Inès MARTINS 22401511
Mickaëla MASTRODICASA 22403748
Jourdan WILSON 22405088
M1 TAL
2024/2025

Rendu le 9/05/2025
M. DUPONT
LZT001 - Introduction à la fouille de texte

Rapport de projet : Classification de chansons par genre à partir des paroles

Sorbonne Nouvelle université des cultures
Université Paris Nanterre
inalco Institut national des langues et civilisations orientales
WEKA

Sommaire

❖ Introduction

Le Traitement Automatique des Langues (Natural Language Processing en anglais) est un domaine vaste que l'on peut diviser en plusieurs disciplines. On y retrouve par exemple des disciplines telles que : la linguistique, l'informatique et l'intelligence artificielle, ces disciplines s'alliant dans le but de créer des outils de traitement de textes et de la parole (incluant la parole signée) pour diverses applications. Dans ce domaine, on retrouve donc des « sous-domaines » qui sont liés à ces disciplines, notamment la fouille de texte (text mining en anglais).

La fouille de texte se révèle en effet représenter une grande part de ce que l'on effectue en TAL, car avec l'essor des technologies numériques, la quantité de textes disponibles en ligne et dans les systèmes d'information n'a cessé d'augmenter. Aussi, pour exploiter efficacement toutes ces données, la fouille de texte s'est imposée comme un domaine de recherche particulièrement important. Elle vise à analyser et extraire des informations pertinentes à partir de documents écrits.

La classification est une tâche de fouille de texte que l'on retrouve parmi les quatre tâches principales de cette discipline qui sont : la recherche d'information, l'extraction d'information, l'annotation et enfin la classification. Cette tâche consiste à assigner une catégorie à un document selon son contenu. Elle peut être utilisée dans de nombreux contextes comme le tri de courriels, l'analyse d'opinions ou la catégorisation d'articles selon certains critères par exemple. Pour cela, il est nécessaire de transformer les textes en données exploitables, puis d'utiliser des techniques d'analyse pour apprendre à les classifier de manière correcte le plus précisément possible.

Notre projet vise à classifier des chansons par genre à partir de leurs paroles. C'est-à-dire que l'on cherche à prédire le genre musical (ici pop, rap ou hip-hop) d'une chanson en analysant uniquement ses paroles (son contenu lexical). Pour cela, il faut dans un premier temps constituer un corpus de données. Dans notre cas il s'agit d'un corpus composé de textes étant des paroles de chansons de divers genres (pop, rap et R&B).

Nous n'avons pas choisi ces genres par hasard, en effet il a semblé cohérent de faire en sorte qu'il s'agisse de genres assez proches mais sans pour autant qu'ils ne le soient trop, auquel cas la prédiction pourrait s'avérer bien trop complexe. Nous avons également fait le choix de créer un corpus composé exclusivement de paroles de chansons en anglais. Car traiter plusieurs langues dans un seul corpus n'aurait pas été judicieux étant donné que selon les langues certains critères sont variables (ponctuation,ect.).

Dans le cadre de ce projet, nous avons utilisé non seulement Python pour la création de nos scripts mais également l'outil Weka. Cet outil est un logiciel d'apprentissage automatique développé par l'université de Waikato en Nouvelle-Zélande. Il contient un ensemble

d'algorithmes d'analyse et de classification de données, qui nous ont été utiles dans le cadre de la réalisation de ce projet.

- ❖ Méthodologie
  - ➢ Méthodologie globale

Afin de travailler en collaboration sur ce projet, nous avons fait le choix de créer un répertoire sur GitHub pour pouvoir partager l'avancée de notre travail et nos scripts. De même, dans le cadre de l'écriture de ce rapport nous avons utilisé les outils de travail collaboratifs de Google, à savoir Google Docs, nous permettant de pouvoir toutes participer à la rédaction.

- ➢ Scrapping pour la constitution du corpus

Dans un premier temps, dans le but de constituer un corpus de 453 fichiers textes, 151 pour le genre rap, 151 pour la pop et 151 pour le R&B, afin d'avoir une égalité dans le nombre de textes de chacun des corpus par genre. Chacune d'entre nous a dans un premier temps récolté des paroles de chansons d'un genre déterminé.

Nous avons globalement toutes utilisé le même processus en créant un script de récupération des paroles de chansons à l'aide de L'API Genius. Cette API permettant d'accéder aux paroles des différents titres musicaux en interrogeant soit un artiste donnée, soit des combinaisons titre et artiste. Il faut donc donner au script des noms d'artistes afin qu'il puisse effectuer ses recherches, ensuite deux modes d'utilisation sont possibles. Le premier permet d'obtenir les chansons les plus populaires d'un artiste donné, en utilisant une fonction qui interroge l'API en fournissant le nom de l'artiste, et renvoie une liste de chansons sous forme de paires titre/artiste. L'utilisateur peut également choisir l'artiste à interroger grâce à un argument en ligne de commande (--artist) et préciser le genre associé (--genre). Une fois la liste des chansons obtenue, une nouvelle fonction va interroger Genius chanson par chanson. Pour chaque titre, elle télécharge les paroles si elles sont disponibles, puis les enregistre dans un fichier texte à l'aide d'une fonction de sauvegarde. Cette dernière enregistre les fichiers sous le nom : <artiste>_<titre>.txt. Le script utilise également argparse pour choisir l'artiste en ligne de commande. Il est donc possible de l'exécuter avec une commande comme : '''python3 recup_parole.py --artist "21 Savage" --genre rap'''

- ➢ Pré-traitement des données

Le but de ce projet étant de classifier des textes, il fallait dans un premier temps construire un jeu de données propre et structuré à partir des fichiers texte contenant les paroles de chansons issues de trois genres musicaux. Ces données devaient être préparées en vue d'une tâche de classification automatique selon le genre musical. Pour cela, un corpus de fichiers textes a été

constitué et organisé en trois dossiers distincts (corpus/rap, corpus/pop et corpus/R&B). Dans chacun des dossiers, chaque fichier contient les paroles d'une chanson, mais également diverses informations parasites telles que les noms des contributeurs, les liens vers des traductions, ou encore des métadonnées générées par les plateformes de lyrics. Le premier enjeu a donc été de filtrer ces fichiers afin d'en extraire uniquement les paroles.

Un script Python a été développé pour automatiser cette étape de prétraitement. À l'aide du module glob, les fichiers ont été récupérés dans chaque dossier et organisés dans un dictionnaire par genre. Une fonction principale a ensuite été chargée de lire chaque fichier, d'y appliquer un nettoyage ciblé, puis de stocker les résultats dans un fichier CSV. Le nettoyage du texte a ensuite été confié à une autre fonction ayant pour rôle d'ignorer toutes les lignes de texte inutiles en amont des paroles, et de supprimer les balises comme [Verse], [Chorus], [Bridge], etc. Cependant, avec les fichiers du genre rap, contrairement aux morceaux pop ou R&B, ces balises apparaissaient très rarement mais ils incluaient presque systématiquement une première ligne non pertinente (contributeurs, tags de plateforme, etc.). Pour remédier à cela, une exception a été ajoutée dans le script : la première ligne de chaque fichier du genre rap est systématiquement supprimée avant tout traitement. Une fois les paroles nettoyées, elles ont été enregistrées dans un fichier CSV avec trois colonnes : Paroles(le texte nettoyé), Genre(le genre musical), et Titre + artiste (le nom original du fichier .txt, permettant d'identifier la chanson).

Ce fichier constitue après cette étape de pré-traitement un jeu de données propre, prêt à être utilisé pour des étapes d'analyse exploratoire ou d'apprentissage automatique.

➢ Analyse préliminaire du corpus avec des nuages de mots

Avant d'aborder l'évaluation des modèles de classification, une première exploration du corpus a été menée à partir de nuages de mots générés par genre musical. Ces visualisations permettent d'identifier les régularités lexicales dominantes dans chaque classe et d'anticiper les difficultés potentielles que pourraient rencontrer les algorithmes.

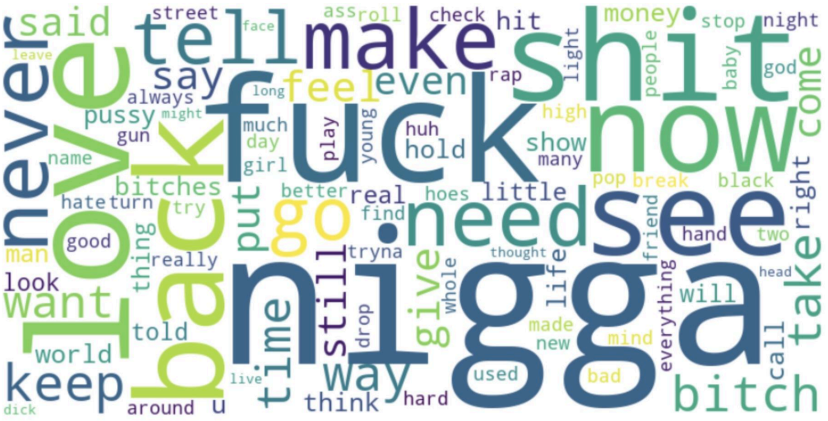
Nuage de mots pour le genre pop

Le nuage de mots du genre pop met en évidence un lexique centré sur les sentiments, les relations personnelles et les émotions. Les plus visibles concernent des actions affectives ou des termes états émotionnels comme "love", "feel", "want", "make", ou encore "kiss" et "heart". Ce lexique exprime un registre plutôt doux, introspectif et orienté vers l'intimité, avec une tonalité générale positive ou mélancolique.

Nuage de mots pour le genre R&B

Le nuage R&B présente un registre très similaire. On y retrouve des verbes liés à l'attachement ou au besoin, comme "need", "tell", "see", "give", "go", ou encore "baby", très récurrents. Ce vocabulaire, comme dans le genre pop, traduit des interactions interpersonnelles, souvent amoureuses ou sensuelles, avec une prédominance de formulations simples et émotionnelles. La proximité lexicale entre pop et R&B est manifeste, ce qui rend leur séparation algorithmique potentiellement difficile.

Nuage de mots pour le genre rap



En revanche, le genre rap se distingue immédiatement par un lexique plus brut, plus oral, et parfois explicite transgressif. Des mots comme "f *ck", "sh* t", "n *gga", ou "b* tch" sont

visuellement dominants. À côté de cela, on note aussi une fréquence importante de termes liés à l'action ou à l'affirmation de soi comme "back", "hit", "real", "money", "put". Ce champ lexical s'éloigne fortement des deux autres, en s'appuyant sur une langue plus familière, plus directe, souvent issue du registre de l'oralité urbaine.

Ces différences, visibles dès cette première observation, permettent d'anticiper certains comportements des classificateurs. La séparation du genre rap devrait être facilitée par son vocabulaire spécifique. En revanche, la distinction entre pop et R&B, qui partagent un fond lexical et thématique très proche, constituera probablement la principale difficulté pour les modèles.

➢ Classification avec Python

Pour la classification avec Python, il est nécessaire d'écrire un script en utilisant des modèles de la bibliothèque scikit-learn. L'objectif du script "algo.py" était de tester un algorithme d'apprentissage supervisé pour classifier automatiquement des paroles de chansons en trois genres musicaux. Les paroles ont dans un premier temps été vectorisés à l'aide de la méthode TF-IDF (TfidfVectorizer), qui permet de pondérer les termes selon leur fréquence dans un document et leur rareté dans l'ensemble du corpus. Dans le cadre de cette classification les données ont été divisées en deux sous-ensembles : un ensemble d'entraînement représentant 70% du corpus total, et un ensemble de test représentant les 30% restants. Ce découpage a été effectué à l'aide de la fonction train_test_split de scikit-learn, en veillant à maintenir la proportion des classes (rap, pop, R&B) dans chaque sous-ensemble grâce à l'argument stratify. C'est pour que l'évaluation du modèle ne soit pas faussée par une répartition déséquilibrée des genres musicaux dans les jeux de données.

Naive Bayes, SVM et Random Forest sont ici les algorithmes sélectionnés pour la tâche de classification. Cependant, chaque modèle comporte des hyperparamètres qui influencent fortement son comportement et sa capacité de généralisation. Afin de sélectionner les meilleures combinaisons de paramètres pour chaque algorithme, nous avons utilisé un outil fourni par scikit-learn: GridSearchCV.

La méthode GridSearchCVpermet de tester automatiquement un ensemble de combinaisons d'hyperparamètres et de sélectionner celle qui maximise une métrique de performance (ici, l'accuracy), en utilisant une validation croisée à 5 plis (5-fold cross-validation). Concrètement, les données d'entraînement sont elles-mêmes divisées en cinq sous-ensembles : à chaque itération, le modèle est entraîné sur quatre d'entre eux et évalué sur le cinquième. Ce processus est répété cinq fois, ce qui permet d'évaluer la robustesse du modèle tout en entraînant les biais dus à un découpage unique.

Pour Naive Bayes, nous avons testé différentes valeurs du paramètre de lissage alpha: 0.1, 0.5 et 1.0. Le modèle Naive Bayes est basé sur des probabilités : il estime pour chaque mot sa probabilité d'apparaître dans chaque classe (ici, chaque genre musical). Mais il arrive qu'un mot présent dans une chanson du test ne soit jamais apparu dans les chansons du corpus d'entraînement. Dans ce cas, sa probabilité est considérée comme nulle, ce qui peut faire chuter la probabilité globale de la classe, même si le reste du texte est très représentatif. Pour résoudre ce problème, on utilise un lissage appelé lissage de Laplace, qui consiste à ajouter artificiellement une petite probabilité à tous les mots, même ceux qu'on n'a jamais vus pendant l'apprentissage. Cela évite les zéros dans les calculs. Ce lissage est contrôlé par un paramètre alpha : Si alpha est petit (ex: 0.1), on ajoute une petite quantité de probabilité fictive i.e. le modèle reste très sensible aux mots rares. Si alpha est grand (ex : 1.0), on ajoute plus de probabilité à tous les mots i.e. le modèle devient plus « conservateur » et moins influencé par les mots rares.

Dans notre expérimentation, nous avons testé plusieurs valeurs d'alpha pour trouver le bon équilibre entre spécificité (mots rares) et généralisation. Cela nous permet de voir si un modèle très sensible au lexique est plus performant qu'un modèle plus lissé.

Pour Random Forest, nous avons testé les combinaisons suivantes : nombre d'arbres (n_estimators de 100 à 200). Un plus grand nombre d'arbres permet généralement de lisser les prédictions et de mieux généraliser, mais au prix d'un temps de calcul plus long. 100 est souvent un bon point de départ, tandis que 200 permet de vérifier si la performance continue à s'améliorer ou si elle se stabilise.

Il y a aussi la profondeur maximale des arbres (max_depth: Aucun, 10, 20). Lorsqu'il est fixé à None, chaque arbre peut continuer à croître tant qu'il trouve des subdivisions possibles dans les données. Cela permet une modélisation très fine, mais peut aussi entraîner un surapprentissage. En testant également des profondeurs limitées (10 et 20), nous cherchons à contraindre la complexité du modèle pour qu'il reste plus simple et plus généralisable.

Et enfin la taille minimale des sous-ensembles qui définissent le nombre minimum d'échantillons requis pour diviser un nœud. Si ce nombre est trop bas (ex : 2), les arbres peuvent se subdiviser de manière excessive, créant des règles trop spécifiques. En augmentant cette valeur à 5, on force l'arbre à regrouper un peu plus d'exemples avant de se diviser pour effectuer une division (min_samples_split: 2 ou 5).

Une fois la recherche par grille effectuée pour chaque algorithme, nous avons entraîné les meilleurs modèles trouvés (best_estimator_) sur l'ensemble d'entraînement. Chaque modèle a ensuite été évalué sur le jeu de test à l'aide de plusieurs indicateurs : l'exactitude, la matrice de confusion, ainsi qu'un rapport de classification affichant la précision, le rappel et le score F1 pour chaque classe.

Pour SVM, nous avons utilisé le type de noyau (kernel) et le paramètre de régularisation C. Le noyau est une fonction mathématique qui permet au modèle de tracer une frontière de séparation entre les classes (ici : rap, pop, R&B) dans l'espace vectoriel défini par les représentations TF-IDF des chansons.

Nous avons testé deux types de noyaux : le noyau linéaire (kernel="linear") et le noyau RBF (kernel="rbf"pour Radial Basis Function). Le noyau linéaire suppose que les classes peuvent être séparées par un hyperplan, c'est-à-dire une « ligne droite » dans un espace multidimensionnel. Ce type de noyau fonctionne bien avec les données textuelles, car les représentations TF-IDF vivent déjà dans des espaces de très grande dimension, où les classes sont souvent naturellement séparables. Le noyau linéaire présente également l'avantage d'être rapide à entraîner et de limiter les risques de surapprentissage.

Le noyau RBF, en revanche, permet une séparation non linéaire : il est capable de tracer des frontières de décision plus flexibles, en courbe, pour mieux épouser la forme des données. Cela peut être utile lorsque les classes ne sont pas bien séparées dans l'espace initial. Toutefois, ce noyau est plus coûteux en calcul, et surtout plus sensible à l'overfitting, c'est-à-dire au risque de trop bien s'ajuster aux exemples d'entraînement sans bien généraliser.

Aussi, nous avons testé différentes valeurs du paramètre C, qui contrôlent la régularisation du modèle. Un faible C(par exemple 0.1) indique au modèle qu'il peut tolérer quelques erreurs de classification sur les données d'entraînement, en échange d'une frontière de séparation plus simple. À l'inverse, un grand C (comme 10) demande au modèle de minimiser au maximum les erreurs, quitte à créer une frontière plus complexe, ce qui peut améliorer la précision à court terme, mais parfois au détriment de la généralisation.

Après avoir sélectionné les meilleurs hyperparamètres pour chaque modèle, il est nécessaire de vérifier que les performances enregistrées ne sont pas simplement dues au hasard d'un découpage particulier des données. Pour cela, nous avons mis en place une évaluation par validation multi-runs, qui consiste à :

- Répéter plusieurs fois le processus d'entraînement et d'évaluation,
- En variante le random_state utilisé pour le découpage des données en jeu d'entraînement et de test,
- Et en mesurant à chaque fois l'exactitude des modèles sur un test indépendant.

Nous avons défini une fonction multi_run_evaluation() qui prend en entrée :

- Le Data Frame contenant les données (df),
- Les meilleurs modèles (best_nb, best_svm, best_rf) trouvés via GridSearchCV,
- Et le nombre de répétitions souhaitées (n_runs, fixé ici à 10).

À chaque exécution, le processus est donc le suivant :

1. Nouveau split aléatoire du corpus en 70 % entraînement et 30 % test, avec un random_state différent à chaque itération.
2. Vectorisation TF-IDF des paroles.
3. Réentraînement de chaque modèle sur ce nouveau jeu d'entraînement.
4. Prédiction sur le jeu de test, et calcul de l'exactitude obtenu.
5.

➢ Classification avec Weka

Pour la classification avec Weka, nous avons repris directement le script «vectorisation.py» fourni, qui contient déjà un pré-traitement léger (mise en minuscules, suppression de la plupart des signes de ponctuation, tokenisation et calcul de TF). Il n'a pas été nécessaire d'écrire un script de nettoyage supplémentaire (stop-words, etc.) Ce script rend les données en .arff aussi, la préparation initiale suffisante pour démarrer Weka.

Pour les protocoles d'évaluation, on avait le test set 80 %/20 % créé par le script «shuffle_and_split.py» qui mélange les fichiers et répartit 80 % en «corpus_train» et 20 % en «corpus_test». Pour garantir le même vocabulaire on ajoute la commande: python3 vectorisation.py --lexicon train.arff corpus_test/ test.arff pour garder le même lexicon entre les deux corpus. Nous avons aussi testé l'option "Cross-validation" (10 folds) ce qui permet d'obtenir une estimation rapide et stable des performances moyennes de chaque modèle sans split manuel pour comparer les résultats. On veut montrer à la fois une estimation robuste « interne » (CV) et une vérification « externe » (held-out) pour couvrir deux angles méthodologiques. Étant donné l'apparition de "?" dans les métriques sur Weka avec l'option "Supplied test set", on n'a pas pu confirmer que Weka utilisait correctement le jeu de test fourni, et que le jeu de test était correctement pris en compte. On a donc ajouté les résultats de la validation croisée à 10 plis (10-fold) pour assurer une évaluation fiable. De même, nous pouvons tester le corpus sur une rotation de données de test.

Les classifieurs testés sont les suivants. «ZeroR» (baseline) qui prédit toujours la classe majoritaire et sert de vérification que Weka lit bien les ARFF et génère un résultat minimal. «NaiveBayesMultinomial» qui calcule la probabilité multinomiale pour chaque genre. SMO, l'implémentation SVM de Weka avec noyau linéaire. L'Arbre de décision J48 (C4.5) et l'ensemble d'arbres de décision RandomForest.

Pour chacun des modèles, Weka fournit un résumé incluant l'accuracy (taux de classification correcte), Precision, Recall, F-Measure par classe et moyenne pondérée, AUC (aire sous la courbe ROC) et la matrice de confusion. La courbe ROC (Receiver Operating Characteristic) représente, pour chaque seuil de décision possible, le taux de vrais positifs (True Positive Rate = Recall) versus le taux de faux positifs (False Positive Rate = 1 – Specificity) (nous ne les mentionnerons pas beaucoup dans l'analyse). Plus l'AUC se rapproche de 1, plus le modèle distingue bien les classes indépendante du choix d'un seuil fixe. Weka fournit aussi

de nombreux indicateurs (aire sous la courbe ROC, PRC Area, MCC, etc.) que nous n'avons pas vraiment étudiés en cours. Nous ne les utiliserons donc pas pour l'analyse. Les sorties complètes (buffers de résultats et prédictions) sont stockées dans «results» dossier sur github pour comparaison. Les images des résumés des analyses seront également incluses dans ce document.

Cette comparaison Weka permet de confirmer et approfondir les observations faites côté Python. Utiliser Weka en parallèle de Python présente plusieurs avantages méthodologiques et pratiques. Weka offre une interface visuelle intuitive (Explorer / Experimenter) qui permet de charger un ARFF, sélectionner des classifieurs, configurer en quelques clics les options de test (CV, supplied test set, percentage split (que je n'ai découvert qu'après avoir créé le script «shuffle_and_split.py» et divisé le corpus de manière externe) et lancer immédiatement les expériences. Cela facilite les essais rapides de nouveaux algorithmes ou de réglages et la comparaison de leurs performances. Nous avons pensé qu'il était pertinent de l'utiliser surtout parce que nous en avons parlé et testé en classe. Weka fonctionne dans ce projet comme un outil de validation externe. Il permet de confirmer dans un environnement différent que les choix en Python et d'explorer d'autres options de classifieurs sans réécrire de code.

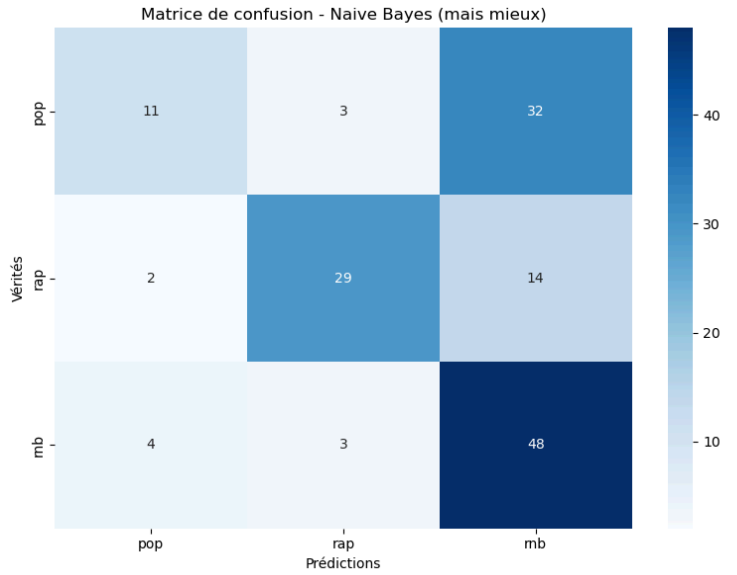
Pour les visualisations de Weka on inclut des visualisations des erreurs de classification qui sont proposées dans l'onglet « visualize classifier errors ». Weka affiche chaque instance selon sa classe réelle et sa classe prédite (axe X vs axe Y). Le «jitter» ajoute un léger déplacement aléatoire aux points pour éviter la superposition ce qui permet de mieux voir la densité des erreurs et des bonnes classifications plutôt que d'avoir un unique point. Ces plots illustrent graphiquement très clairement les capacités des classificateurs utilisés. C'est pour SMO (SVM) que ce type de visualisation est le plus parlant, les autres sont inclus dans l'annexe. Les classifieurs basés sur les arbres de décision (J48 et Random Forest) montre souvent des blocs de points plus compacts, Naive Bayes Multinomial donne un nuage plus diffus (sa décision repose sur des probabilités par mot qui rend les frontières moins nettes), et LazyIBK (k-NN) donne une vue pour voir localement si les erreurs sont isolées ou groupées. Même dans ces cas, k-NN montre une distribution intéressante à visualiser (en annexe), tandis que ZeroR confirme le bon chargement des données sans apport prédictif.

- ❖ Résultats : Interprétation des matrices

Après observation des résultats obtenus à l'aide de Python et Weka nous avons pu effectuer plusieurs constats que nous allons évoquer dans cette partie.

- ➢ Naive Bayes :
  - ■ Sur les résultats obtenus avec Python

Matrice de confusion obtenue avec l'algorithme Naive Bayes en Python



Le modèle Naive Bayes montre d'après la matrice de confusion une nette tendance à confondre les chansons pop avec le genre R&B. En effet, sur 46 chansons réellement de genre pop, seules 11 sont correctement classées comme telles, tandis que 32 sont classées à tort comme R&B. Cette confusion s'observe également, dans une moindre mesure, pour le genre rap : 14 chansons rap sont elles aussi mal étiquetées en R&B. En revanche, le modèle reconnaît assez bien les chansons R&B, avec 48 prédictions correctes sur 55. Ce comportement s'explique sans doute par le manque de sensibilité aux structures globales du texte ou aux cooccurrences complexes de termes, ce qui peut nuire à sa capacité à distinguer des genres proches lexicalement.

- Sur les résultats obtenus avec Weka

```
=== Confusion Matrix ===

 a  b  c   <-- classified as
10  0 11 |  a = pop
 2 17  1 |  b = rap
 6  0 17 |  c = rnb
```

Held-Out 80%/20%

```
=== Confusion Matrix ===

 a  b  c   <-- classified as
67  3 34 |  a = pop
10 79 10 |  b = rap
28  3 82 |  c = rnb
```

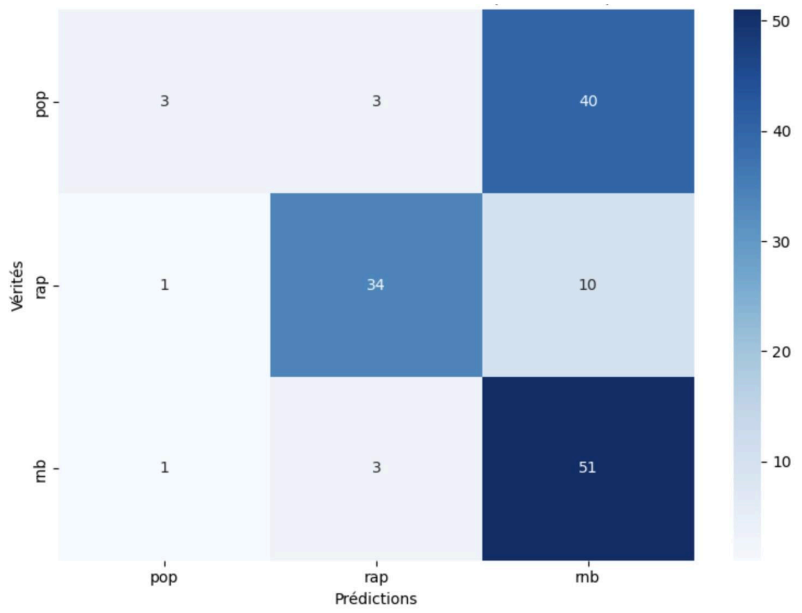
10-fold Cross Validation

Comme en Python, on observe une tendance à confondre la pop avec le R&B (beaucoup de titres pop vont dans la colonne c = rnb). Cependant, Weka améliore la classification pour la

pop tout en maintenant des bonnes performances sur le rap. Le modèle multinomial de Weka se montre ainsi plus équilibré et moins biaisé vers le R&B qu'en Python.

- ➢ Arbres de décision
  - ■ Sur les résultats obtenus avec Python

Matrice de confusion obtenue avec l'algorithme d'arbre de décision Python (RandomForestClassifier)



La matrice de confusion du modèle Random Forest affiche les meilleures performances en ce qui concerne la classification des genres rap et R&B. Il parvient à identifier 34 chansons rap sur 45, et 51 chansons R&B sur 55, ce qui représente une excellente précision pour ces deux classes. Cependant, il échoue presque totalement sur la classe pop, avec seulement 3 bonnes prédictions. La quasi-totalité des chansons pop (40 sur 46) sont classées à tort comme R&B. Ce défaut peut s'expliquer par un surapprentissage du modèle sur certaines caractéristiques lexicales propres au R&B, qui pourraient également apparaître dans les chansons pop, rendant la frontière entre ces deux genres trop fine pour être bien saisie par les arbres de décision.

- Sur les résultats obtenus avec Weka

```
=== Confusion Matrix ===

 a  b  c   <-- classified as
54  4 46 |  a = pop
 4 76 19 |  b = rap
33  1 79 |  c = rnb
```

10-fold Cross Validation

```
=== Confusion Matrix ===

 a  b  c   <-- classified as
12  0  9 |  a = pop
 0 18  2 |  b = rap
 6  0 17 |  c = rnb
```

Held-Out 80%/20%

Sous Weka, le RandomForest donne une classification beaucoup plus équilibrée qu'en Python. La détection de la pop est beaucoup mieux avec 10-fold CV et 80/20 Held-Out. Tous les trois montrent un très bon rappel rap. On voit une baisse du rappel et la détection de R&B comparée au python mais le calcul reste satisfaisant.

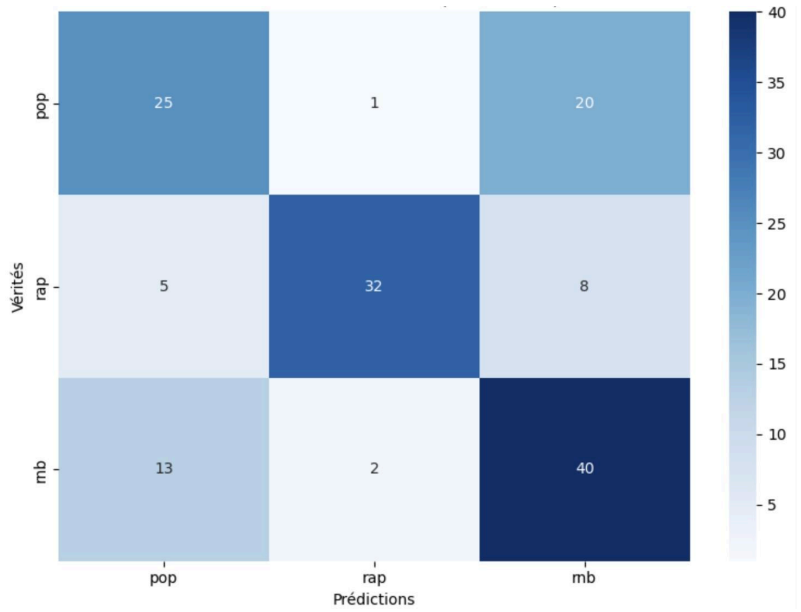
	Accuracy	F_1-score pondéré	Rappel pop	Rappel rap	Rappel R&B
Python	60 %	0,53	0,07	0,76	0,93
Weka CV	64 %	0,65	0,52	0,77	0,70
Weka 80/20	65 %	0,66	0,57	0,80	0,77

En résumé, le RF de Weka sacrifie un peu la performance sur le R&B mais finit par être plus équilibré sur les trois genres.

➢ SVM

- Sur les résultats obtenus avec Python

Matrice de confusion obtenue avec l'algorithme SVM en Python



Le modèle SVM présente une particularité. Il reconnaît bien les chansons pop (25 sur 46) et montre une bonne capacité à classifier les chansons rap (32 sur 45). Comme pour Naive Bayes, la principale source d'erreurs se situe entre pop et R&B, avec 20 chansons pop classées comme R&B, et 13 chansons R&B prédites comme pop. Toutefois, la séparation entre le rap et les deux autres genres est bien plus nette qu'avec Naive Bayes. Ce résultat visible dans la matrice de confusion confirme que le SVM, surtout avec un noyau linéaire, est bien adapté aux représentations TF-IDF, car il exploite efficacement la géométrie de l'espace vectoriel pour tracer des frontières de décision claires.

- Sur les résultats obtenus avec Weka

```
=== Confusion Matrix ===

  a  b  c   <-- classified as
 59  1 44 |  a = pop
  2 96  1 |  b = rap
 26  1 86 |  c = rnb
```

10-fold Cross Validation

```
=== Confusion Matrix ===

  a  b  c   <-- classified as
 11  1  9 |  a = pop
  0 20  0 |  b = rap
  6  0 17 |  c = rnb
```

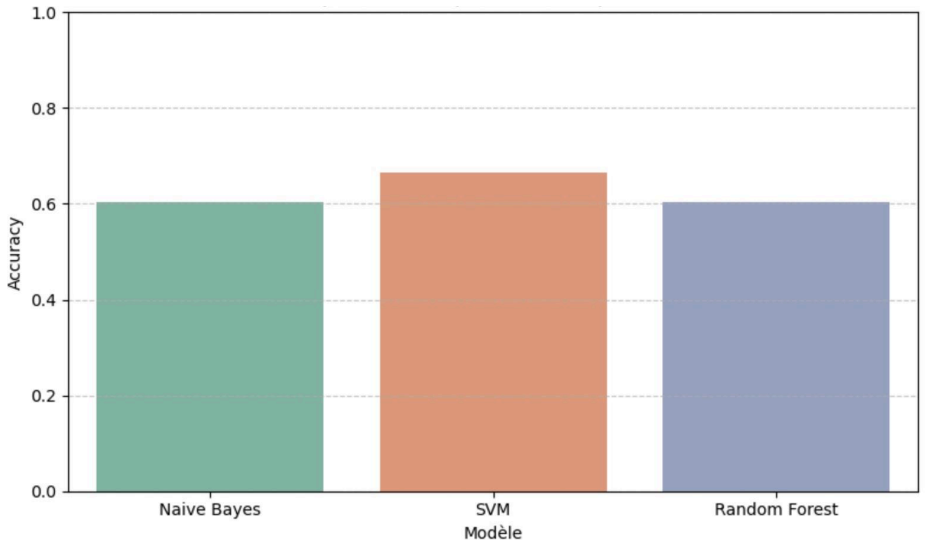
Held-Out 80%/20%

Pour Weka 10-fold, le SVM montre d'excellentes performances sur le rap (96/99) et le rnb (86/113). La pop reste plus difficile (59/104), avec 44 chansons confondues en rnb. Weka – Held-Out montre une baisse. Le rap est parfaitement classifié (20/20), la pop l'est modérément (11/21) et le rnb assez bien (17/23). La confusion entre pop et rnb persiste, qui montre la frontière parfois floue entre ces deux genres.

❖ Évaluation des performances des modèles de calcul

➢ Pour les algorithmes Python

Graphique d'évaluation des performances des algorithmes Python



Les résultats montrent que le modèle SVM obtient la meilleure performance globale avec une précision de 66 %, suivi de très près par Random Forest et Naive Bayes, tous deux autour de 60%. Ces scores sont confirmés à la fois par les rapports de classification et le graphique de comparaison des précisions. La différence entre les modèles est modérée mais nette, avec SVM en tête.

Naive Bayes

```
>>> Naive Bayes best parms: {'alpha': 0.1}
 Naive Bayes (mais mieux)
Accuracy : 0.60
              precision    recall  f1-score   support

         pop       0.65      0.24      0.35        46
         rap       0.83      0.64      0.72        45
         rnb       0.51      0.87      0.64        55

    accuracy                           0.60       146
   macro avg       0.66      0.59      0.57       146
weighted avg       0.65      0.60      0.58       146
```

Naive Bayes obtient une précision de 60 %. Il présente de bons résultats sur la classe rap avec un rappel de 0.64 et une précision de 0.83, mais il sous-performe sur la classe pop, avec un rappel très faible de 0.24. La classe rnb est relativement bien reconnue avec un rappel de 0.87, mais une précision plus faible (0.51), ce qui indique une tendance à surclasser en rnb. Le F1-score global (pondéré) est de 0,58, ce qui confirme une performance correcte mais limitée.

Arbre de décision

```
>>> Random Forest best parms: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 200}
 Random Forest (mais mieux)
Accuracy : 0.60
              precision    recall  f1-score   support

         pop       0.60      0.07      0.12        46
         rap       0.85      0.76      0.80        45
         rnb       0.50      0.93      0.65        55

    accuracy                           0.60       146
   macro avg       0.65      0.58      0.52       146
weighted avg       0.64      0.60      0.53       146
```

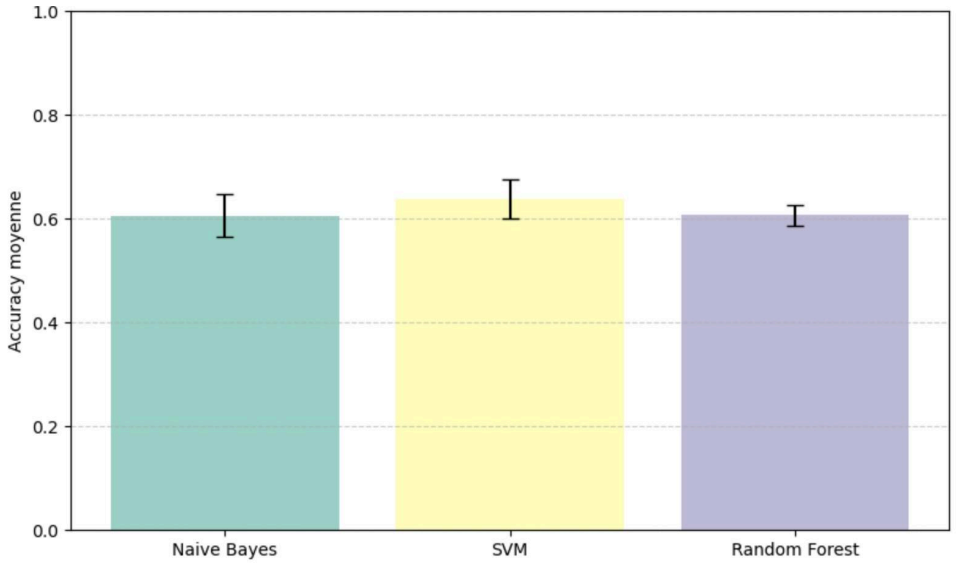
Random Forest atteint lui aussi une précision de 60 %, mais avec une répartition des performances très déséquilibrée. La classe rap est très bien reconnue avec un rappel de 0,76 et une précision de 0,85. La classe rnb est également bien identifiée avec un rappel de 0.93, mais une précision plus faible de 0.50, ce qui a traduit une surclassification vers cette classe. En revanche, la classe pop est presque ignorée, avec un rappel de 0,07 malgré une précision artificiellement élevée de 0,60. Cela indique que le modèle ne reconnaît presque jamais correctement les exemples pop, ce qui dégrade fortement la performance globale et empêche une classification équilibrée. Le score F1 pondéré est de 0,53, inférieur aux deux autres modèles.

SVM

```
>>> SVM best parms: {'C': 1, 'kernel': 'linear'}
 SVM (mais mieux)
Accuracy : 0.66
              precision    recall  f1-score   support

         pop       0.58      0.54      0.56        46
         rap       0.91      0.71      0.80        45
         rnb       0.59      0.73      0.65        55

    accuracy                           0.66       146
   macro avg       0.69      0.66      0.67       146
weighted avg       0.69      0.66      0.67       146
```

SVM atteint une précision de 66 %. Il améliore nettement la classification du genre pop par rapport à Naive Bayes, avec un rappel de 0.54 et une précision de 0.58. La classe rap est bien reconnue, avec un rappel élevé de 0,71 et une précision de 0,91, ce qui en fait la classe la mieux maîtrisée par ce modèle. Le genre obtient également de bons résultats, avec un rappel de 0,73 et une précision de 0,59. Le score F1 moyen pondéré atteint 0,67.

Graphique d'évaluation des performances des algorithmes Python sans random state



Le graphique montre les précisions moyennes obtenues par les trois modèles sur dix exécutions différentes du pipeline de classification, avec un découpage aléatoire des données à chaque fois. Les barres verticales indiquent l'écart-type des performances, c'est-à-dire la variabilité d'une course à l'autre. Ce type de graphique permet d'évaluer à la fois la performance moyenne et la stabilité des modèles.

SVM est le modèle avec la meilleure précision moyenne, aux alentours de 0.64. Il est également relativement stable, avec une variation modérée entre les pistes. Naive Bayes et Random Forest atteignent tous deux une précision moyenne proche de 0.61. L'écart-type de Naive Bayes est légèrement supérieur à celui de Random Forest, ce qui signifie qu'il est un peu plus sensible à la répartition aléatoire des données. Random Forest est le modèle le plus stable, avec la barre d'erreur la plus courte, bien que son score moyen soit légèrement inférieur à celui du SVM.

Ces résultats confirment que SVM est globalement le modèle le plus performant sur cette tâche. Il conserve une bonne précision même lorsqu'on change le découpage des données, ce qui indique une bonne capacité de généralisation. En revanche, bien que Random Forest soit très stable, sa performance plafonne à un niveau similaire à celui de Naive Bayes, ce qui limite son intérêt dans ce contexte.

➢ Pour Weka

Naive Bayes

NaiveBayesMultinomial (Weka – held-out 80/20) la précision globale est 71 % et le F_1-score pondéré est 0,69. Rap a un rappel élevé et précision parfaite (1,00, ce qui est méfiante), le modèle détecte très bien le rap. Le RnB a bon rappel mais quelques faux positifs (précision

0,59), il surclasse ou sur-prévoit le genre RnB et classe à tort certains titres non-RnB comme appartenant à ce genre. Pop reste la classe la plus délicate, avec un rappel à 0,48, signe que près de la moitié des titres pop sont confondus, même si la précision (0,56) reste correcte quand il prédit pop.

Interprétation des résultats Naive Bayes Multinomial Held-Out (Weka)

```
=== Summary ===

Correctly Classified Instances          44               68.75   %
Incorrectly Classified Instances        20               31.25   %
Kappa statistic                          0.5278
Mean absolute error                      0.2084
Root mean squared error                  0.4546
Relative absolute error                 46.9595 %
Root relative squared error             96.5216 %
Total Number of Instances               64

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.476    0.186    0.556      0.476    0.513      0.303    0.764     0.599     pop
                 0.850    0.000    1.000      0.850    0.919      0.892    0.981     0.968     rap
                 0.739    0.293    0.586      0.739    0.654      0.430    0.782     0.583     rnb
Weighted Avg.    0.688    0.166    0.705      0.688    0.690      0.533    0.838     0.709
```

Interprétation des résultats Naive Bayes Multinomial CV (Weka)

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         209               66.1392 %
Incorrectly Classified Instances       107               33.8608 %
Kappa statistic                          0.4885
Mean absolute error                      0.3673
Root mean squared error                  0.4087
Relative absolute error                 82.758  %
Root relative squared error             86.756  %
Total Number of Instances               316

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area
Class
                 0.519    0.175    0.593      0.519    0.554      0.358    0.777     0.581
pop
                 0.768    0.023    0.938      0.768    0.844      0.791    0.977     0.956
rap
                 0.699    0.320    0.549      0.699    0.615      0.365    0.777     0.633
rnb
Weighted Avg.    0.661    0.179    0.685      0.661    0.667      0.496    0.840     0.717
```

En Python, Naive Bayes “classique” affichait 66 % d’accuracy. Avec Weka Multinomial, on gagne d’accuracy et les rappels de pop et rap sont meilleurs, ce qui montre l’intérêt de l’implémentation multinomiale sur des données vectorisées. La version Multinomial de Naive Bayes (qui modélise directement les fréquences de mots dans chaque document) profite du fait que nos paroles sont déjà vectorisées. L’algorithme Multinomial est conçu pour exploiter ces comptages et retire de meilleurs résultats qu’une version naïve “classique” qui ne tient pas compte de cette structure. Finalement les résultats de validation croisée 10-fold pour NaiveBayesMultinomial. Bien qu’ils soient légèrement inférieurs en général à ceux de la

répartition 80 %/20 %, ils confirment la stabilité du modèle et on les inclut parce qu'il fournit une évaluation différente de la généralisation étant donné les jeux de test tournants.

Arbre de décision

Interprétation des résultats Random Forest Held-Out (Weka)

```
=== Summary ===

Correctly Classified Instances          47               73.4375 %
Incorrectly Classified Instances        17               26.5625 %
Kappa statistic                          0.5991
Mean absolute error                      0.3606
Root mean squared error                  0.4023
Relative absolute error                 81.2676 %
Root relative squared error             85.4182 %
Total Number of Instances               64

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.571    0.140    0.667      0.571    0.615      0.451    0.757     0.619     pop
                 0.900    0.000    1.000      0.900    0.947      0.928    0.992     0.985     rap
                 0.739    0.268    0.607      0.739    0.667      0.455    0.820     0.619     rnb
Weighted Avg.    0.734    0.142    0.749      0.734    0.738      0.602    0.853     0.733
```

Interprétation des résultats RandomForest CV (Weka)

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         209               66.1392 %
Incorrectly Classified Instances       107               33.8608 %
Kappa statistic                          0.4885
Mean absolute error                      0.3673
Root mean squared error                  0.4087
Relative absolute error                 82.758  %
Root relative squared error             86.756  %
Total Number of Instances              316

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.519    0.175    0.593      0.519    0.554      0.358    0.777     0.581     pop
                 0.768    0.023    0.938      0.768    0.844      0.791    0.977     0.956     rap
                 0.699    0.320    0.549      0.699    0.615      0.365    0.777     0.633     rnb
Weighted Avg.    0.661    0.179    0.685      0.661    0.667      0.496    0.840     0.717
```

Le score en held-out est un peu plus élevé que celui en CV, ce qui peut s'expliquer par un jeu de test fixe (plus proche du train) contre une vraie rotation des données en CV. Le CV donne généralement une estimation plus robuste de la généralisation. En somme, RandomForest dans Weka produit peu d'erreurs, surtout pour le rap et le rnb, et un peu plus pour la pop.

SMO

Dans Weka, l'algorithme SVM est implémenté par SMO (Sequential Minimal Optimization) avec un noyau linéaire par défaut. Il cherche un hyperplan droit dans l'espace des mots pour séparer les genres et non par le produit scalaire de leurs vecteurs de caractéristiques comme

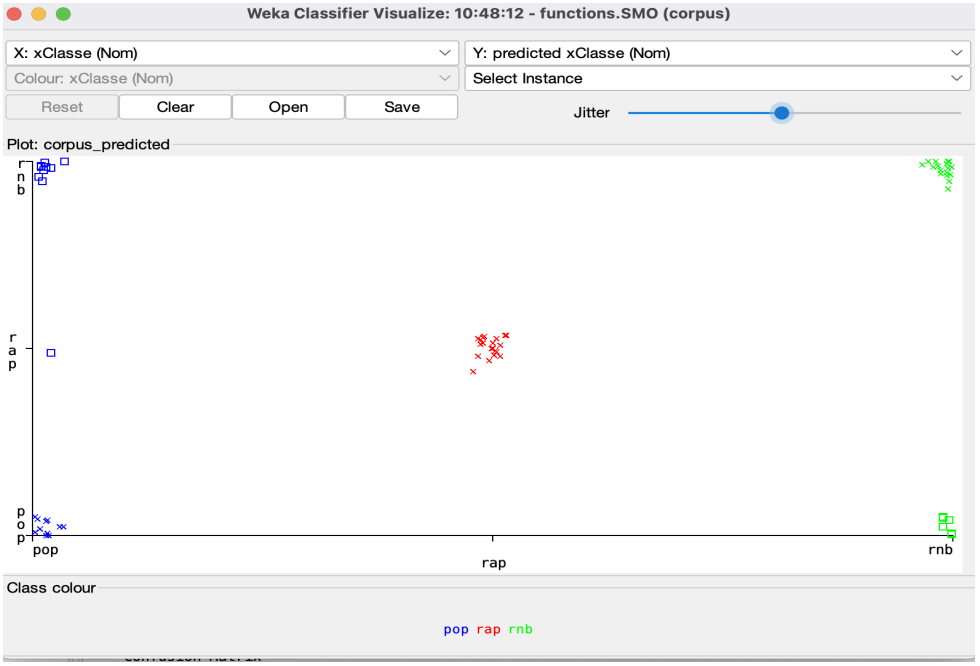
SVM. SMO normalise automatiquement chaque attribut avec une régularisation standard (C=1.0 par défaut) pour une meilleure généralisation. Ces différences de noyau et de normalisation peuvent expliquer en partie pourquoi les scores SMO (75 % précision) et SVM (69 % en Python) ne sont pas identiques.

Interprétation des résultats SMO Held-Out (Weka)

```
=== Summary ===

Correctly Classified Instances          48               75      %
Incorrectly Classified Instances        16               25      %
Kappa statistic                          0.6237
Mean absolute error                      0.2812
Root mean squared error                  0.3648
Relative absolute error                 63.3803 %
Root relative squared error             77.4581 %
Total Number of Instances               64

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.524    0.140    0.647      0.524    0.579      0.409    0.733     0.519     pop
               1.000    0.023    0.952      1.000    0.976      0.965    0.989     0.952     rap
               0.739    0.220    0.654      0.739    0.694      0.508    0.788     0.592     rnb
Weighted Avg.  0.750    0.132    0.745      0.750    0.744      0.618    0.833     0.681
```

Le SMO (SVM) se distingue par une performance globale solide (75 % de précision, F_1 = 0,74 moyenne pondérée), et particulièrement sur la classe rap. Il gomme en grande partie les erreurs massives de ZeroR et LazyIBK (ces scores ne valent pas la peine d'être inclus), et surpasse Naive Bayes Multinomial, grâce à sa capacité à trouver la limite de décision la plus centrale possible. Les légères confusions entre pop et rap indiquent qu'un affinage du prétraitement (par exemple suppression de mots fréquents ambigus) pourrait encore améliorer les résultats.

Nuage de Points SMO (Weka) Held-Out



Ce nuage de points est une visualisation des instances réelles (axe X) vs. leurs prédictions (axe Y) sous SMO (SVM) dans Weka, avec un peu de « jitter » pour séparer les points qui superpose. On voit une diagonale idéale et trois genres distincts. En bas à gauche, les croix bleues (pop) prédites comme pop. Au centre, les croix rouges (rap) prédites comme rap. En haut à droite, les croix vertes (rnb) prédites comme rnb.Quelques bleus apparaissent à gauche (rnb classé pop), et quelques verts dans l'autre sens (à droite, pop classé rnb) ce sont les cas mal classés. Même avec ces erreurs on peut dire qu'ils sont en faible nombre et que SMO se trompe moins que les autres.

❖ Conclusion

Pour conclure ce rapport nous pouvons dire que Python et Weka peuvent tous deux être efficaces dans le cadre de la classification de chansons par genre cependant Weka permet d'obtenir des résultats plus fiables sur certains calculs (gain d'accuracy pour Naive Bayes, et peu d'erreurs produites avec RandomForest par exemple). Aussi est-il intéressant de noter cela car il peut donc être plus intéressant de se tourner vers Weka lors d'une tâche de classification telle que la nôtre plutôt que vers un algorithme Python.

Weka facilite l'expérimentation rapide (CV, held-out, visualisations), tandis que Python offre plus de flexibilité pour le prétraitement, la recherche d'hyperparamètres et la création de pipelines reproductibles.

Les expériences menées dans Weka confirment plusieurs tendances observées avec Python. Comme la confusion entre pop et R&B. Pop et R&B partagent un lexique et des thématiques proches donc on voit de nombreuses chansons pop mal classées en R&B et vice-versa. Cette

confusion persiste tant en 10-fold CV qu'en held-out 80/20 en Weka, ce qui montre que les différences lexicales entre ces deux genres reste floue pour un classifieur basé uniquement sur la présence ou absence de mots. Le rap est systématiquement la classe la mieux car son vocabulaire (argot, références) est plus spécifique et moins partagé avec les deux autres genres. Quel que soit l'outil, SVM linéaire est la meilleure approche sur nos données TF-IDF.

L'utilisation du modèle TF-IDF avec des unigrammes montre ses limites. Il ne prend pas en compte l'ordre des mots, la syntaxe ou le sens des phrases. Cela réduit la qualité des informations transmises aux modèles, ce qui peut limiter leurs performances.

En ce qui concerne l'impact du protocole d'évaluation, la validation croisée à 10 plis offre des scores un peu plus élevés et stables que le held-out 80/20, car chaque document de test passe un tour dans l'ensemble d'entraînement. Le held-out donne une évaluation plus « réaliste » sur des données strictement nouvelles. Nous avions initialement pensé à classifier les chansons en pop, rock et rap, mais nous avons finalement choisi le R&B précisément pour rendre la séparation plus difficile. Il est certain que le rock aurait peut-être été plus simple à distinguer, mais nos résultats restent néanmoins satisfaisants, même face à ce défi supplémentaire.

Au cours de ce projet nous avons rencontré quelques difficultés, une erreur time out a au départ faussé le nombre de textes obtenus par genres, nous avions ainsi 485 fichiers textes, 151 pour le genre rap, 151 pour la pop et 183 pour le R&B. Ce nombre de textes ne nous permettant pas d'avoir une égalité pour chaque genre dans le corpus, il nous a paru plus judicieux de rééquilibrer le corpus en effaçant des textes pour le R&B. Cependant, il semblait que malgré cette inégalité cela n'influence pas les résultats au final.

Pour améliorer les résultats, plusieurs solutions peuvent être mises en place. Appliquer une lemmatisation permettrait de regrouper les variantes d'un même mot. L'ajout de n-grammes permettrait de capturer des expressions typiques propres à chaque genre. Il serait aussi utile d'adapter la liste de stopwords au vocabulaire spécifique des paroles de chansons pour mieux filtrer le bruit. Utiliser des représentations sémantiques plus riches, comme des embeddings de mots ou des modèles préentraînés comme BERT, permettrait de mieux saisir le sens des textes. Enfin, équilibrer le corpus sur la longueur et la richesse lexicale, et activer la pondération des classes dans les modèles, aiderait à limiter le surapprentissage, en particulier chez Random Forest. En combinant ces pistes (prétraitement, enrichissement des caractéristiques, ajustement du jeu de données), on pourrait peut-être réduire la confusion pop–R&B et augmenter globalement la précision du modèle.

- ❖ Bibliographie

https://docs.genius.com/

https://waikato.github.io/weka-wiki/documentation/

https://scikit-learn.org/stable/user_guide.html

Isabelle Tellier & Loïc Grobol (crédits cours et slides), Yoann Dupont, Introduction à la fouille de textes Cours 8 - Arbres de décision, (2024-2025)

Isabelle Tellier & Loïc Grobol (crédits cours et slides), Yoann Dupont, Introduction à la fouille de textes Cours 9 - Naïve Bayes, (2024-2025)

Isabelle Tellier & Loïc Grobol (crédits cours et slides), Yoann Dupont, Introduction à la fouille de textes Cours 10 - SVM, (2024-2025)

Lien git vers le projet : https://github.com/mickaela-mstr/Fouille_de_texte

❖ Annexes

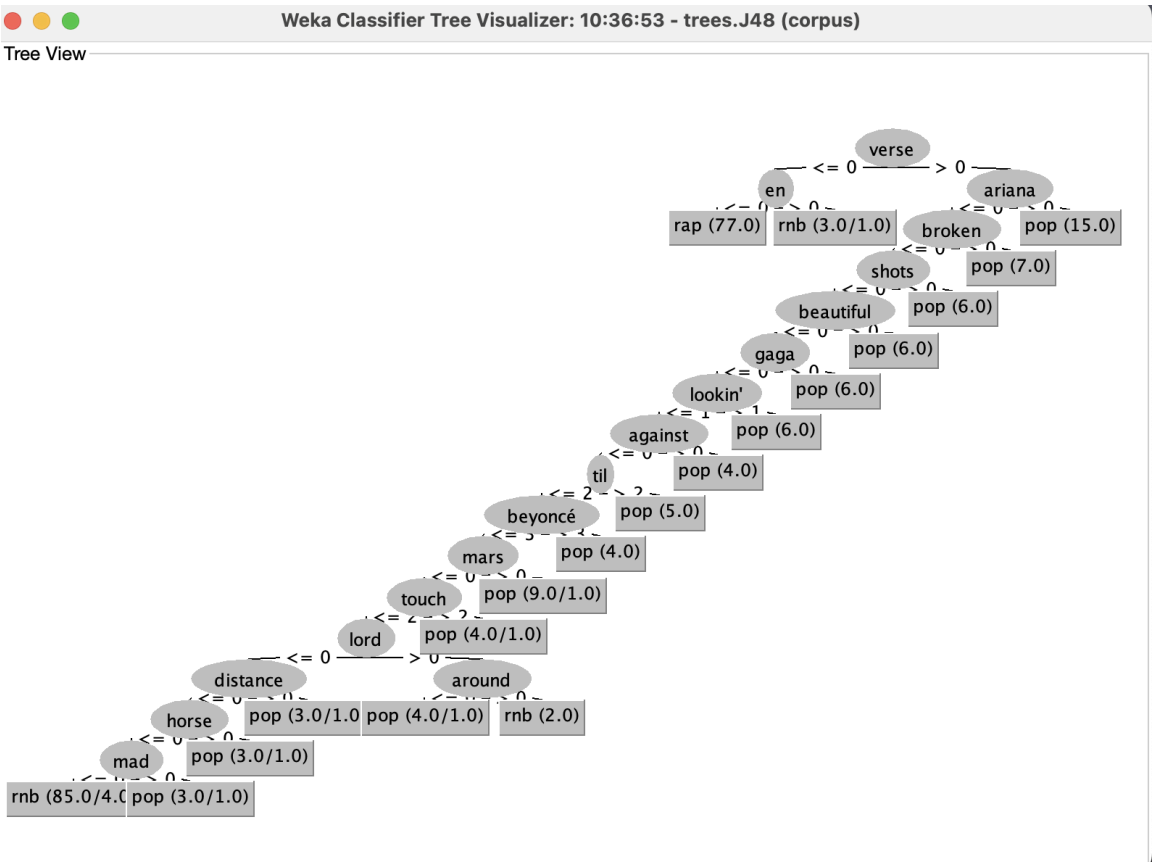
ZeroR visuel Held-Out (Baseline)

```
=== Summary ===

Correctly Classified Instances          23               35.9375 %
Incorrectly Classified Instances        41               64.0625 %
Kappa statistic                          0
Mean absolute error                      0.4437
Root mean squared error                  0.471
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances               64

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    0.000    ?          0.000    ?          ?        0.500     0.328     pop
                 0.000    0.000    ?          0.000    ?          ?        0.500     0.313     rap
                 1.000    1.000    0.359      1.000    0.529      ?        0.500     0.359     rnb
Weighted Avg.    0.359    0.359    ?          0.359    ?          ?        0.500     0.334

=== Confusion Matrix ===

  a  b  c   <-- classified as
  0  0 21 |  a = pop
  0  0 20 |  b = rap
  0  0 23 |  c = rnb
```

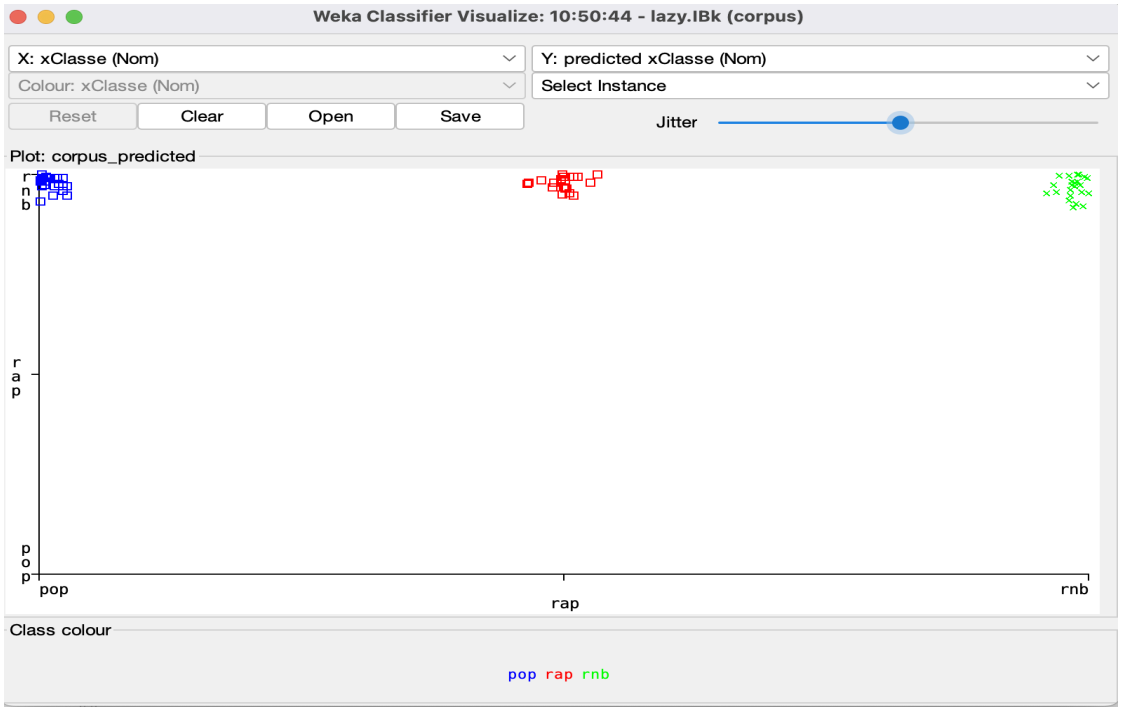
ZeroR visuel CV (Baseline)

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         113               35.7595 %
Incorrectly Classified Instances       203               64.2405 %
Kappa statistic                          0
Mean absolute error                      0.4438
Root mean squared error                  0.4711
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances              316

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    0.000    ?          0.000    ?          ?        0.481     0.320     pop
                 0.000    0.000    ?          0.000    ?          ?        0.491     0.309     rap
                 1.000    1.000    0.358      1.000    0.527      ?        0.486     0.351     rnb
Weighted Avg.    0.358    0.358    ?          0.358    ?          ?        0.486     0.327

=== Confusion Matrix ===

   a   b   c   <-- classified as
   0   0 104 |   a = pop
   0   0  99 |   b = rap
   0   0 113 |   c = rnb
```

Arbre de décision J48 Held-Out visuel (Weka)



Nuage de points Lazy IBK Held-Out (Weka)



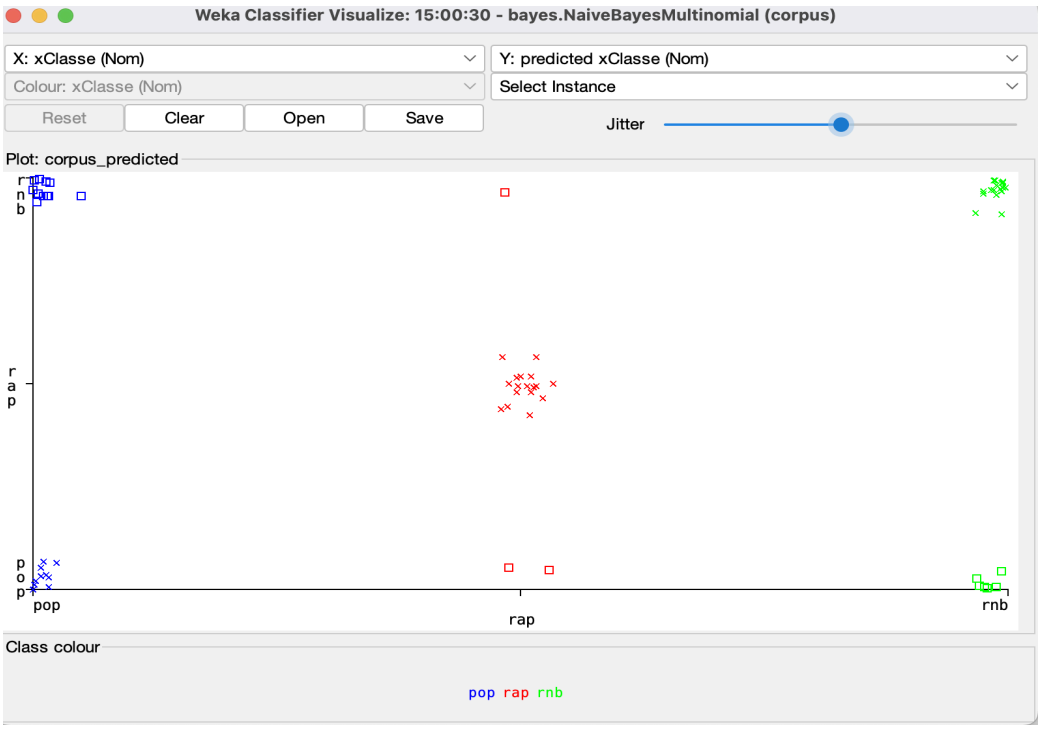
Résumé de la classification LazyIBK Held-Out (Weka)

```
=== Summary ===

Correctly Classified Instances          23               35.9375 %
Incorrectly Classified Instances        41               64.0625 %
Kappa statistic                          0
Mean absolute error                      0.4273
Root mean squared error                  0.6497
Relative absolute error                 96.2902 %
Root relative squared error            137.9358 %
Total Number of Instances               64

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    0.000    ?          0.000    ?          ?        0.500     0.328     pop
                 0.000    0.000    ?          0.000    ?          ?        0.500     0.313     rap
                 1.000    1.000    0.359      1.000    0.529      ?        0.500     0.359     rnb
Weighted Avg.    0.359    0.359    ?          0.359    ?          ?        0.500     0.334

=== Confusion Matrix ===

  a  b  c   <-- classified as
  0  0 21 |  a = pop
  0  0 20 |  b = rap
  0  0 23 |  c = rnb
```

Résumé de la classification LazyIBK CV (Weka)

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         115               36.3924 %
Incorrectly Classified Instances       201               63.6076 %
Kappa statistic                          0.0106
Mean absolute error                      0.4243
Root mean squared error                  0.6478
Relative absolute error                 95.5951 %
Root relative squared error            137.5127 %
Total Number of Instances              316

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.010    0.005    0.500      0.010    0.019      0.029    0.498     0.329     pop
                 0.020    0.000    1.000      0.020    0.040      0.118    0.510     0.329     rap
                 0.991    0.985    0.359      0.991    0.527      0.025    0.498     0.356     rnb
Weighted Avg.    0.364    0.354    0.606      0.364    0.207      0.056    0.502     0.339

=== Confusion Matrix ===

   a   b   c   <-- classified as
   1   0 103 |   a = pop
   0   2  97 |   b = rap
   1   0 112 |   c = rnb
```

Nuage de points NativeBayesMultinomial Held-Out (Weka)



Nuage de points RandomForest Held-Out (Weka)