

Rapport

TP04 — Développement d'un système intelligent

Equipe 7:

Kaggle username: musicmusic

BouzabiaHamda

Mefteh Aida

Belhadj Mohamed Nahed

Introduction

Le domaine de musique a créé des grandes nécessités de construire des classificateurs performants qu'aident les utilisateurs à bien gérer leurs collections désirées en se basant à des moyens faciles à manipuler.

Dans ce travail dirigé nous avons été demandés de concevoir un algorithme intelligent afin de résoudre le problème de classification de genres audio. Cet algorithme est le fruit de combinaison des plusieurs algorithmes d'apprentissage les plus pertinentes pour ce genre de problème en se basant à un processus de « voting ».

Afin de concevoir cet algorithme, plusieurs ensembles de données d'entraînement incluant différents ensembles de primitives audio ont été fournis, ses données contenant 179 555 fichiers audio MP3 étiquetés selon 25 genres musicaux. De plus, pour apprendre nos algorithmes, ces données ont besoin d'une phase de préparation telles que la normalisation et la réduction de la dimensionnalité que vont être bien expliqué dans la suite du rapport.

Question 2 :

Pour ce travail dirigé nous avons utilisés :

Environnement matériel :

Un ordinateur avec les performances suivantes :

- Modèle HP
- OS Windows 10-64 bits
- RAM 16GB
- Processeur Intel(R) Core(TM) i7-3770 CPU@ 3.40GHz 3.40GHz

Environnement logiciel :

Anaconda : nous avons travaillé avec python 3 sur un notebook Jupyter.

Partitionnement des données

Comme dans le TP précédent, nous avons partagé les données comme suit : 80% des données pour l'apprentissage et 20% pour la validation.

Prétraitement de donnée

Nous avons utilisé 3 techniques que sont :

Normalisation :

Nous avons utilisé la normalisation dans la plage [0,1] afin d'empêcher que les attributs avec des grandes plages prédominent sur les attributs avec des petites plages.

La mise à l'échelle des fonctionnalités par la normalisation (ou la normalisation Z-score) peut constituer une étape de prétraitement importante pour de nombreux algorithmes d'apprentissage automatique. En effet, la normalisation implique de redimensionner les caractéristiques de manière à ce qu'elles présentent les propriétés d'une distribution normale standard avec une moyenne de 0 et un écart type de 1.

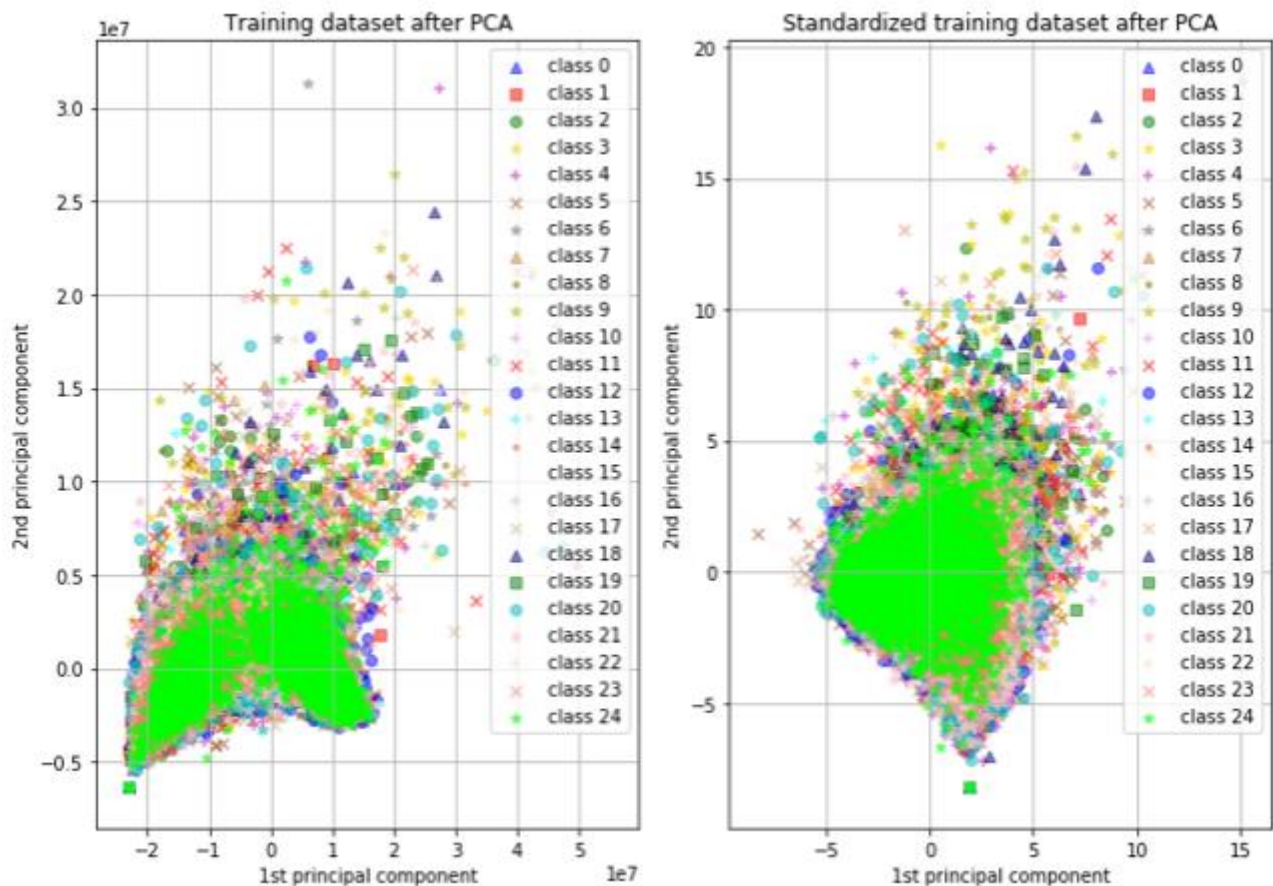
Réduction de la dimensionnalité (PCA)

Bien que de nombreux algorithmes (tels que SVM, les voisins K-les plus proches et la régression logistique) nécessitent la normalisation de fonctionnalités, intuitivement, nous pouvons penser à la PCA (Principle Component Analysis) comme un excellent exemple du moment où la normalisation est importante. En PCA, nous nous intéressons aux composants qui maximisent la variance. Si un composant varie moins qu'un autre en raison de leurs échelles, la PCA pourrait déterminer que la direction de la variance maximale correspond au composant que varie le plus.

Pour illustrer cela, la PCA est réalisée en comparant l'utilisation de données avec des données **StandardScaler** appliquées, à des données non mises à l'échelle. Les résultats sont visualisés et une nette différence est notée.

Le jeu de données utilisé est le jeu de données de musique. Cet ensemble de données présente des caractéristiques continues d'échelle hétérogène en raison des propriétés différentes qu'ils mesurent.

Les données transformées sont ensuite utilisées pour former un classifieur naïf de Bayes, KNN et RNA. Une différence nette en termes de précision de prédiction est observée, l'ensemble de données mis à l'échelle avant PCA surpassant considérablement la version non mise à l'échelle.



Dans cette section, nous explorons l'un des algorithmes non supervisés les plus largement utilisés, l'analyse en composantes principales (PCA). PCA est fondamentalement un algorithme de réduction de dimensionnalité, mais il peut également être utile en tant qu'outil de visualisation, de filtrage du bruit, d'extraction de caractéristiques et d'ingénierie, et bien plus encore.

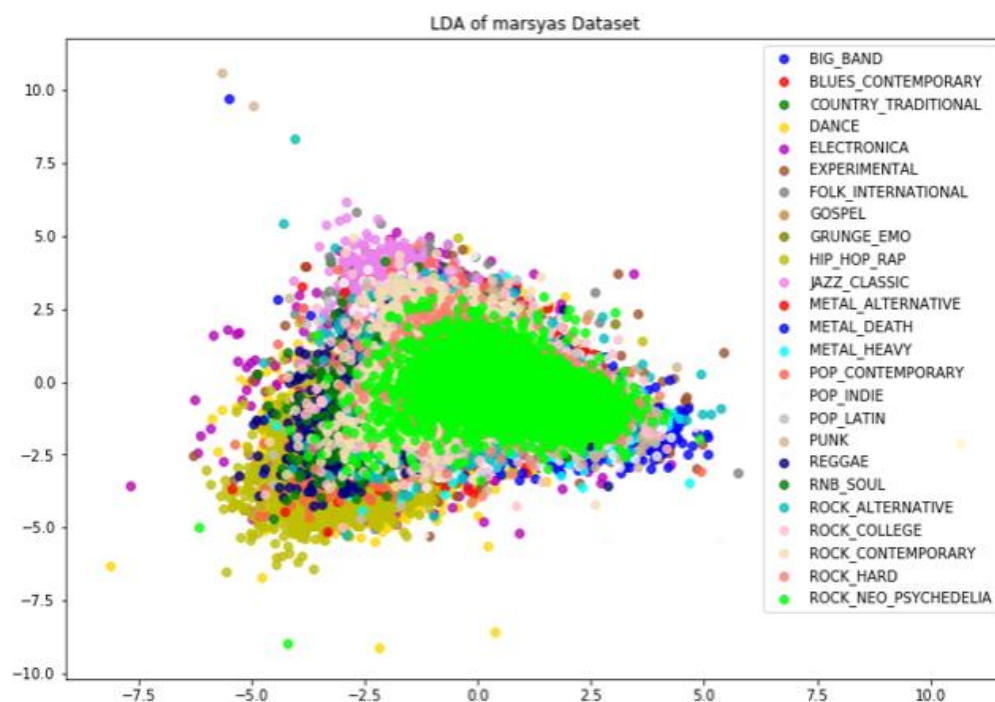
L'importance d'ajout de PCA pour les trois modèles :

Nous allons calculer dans cette partie la précision et le f1-score pour les trois modèles avec et sans PCA pour bien montrer l'effet positive de l'ajout de PCA dans notre solution :

KNN				RNA				Bayes			
datasv		acc for KNN with pca	acc KNN	datasv		acc for RNA with pca	acc RNA	datasv		acc for bayes with pca	acc bayes
0	msd-jmirderivatives_dev	0.096071	0.037147	0	msd-jmirderivatives_dev	0.270335	0.269305	0	msd-jmirderivatives_dev	0.128150	0.127203
1	msd-jmiripc_dev	0.063797	0.070842	1	msd-jmiripc_dev	0.197488	0.213500	1	msd-jmiripc_dev	0.160759	0.133079
2	msd-jmirmfccs_dev	0.122135	0.096962	2	msd-jmirmfccs_dev	0.231545	0.251399	2	msd-jmirmfccs_dev	0.169419	0.149787
3	msd-jmirmoments_dev	0.046476	0.019103	3	msd-jmirmoments_dev	0.178191	0.202696	3	msd-jmirmoments_dev	0.152182	0.133552
4	msd-jmirspectral_dev	0.066832	0.041074	4	msd-jmirspectral_dev	0.200301	0.232909	4	msd-jmirspectral_dev	0.162596	0.141711
5	msd-marsyas_dev_new	0.179583	0.090056	5	msd-marsyas_dev_new	0.295146	0.302971	5	msd-marsyas_dev_new	0.109409	0.065718
6	msd-mvd_dev	0.061680	0.020941	6	msd-mvd_dev	0.226003	0.233383	6	msd-mvd_dev	0.095737	0.122609
7	msd-rh_dev_new	0.061318	0.060037	7	msd-rh_dev_new	0.201108	0.205954	7	msd-rh_dev_new	0.127259	0.123973
8	msd-ssd_dev	0.155328	0.135752	8	msd-ssd_dev	0.288825	0.291164	8	msd-ssd_dev	0.168054	0.138398
9	msd-trh_dev	0.056891	0.051683	9	msd-trh_dev	0.210660	0.213027	9	msd-trh_dev	0.083930	0.094177

Analyse de discrimination linéaire (LDA)

De plus de la méthode PCA, nous avons essayé la méthode LDA, cette dernière est une approche supervisée permet de trouver l'espace dans lequel la projection d'entrée donne une bonne séparation entre les classes. Cette espace est de dimension inférieure à l'espace d'entrée.



Cependant, l'utilisation de LDA n'a pas donnée une amélioration sur les performances de nos classificateurs.

Question 3 :

Nous avons travaillé avec plusieurs modèles d'apprentissage comme SVM, Random Forest, KNN, Bayes Naif et RNA.

Tout d'abord, nous avons appliqué la méthode **Random Forest** sur notre ensemble de données, qui nous a donné une précision assez faible, D'une part, cette méthode consomme beaucoup de temps lors de la phase d'apprentissage ainsi que dans la phase de prédiction. D'autre part, l'ajout d'une autre primitive entraîne une complexité au niveau de l'entraînement et la prédiction. Pour conclure, l'arbre aléatoire est considéré comme une boîte noire difficilement interprétable et difficilement améliorable. C'est pour ces raisons que nous n'avons pas pris cette méthode comme l'un de nos modèles.

De plus, nous avons appliqué la méthode **SVM** sur un seul fichier (msd-marsays_dev_new) que nous a donné une précision égale à 12% cependant elle a pris beaucoup de temps pour s'entraîner. En effet, cette méthode possède une performance assez élevée ainsi qu'elle traite des problèmes non linéaires (la distribution entre les classes est irrégulière, non balancé). Cependant, nous n'avons pas choisi cette méthode vu qu'elle est plus lente (179555 fichiers MP3 et 1360 primitives) et qu'elle présente un calcul matriciel important. Finalement, notre problème vise la classification de 26 classes celle-ci reste une question ouverte dans le SVM.

Donc, nous avons choisi de travailler avec KNN, Bayes Naif et RNA à titre de classificateur de base, comme décrits ci-dessous :

Le **KNN** est un modèle facile à comprendre, il permet un apprentissage rapide et applicable aux données non linéaires(notre cas).

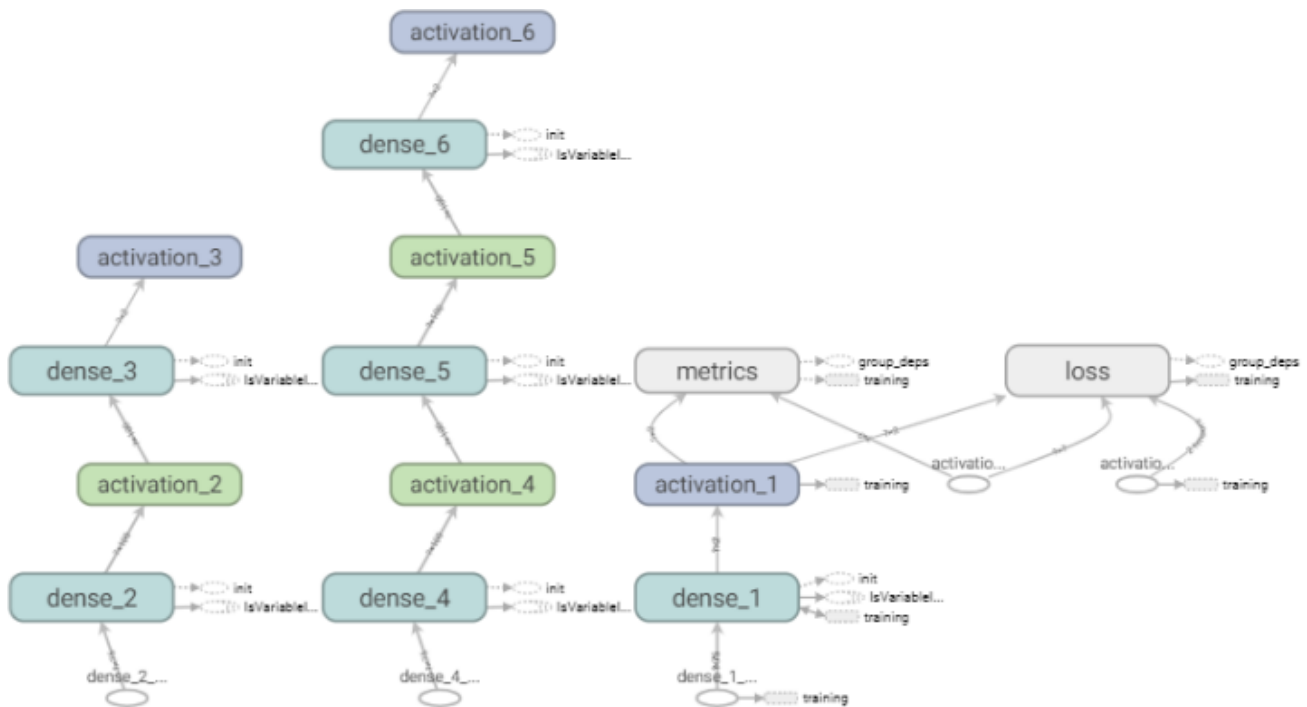
Le **Bayes Naif** est un modèle simple, il présente un coût de calcul rapide et il est très efficace et robuste pour les grandes bases de données (données et primitives), d'où aucun risque de plantage.

Le **réseau de neurones RNA** est un modèle qui possède une capacité de traitement très élevées pour divers problèmes. Il peut travailler avec des données bruitées et aberrantes dans des domaines complexes ; il est plus performant que les statistiques et les arbres de décisions. Dans notre cas, nous avons créé deux couches cachées en utilisant la fonction d'activation « **relu** ». Ce dernier est une fonction simple à être dériver et elle converge rapidement par rapport à la fonction « Sigmoid ». De plus, nous avons choisi **Adam** comme un optimisier. A la sortie de chaque couche cachée, nous avons utilisé BatchNormalization(), ce dernier permet au RNA de converger plus rapidement. Puis, nous avons appliqué Dropout () qui permet de désactiver quelques neurones de la première couche

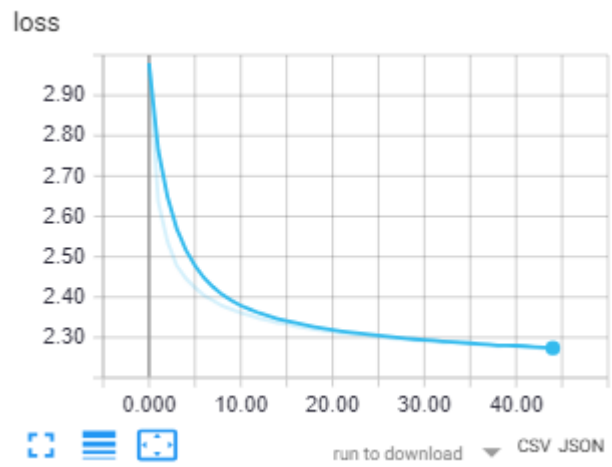
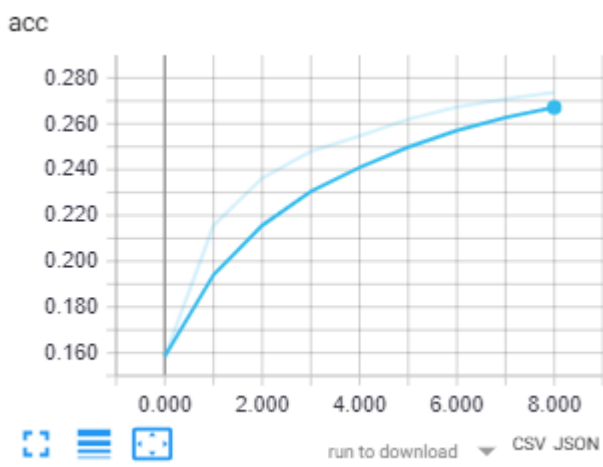
pour éviter le problème de sur-apprentissage et rendre le réseau plus robuste et puissant. Nous avons choisi un taux de 0.2 que nous a donné la meilleure précision.

Enfin nous avons utilisé la fonction d'activation **softmax** qui permet de calculer les probabilités de chaque classe sur toutes les classes cibles possibles.

La figure ci-dessous montre le graphe TensorBoard obtenue pour un MLP à deux couches cachées réalisé avec Tensorflow.

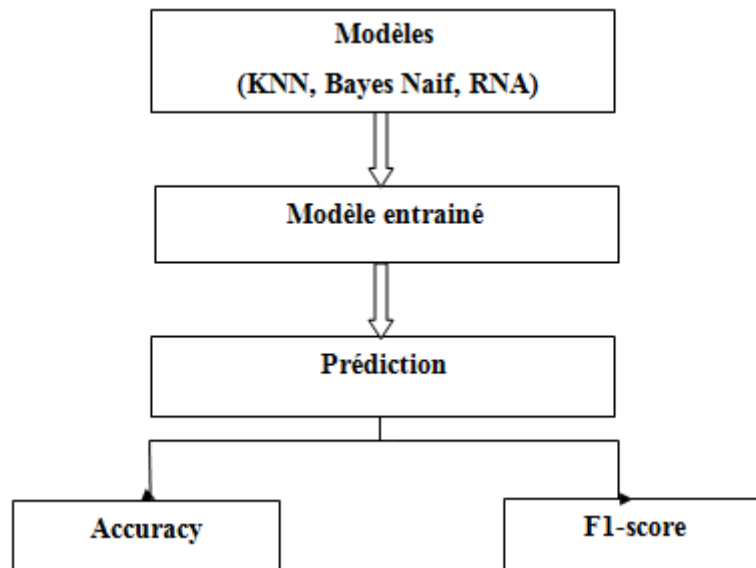


De plus, les deux figures présentés ci-dessous montrent respectivement l'évolution de la courbe de précision et d'affaiblissement en fonction de nos données.



Question 4 :

Le processus que nous avons abordé dans le choix des hyper-paramètres pour chaque modèle d'apprentissage ainsi que les fichiers que nous allons utiliser dans la suite du problème est illustré dans la figure ci-dessous. Ceux qui possèdent les meilleures précisions sont retenus.



K-plus proche voisin (KNN) :

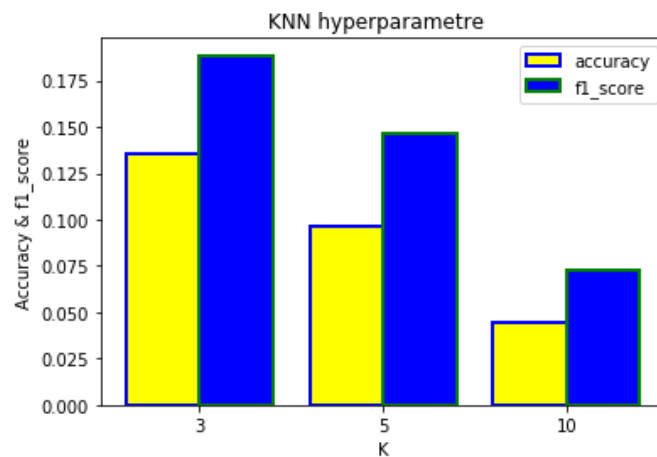
Dans cette partie, nous avons appliqué le modèle KNN avec différents hyper-paramètres sur tous les ensembles de données afin de choisir le fichier qui présente la meilleure prédiction dont l'objectif de l'utiliser dans l'entraînement de notre modèle.

K=3				K=5				K=10			
datacsv acc KNN avec K=3 f1 KNN avec K=3				datacsv acc KNN avec K=5 f1 KNN avec K=5				datacsv acc KNN avec K=10 f1 KNN avec K=10			
0	msd-jmirderivatives_dev	0.037147	0.057159	0	msd-jmirderivatives_dev	0.021860	0.036743	0	msd-jmirderivatives_dev	0.007463	0.013441
1	msd-jmiripc_dev	0.070842	0.103866	1	msd-jmiripc_dev	0.045919	0.074273	1	msd-jmiripc_dev	0.016318	0.029187
2	msd-jmirmfccs_dev	0.096962	0.141098	2	msd-jmirmfccs_dev	0.064799	0.104815	2	msd-jmirmfccs_dev	0.022611	0.040754
3	msd-jmirmoments_dev	0.019103	0.031059	3	msd-jmirmoments_dev	0.006433	0.011761	3	msd-jmirmoments_dev	0.000585	0.001147
4	msd-jmirspectral_dev	0.041074	0.062043	4	msd-jmirspectral_dev	0.024700	0.040718	4	msd-jmirspectral_dev	0.007936	0.014372
5	msd-marsyas_dev_new	0.090056	0.130195	5	msd-marsyas_dev_new	0.060817	0.096826	5	msd-marsyas_dev_new	0.025842	0.044482
6	msd-mvd_dev	0.020941	0.035433	6	msd-mvd_dev	0.010470	0.019325	6	msd-mvd_dev	0.004316	0.008230
7	msd-rh_dev_new	0.060037	0.087279	7	msd-rh_dev_new	0.042995	0.065284	7	msd-rh_dev_new	0.027318	0.042357
8	msd-ssd_dev	0.135752	0.188593	8	msd-ssd_dev	0.096711	0.146451	8	msd-ssd_dev	0.044610	0.072982
9	msd-trh_dev	0.051683	0.077257	9	msd-trh_dev	0.036507	0.056853	9	msd-trh_dev	0.022751	0.036138
Le fichier que donne la meilleure précision et f1_score est :msd-				Le fichier que donne la meilleure précision et f1_score est : msd-				Le fichier que donne la meilleure précision et f1_score est :			

ssd-dev	ssd-dev	msd-ssd-dev
---------	---------	-------------

Nous avons présenté dans le tableau ci-dessous les valeurs de la précision et de F1-score pour le fichier « msd-ssd-dev » en fonction du nombre de voisins {3, 5, 10}

Hyper paramètre	K=3	K=5	K=10
Accuracy	13.57 %	9.67 %	4.46 %
F1- measure	18.85 %	14.64 %	7.3 %



A partir de cette figure il est clair que la précision et le f1-score sont meilleures pour k=3

Bayes naif

De même pour le Bayes Naif, nous avons appliqué ce modèle avec différents hyper-paramètres sur tous les ensembles de données afin de choisir le fichier ou les fichiers qui présentent la meilleure prédiction dont l'objectif de l'utiliser dans l'entraînement de notre modèle. Comme présenté dans le tableau suivant :

GaussianNB				MinMaxScaler				KBinsDiscretizer			
	datacsv	acc Gaussien	f1 Gaussien		datacsv	acc MinMaxScaler	f1 MinMaxScaler		datacsv	acc KBins	f1 KBins
0	msd-jmirderivatives_dev	0.127203	0.093697	0	msd-jmirderivatives_dev	12.798307	6.708071	0	msd-jmirderivatives_dev	10.578931	6.658869
1	msd-jmiripc_dev	0.133079	0.096914	1	msd-jmiripc_dev	9.245078	2.900868	1	msd-jmiripc_dev	13.032219	7.323158
2	msd-jmirmfccs_dev	0.149787	0.116703	2	msd-jmirmfccs_dev	6.961655	1.443143	2	msd-jmirmfccs_dev	9.206093	4.573911
3	msd-jmirmoments_dev	0.133552	0.090303	3	msd-jmirmoments_dev	6.811283	1.372364	3	msd-jmirmoments_dev	10.631840	6.271530
4	msd-jmirspectral_dev	0.141711	0.102499	4	msd-jmirspectral_dev	7.785915	2.571337	4	msd-jmirspectral_dev	10.701456	6.048375
5	msd-marsyas_dev_new	0.065718	0.021114	5	msd-marsyas_dev_new	14.641753	8.824182	5	msd-marsyas_dev_new	14.457966	11.214531
6	msd-mvd_dev	0.122609	0.104360	6	msd-mvd_dev	13.210437	7.873435	6	msd-mvd_dev	12.675782	10.421407
7	msd-rh_dev_new	0.123973	0.096281	7	msd-rh_dev_new	10.077692	4.929495	7	msd-rh_dev_new	11.573056	6.911138
8	msd-ssd_dev	0.138398	0.110653	8	msd-ssd_dev	16.471276	11.223404	8	msd-ssd_dev	14.730862	11.914514
9	msd-trh_dev	0.094177	0.085878	9	msd-trh_dev	12.397316	7.030457	9	msd-trh_dev	11.456100	9.443493
Le fichier que donne la meilleure précision et f1_score est :Msd-jmirmfccs_dev				Le fichier que donne la meilleure précision et f1_score est : msd-ssd-dev				Le fichier que donne la meilleure précision et f1_score est :msd-ssd-dev			

Nous avons conclu que les fichiers, présentés ci-dessous, ont les meilleures valeurs de précision et de F1-score pour les différents hyper-paramètres

Hyper paramètre	GaussianNB (Msd-jmirmfccs_dev)	Multinomial :MinMaxScaler (msd-ssd-dev)	Multinomial :KBinsDiscretizer (msd-ssd-dev)
Accuracy	14.97 %	16.47 %	14.73 %
F1- measure	11.67 %	11.22 %	11.91 %

Au total, nous avons choisi de travailler avec **Multinomial : MinMaxScaler** avec le fichier csv : msd-ssd-dev

Le Réseau de Neurones (RNA)

Variation de nombres de couches :

Nous avons utilisé nombre de couches égale à 2 et 4 couches

Nombre de couche=2				Nombre de couche=4																																																																																											
<table><tr><th></th><th>datacsv</th><th>acc RNA 2 couches</th><th>f1 RNA 2 couches</th></tr><tr><td>0</td><td>msd-jmirderivatives_dev</td><td>27.465122</td><td>25.112657</td></tr><tr><td>1</td><td>msd-jmiripc_dev</td><td>21.876305</td><td>18.761342</td></tr><tr><td>2</td><td>msd-jmirmfccs_dev</td><td>25.891788</td><td>23.096507</td></tr><tr><td>3</td><td>msd-jmirmoments_dev</td><td>20.598145</td><td>17.323976</td></tr><tr><td>4</td><td>msd-jmirspectral_dev</td><td>23.675197</td><td>20.995545</td></tr><tr><td>5</td><td>msd-marsyas_dev_new</td><td>30.895826</td><td>28.848861</td></tr><tr><td>6</td><td>msd-mvd_dev</td><td>23.803291</td><td>21.392255</td></tr><tr><td>7</td><td>msd-rh_dev_new</td><td>21.021414</td><td>17.707420</td></tr><tr><td>8</td><td>msd-ssd_dev</td><td>30.247000</td><td>28.276837</td></tr><tr><td>9</td><td>msd-trh_dev</td><td>21.834535</td><td>19.287921</td></tr></table>					datacsv	acc RNA 2 couches	f1 RNA 2 couches	0	msd-jmirderivatives_dev	27.465122	25.112657	1	msd-jmiripc_dev	21.876305	18.761342	2	msd-jmirmfccs_dev	25.891788	23.096507	3	msd-jmirmoments_dev	20.598145	17.323976	4	msd-jmirspectral_dev	23.675197	20.995545	5	msd-marsyas_dev_new	30.895826	28.848861	6	msd-mvd_dev	23.803291	21.392255	7	msd-rh_dev_new	21.021414	17.707420	8	msd-ssd_dev	30.247000	28.276837	9	msd-trh_dev	21.834535	19.287921	<table><tr><th></th><th>datacsv</th><th>acc RNA 4 couches</th><th>f1 RNA 4 couches</th></tr><tr><td>0</td><td>msd-jmirderivatives_dev</td><td>26.980591</td><td>24.210320</td></tr><tr><td>1</td><td>msd-jmiripc_dev</td><td>21.135585</td><td>17.272875</td></tr><tr><td>2</td><td>msd-jmirmfccs_dev</td><td>25.262454</td><td>22.146946</td></tr><tr><td>3</td><td>msd-jmirmoments_dev</td><td>20.314110</td><td>16.346559</td></tr><tr><td>4</td><td>msd-jmirspectral_dev</td><td>23.226866</td><td>19.875031</td></tr><tr><td>5</td><td>msd-marsyas_dev_new</td><td>30.210799</td><td>27.481467</td></tr><tr><td>6</td><td>msd-mvd_dev</td><td>23.555457</td><td>20.524206</td></tr><tr><td>7</td><td>msd-rh_dev_new</td><td>20.709532</td><td>17.174741</td></tr><tr><td>8</td><td>msd-ssd_dev</td><td>29.497925</td><td>26.276108</td></tr><tr><td>9</td><td>msd-trh_dev</td><td>21.561638</td><td>18.437555</td></tr></table>					datacsv	acc RNA 4 couches	f1 RNA 4 couches	0	msd-jmirderivatives_dev	26.980591	24.210320	1	msd-jmiripc_dev	21.135585	17.272875	2	msd-jmirmfccs_dev	25.262454	22.146946	3	msd-jmirmoments_dev	20.314110	16.346559	4	msd-jmirspectral_dev	23.226866	19.875031	5	msd-marsyas_dev_new	30.210799	27.481467	6	msd-mvd_dev	23.555457	20.524206	7	msd-rh_dev_new	20.709532	17.174741	8	msd-ssd_dev	29.497925	26.276108	9	msd-trh_dev	21.561638	18.437555
	datacsv	acc RNA 2 couches	f1 RNA 2 couches																																																																																												
0	msd-jmirderivatives_dev	27.465122	25.112657																																																																																												
1	msd-jmiripc_dev	21.876305	18.761342																																																																																												
2	msd-jmirmfccs_dev	25.891788	23.096507																																																																																												
3	msd-jmirmoments_dev	20.598145	17.323976																																																																																												
4	msd-jmirspectral_dev	23.675197	20.995545																																																																																												
5	msd-marsyas_dev_new	30.895826	28.848861																																																																																												
6	msd-mvd_dev	23.803291	21.392255																																																																																												
7	msd-rh_dev_new	21.021414	17.707420																																																																																												
8	msd-ssd_dev	30.247000	28.276837																																																																																												
9	msd-trh_dev	21.834535	19.287921																																																																																												
	datacsv	acc RNA 4 couches	f1 RNA 4 couches																																																																																												
0	msd-jmirderivatives_dev	26.980591	24.210320																																																																																												
1	msd-jmiripc_dev	21.135585	17.272875																																																																																												
2	msd-jmirmfccs_dev	25.262454	22.146946																																																																																												
3	msd-jmirmoments_dev	20.314110	16.346559																																																																																												
4	msd-jmirspectral_dev	23.226866	19.875031																																																																																												
5	msd-marsyas_dev_new	30.210799	27.481467																																																																																												
6	msd-mvd_dev	23.555457	20.524206																																																																																												
7	msd-rh_dev_new	20.709532	17.174741																																																																																												
8	msd-ssd_dev	29.497925	26.276108																																																																																												
9	msd-trh_dev	21.561638	18.437555																																																																																												
Le meilleur fichier est :msd-marsays_dev_new				Le meilleur fichier est :msd-marsays_dev_new																																																																																											

Pour le reste des hyper-paramètres nous allons choisir un nombre de couches égale à 2.

Variation de learning rate (taux d'apprentissage)

Lr=0.05				Lr=0.0005			
datacsv acc RNA avec Lr=0.05 f1 RNA avec Lr=0.05				datacsv acc RNA avec Lr=0.0005 f1 RNA avec Lr=0.0005			
0	msd-jmirderivatives_dev	26.186962	22.812897	0	msd-jmirderivatives_dev	27.465122	25.112657
1	msd-jmiripc_dev	20.851550	17.436207	1	msd-jmiripc_dev	21.876305	18.761342
2	msd-jmirmfccs_dev	24.758431	22.321836	2	msd-jmirmfccs_dev	25.891788	23.096507
3	msd-jmirmoments_dev	19.534405	15.339402	3	msd-jmirmoments_dev	20.598145	17.323976
4	msd-jmirspectral_dev	22.413745	19.916514	4	msd-jmirspectral_dev	23.675197	20.995545
5	msd-marsyas_dev_new	29.923979	27.569428	5	msd-marsyas_dev_new	30.895826	28.848861
6	msd-mvd_dev	22.642087	19.554227	6	msd-mvd_dev	23.803291	21.392255
7	msd-rh_dev_new	20.102476	17.648799	7	msd-rh_dev_new	21.021414	17.707420
8	msd-ssd_dev	29.127565	27.104335	8	msd-ssd_dev	30.247000	28.276837
9	msd-trh_dev	20.328033	17.914772	9	msd-trh_dev	21.834535	19.287921
Le meilleur fichier est :msd-marsays_dev_new				Le meilleur fichier est :msd-marsays_dev_new			

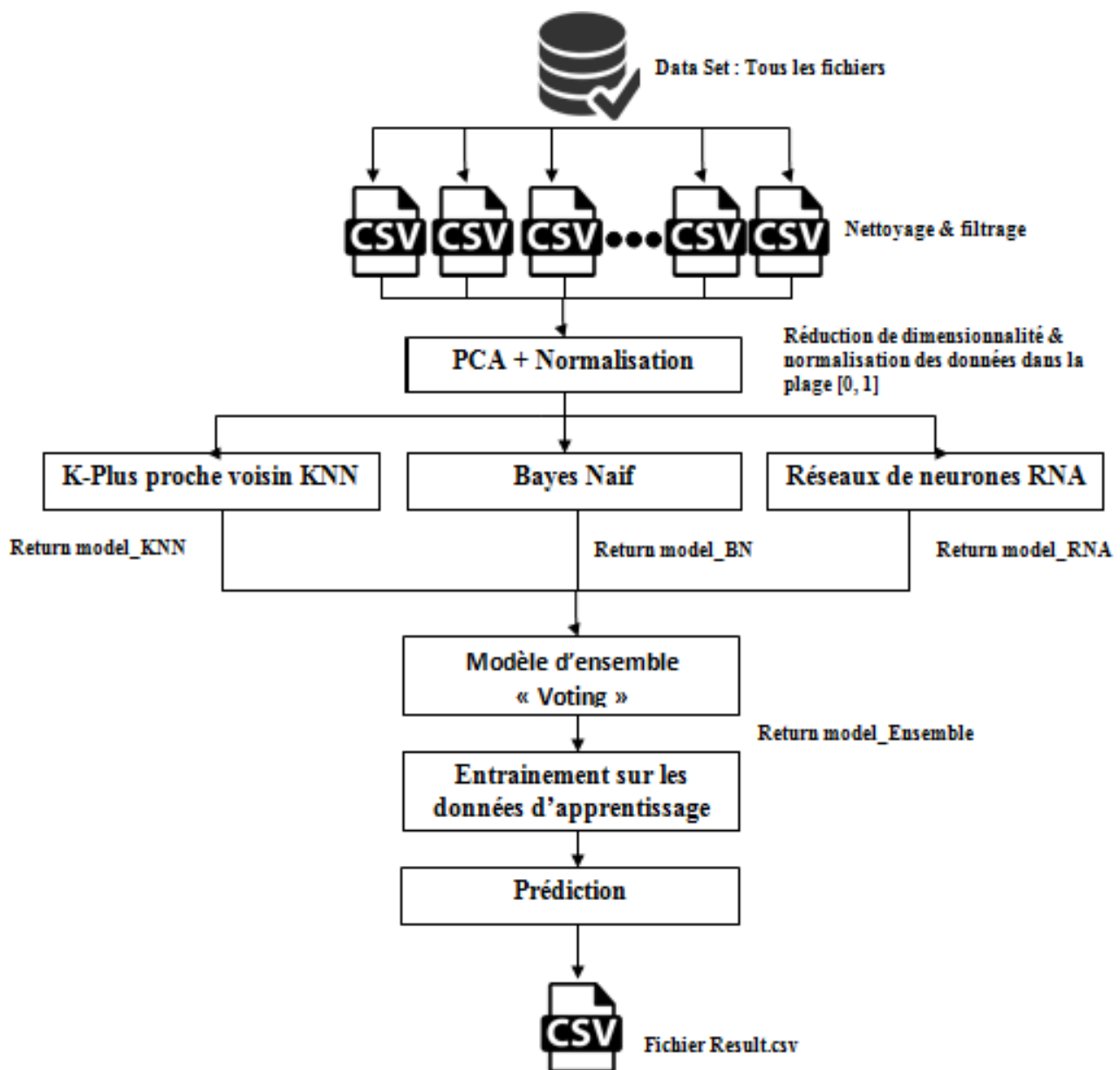
Le meilleur learning rate est Lr=0.0005

Donc, pour le reste de laboratoire nous allons évaluer la performance de modèle RNA principalement sur le fichier msd-marsays_dev_new en utilisant un nombre de couche égale à 2, et un taux d'apprentissage égale à 0.0005 puisque ces derniers ont donné la meilleure précision (30.89%).

Question 5 :

Nous allons définir dans cette partie les étapes que nous avons abordées pour élaborer la solution de ce problème. Comme première étape, nous avons appliqué des prétraitements sur nos données tels que la normalisation et la PCA afin de réduire la dimensionnalité et extraire les caractéristiques

pertinentes à partir desquelles l'apprentissage sera plus facile qu'à partir des entrées brutes. La deuxième étape consiste à construire un algorithme d'apprentissage intelligent. En effet, cet algorithme combine plusieurs algorithmes de classification suivant un processus de voting. Comme vous voyez dans la figure, ci-dessous, le modèle de combinaison des trois algorithmes KNN, Bayes Naïf et RNA vont subir un entraînement afin de construire le modèle d'apprentissage. Ensuite nous avons fait une prédiction sur différents ensembles de validation. Un ensemble « **du fichier tagget** » qui contient les labels (les classes sont définies) afin de valider la précision de notre algorithme. Au contraire, un ensemble « **du fichier untagget** » dont les valeurs cibles sont inconnus. Le résultat du deuxième ensemble est enregistré dans un fichier appelé « **result.csv** » qui sera soumis et prédit à la compétition Kaggle.



Pour la combinaison entre les modèles, nous avons utilisé **hard voting** (majority voting) qui permet de prédire la classe qui obtient la plupart des votes. La figure, présenté, ci-dessous illustre le bout de code de celle-ci.

```
def ENSEMBLE_model_3():
    print("Majority Vote of RNA, Bayes and KNN\n")
    model = VotingClassifier(estimators = [ ('RNA',model_RNA ), ('NB', model_Bayes) , ('KNN', model_KNN) ],
                            voting = 'hard',
                            n_jobs = 3)
    return model
```

Cette combinaison a amélioré nos résultats. En effet pour l'ensemble de données 'msd-jmirmfccs_dev' nous avons obtenu un pourcentage égal à 49.35 % pour les données d'apprentissage et 17.34 % pour les données validation.

Question 6 :

Les différents hyper paramètres finaux de nos modèles sont :

	KNN	Bayes Naïf	RNA
Hyper paramètre	K=3	Multinomial=MinMaxScalar	Nombre de couche = 2 Fonction d'activation=relu Taux d'apprentissage=0.0005

	Précision (%)	F1-score (%)
RNA	30.89	28.84
KNN	13.57	18.85
Bayes Naïf	16.47	11.22

Question 7 :

Comme piste d'amélioration nous proposons de mieux préparer les données initiales afin de réduire les données bruitées et aberrantes que peuvent parfois fausser l'apprentissage de nos modèles. Cette préparation permet aussi de choisir les primitives le plus discriminant parmi tous les fichiers proposés. De plus, précisément, pour les réseaux neurones nous suggestionnons que la combinaison



entre tous les hyper-paramètres peuvent donner des résultats meilleurs en terme de précision et f1-score.

Compétition sur Kaggle

Le fichier result.csv utilisé pour cette compétition est le résultat d'un classificateur RNA dont les hyperparamètres sont :

- 2 couches d'activation
- $Lr=0.0005$
- Fonction coût : categorical_crossentropy

Nous avons obtenu un pourcentage égal à 19.26 %

8	musicmusic	 	0.19266	11
---	------------	---	---------	----

Conclusion

Ce projet consiste à construire un algorithme intelligent qui combine trois algorithmes d'apprentissage afin de traiter les problèmes de classifications des genres de musique.

Pour se faire, nous avons commencé tout d'abord par des prétraitements sur les données comme la normalisation et le PCA pour les préparer à la phase d'apprentissage. Ensuite, suite à des évaluations effectuées en utilisant plusieurs algorithmes d'apprentissages, nous avons retenu trois classificateurs qui sont KNN, Bayes naïf et réseaux de neurones respectivement avec $k=3$, MinMaxScaler et 2 couches d'activation qui présentent les meilleurs hyperparamètres. Une fois que nous avons choisi ses modèles nous avons les combinés ensemble ce qui permet d'améliorer de plus nos résultats.