Classification de musiques par genre

Elise CHIN, Télio CROPSAL

24 avril 2020

Matière : Projet analyse de données
Enseignant : Nicolas BORIA

1. Introduction

L'idée de départ était de trouver un jeu de données qui soit à la fois intéressant pour nous tout en étant un défi accessible avec nos compétences.

L'envie de travailler autour de la musique fit son apparition assez vite car comme la grande majorité des humains nous apprécions la musique et nous étions curieux de voir comment la machine dénuée de sensibilité artistique pourrait appréhender ces données.

De plus avoir comme sujet d'étude des titres musicaux permet d'aborder le processus d'un projet d'analyse des données de A à Z. En effet il faut disposer d'un ensemble de musiques, extraire de nouvelles données par-dessus, c'est à dire quantifier les musiques, puis écrire et comparer les performances de différents algorithmes de classification et enfin créer une application concrète de son travail avec un classificateur pour une musique quelconque d'un utilisateur.

Ce dernier point est très important dans notre choix car beaucoup d'autres jeux de données disponibles sur Internet ne permettent pas facilement de tester son projet avec de nouvelles données. Par exemple pour le jeu de données sur les fleurs d'Iris il faudrait en trouver une à l'extérieur et prendre ces mesures, pour les musiques il suffit d'en télécharger des nouvelles.

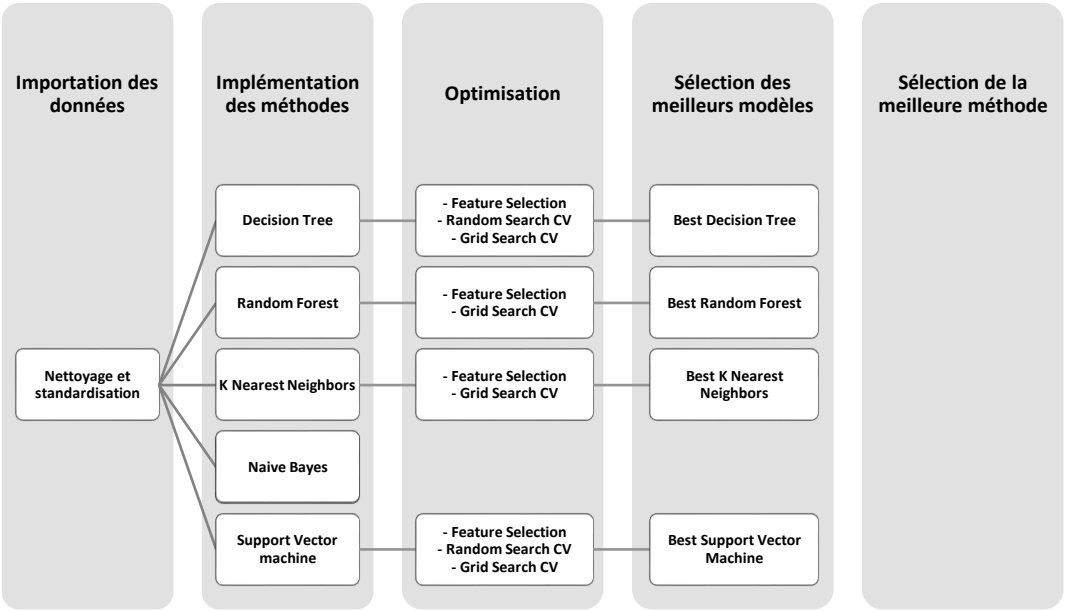
Nous nous sommes donc orientés vers la classification des musiques par genre après une hésitation avec la classification des musiques traditionnels par pays et ceci pour deux raisons : la première est que nous avions trouvé un jeu de données brut avec seulement les musiques et non avec l'extraction déjà faite, ce qui nous obligeaient à utiliser à la place de Librosa, MARSYAS une librairie peu adaptée à Python. La seconde est que la palette des musiques touchées est plus grande et plus proche de nos préférences musicales.

2. Présentation du projet

La classification de musiques par genre n'est pas un récent problème d'apprentissage automatique. De nombreuses personnes se sont déjà penchées sur la question et ont implémenté des algorithmes diverses et variées.

Notre angle d'attaque

Nous avons choisi pour ce projet d'implémenter uniquement des méthodes de classification que nous avons vu en cours et d'autres algorithmes que nous estimions accessibles.



3. Jeu de données

Le jeu de données que nous avons choisi est GTZAN de G. Tzanetakis and P. Cook publié en 2002. Il est constitué de dix genres de musiques : blues, classique, country, disco, hiphop, jazz, métal, pop, reggae et rock. Pour chaque genre on dispose de 100 extraites de 30 secondes. C'est donc un petit jeu de données en comparaison avec d'autres.

3.1 Extraction des caractéristiques

Pour extraire les données des musiques, on utilise Librosa, une librairie pour faire du traitement audio avec Python.

3.2 Sélection de caractéristiques

La question que l'on peut désormais se poser est la suivante : quelles sont les informations les plus pertinentes pour la création d'un modèle prédictif ? Le processus qui permet de trouver ces caractéristiques s'intitule la sélection de caractéristiques, ou sélection d'attribut. Cela consiste, étant donné des données dans un espace de grande dimension, à déterminer un sous-ensemble de caractéristiques pertinentes pour un problème donné. L'objectif est multiple. En effet, en supprimant les attributs redondants ou qui ne contribuent pas à la précision d'un modèle, la durée d'apprentissage est d'une part réduite et d'autre part il en résulte un modèle plus "simple". Il permet également de diminuer le surapprentissage.

Il existe trois types d'approches : par filtrage, par emballage et par intégration. Nous nous intéressons ici à la **méthode** dite **par emballage**. Contrairement à l'approche par filtrage qui consiste à présélectionner les caractéristiques indépendamment de l'algorithme d'apprentissage qui va être appliqué par la suite, la méthode par emballage utilise le modèle prédictif pour estimer la précision d'un sous-ensemble de caractéristiques. Ainsi l'importance d'une caractéristique est calculée à partir d'un modèle.

Algorithme de calcul de l'importance des caractéristiques (1)

On entraîne tout d'abord le modèle sur l'ensemble d'apprentissage, puis on effectue les prédictions sur le jeu test et on calcule l'erreur entre les estimations et les réels labels. Cette valeur constitue l'erreur résultante des données tests sans modification des colonnes. Ensuite, pour chaque colonne du jeu de données, on permute ses valeurs, on effectue une nouvelle prédiction et on calcule la variation de l'erreur en comparaison de celle calculée avant permutation de la colonne. Plus la différence est grande, plus la caractéristique associée à la colonne est importante. En effet, intuitivement si une colonne est importante lors de l'étape de prédiction, alors la permutation des valeurs de la colonne devrait entraîner une augmentation de l'erreur et vice versa.

4. Optimisation des hyperparamètres

Nous avons procédé pour chacune des méthodes décrites dans la section suivante, à une recherche des meilleures combinaisons d'hyperparamètres (2) afin d'améliorer la performance de nos modèles.

L'optimisation peut se décomposer en deux étapes :

i. Recherche randomisée d'hyperparamètres (*Random Search Cross Validation*)
ii. Recherche exhaustive d'hyperparamètres (*Grid Search Cross Validation*)

Selon la méthode, la première étape a été supprimée car jugée non nécessaire.

4.1 K-fold Cross Validation

Dans les deux étapes, on évalue notre modèle à l'aide d'une des méthodes les plus courantes de validation croisée, la ***k-fold cross validation***. Celle-ci consiste à diviser l'échantillon original en k échantillons, appelés *fold.* Chaque fold est ensuite considéré comme ensemble de validation et les k-1 autres échantillons constitueront l'ensemble d'apprentissage. On calcule ensuite le score de performance, puis on répète l'opération en sélectionnant un autre échantillon de validation parmi les k-1 échantillons qui n'ont pas encore été utilisés pour la validation du modèle. L'opération se répète ainsi k fois pour qu'en fin de compte chaque échantillon ait été utilisé exactement une fois comme ensemble de validation. La moyenne des k scores est enfin calculée pour le score final du modèle.

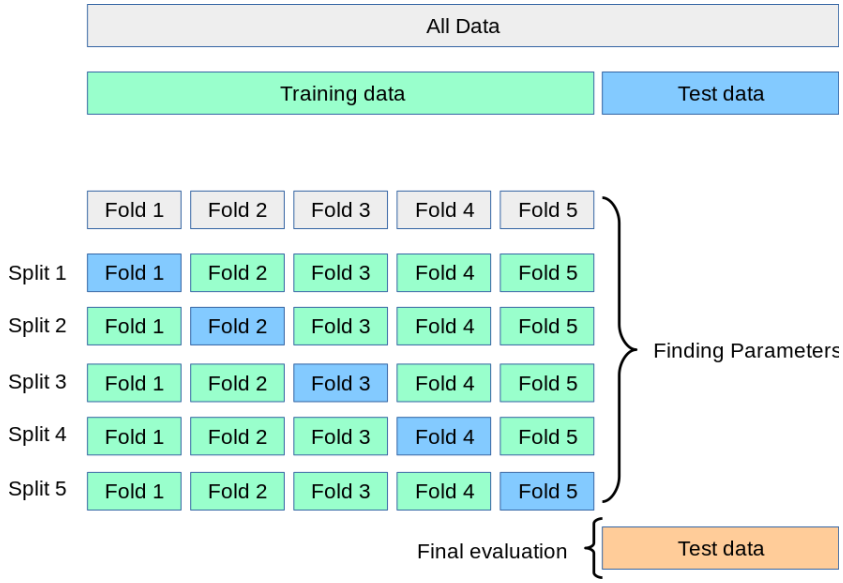


Schéma 1. *Principe du k-fold cross validation (source : scikit-learn).*

4.2 Random Search Cross Validation

Généralement, nous n'avons qu'une vague idée des valeurs des meilleurs hyperparamètres. Afin de restreindre notre recherche, on évalue donc un large choix de combinaisons qui ont été formées en tirant aléatoirement les valeurs dans une grille d'hyperparamètres. Le nombre de combinaisons à tester est défini à l'avance et l'évaluation se fait à l'aide d'une *k-fold cross validation*.

Nous avons codé notre propre classe `RandomizedSearchCV`, sur le modèle de celle de Scikit-learn. L'algorithme se définit comme suit :

Pour chaque itération
1. Choix aléatoire d'une combinaison d'hyperparamètres
2. Instanciation du modèle avec ces hyperparamètres
3. Application du k-fold cross validation sur l'ensemble d'apprentissage
4. Sauvegarde du score moyen du modèle

Comparaison des scores et sélection du meilleur modèle

4.3 Grid Search Cross Validation

La recherche randomisée a permis de réduire les intervalles de recherche pour chaque hyperparamètre. Nous pouvons donc désormais effectuer une recherche plus exhaustive en spécifiant l'ensemble des combinaisons d'hyperparamètres à essayer : c'est la *Grid Search Cross Validation*. On procède de la même manière que précédemment mais au lieu de choisir aléatoirement une combinaison dans une grille d'hyperparamètres, on évalue toutes les combinaisons qui puissent être créer à partir de cette grille.

De même, nous avons codé notre propre classe `GridSearchCV`. L'algorithme se définit comme suit :

Pour chaque combinaison de d'hyperparamètres :
1. Instanciation du modèle avec ces hyperparamètres
2. Application du k-fold cross validation sur l'ensemble d'apprentissage
3. Sauvegarde du score moyen du modèle

Comparaison des scores et sélection du meilleur modèle

Finalement à l'issue de cette procédure, nous espérons obtenir un meilleur modèle.

5. Méthodes

5.1 Arbre de décision

Notre première approche fut l'utilisation d'un arbre de décision. Il a l'avantage d'être simple à implémenté, d'être lisible et plus ou moins rapide à exécuter. Chaque nœud de l'arbre consiste en une question du vecteur d'entrée et les différentes classes possibles du problème sont situées aux extrémités des branches, ou feuilles de l'arbre. Suivant l'entrée du problème, on parcourt l'arbre de décision jusqu'à arriver à une feuille, qui correspond à une prédiction sur la classe de l'entrée. Voici un exemple :

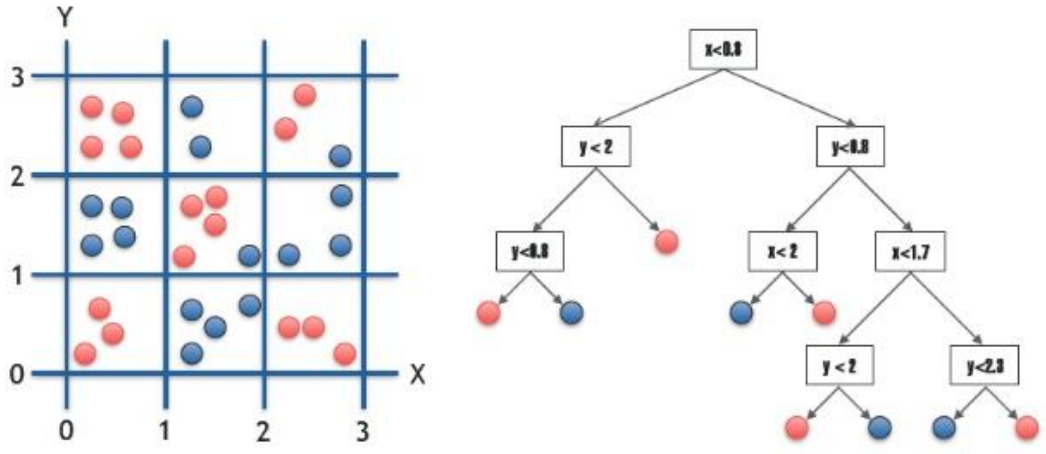


Schéma 2. Un arbre de décision permettant de classifier les boules rouges et bleues (source : slideshare)

Notre modèle a trois paramètres que nous avons optimisé :
- `n_cuts` -- le nombre de coupes à tester pour trouver la meilleure
- `max_depth` -- la profondeur maximale de chaque arbre
- `max_features` -- le nombre de caractéristiques à choisir lors de l'entraînement du modèle

5.2 Forêt aléatoire

La seconde approche est la suite logique de l'arbre de décision puisqu'il s'agit non pas d'élaborer un classifieur à partir d'un unique arbre mais de multiples arbres de décision entrainés sur des différents sous-ensembles de données du jeu de départ. Plus précisément :

Pour chaque arbre que l'on souhaite construire :

1. Tirage sans remise d'un nombre `n_samples` d'observations dans le jeu de données initial
2. Sur les 30 caractéristiques, ne retenir qu'un nombre `max_features`
3. Entraînement d'un arbre de décision sur cet échantillon

La prédiction de la forêt aléatoire se base sur un vote majoritaire des prédictions de chaque arbre qui la compose.

Notre random forest a cinq paramètres, dont trois pour définir un arbre :
- `n_trees` -- le nombre d'arbres de la forêt
- `n_samples` -- le nombre de données à placer dans le noeud de chaque arbre avant qu'il ne soit partitionné
- `n_cuts` -- le nombre de coupes à tester pour trouver la meilleure
- `max_depth` -- la profondeur maximale de chaque arbre
- `max_features` -- le nombre de caractéristiques à choisir lors de l'entraînement du modèle

5.3 Méthode des k plus proches voisins

La troisième approche est l'une des premières techniques appliquées aux problèmes de classification en raison de son accessibilité. Comme son nom l'indique, l'idée ici est de déterminer les k plus proches voisins labellisés d'une nouvelle entrée, selon une distance définie. La prédiction correspond alors à la classe la plus représentée parmi les voisins (3).

Les paramètres de la méthode sont la fonction de distance et l'ordre p si la distance choisie est celle de Minkowski. C'est cette dernière que nous avons choisie pour obtenir le meilleur modèle. La distance de Minkowski d'ordre p (où p est un entier) entre deux points x et y $\in \mathbb{R}^n$ est :

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

Si p = 1, alors elle correspond à la distance de Manhattan et si p = 2, alors c'est la distance euclidienne. Lorsque p tend vers l'infini, nous obtenons la distance de Chebyshev :

$$\lim_{p \to \infty} \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}} = \max_{i=1,\dots,n} |x_i - y_i|$$

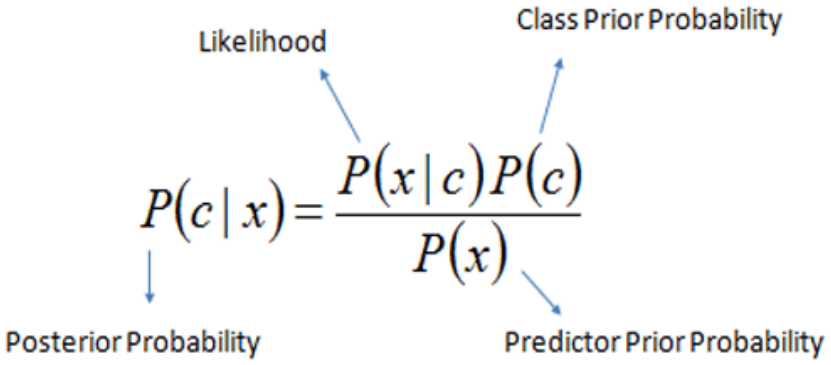
$$\lim_{p \to -\infty} \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}} = \min_{i=1,\dots,n} |x_i - y_i|$$

Dans le cadre du projet, nous nous sommes intéressés à la valeur de l'erreur de prédiction en fonction de k et de p dans {1,2, ∞}.

5.4 Bayes naïf gaussien

La quatrième approche fut d'utiliser un classifieur naïf bayésien gaussien, ce classifieur ne prend pas d'hyperparamètres.

Pour fonctionner ce classificateur suppose que les caractéristiques qui décrivent le jeu de données sont indépendantes et suivent chacune une loi normale de paramètre : la moyenne et la variance empiriques de cette même caractéristique. Il est donc possible de calculer la probabilité d'une classe (i.e. un genre), d'une caractéristique sachant une certaine valeur donnée (on connait la loi). Il en résulte qu'avec la formule de Bayes, on peut calculer pour chaque entrée dans le jeu de données, la probabilité d'appartenir à une classe sachant les données caractéristiques. Il suffit donc de prendre la probabilité de la plus grande en fonction des classes pour définir la classe de l'entrée.



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Pour ce classifieur nous avons décidé de le programmer quasiment exclusivement avec Pandas pour s'entrainer et changer de ce qu'on trouve sur Internet (4).

5.5 Machine à vecteurs de support (5)

La cinquième approche fut d'utiliser les SVM (Support Vector Machine, en français : Machine à vecteurs de support). Le SVM un algorithme de classification binaire, son objectif est de séparer l'espace des données en deux à l'aide d'un hyperplan. Or la plupart des temps il est impossible d'effectuer une séparation linéaire et on veut éviter une surinterprétation des données. Pour résoudre ce problème on introduit une notion de marge, qui autorise des erreurs mais en les pénalisant.

Le problème revient donc à minimiser cette fonction de score :
(Avec en plus une borne +b dans notre cas)

$$\min_{\mathbf{w} \in \mathbb{R}^d} C \sum_{i}^{N} \max(0, 1 - y_i f(\mathbf{x}_i)) + ||\mathbf{w}||^2$$

Pour minimiser on effectue une descente de gradient stochastique, c'est à dire une descente qui se fait sur des lots de points d'une certaine taille tirés aléatoirement (6) (7).

L'expression de l'hyperplan est :

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

Pour définir la classe, en fonction des caractéristiques d'un vecteur x, il suffit de regarder le signe de f(x).

Maintenant que le SVM fonctionne, il faut pouvoir utiliser ce classificateur binaire dans le cadre de notre problème qui est multi-classe. Pour ce faire nous avons utilisé deux approches, la première est le "Un contre tout le reste" et l'autre le "Un contre un". Pour le "Un contre tout le reste", on instancie un nombre de classifieurs égal au nombre de classe et chaque classifieur est entrainé avec une classe qui a comme label 1 et les autres à - 1.
Pour déterminer une classe, chaque entrée est testée avec chaque classificateur et celui avec la plus grande marge désigne la classe de l'entrée.
Calcul de la marge : $\frac{2}{||\mathbf{w}||}$

Pour le "Un contre un", on instancie un nombre de classificateur égal au nombre de combinaison possible entre deux classes. Puis on procède par un vote pour une nouvelle entrée, chaque classifieur votant pour la classe qui lui semble appropriée.

6. Résultats et discussions

Nous avons testé ces algorithmes pour classifier les musiques selon dix genres. L'ensemble d'apprentissage était constitué de 80 chansons de chaque genre, les 20 restants ont été utilisés pour estimer la performance du modèle. De manière générale, nous remarquons plusieurs points.

6.1 Matrices de confusion

La qualité des prédictions varie significativement en fonction des genres, par exemple le classique donne des meilleures chances de réussite que le rock. Ce qui semble logique car la musique classique est assez différente des autres types de musiques. On peut aussi noter que le rock et le blues sont assez fréquemment confondus comme le montre les matrices de confusion ci-dessous et qu'il est particulièrement difficile de classifier le rock.

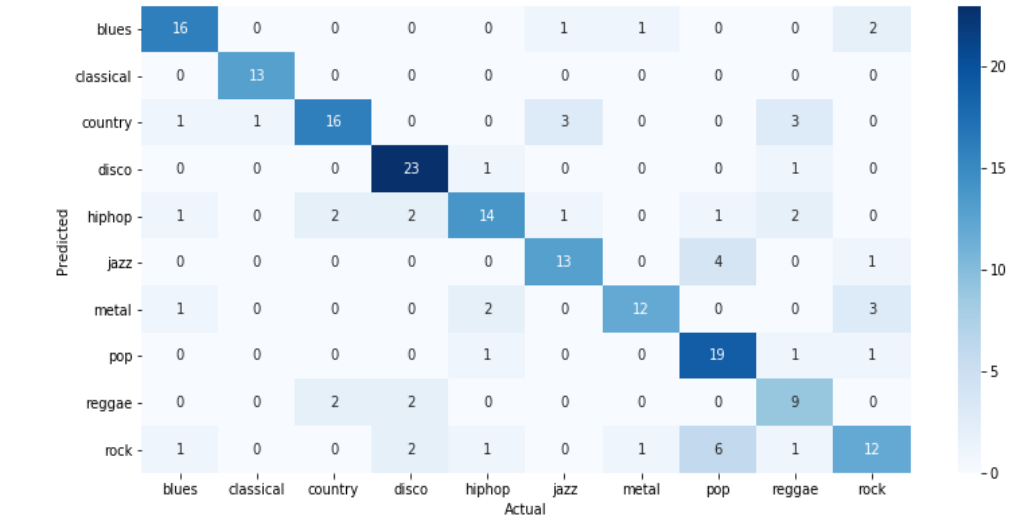


Tableau 1. Matrice de confusion de la méthode des k plus proches voisins

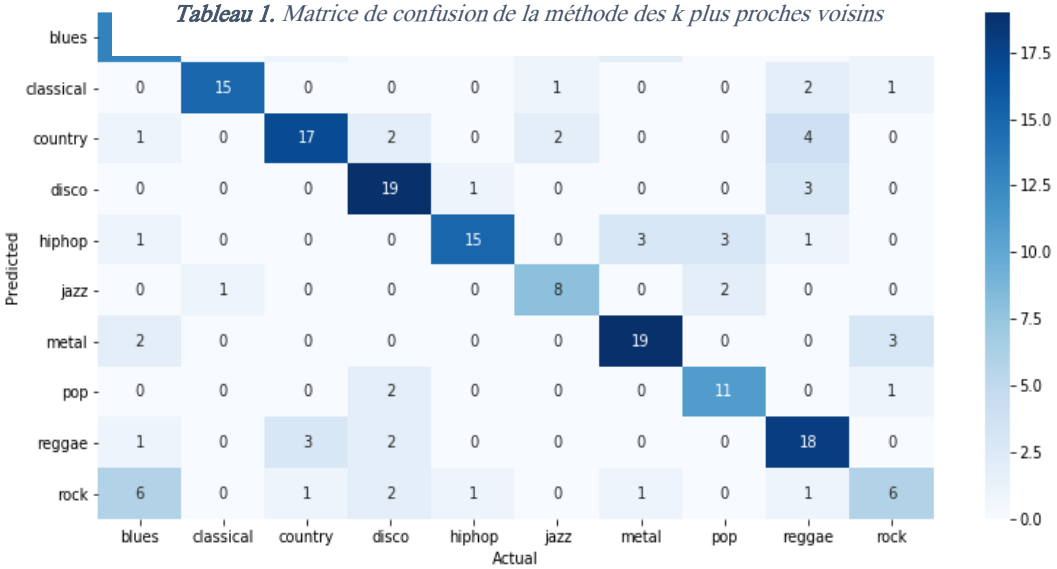


Tableau 2. Matrice de confusion de la forêt aléatoire

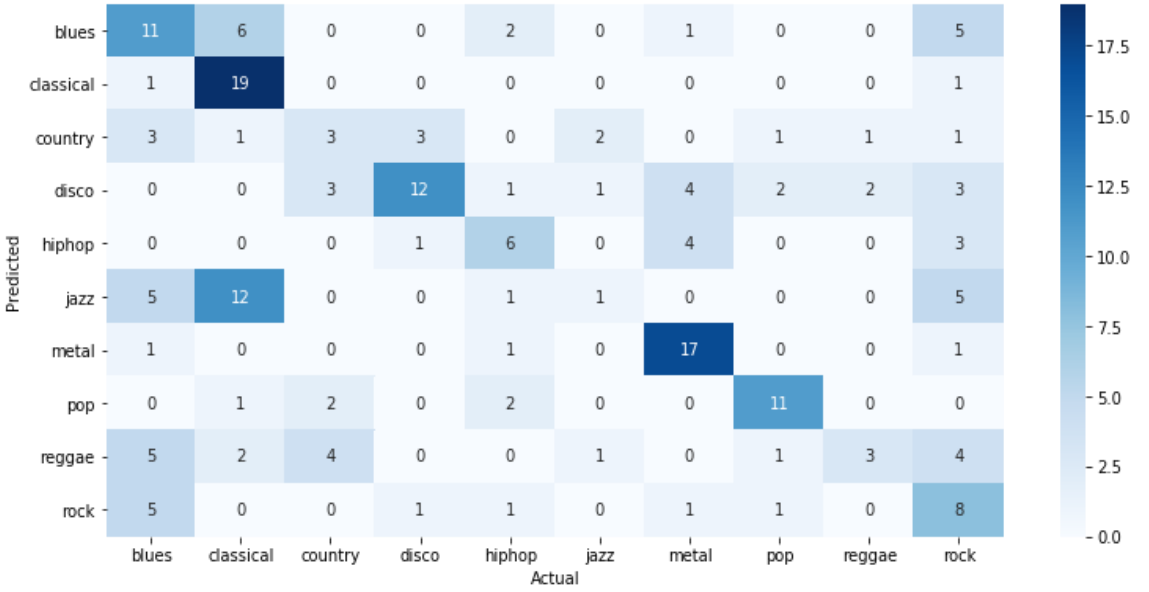


Tableau 3. Matrice de confusion du classifieur naïf bayésien

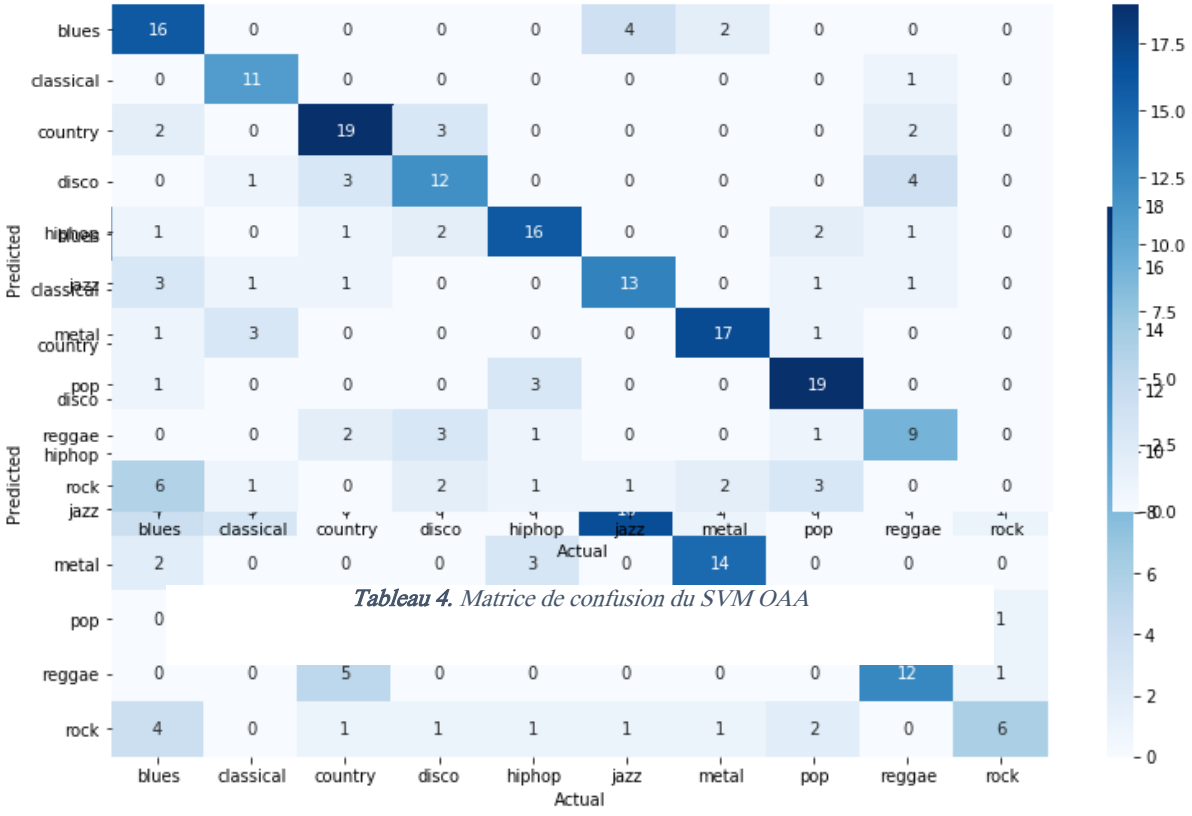


Tableau 4. Matrice de confusion du SVM OAA

Tableau 5. Matrice de confusion du SVM OVO

6.2 Performance des modèles

Voici un tableau comparatif des différents modèles :

Méthode	Taux d'exactitude (%)			Commentaires
	Modèle de base, sans SC[1]	Modèle de base, avec SC	Meilleur modèle sans SC	
Decision Tree	28,9	34,5	54	Taux est supérieur dans le cas avec SC mais tend à baisser après optimisation des hyperparamètres.
Random Forest	66	64	70,5	
kNN	67	68	73,5	
Naive Bayes	45,5	27,5	45,5	Pas d'optimisation des hyperparamètres, donc le meilleur modèle est le modèle de base sans SC.
SVM OAA	71	65,5	67,5	Le résultat peut paraître surprenant puisque nous obtenons un score inférieur pour le "meilleur" modèle. Nous obtenons déjà un bon taux avec le modèle de base car nous avions au préalable essayé exhaustivement quelques valeurs d'hyperparamètres.
SVM OVO	67	63	72	

Etonnamment, la sélection de caractéristiques n'a pas permis, dans la majorité des cas, d'améliorer les modèles comme nous l'espérions. Cela peut s'expliquer par le fait que nous nous retrouvions avec trop peu de caractéristiques (11/30 pour Random Forest par exemple) et donc trop peu d'informations qui permettraient de différencier les genres. Le nombre de caractéristiques sélectionnés doit se situer dans une fourchette acceptable, et plus particulièrement pour notre problème.

Les résultats théoriques sont assez bons avec KNN et le OVO-SVM mais en pratique il y a beaucoup de limitations. En effet dans la littérature on trouve des critiques du jeu de données, qui lui reproche une trop grande uniformité au niveau des artistes et des musiques pour un même genre (8), ce qui augmente artificiellement le score des algorithmes sans le rendre meilleur pour un usage pratique.

Toutefois, à notre surprise, tous les algorithmes sont tout de même assez efficaces malgré quelques similitudes entre certains genres. Obtenir 70% de taux de précision est une très bonne nouvelle, contre 10% au départ si nous n'avions implémenté aucune méthode. Cependant, bien qu'ils apprennent les facteurs distinctifs des genres musicaux, ils se heurtent comme nous en tant qu'être humain aux mêmes difficultés : distinguer la nature des genres musicaux qui partagent des caractéristiques avec d'autres genres !

7. Conclusion

Nous avons implémenté avec succès de nombreuses méthodes pour classifier une musique selon son genre. Cela peut constituer un premier pas vers la conception d'un algorithme plus robuste pour reconnaître une musique ou encore pour construire un système de recommandation.

Que ce serait-il passé si nous avions supprimé certains genres partageant des caractéristiques avec d'autres genres du jeu de données initial (par exemple le blues et le jazz) ? En conservant les genres qui se distinguent réellement des autres, nous pourrions améliorer encore plus la performance de nos modèles au risque d'être moins précis dans la dénomination.

Avec un peu plus de temps et de compétences, nous aurions également pu explorer la voie de l'apprentissage profond en implémentant des réseaux de neurones, méthode fréquemment apparue lors de nos recherches sur le sujet.

[1] Sélection de caractéristiques

Références

1. **Arora, Aman.** Implementing Feature Importance in Random Forests from Scratch. *Medium.* [Online] 1 Janvier 2019. https://medium.com/@amaarora/implementing-feature-importance-in-random-forests-from-scratch-2216e031ff74.
2. **Koehrsen, Will.** Hyperparameter Tuning the Random Forest in Python. *Medium.* [Online] 10 Janvier 2018. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74.
3. **Zakka, Kevin.** A Complete Guide to K-Nearest-Neighbors with Applications in Python and R. [Online] 13 Juillet 2016. https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/.
4. **Brownlee, Jason.** Naive Bayes Classifier From Scratch in Python. *Machine Learning Mastery.* [Online] 25 Octobre 2019. https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/.
5. **Utah, The University of.** Support Vector Machines: Training with Stochastic Gradient Descent. [Online] https://svivek.com/teaching/lectures/slides/svm/svm-sgd.pdf.
6. Gradient descent and stochastic gradient descent from scratch. *Gluon.* [Online] https://gluon.mxnet.io/chapter06_optimization/gd-sgd-scratch.html.
7. ML | Mini-Batch Gradient Descent with Python. *Geeks for Geeks.* [Online] https://www.geeksforgeeks.org/ml-mini-batch-gradient-descent-with-python/.
8. **Sturm, Bob L.** The GTZAN dataset: Its contents, its faults,their effects on evaluation, and its future use. [Online] 10 Juin 2013. https://arxiv.org/pdf/1306.1461.pdf.