	mtgillikin	i love lebron. http://bit.ly/PdHur
19	0	21	Mon May 11 05:21:37 UTC 2009	lebron	ursecretdezire	@ludajuice Lebron is a Beast, but I'm still cheering 4 the A..til the end.
20	4	22	Mon May 11 05:21:45 UTC 2009	lebron	Native_01	@Pmillzz lebron IS THE BOSS
21	4	23	Mon May 11 05:22:03 UTC 2009	lebron	princezzcutz	@sketchbug Lebron is a hometown hero to me, lol I love the Lakers but let's go Cavs, lol

Figure 4: Dataset de base fourni sur moodle

Le processus de nettoyage implique plusieurs méthodes pour transformer le texte brut du tweet. La méthode 'link_process' supprime tout hyperlien, la méthode 'username_process' élimine les noms d'utilisateur ou les mentions '@', et la méthode 'rt_process' supprime la syntaxe de retweet ('RT'). Les caractères de ponctuation sont supprimés par la méthode 'punc_process', et les valeurs en pourcentage sont remplacées par 'XX%' à l'aide de la méthode 'pct_process'. Les valeurs en dollars et en euros sont remplacées par 'XX$' et 'XX€', respectivement, avec la méthode 'dollar_process'.

Pour la tokenisation, la méthode 'tokenization' utilise la bibliothèque 'nltk' pour diviser le texte en mots individuels. Les 'stop_words', des mots courants qui ne contribuent pas beaucoup à la classification, sont supprimés à l'aide de la méthode 'remove_stopwords'.

La méthode 'clean' est la principale fonction de nettoyage. Si la classe se trouve dans le contexte du traitement d'un ensemble de données complet, elle applique toutes les méthodes de nettoyage à chaque tweet de l'ensemble de données, y compris la tokenisation et la suppression des mots vides. Les colonnes telles que la date, le flag du tweet et l'utilisateur sont supprimées pour simplifier le dataframe. Si la classe est dans le contexte d'une seule entrée, elle applique les étapes de nettoyage nécessaires à cette entrée.

A l'issue du pré-processing, voici le type dataset obtenu :

	A	B	C	
1	target	ids	Tweet_Tokenized	Tweet_no_stop
2	4	4	['Reading', 'my', 'kindle2', 'Love', 'it', 'Lee', 'childs', 'is', 'good', 'read', '']	['Reading', 'kindle2', 'Love', 'Lee', 'childs', 'good', 'read', '']
3	4	5	['Ok', 'first', 'assesment', 'of', 'the', 'kindle2', 'it', 'fucking', 'rocks', '']	['Ok', 'first', 'assesment', 'kindle2', 'fucking', 'rocks', '']
4	4	6	['', 'You', 'll', 'love', 'your', 'Kindle2', 'I', 've', 'had', 'mine', 'for', 'a', 'few', 'months', 'and', 'never', 'looked', 'back', 'The', 'new',	['', 'You', 'love', 'Kindle2', 'I', 'mine', 'months', 'never', 'looked', 'back', 'The', 'new', 'big', 'one', 'huge', 'N
5	4	7	['', 'Fair', 'enough', 'But', 'i', 'have', 'the', 'Kindle2', 'and', 'I', 'think', 'it', 's', 'perfect', '']	['', 'Fair', 'enough', 'But', 'Kindle2', 'I', 'think', 'perfect', '']
6	4	8	['', 'no', 'it', 'is', 'too', 'big', 'I', 'm', 'quite', 'happy', 'with', 'the', 'Kindle2', '']	['', 'big', 'I', 'quite', 'happy', 'Kindle2', '']
7	0	9	['Fuck', 'this', 'economy', 'I', 'hate', 'aig', 'and', 'their', 'non', 'loan', 'given', 'asses', '']	['Fuck', 'economy', 'I', 'hate', 'aig', 'non', 'loan', 'given', 'asses', '']
8	4	10	['Jquery', 'is', 'my', 'new', 'best', 'friend', '']	['Jquery', 'new', 'best', 'friend', '']
9	4	11	['Loves', 'twitter']	['Loves', 'twitter']
10	4	12	['how', 'can', 'you', 'not', 'love', 'Obama', 'he', 'makes', 'jokes', 'about', 'himself', '']	['love', 'Obama', 'makes', 'jokes', '']
11	2	13	['Check', 'this', 'video', 'out', 'President', 'Obama', 'at', 'the', 'White', 'House', 'Correspondents', 'Dinner', 'http', 'bit', 'ly', 'IM	['Check', 'video', 'President', 'Obama', 'White', 'House', 'Correspondents', 'Dinner', 'http', 'bit', 'ly', 'IMX
12	0	14	['', 'I', 'firmly', 'believe', 'that', 'Obama', 'Pelosi', 'have', 'ZERO', 'desire', 'to', 'be', 'civil', 'It', 's', 'a', 'charade', 'and', 'a', 'slo	['', 'I', 'firmly', 'believe', 'Obama', 'Pelosi', 'ZERO', 'desire', 'civil', 'It', 'charade', 'slogan', 'want', 'destroy
13	4	15	['House', 'Correspondents', 'dinner', 'was', 'last', 'night', 'whoopi', 'barbara', 'amp', 'sherri', 'went', 'Obama', 'got', 'a', 'standi	['House', 'Correspondents', 'dinner', 'last', 'night', 'whoopi', 'barbara', 'amp', 'sherri', 'went', 'Obama', 'g
14	4	16	['Watchin', 'Espn', 'Jus', 'seen', 'this', 'new', 'Nike', 'Commerical', 'with', 'a', 'Puppet', 'Lebron', 'sh', 't', 'was', 'hilarious', 'LM	['Watchin', 'Espn', 'Jus', 'seen', 'new', 'Nike', 'Commerical', 'Puppet', 'Lebron', 'sh', 'hilarious', 'LMAO',
15	0	17	['dear', 'nike', 'stop', 'with', 'the', 'flywire', 'that', 'shit', 'is', 'a', 'waste', 'of', 'science', 'and', 'ugly', 'love', '']	['dear', 'nike', 'stop', 'flywire', 'shit', 'waste', 'science', 'ugly', 'love', '']
16	4	18	['', 'lebron', 'best', 'athlete', 'of', 'our', 'generation', 'if', 'not', 'all', 'time', 'basketball', 'related', 'I', 'don', 't', 'want', 'to', 'get', 'in	['', 'lebron', 'best', 'athlete', 'generation', 'time', 'basketball', 'related', 'I', 'want', 'get', 'inter', 'sport', 'deb
17	0	19	['I', 'was', 'talking', 'to', 'this', 'guy', 'last', 'night', 'and', 'he', 'was', 'telling', 'me', 'that', 'he', 'is', 'a', 'die', 'hard', 'Spurs', 'fan'	['I', 'talking', 'guy', 'last', 'night', 'telling', 'die', 'hard', 'Spurs', 'fan', 'He', 'also', 'told', 'hates', 'LeBron', 'J
18	4	20	['i', 'love', 'lebron', 'http', 'bit', 'ly', 'PdHur']	['love', 'lebron', 'http', 'bit', 'ly', 'PdHur']
19	0	21	['', 'Lebron', 'is', 'a', 'Beast', 'but', 'I', 'm', 'still', 'cheering', '4', 'the', 'A', 'til', 'the', 'end', '']	['', 'Lebron', 'Beast', 'I', 'still', 'cheering', '4', 'A', 'til', 'end', '']
20	4	22	['', 'lebron', 'IS', 'THE', 'BOSS']	['', 'lebron', 'IS', 'THE', 'BOSS']
21	4	23	['', 'Lebron', 'is', 'a', 'hometown', 'hero', 'to', 'me', 'lol', 'I', 'love', 'the', 'Lakers', 'but', 'let', 's', 'go', 'Cavs', 'lol']	['', 'Lebron', 'hometown', 'hero', 'lol', 'I', 'love', 'Lakers', 'let', 'go', 'Cavs', 'lol']

Figure 5: Database obtenue après préprocessing

Notre classe offre une approche complète et modulaire pour le nettoyage d'ensembles de données de tweets, les rendant adaptées aux tâches ultérieures de classification. Elle nous permet d'obtenir une nouvelle base contenant le même nombre d'exemple (ici 498 tweets) équilibrée entre les différentes classes (181 positifs, 139 neutres, 177 négatifs).

- **Difficultés rencontrées**

Le formatage des données et les différentes fonctions de préprocessing employées n'ont pas fait l'objet de difficultés d'implémentation particulières, si ce n'est la manipulation d'expressions régulières, en particuliers pour les données textuelles tokenisées. En effet, l'usage de la bibliothèque 'nltk' transforme les chaînes de caractères en liste de chaînes de caractère (une chaîne par mot). Or, ces listes étaient encodées en format string dans notre base de données issue du pre-processing. Nous avons donc eu besoin de les transformer à nouveau en liste, et avons pour cela eu recours à la librairie 'ast' combinée à une expression régulière pour re-spécifier le format des tweets encodés à chaque ligne.

Si les difficultés liées au formatage était réduites, le travail de conception de notre dataframe a en revanche nécessité une réflexion pour sélectionner des filtres pertinents et anticiper les informations qu'il était nécessaire de préserver pour entraîner nos modèles. L'étape de pré-processing nous a ainsi permis de prévoir les besoins de nos modèles pour optimiser leur performance. A titre d'exemple, le choix d'inclure une colonne avec et sans stop-words dérive justement de l'idée de se concentrer uniquement sur des mots dont l'emploi sera davantage liés au contexte émotionnel. Le fait de conserver une colonne où l'on sépare les pronoms et mots de conjonctions avait ainsi pour but d'éviter de possibles biais dans nos données. Si, par exemple, la longueur des tweets n'est pas normalisée dans une database, et qu'une étiquette contient en moyenne des tweets plus long, alors

les stop-words auraient plus de chance d'y être associés du fait, simplement, d'un biais statistique. Le travail de conception et d'implémentation du modèles Naïves Bayes a, dans la suite du projet, confirmer le bien-fondé de cette réflexion : le prise en compte ou non des stop-words est un paramètre à part entière de ce dernier.

2.2 Algorithmes de classification

2.2.1 Classification Dictionnaire (Naïve)

La classe 'NaiveClassification' représente un classificateur naïf qui classe les tweets en fonction du nombre de mots positifs ou négatifs qu'ils contiennent. Si le nombre de mots négatifs est supérieur au nombre de mots positifs, la fonction renvoie 0, indiquant un tweet globalement négatif. Si le nombre de mots négatifs est inférieur au nombre de mots positifs, la fonction renvoie 4, indiquant un tweet globalement positif. Si les deux nombres sont égaux, la fonction renvoie 2, indiquant un tweet globalement neutre.

La classe est initialisée avec des listes de mots positifs et négatifs provenant de fichiers texte. L'initialisation prend également en compte si la classification concerne un seul tweet (paramètre 'single_input_classification'). La fonction 'count_positives' compte le nombre de mots positifs dans le tweet, tandis que la fonction 'count_negatives' compte le nombre de mots négatifs. La fonction 'classify' effectue la classification en comparant les nombres de mots positifs et négatifs, et la fonction 'get_classified' renvoie le dataframe mis à jour avec une colonne de classification ajoutée.

Cette approche de classification est naïve car elle se base uniquement sur le nombre de mots positifs et négatifs, sans tenir compte du contexte (ex : ironie) ou de la pondération des mots. La classe offre une manière simple de classer les tweets en fonction de la présence de mots clés positifs et négatifs. En d'autres termes, cette classe (et la Naîve Bayes) représente notre baseline en termes d'évaluation des classifieurs.

2.2.2 K plus proches voisins (KNN)

La classe 'KNN' implémente un classificateur des k plus proches voisins (KNN) from scratch. Cette classe utilise une bibliothèque externe pour calculer les distances, avec le crédit d'auteur attribué dans le readme.

L'initialisation de la classe prend en compte divers paramètres, notamment les données sur lequel notre modèle va se baser pour calculer les distance (dataset d'entraînement), le

nombre de voisins à considérer dans l'algorithme, le type de distance (parmi une liste de distances disponibles), et le type de vote (majoritaire ou pondéré).

La méthode 'liste_distance' calcule la distance entre un tweet donné et tous les autres tweets dans l'ensemble de données du dataset d'entraînement en fonction du type de distance spécifiée lors de l'initialisation. Les types de distances disponibles incluent "naïve", "levenshtein", "lcs", "damerau_levenshtein", "hamming", "jaro", "cosine", "jaccard", "sorensen_dice", et "qgram_dice". Il est à noter que nous avons utilisé des distances normalisée pour calculer les similarité entre un tweet non labelisé et ceux de notre dataset d'entraînement.

La méthode 'classification' effectue la classification du tweet en fonction de sa distance par rapport aux autres tweets, du type de vote et du nombre de voisins. Les voisins sont triés par ordre croissant de distance, et les votes peuvent être majoritaires ou pondérés en fonction de la méthode de vote spécifiée. Dans le cas où ils sont pondérés, on les multiplie par l'inverse de la distance au carré. Donc plus la distance entre deux tweet est importante, plus la valeur du vote sera petite. On somme ensuite les votes pour chaque étiquette. Le label retourné correspond alors à la somme la plus importante.

Pour labelliser un ensemble de tweets dans une base de donnée, on itère la méthode 'classification' sur la liste des tweets tokénisés d'un dataset donné après avoir paramétré le calcul des distances dans la méthode 'liste_distance'.

2.2.3 Naïve Bayes

La classe 'NaiveBayes' met en œuvre un classificateur Naive Bayes pour la classification de tweets en fonction de divers paramètres spécifiés à l'initialisation.
Le premier paramètre modifiable est le type de représentation des données ("fréquence" ou "présence"). Avec l'option "présence", on récupère les probabilités conditionnelles associées à chaque étiquette pour chaque mot présent dans le tweet. Avec l'option "fréquence", chaque probabilité sera calculée pour chaque étiquette, mais sera en plus pondérée par le nombre d'occurrences de ce mot dans le tweet en les élevant une puissance correspondant à l'occurrence. Si un mot a une faible probabilité d'apparaître sous une étiquette (donc une probabilité conditionnelle proche de 0) alors l'élever à un puissance fera considérablement baisser sa valeur. Au contraire, si sa probabilité est élevée (proche de 1), cette baisse sera moins importante.
Le second paramètre concerne le fait de prendre en compte ou pas stop-words. Dans les consignes données, cela correspond à tous mot de longueur strictement inférieure à 3. Comme indiqué dans la partie sur le pré-processing, cela permet de pouvoir considérer

uniquement les mots dont l'usage refléterait davantage le contexte sémantique et donc émotionnel du tweet (les stop-words étant souvent des pronoms ou mots de conjonction). Enfin, le dernier paramètre est le type de n-gramme utilisé : "uni-gramme" si l'on découpe le tweet en mot uniques, "bi-gramme" si on le découpe en ensembles de deux mots, ou "both" si utilise ces deux découpages en même temps. L'idée ici est de pouvoir prendre en compte des expressions (ici des groupes de deux mots) dont le sens global diffère de la simple addition des mots qui les composent (ex: "chef d'oeuvre"). On suppose ici que la prise en compte de cette variation sémantique pourrait également être liée à une variation du contexte et donc d'émotion associée.

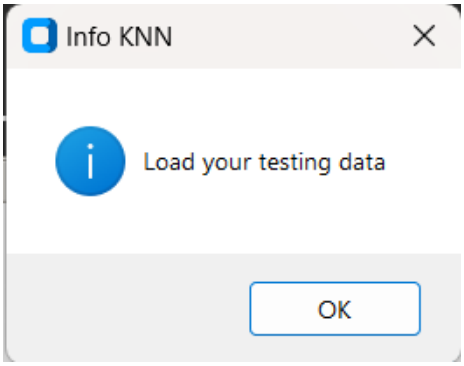
Notre classe 'NaiveBayes' initialise en calcule les probabilités conditionnelles à l'aide d'un ensemble d'entraînement. Elle comporte la méthode 'dictionnaire' qui crée, pour chaque étiquette, un dictionnaire des mots et de leurs occurrences dans la base de données en fonction des paramètres spécifiés. On découpe ainsi chaque tweet tokenisé en mots ('unigramme'), groupe de deux mots ('bi-grammes') ou les deux ('both'). On exclut les stop-word des tweets tokénisés si on ne souhaite pas les prendre en compte. Bien que nous ayons effectué une tokenisation de nos tweet sans stop word dans le pre-processing via la librairie 'nltk', nous avons ici respecté la consigne donnée en cours pour l'élimination de ces mots qui se veut plus stricte ("tous mots de moins de 3 caractères"). Une fois le découpage du tweet tokénisé effectué, on additionne les occurrences des mots en fonction de l'étiquette attribuée au tweet. On obtient au final un dictionnaire pour chaque étiquette qui répertorie le nombre de fois où un mot a été employé avec ce contexte émotionnel. On calcule, à partir de ces fréquence, les probabilité conditionnelles d'observer un mot donnée au sein d'une étiquette. Il est à noté ici que nous avons utilisé un estimateur de Laplace pour éviter que cette probabilité soit égale à 0 dans le cas où un mot n'aurait pas était présent sous une étiquette.

Une fois le modèle "entraîné" via la méthode "dictionnaire", on utilise la méthode 'classification' pour attribuer une étiquette à un tweet test. Elle se réfère aux dictionnaires construits à l'aide des paramètres spécifiés lors de l'initialisation, et calcule pour un tweet donné les probabilités pour chaque classe. Elle retourne l'étiquette dont la probabilité associée est la plus grande. Pour labelliser un dataset test, on appliquera cette méthode pour chaque tweet tokénisé de ce dernier.

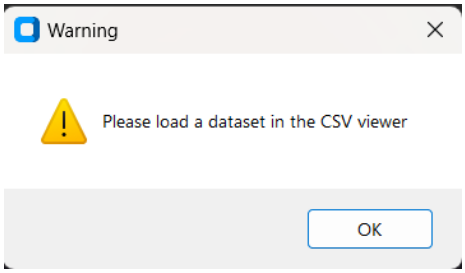
La classe ‘NaiveBayes‘ offre donc une implémentation personnalisée du classificateur : 12 combinaisons possibles de paramètre sont possibles pour notre modèle.

2.3 Interface graphique

La partie principale de notre interface graphique est présentée en 1.3. Tout au long de l’utilisation de l’application l’utilisateur sera ammené à intéragir avec des fenêtre d’information:



(a) Information panel



(b) Warning panel

Lors du test de ses modèles sur des nouvelles données, l’utilisateur sera ammené à remplir des fenêtres de configuration pour les modèles :

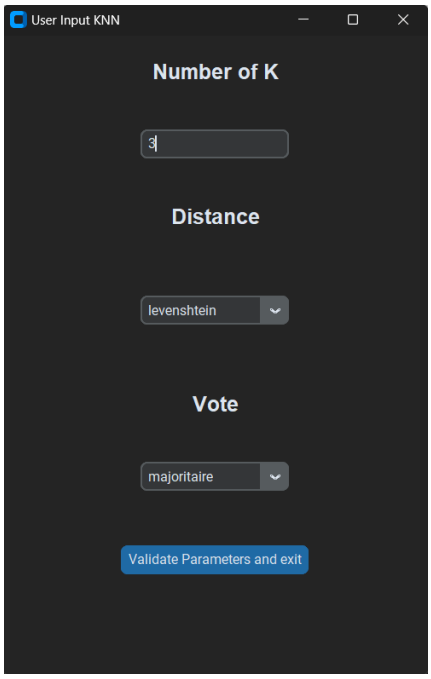


Figure 7: Fenêtre de configuration du modèle KNN

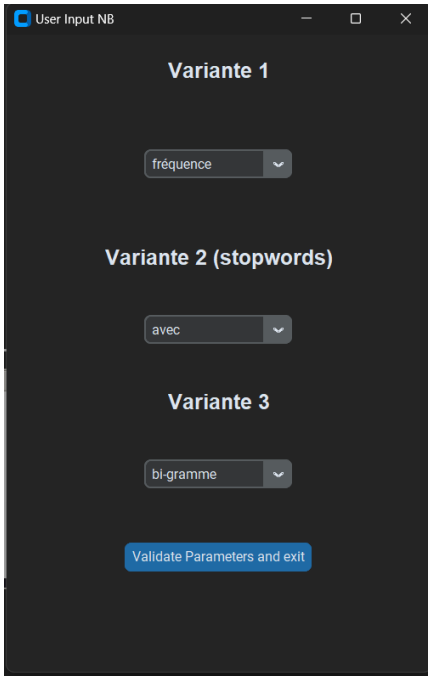


Figure 8: Fenêtre de configuration du modèle Naive Bayes

Enfin, lorsque l'utilisateur veut classifier un input personnalisé, il peut le faire en cliquant sur "Test on single input", remplir la configuration de modèle qu'il souhaite et écrire son tweet :

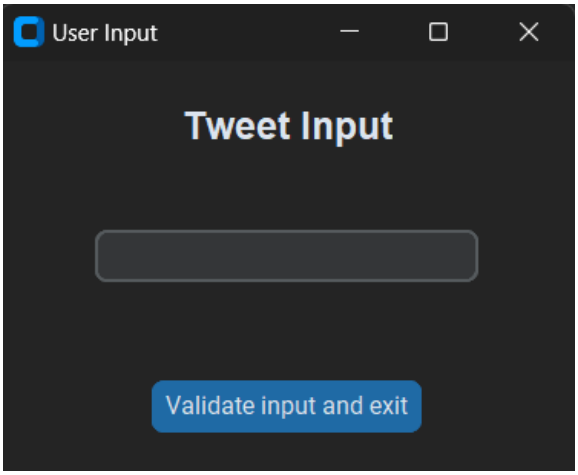


Figure 9: Fenêtre d'entrée pour l'utilisateur lors de la classification d'un seul tweet.

3 Résultats de la classification avec les différentes méthodes et analyse

3.1 Entraînement sans cross-validation

Pour évoluer la performance de nos modèles, la première méthode a consisté à former un ensemble d'entraînement regroupant 70% des données randomisées issues du premier dataset fourni sur moodle (498 tweets). Les 30% de données randomisées restantes ont constitué l'ensemble de test.
Nous avons testé les différentes configurations d'hyperparamètres pour entraîner nos modèles afin de trouver celle qui maximiserait le score de prédiction sur l'ensemble de test.
Pour comparer nos modèles, nous nous sommes basé sur un score d'accuracy calculé sur l'échantillon test. Il correspond au nombre de classifications correctes sur cet échantillon divisé par sa taille.
Le modèle naïf (basé sur le dictionnaire des mots positifs et négatif) a obtenu un score équivalent à **0.49**. Compte-tenu que nous disposons de 3 étiquettes, une classification au hasard correspondrait à un score de 33%. Ici, ce modèle performe donc mieux qu'une labellisation aléatoire. On peut supposer que cela soit du à une bonne performance pour les étiquettes positives et négatives. En effet, si l'on suppose que notre dataset contiennent peu de tweet ironique, la méthode de classification via dictionnaire semble pertinente. En revanche, l'attribution d'étiquette "neutre" dans le modèle (tout du moins dans l'architecture présentée en cours et que nous avons implémentée) semble moins probable que les deux autres compte tenu du fait qu'il faille,dans ce cas, avoir exactement le même nombre de mots postifs et négatifs dans le tweet. Une piste pour améliorer ce modèle serait par exemple de lisser ce critère en tolérant un certains écart entre le nombre de mots positifs ou négatifs pour une labellisation sous l'étiquette "neutre" ou d'attribuer un poids/une valeur à chaque mot du vocabulaire en fonction de son "expressivité" (distinguer les mots à connotation très positives et ceux simplement positifs par exemple).
Pour les modèles KNN et Naives Bayes, les résultats sont présentés dans les deux graphes suivants et correspondent aux tests des différentes combinaisons de paramètres que nous avons testés. Les scores reportés sont ceux d'accuracy.

Selon le premier graphe, le modèle Bayésien Naïf le plus performant serait celui paramétré sans prendre en compte le nombre d'occurences du mot (attribut "présence"), mais qui prend en compte les "stop-words" et qui considère seulement des uni-grammes (i.e. les mot entier uniquement). Ce modèle atteint un score d'accuracy de **0.687**.

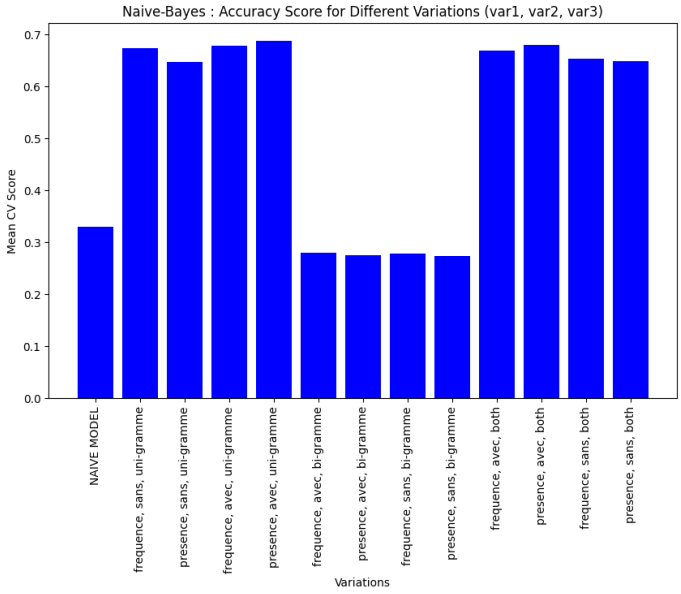


Figure 10: Accuracy score for various Naive Bayes models.

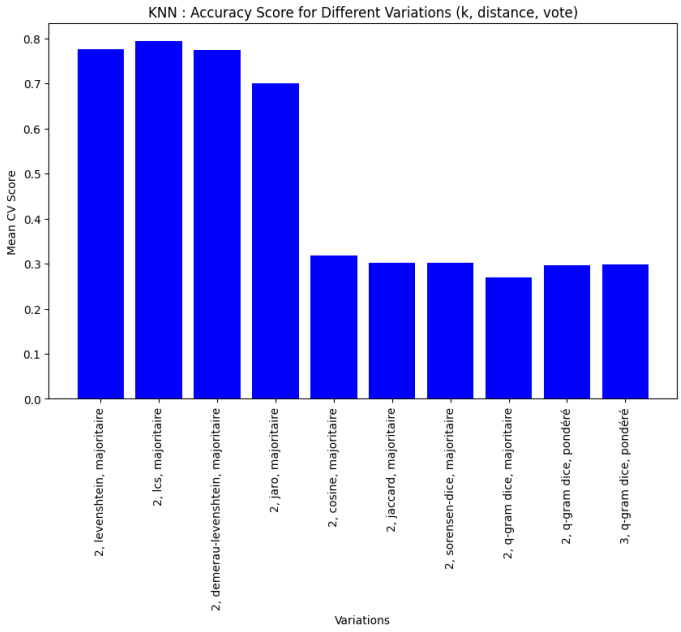


Figure 11: Accuracy score for various KNN models.

Selon le second graphe, le modèle KNN le plus performant serait celui paramétré avec $k = 2$, la distance LCS et un vote majoritaire pour la sélection du voisin le plus proche. Ce modèle atteint un score d'accuracy de **0.793**.

Dans la littérature, nous retrouvons souvent que le classifieur KNN est plus performant que la plupart des autres classifieurs (sans compter le deep learning), ceci s'explique par

un contexte et des données très appropriées pour l'application d'un modèle KNN. On peut supposer également que la variété de distances disponibles pour estimer la similarité entre les tweets permet de capter efficacement les variations d'informations entre deux contenus textuels. Disposer d'une base suffisamment importante et correctement labelisée permettrait ainsi de disposer d'un nombre de voisins suffisamment dense pour estimer correctement l'étiquette des données. En revanche, l'estimation de similarité entre les données est gourmands en terme de temps de calcul. Nous avons pu constaté, avec seulement 500 données environ, que le temps de calcul pour les modèles KNN tournaient en général autour de 1 minute. Ceux pour naîves étaient de l'ordre de quelques secondes.

3.2 Cross Validation

À la suite de l'évaluation de nos modèles sur deux ensembles distincts, nous avons souhaité effectuer une validation croisée avec un nombre de splits fixé à 5. Le but d'une validation croisée étant de pouvoir estimer l'accuracy des modèles indépendamment des données sur lesquels ils sont entraînés et testés (donc indépendamment des biais statistiques pouvant y être présents).
En d'autres termes, nous évaluons successivement les différentes combinaisons de modèles sur divers sous-ensembles de nos données. Il est à noter que lors du processus d'entraînement, la validation croisée est appliquée à toutes les combinaisons de modèles possibles, ce qui peut occasionner des durées d'entraînement importantes : environ 10 minutes pour KNN (attribuable au calcul exhaustif de toutes les distances pour des nombres de voisins différents) et de l'ordre de quelques secondes pour Naïve Bayes.

Cette approche, bien que demandant un investissement en temps, offre une évaluation plus robuste des performances des modèles, permettant ainsi de mieux appréhender leur capacité de généralisation. La disparité observée dans les temps d'entraînement entre les modèles KNN et Naive Bayes souligne également la complexité algorithmique inhérente à chaque approche, et met en évidence l'impact de ces choix sur les performances et la rapidité d'exécution du processus d'apprentissage automatique.

Notre démarche a été la suivante : nous avons randomisé notre dataset et découpé en 5 sous-ensemble différents de données pour formé les ensembles de validation. Pour chaque fold, le modèle était entraîné sur 80% de données restantes vis-à-vis de chaque sous-ensembles de validation. Enfin, pour le dataset de test, nous avons utilsé les tweets "avengers" fournis sur moodle (50 en tout) et labellisés à la main (3.3).

Les scores d'accuracy moyens obtenus sont les suivants :

Algorithm	Mean CV Score
Naive Bayes (présence, avec, uni-gramme)	0.646
KNN (k=2, LCS, Pondéré)	0.50
Naive Classification	0.49

Table 1: Mean CV Score des modèles les plus performants après une cross validation, cv=5.

Nous constatons une concordance notable de nos résultats, notamment en ce qui concerne le modèle Naive Bayes. Le modèle présentant les performances les plus élevées à la suite de la validation croisée est celui configuré selon les paramètres suivants : Naive Bayes (présence, avec, uni-gramme). Toutefois, il est important de noter une légère diminution du score (0,654 contre 0,687), attribuable à une quantité réduite de données utilisées lors de l'évaluation au cours de la validation croisée (les jeux de données étant plus restreints que ceux présentés dans 3.1).

Pour le modèle KNN, la configuration déterminée par la validation croisée diffère légèrement avec celle obtenue précédemment pour le type de vote. Le meilleur score moyen d'accuracy est obtenu avec un vote pondéré et non pas majoritaire (le nombre de voisin k=2 et le distance LCS choisie sont en revanche identiques). Ici, il est important de noter la chute du score d'accuracy comparé à l'évaluation du score sans cross-validation. Cette baisse de performance semble le plus probablement liée à l'insuffisance de données pour le calcul et la généralisation des distances.

Il est évident que la dimension de nos ensembles de données influence significativement la performance des modèles, soulignant ainsi l'importance cruciale de la quantité de données dans le processus d'entraînement et d'évaluation des modèles d'apprentissage automatique.

3.3 Test sur données non labellisées

Pour obtenir un échantillon test dont les données n'auraient pas été utilisées pour l'évaluation des scores d'accuracy en cross-validation, nous avons utilisé le jeu de données 'avengers' non-labélisées disponibles sur le site de l'UE. Nous avons labélisé 50 tweets a l'aide de notre application. Puis en lancant les modèles nous obtenons les scores suivants :

Il est important de noter que l'annotation manuelle introduit un biais. A priori, la labellisation n'est jamais optimale.

Les scores de généralisation(respectivement **0.5**, **0.52**, **0.36** pour le classifieur naïf, le KNN(k=2, LCS,vote pondéré) et le Naive Bayes(présence, avec stop-word et uni-gramme)

Algorithm	Accuracy (acc)	Error Rate (er)
Naive Classification	0.5	0.5
KNN (k=2, LCS, Pondéré)	0.529	0.481
Naive Bayes (presence, avec, uni-gramme)	0.366	0.634

Table 2: Mesures de performances des différents algorithmes sur des données manuellement labéllisées.

de nos modèles sont cohérents avec ceux observés avec les scores de validation (**0.49**, **0.50**, **0.64**) pour la classification naïve par dictionnaire et le KNN. Le score de généralisation pour le Naives Bayes chute en revanche. Il est possible d'attribuer cette baisse à une modification du vocabulaire employé entre les tweets de notre dataset d'entraînement et celui de test. Les distances employées dans le KNN étant moins sensible à cet effet. Une meilleure de vérifier notre hypothèse serait d'augmenter la taille de notre dataset d'entraînement pour enrichir les dictionnaires associés à chaque étiquette et à partir desquels sont calculées les différentes probabilités conditionnelles. Le rôle de la labellisation manuelle peut également influer sur le score en lien avec la perception des tweets jugés neutres et/ou positifs. Le modèle n'a pas la capacité de comprendre les doubles sens ou l'implicite par exemple. La labellisation doit donc être la plus objective possible.

4 Conclusion

Au cours de cette étude, nous avons réalisé la mise en œuvre de divers mécanismes essentiels à l'analyse de données, focalisant particulièrement notre attention sur l'analyse de tweets. Ces mécanismes englobent des processus de traitement des données ainsi que des approches de classification utilisant différents modèles d'apprentissage supervisés tels que le KNN et le Naïve Bayes, en plus d'un modèle automatique basé sur la Classification par Dictionnaire. Ces stratégies ont été intégrées au sein d'une application dotée d'une interface graphique, offrant une gamme diversifiée d'actions à l'utilisateur.

Les résultats obtenus avec ou sans cross-validation convergent de manière cohérente. Les disparités de performance entre l'entraînement indépendant des modèles et la validation croisée sont attribuables à la disparité de volume de données utilisées au cours des phases d'entraînement et de validation. En ce qui concerne le déploiement des modèles sur des données annotées manuellement, les performances se révèlent moins convaincantes. Une fois de plus, l'acquisition d'un ensemble de données plus substantiel se révélerait préférable pour une amélioration des performances des modèles.

Ce projet nous a offert l'opportunité de conduire une étude de manière exhaustive et autonome, nous familiarisant ainsi avec la rigueur et la méthodologie inhérentes à la science des données.

5 Annexes

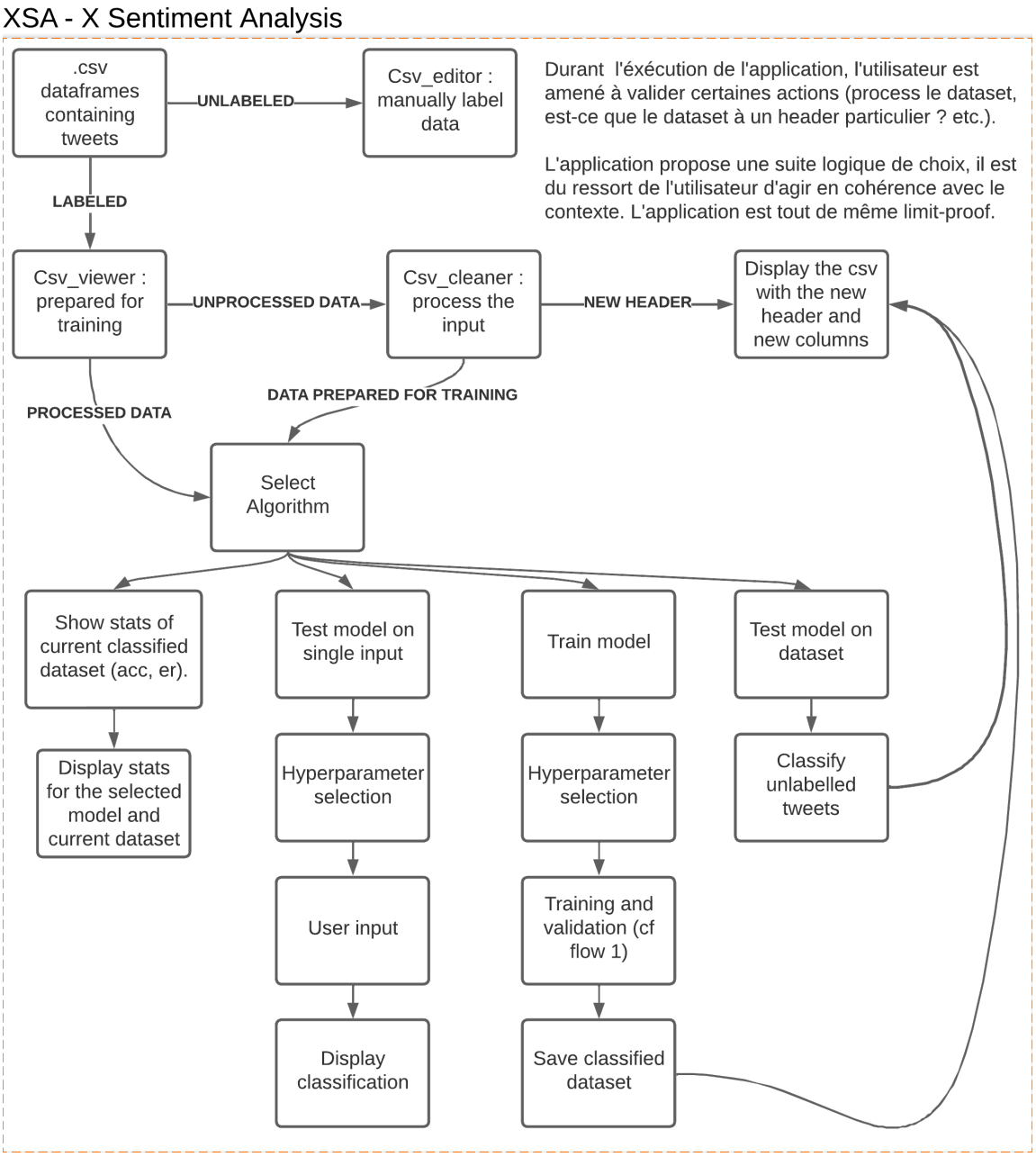


Figure 12: Workflow de l'application : potentiels scénarios que l'utilisateur est amené à suivre.

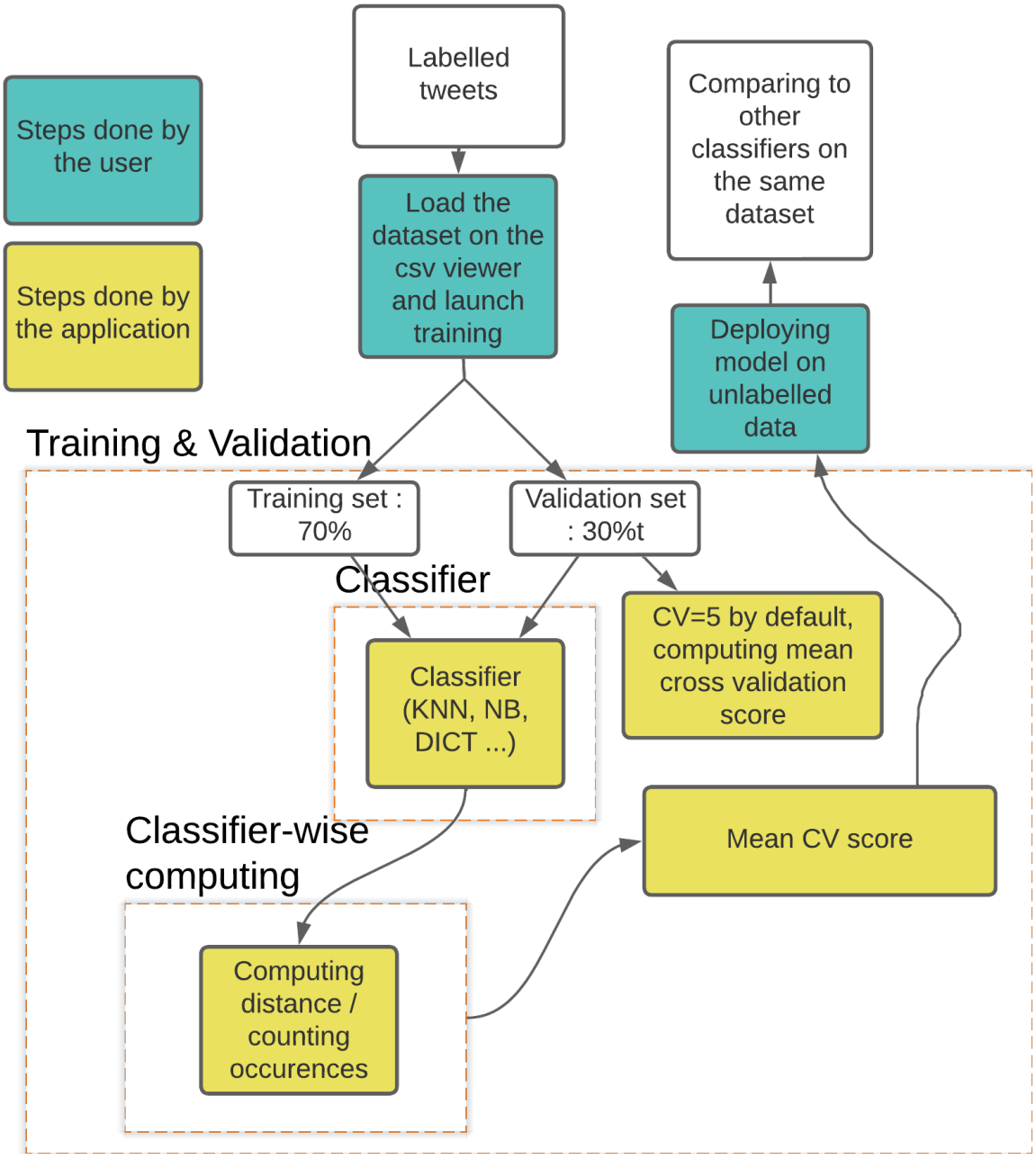


Figure 13: Worflow détaillé de la phase d'entraînement et de cross validation.