Final_ML_projet

May 12, 2023

1 Classification du genre musical d'une chanson ou d'un artiste

1.1 I. Introduction

1.1.1 But de l'étude

L'identification du genre musical auquel appartient une musique ou un artiste est une tâche complexe qui peut être abordée à travers des méthodes de machine learning. Cette problématique soulève des enjeux importants pour l'industrie musicale, notamment dans le cadre de la **recommandation de musique personnalisée, de la création de playlists automatisées, ou encore de la classification de contenu pour les plates-formes de streaming**.

De nombreuses études ont exploré différentes techniques de classification de genre musical à travers des méthodes de traitement du signal et d'analyse acoustique, mais également à travers des approches basées sur les réseaux de neurones et l'apprentissage automatique. Par exemple, l'étude de Tzanetakis et Cook (2002) a proposé une méthode de classification de genre musical basée sur l'analyse de caractéristiques temporelles et fréquentielles des signaux audio. D'autres études, comme celle de Lee et al. (2018), ont exploré l'utilisation de réseaux de neurones pour la classification de genre musical à partir de données provenant de Spotify.

Cependant, malgré ces avancées, l'identification du genre musical reste une tâche complexe en raison de la variabilité des caractéristiques acoustiques des genres musicaux, mais aussi de la subjectivité de la classification de genre musical en général. C'est pourquoi de nouvelles approches et méthodes sont nécessaires pour améliorer la précision de la classification de genre musical.

Dans ce contexte, ce projet de machine learning a pour objectif **d'explorer différentes techniques de classification de genre musical** à partir de données de musique et d'artistes, en utilisant des approches basées sur **l'inférence bayésienne et l'apprentissage supervisé**. Nous chercherons à développer un modèle précis et robuste capable de classifier automatiquement le genre musical auquel appartient une musique ou un artiste.

1.1.2 Source des données

Pour ce projet de classification de genre musical, nous avons utilisé l'API Spotify pour créer une playlist contenant environ 2800 musiques de 4 genres différents : **le rap, le rock, l'électro et la musique classique**. Ces genres musicaux ont été choisis afin de ne pas alourdir le sujet avec une multitude de genres et de sous-genres, tout en permettant une mise en pratique efficace des méthodes de classification étudiées en cours. Ils ont également été sélectionnés car ils présentent des

caractéristiques acoustiques distinctes qui permettront à notre modèle de classification de mieux apprendre les différences entre les genres et de les classifier avec précision. De plus, ces genres sont très populaires et représentatifs de l'industrie musicale, ce qui leur confère une certaine pertinence pour notre étude.

Nous avons extrait les données pour notre projet de classification de genre musical en utilisant **l'API Spotify**. Nous avons commencé par extraire les caractéristiques audio des playlists de Spotify de chacun des 4 genres sélectionnés (rap, rock, électro et musique classique) et avons regroupé ces données dans un fichier CSV. Ensuite, nous avons extrait les mêmes caractéristiques pour plusieurs playlists d'artistes sur lesquels nous avons tester nos modèles de classification. Le code source de la création des données est présente dans le dossier 'creation_data'. Nous avons fait le choix d'utiliser l'API Spotify afin d'avoir la main complète sur notre dataset et d'en connaître parfaitement les biais.

1.2 II. Visualisation

1.2.1 La forme des fichiers

Nous disposons donc d'un fichier csv avec un medley de chansons des 4 genres musicaux et leurs caractéristiques acoustiques. Ces caractéristiques ont été créées par Spotify et fournissent des informations telles que le tempo, l'énergie de la musique… Le détail de ces caractéristiques est donné sur le site de l'API : https://developer.spotify.com/documentation/web-api/reference/get-several-audio-features. Dans la création du fichier csv nous avons gardé les features qui concernaient la partie musicale et supprimé les données qualitatives telles que l'uri (url)… Ainsi il prend la forme :

[1]:
```
import pandas as pd
data = pd.read_csv('data/all.csv')
data
```

[1]:

	popularity	danceability	energy	key	loudness	mode	acousticness	\
0	35	0.581	0.6940	4	-7.065	0	0.04310	
1	35	0.429	0.9830	5	-1.962	1	0.00135	
2	49	0.629	0.9860	6	-0.565	0	0.02490	
3	58	0.758	0.7460	0	-7.209	0	0.44200	
4	24	0.647	0.6740	1	-5.345	1	0.02500	
…	…	…	…	…	…	…	…	
2821	0	0.273	0.1510	2	-23.881	0	0.82800	
2822	0	0.088	0.0314	0	-32.668	0	0.85200	
2823	0	0.225	0.2250	2	-21.485	0	0.78700	
2824	34	0.290	0.0126	4	-27.648	0	0.98700	
2825	4	0.136	0.1620	5	-19.418	1	0.99200	

	instrumentalness	speechiness	liveness	valence	tempo	duration_ms	\
0	0.091500	0.0719	0.6780	0.3490	159.970	230688	
1	0.858000	0.1250	0.1290	0.4780	149.964	224000	
2	0.000001	0.4650	0.8410	0.0559	102.513	268098	

```
3             0.000834       0.0285    0.1540   0.6580  137.968        164893
4             0.220000       0.2830    0.0871   0.3080  148.011        196391
…                …            …         …        …       …              …
2821          0.787000       0.0492    0.1670   0.0832  131.268        400427
2822          0.190000       0.0487    0.1130   0.0513   84.687        526000
2823          0.756000       0.0434    0.1440   0.0649  153.593        570000
2824          0.943000       0.0592    0.0973   0.0333  117.779        339733
2825          0.948000       0.0471    0.0937   0.0297   99.082        621093

      time_signature      genre
0                  4    electro
1                  4    electro
2                  4    electro
3                  4    electro
4                  4    electro
…                …          …
2821               3  classique
2822               3  classique
2823               3  classique
2824               4  classique
2825               4  classique

[2826 rows x 15 columns]
```

Pour ce qui est des artistes dont nous allons étudier l'appartenance à un genre leur fichier csv se présente ainsi :

[2]:
```
#Playlist Beethoven de Spotify
pd.read_csv('data/artists/beethoven.csv')
```

[2]:
```
    popularity  danceability   energy  key  loudness  mode  acousticness  \
0           64         0.250  0.31800    0   -15.605     0         0.956
1           66         0.289  0.03060    9   -30.790     0         0.987
2           60         0.278  0.00245    1   -41.530     0         0.995
3           56         0.247  0.03830    5   -22.760     1         0.955
4           44         0.363  0.19300    7   -19.428     1         0.990
..         …             …      …  …       …    …             …
82          30         0.265  0.11800    3   -17.646     1         0.883
83          36         0.276  0.09380    0   -24.977     1         0.990
84          30         0.388  0.18100    2   -18.788     1         0.881
85          37         0.157  0.04540    5   -23.924     1         0.966
86          31         0.315  0.04140    5   -23.688     1         0.916

    instrumentalness  speechiness  liveness  valence    tempo  duration_ms  \
0              0.897       0.0411    0.0976   0.1880   98.171       442000
1              0.911       0.0446    0.1020   0.1180  125.610       212067
2              0.881       0.0497    0.1140   0.2930  139.161       403333
```

```
3              0.714       0.0363    0.1080   0.1360  108.252       758267
4              0.910       0.0328    0.0888   0.3250   85.701       373665
..               …           …         …        …        …            …
82             0.531       0.0402    0.1420   0.2070  156.092       401400
83             0.892       0.0381    0.1950   0.1250  101.124       340960
84             0.177       0.0408    0.3300   0.5020   75.943       405707
85             0.918       0.0374    0.1050   0.0497   79.182       623373
86             0.257       0.0444    0.0868   0.1090   83.956       664533

    time_signature      genre
0                4  classique
1                3  classique
2                3  classique
3                4  classique
4                4  classique
..             …        …
82               3  classique
83               4  classique
84               5  classique
85               3  classique
86               4  classique

[87 rows x 15 columns]
```

1.2.2 Analyse des données

Forme générale Pour avoir une visualisation d'ensemble des données, il peut être intéressant de visualiser les relations entre par paires des features. Pour se faire nous pouvons utiliser la fonction pairplot de seaborn.

[3]:
```
import seaborn as sns
#sns.pairplot(data, hue = 'genre')
```

La fonction étant assez longue à exécuter nous avons l'image enregistrée directement affichée ci-dessous.

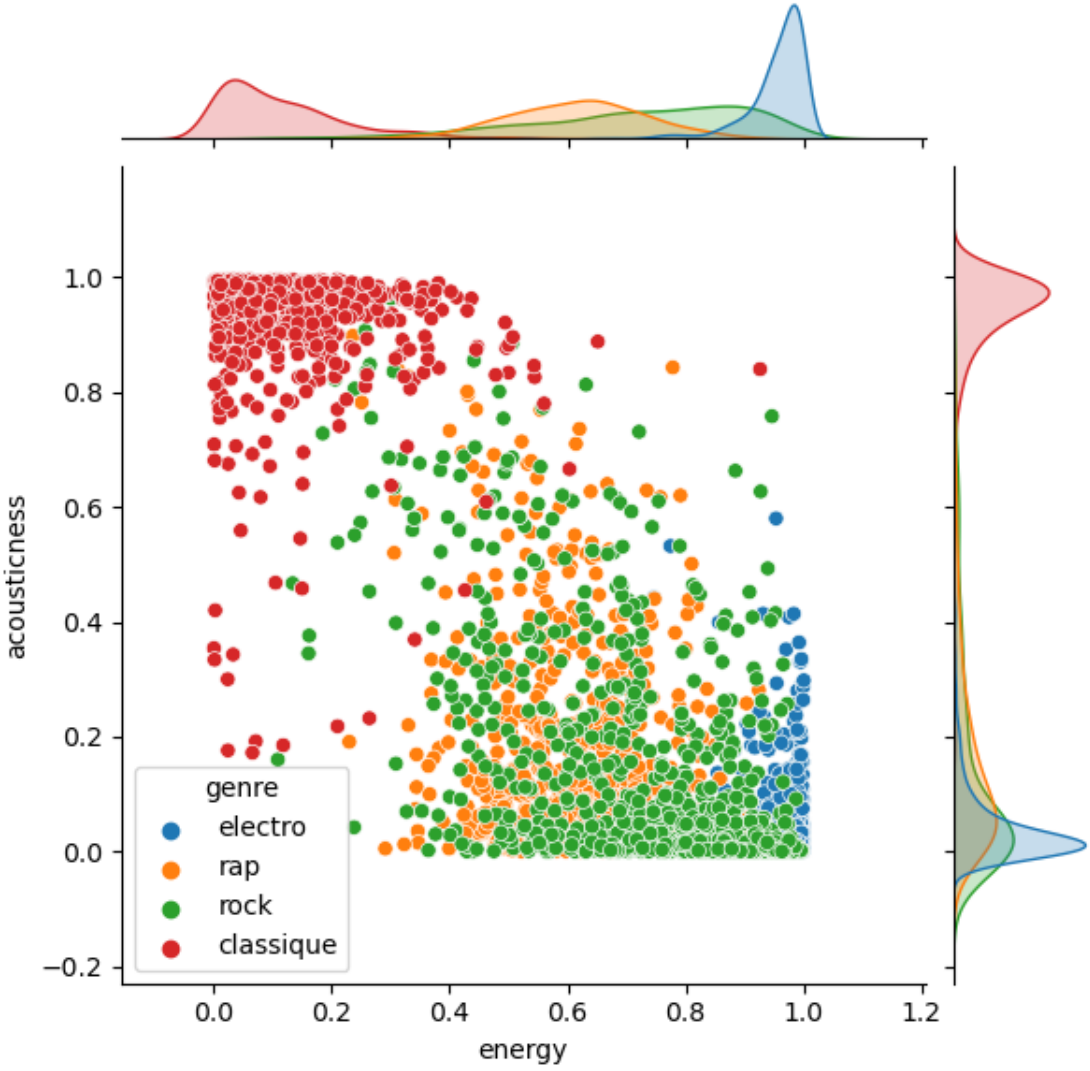
Avec ce premier plot nous pouvons remarquer plusieurs choses :

- Trois de nos features sont discètes : key, mode, time_signature qui correspondent respectivement à la clé, la modalité (majeur ou mineur) et la signature rythmique (nombre de temps dans une mesure)
- L'energy et la loudness semble être correlées
- Pour toutes les variables continues, leur distribution semble gaussienne

Nous voyons également que le classique se distingue généralement des autres genres pour plusieurs features. On peut zoomer sur un graphique en particulier pour mieux observer cette distinction.

[4]:
```
sns.jointplot(data = data, y ='acousticness', x =  'energy', hue = 'genre')
```

[4]: <seaborn.axisgrid.JointGrid at 0x7fa450db35b0>

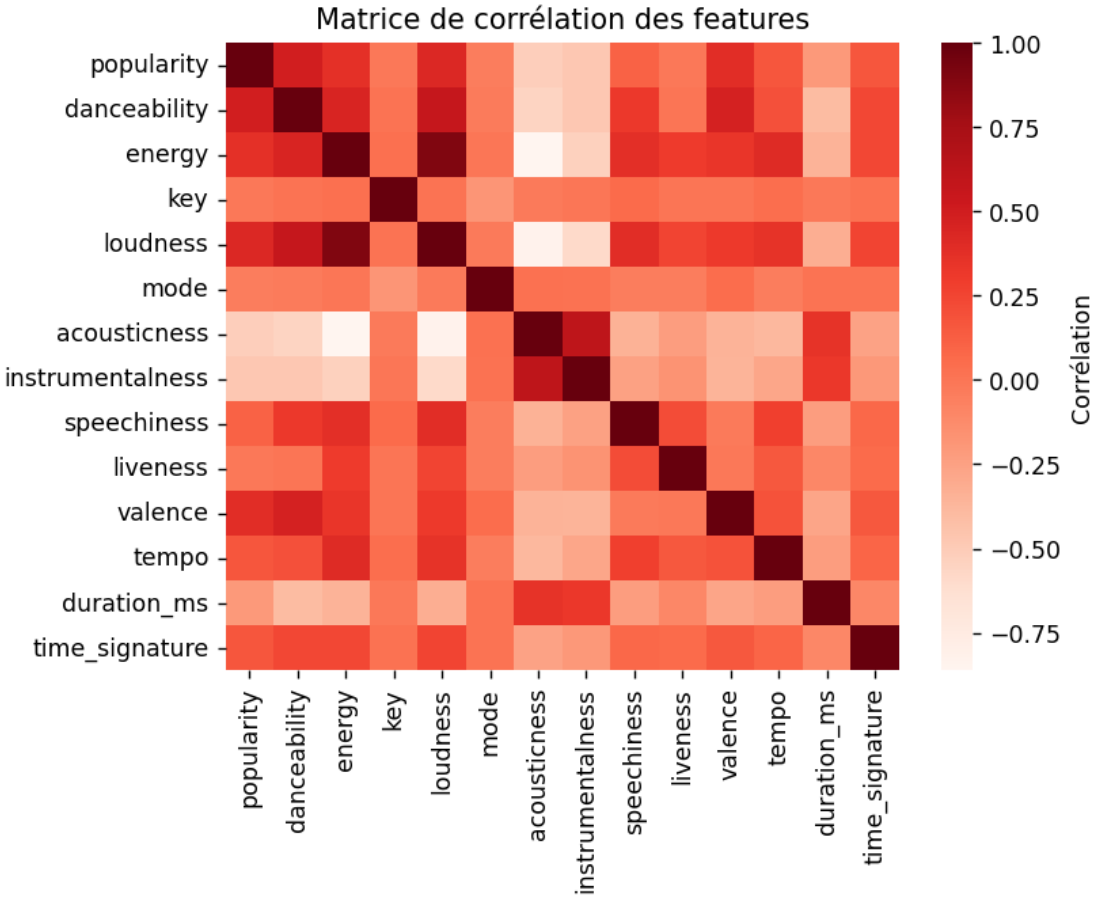


1.2.3 Corrélation

Une fois nos données visualisées, il est important de comprendre les différentes corrélations entre nos features pour l'analyse future. En effet, si deux variables sont corrélées elles ne pourront pas être indépendantes. Ces liens sont importants pour le choix de nos algorithmes. Effectivement, en vertu du théorème de No Free Lunch nous devons comprendre le contexte et les spécificités de nos données. Pour se faire, nous pouvons tracer la matrice de corrélation entre nos features.

[5]:
```
import matplotlib.pyplot as plt

plt.figure(dpi = 125)
```

```
plt.title('Matrice de corrélation des features')
sns.heatmap(data.corr(), cbar_kws={'label': 'Corrélation'},cmap='Reds')
plt.show()
```



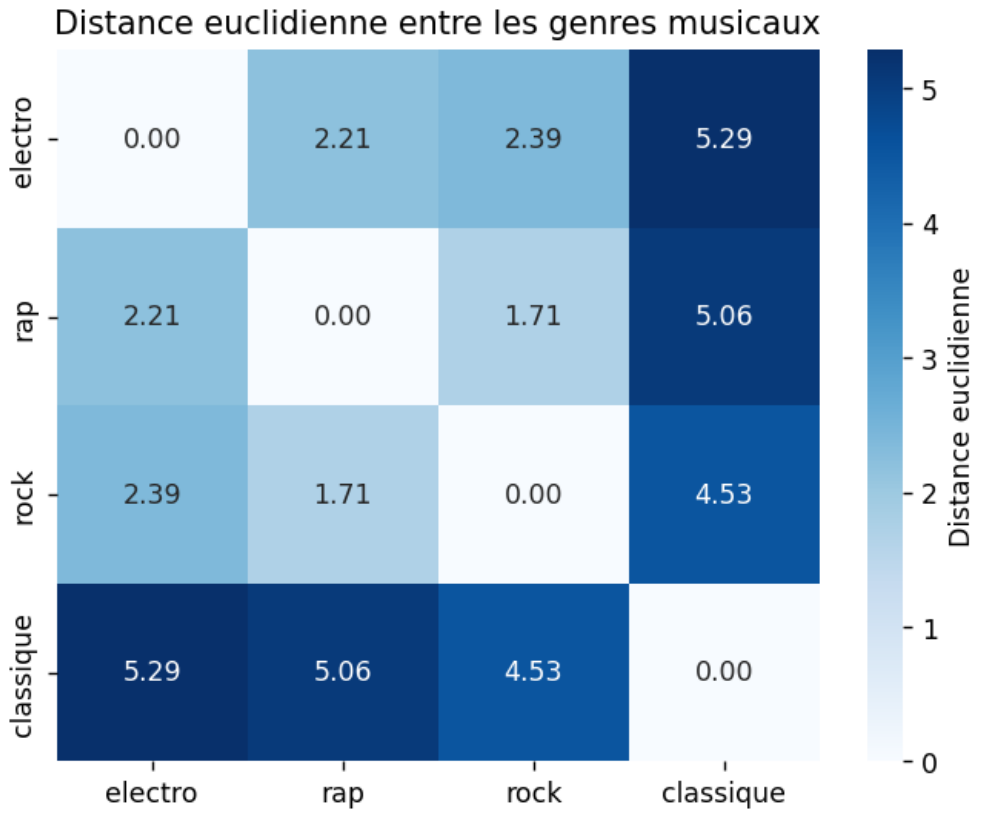
Nous voyons bien que la loudness et l'energy sont fortement correlées. Il faudra avoir cela en tête pour le reste de l'étude.

1.2.4 Distance euclidienne

Enfin, en étude préliminaire il est pertinent de visualiser la matrice des distances euclidiennes entre les genres afin de voir la similarité entre les différents genres musicaux. En utilisant cette matrice, nous pouvons identifier les genres qui sont les plus proches ou les plus éloignés les uns des autres en termes de caractéristiques musicales. Cette information peut nous aider à comprendre comment les différents genres se distinguent les uns des autres et comment les algorithmes de classification pourraient être influencés par ces différences.

Par exemple, si deux genres sont très proches en termes de caractéristiques musicales, cela pourrait signifier que l'algorithme aura plus de difficulté à les distinguer, tandis que si deux genres sont très différents, l'algorithme pourrait avoir une meilleure performance de classification.

[6]:
```
import numpy as np
from scipy.spatial.distance import cdist
from normalisation import normalisation

#On normalise les données grâce à une fonction spécialement écrite pour nos␣
 ↪données
data = pd.read_csv('data/all.csv')
data = normalisation(data)

# On extrait les colonnes correspondant aux features, on élimine la colonne␣
 ↪'genre'
features_cols = data.columns[:-1]

# On crée un dictionnaire de genres et de leurs features
genres = {}
for genre in data["genre"].unique():
    genre_data = data[data["genre"] == genre] #sélectionne un genre
    genre_vector = np.mean(genre_data[features_cols], axis=0) #calcule la␣
 ↪moyenne de chaque feature
    genres[genre] = genre_vector #l'entre dans le dictionnaire


# Calcul de la matrice des distances euclidiennes entre les genres
matrice = cdist(list(genres.values()), list(genres.values()),␣
 ↪metric='euclidean')


plt.figure(dpi = 125)
plt.title("Distance euclidienne entre les genres musicaux")


sns.heatmap(matrice,  annot=True,fmt=".2f",
            xticklabels= data["genre"].unique(),
            yticklabels= data["genre"].unique(),
            cmap = 'Blues',
            cbar_kws={'label': 'Distance euclidienne'})
plt.show()
```



On voit que le rap et le rock sont très proches et que l'électro les suit de près. Cela peut-être dû aux features qui ne sont pas assez pertinentes pour les différencier ou un biais de nos données liées au choix des chansons rap et électro. En effet, ce sont deux genres qui possèdent une multitude de sous-genres on peut donc supposer que les chansons prises sont plus proches de sous-genres se rapprochant du rock.

Si la distance euclidienne entre deux genres est trop faible, cela signifie que les caractéristiques musicales de ces genres sont très similaires et donc plus difficiles à distinguer. Certains algorithmes pourraient avoir du mal dans ce cas tels que les classifieurs linéaires ou la régression logistique ou le SVM linéaire, car ils cherchent une frontière de décision linéaire pour séparer les classes, et des classes similaires peuvent être difficiles à séparer de cette manière ou les algorithmes basés sur les k plus proches voisins comme KNN, car ils se basent sur la proximité des échantillons pour effectuer la classification.

En revanche, d'autres algorithmes peuvent ne pas être affectés par la faible distance euclidienne entre les genres, tels que les arbres de décision, car ils peuvent trouver des séparations non linéaires qui sont plus adaptées aux données qui ont des caractéristiques similaires entre les classes.

[7]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import glob
```

```
import sklearn.metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import validation_curve, KFold
import keras.utils
import keras.models
import keras.layers
from normalisation import normalisation
```

```
2023-05-09 22:54:59.996972: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

1.3 III. Les approches : bayésienne et machine learning

Nous avons selectionné plusieurs algorithmes d'apprentissage supervisé qui sont pertinents dans un problème de classification avec plus de deux classes.

1.3.1 1. L'approche bayésienne

Dans le cas de la classification de chansons d'un artiste dans un genre musical, nous pouvons utiliser une approche bayésienne pour prédire à quel genre appartient chaque chanson.

La formule de Bayes nous permet de calculer la probabilité qu'une chanson appartienne à un genre donné, en fonction de la probabilité de chaque features de la chanson pour ce genre. Nous utilisons également les probabilités a priori des genres musicaux $P(G_k)$, c'est-à-dire la probabilité qu'une chanson appartienne à un genre avant même de connaître ses caractéristiques. Nous supposons ces probabilités pour chaque genre comme équiprobables.

Ainsi, on calcule pour chaque genre G_k sachant la caractéristique F_i à l'itération i la probabilité que la chanson appartienne au genre k par la formule de Bayes suivante :

$$P(G_k|F_i) = \frac{P(F_i|G_k)P(G_k)}{\sum_{k=1}^{4} P(G_k)P(F_i|G_k)}$$

où $P(G_k)$ est remplacée à chaque itération par $P(G_k|F_i)$ et où $P(F_i|G_k)$ est calculée grâce à la fonction scipy qui renvoie la valeur de la densité de probabilité pour la valeur de la feature F_i. Le code suivant permet un tel calcul.

Les features dont la distributions n'est pas gaussienne ont été éliminées. De plus, les deux features corrélées également ('loudness' et 'energy') puisque la formule de Bayes suppose une indépendance des features utilisées. En effet, lorsque deux features sont corrélées, cela signifie qu'elles fournissent

des informations similaires, ce qui peut entraîner une sur-représentation de ces informations dans le modèle.

[8]:
```
import scipy.stats as st

#Formule de bayes appliquée pour une chanson

def bayesian_features(data, i, All):
    """
    data : playlist d'un artiste
    i : numéro de la chanson
    All : dataset

    """

    proba_list = []

    for genre in pd.unique(All['genre'].values): #prend les genres qui␣
 ↪apparaissent dans le data set

        proba = 0.25 #équiprobabilité à priori des genres

        cond = All['genre'] == genre #sélection des données d'un même genre

        #Formule de bayes pour chaque feature
        for features in data.drop(['key','mode','genre'], axis = 1):

            mean = np.mean(All[cond][features].values) #moyenne pour un genre␣
 ↪d'une feature
            sigma = np.std(All[cond][features].values) #idem pour l'écart-type

            feature_value= data[features][i]


            pdf = st.norm.pdf(feature_value, mean, sigma)
            proba = proba*pdf

        proba_list.append(proba)
    proba_list = np.array(proba_list)/np.sum(proba_list)

    return proba_list
```

[9]:
```
def bayesian_tracks(artist, All): #Prédiction sur toutes chansons d'un artiste

    y_pred = []

    for i in range(len(artist)):
```

```
        #Calcul des probabilités pour chaque genre
        proba = bayesian_features(artist, i, All)

        #On associe à la chanson le genre dont la probabilité est la plus haute
        genre = pd.unique(All['genre'].values)[np.argmax(proba)]

        y_pred.append(genre)

    return y_pred
```

On peut ainsi calculer les prédictions correctes pour chaque artiste.

[10]:
```
All = pd.read_csv('data/all.csv')
csv_files = glob.glob('data/artists/*.csv')
```

[11]:
```
bayes_predict = []
bayes_tab_y_pred = np.array([])

print("\nPourcentages de chansons de l'artiste correctement prédites :\n")

for name in csv_files:

    data_artist = pd.read_csv(f'{name}')
    y_artist = data_artist.iloc[:, -1]

    y_pred_artist = bayesian_tracks(data_artist, All)
    bayes_tab_y_pred = np.concatenate((bayes_tab_y_pred, y_pred_artist))

    bayes_predict.append(sklearn.metrics.accuracy_score(y_pred_artist,␣
 ↪y_artist))
    print(round(sklearn.metrics.accuracy_score(y_pred_artist, y_artist),2),␣
 ↪name[13:-4])

print('\n La moyenne du score des prédictions est de : ', round(np.
 ↪mean(bayes_predict),3))
bayes_mean = np.mean(bayes_predict)
```

```
Pourcentages de chansons de l'artiste correctement prédites :

0.8 acdc
0.9 bach
0.93 drake
0.84 rollingstones
0.12 deadmau5
1.0 brahms
```

```
0.05 daftpunk
0.98 cardib
0.08 aphextwins
0.93 kendricklamar
0.84 travisscott
0.7 beatles
0.92 drpeacock
0.06 kraftwerk
0.16 chemicalbrothers
0.08 majorlazer
1.0 chopin
0.65 queen
1.0 jcole
0.86 ledzepplin
0.99 mozart
1.0 beethoven

 La moyenne du score des prédictions est de :  0.677
```

Bien que l'approche bayésienne puisse donner de bons résultats pour la classification des chansons d'un artiste dans un genre musical, nous avons constaté que son taux de réussite est d'environ 68% et qu'elle a des difficultés à classifier le genre musical de la musique électronique. Pour améliorer la précision de notre classification, nous pouvons utiliser des algorithmes d'apprentissage supervisé tels que l'arbre de décision, random forest, K-NN ou les réseaux de neurones.

Ces algorithmes peuvent apprendre à partir des données d'entraînement et généraliser la classification pour de nouvelles données. Par exemple, l'arbre de décision peut apprendre des règles de décision simples à partir des caractéristiques des chansons, tandis que le random forest peut combiner plusieurs arbres de décision pour améliorer la précision. K-NN peut également classifier une chanson en se basant sur les caractéristiques de ses k chansons voisines (dans l'espace des features), tandis que le réseau de neurones peut apprendre des représentations plus complexes et abstraites des caractéristiques de la musique.

En utilisant ces algorithmes d'apprentissage supervisé, nous espérons améliorer la précision de la classification de genre musical de la musique électronique et de manière générale, obtenir de meilleurs résultats de classification pour toutes les genres musicales.

1.3.2 2. Modèles de Machine de learning :

Avant d'aborder les différents modèles d'apprentissage supervisé pour la classification des chansons d'un artiste dans un genre musical, nous allons normaliser nos données pour éviter que certaines caractéristiques aient un poids plus important que d'autres dans la classification et réduire le temps d'apprentissage de nos algorithmes.

La normalisation se fait à partir de la fonction normalisation du fichier éponyme. Elle appelle la fonction StandardScaler() permettant ainsi de normaliser les données csv des artistes de la même façon que le fichier d'entraînement 'all.csv'.

Nous divisons ensuite nos données en données de test et d'entraînement.

Preprocessing :

[12]:
```
data = normalisation(pd.read_csv('data/all.csv'))  #Normalisation des données
```

[13]:
```
data.head()
```

[13]:

	popularity	danceability	energy	key	loudness	mode	\
0	-0.093794	0.321627	0.290657	-0.332501	0.340162	-1.198418	
1	-0.093794	-0.445287	1.180248	-0.051925	0.938663	0.834433	
2	0.405985	0.563810	1.189482	0.228650	1.102509	-1.198418	
3	0.727272	1.214677	0.450722	-1.454803	0.323273	-1.198418	
4	-0.486478	0.654628	0.229094	-1.174227	0.541891	0.834433	

	acousticness	instrumentalness	speechiness	liveness	valence	tempo	\
0	-0.733783	-0.314992	-0.382625	2.662546	-0.076872	1.158535	
1	-0.843618	1.993341	0.070292	-0.497186	0.450933	0.840944	
2	-0.781664	-0.590542	2.970323	3.600681	-1.276096	-0.665155	
3	0.315633	-0.588035	-0.752805	-0.353300	1.187406	0.460190	
4	-0.781400	0.071989	1.417954	-0.738338	-0.244625	0.778955	

	duration_ms	time_signature	genre
0	-0.159014	0.224828	electro
1	-0.200568	0.224828	electro
2	0.073419	0.224828	electro
3	-0.567807	0.224828	electro
4	-0.372106	0.224828	electro

[14]:
```
y = data['genre'] #Segmentation des données
X = data.iloc[:, :-1]
print('X shape:',X.shape,'\ny shape:', y.shape)
```

```
X shape: (2826, 14)
y shape: (2826,)
```

[15]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪shuffle=True)
# Sépartion des données, en train et test set
```

1.3.3 1 - Decision Tree

Le premier algorithme que nous utilisons est l'arbre de décision (Decision Tree). C'est un modèle de classification qui fonctionne en divisant récursivement les données en sous-groupes homogènes en fonction de leurs caractéristiques jusqu'à ce que 'les feuilles de l'arbre' représentent les classes finales.

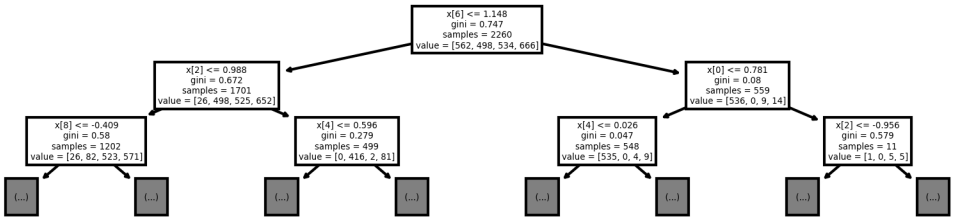
Entrainement :

[16]:
```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train,y_train)
```

[17]:
```
y_pred = clf.predict(X_test)
print('Train score :',clf.score(X_train,y_train))
print('Test score :',clf.score(X_test,y_test))
```

```
Train score : 1.0
Test score : 0.8851590106007067
```

[18]:
```
plt.figure(figsize=(7, 1.5), dpi=300)
tree.plot_tree(clf.fit(X_train, y_train), max_depth=2)
plt.show()
```



Prédiction de l'arbre de décision : On obtient les prédictions suivantes.

[19]:
```
csv_files = glob.glob('data/artists/*.csv')
```

[20]:
```
tree_predict = []
tab_y_vrai = np.array([])
tree_tab_y_pred = np.array([])

for name in csv_files:
    data_artist = normalisation(pd.read_csv(f'{name}'))

    y_vrai = data_artist['genre']
    X_artist = data_artist.drop(['genre'],axis=1)

    tab_y_vrai = np.concatenate((tab_y_vrai, y_vrai))
    tree_tab_y_pred = np.concatenate((tree_tab_y_pred, clf.predict(X_artist)))

    tree_predict.append(clf.score(X_artist, y_vrai))
    print(round(clf.score(X_artist, y_vrai),2), name[13:-4])

print('\n La moyenne du score des prédictions est de : ', round(np.
  ↪mean(tree_predict),3))
```

```
tree_mean = np.mean(tree_predict)
```

```
0.78 acdc
0.92 bach
0.91 drake
0.82 rollingstones
0.26 deadmau5
0.95 brahms
0.11 daftpunk
0.95 cardib
0.08 aphextwins
0.87 kendricklamar
0.89 travisscott
0.82 beatles
1.0 drpeacock
0.1 kraftwerk
0.26 chemicalbrothers
0.22 majorlazer
0.91 chopin
0.84 queen
0.92 jcole
0.88 ledzepplin
0.93 mozart
0.97 beethoven

 La moyenne du score des prédictions est de :  0.699
```

1.3.4 2 - Random Forest

Un autre algorithme couramment utilisé dans les problèmes similaires est l'algorithme Random Forest. Cette méthode combine les résultats de plusieurs arbres de décision pour améliorer la précision de la classification.

Entrainement/Cross Validation : La paramètre déterminant pour cet algorithme est donc le nombre optimal d'arbres pour obtenir les meilleurs résultats.

Ainsi, nous pouvons utiliser une validation croisée pour sélectionner le nombre d'arbres optimal, et nous assurer que notre modèle est bien adapté à nos données.

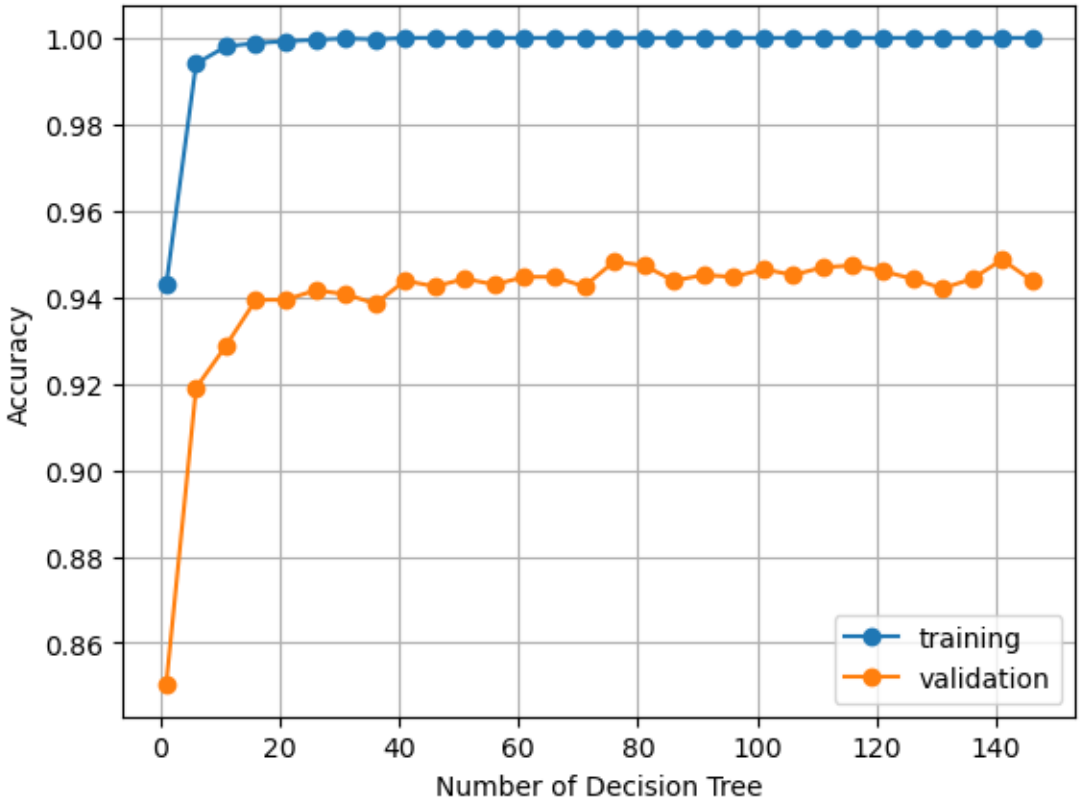
[21]:
```
clf_forest = RandomForestClassifier()
```

[22]:
```
cv = KFold(n_splits=5, shuffle=True)
nb_tree = np.arange(1,150,5)
```

[23]:
```
train_score, val_score = validation_curve(clf_forest, X_train,
                                          y_train,
                                          param_name='n_estimators',
```

```
                                                param_range = nb_tree,
                                                cv = cv)
```

[24]:
```
plt.plot(nb_tree,train_score.mean(axis=1),'o-',label='training')
plt.plot(nb_tree,val_score.mean(axis=1),'o-',label='validation')
plt.xlabel('Number of Decision Tree')
plt.ylabel('Accuracy')
plt.grid()
plt.legend()
plt.show()
```



A partir de 40 arbres, les performances se stabilisent. Ainsi nous fixons pour notre entraînement n_estimators à 50.

[25]:
```
clf_forest = RandomForestClassifier(n_estimators=50)
clf_forest = clf_forest.fit(X_train,y_train)
```

[26]:
```
print('Train score :',clf_forest.score(X_train,y_train))
print('Test score :',clf_forest.score(X_test,y_test))
```

```
Train score : 1.0
Test score : 0.9487632508833922
```

Prédiction Random Forest : On obtient les prédictions suivantes.

[27]:
```
csv_files = glob.glob('data/artists/*.csv')
```

[28]:
```
randf_predict = []
randf_tab_y_vrai = np.array([])
randf_tab_y_pred = np.array([])

for name in csv_files:
    data_artist = normalisation(pd.read_csv(f'{name}'))

    y_vrai = data_artist['genre']
    X_artist = data_artist.drop(['genre'],axis=1)

    randf_tab_y_pred = np.concatenate((randf_tab_y_pred, clf_forest.
 ↪predict(X_artist)))

    randf_predict.append(clf_forest.score(X_artist, y_vrai))
    print(round(clf_forest.score(X_artist, y_vrai),2), name[13:-4])

print('\n La moyenne du score des prédictions est de : ', round(np.
 ↪mean(randf_predict),3))
randf_mean = np.mean(randf_predict)
```

```
0.92 acdc
0.96 bach
0.93 drake
0.98 rollingstones
0.32 deadmau5
1.0 brahms
0.09 daftpunk
0.93 cardib
0.06 aphextwins
0.88 kendricklamar
0.95 travisscott
0.96 beatles
1.0 drpeacock
0.04 kraftwerk
0.14 chemicalbrothers
0.08 majorlazer
1.0 chopin
0.92 queen
0.98 jcole
0.96 ledzepplin
1.0 mozart
1.0 beethoven

 La moyenne du score des prédictions est de :  0.732
```

1.3.5 3 - K-Nearest Neighbors

KNN (K-Nearest Neighbors) est un algorithme d'apprentissage supervisé utilisé pour résoudre des problèmes de classification ou de régression. Il est basé sur la distance entre les échantillons dans un espace de caractéristiques, où chaque caractéristique représente une variable mesurée pour chaque échantillon.

L'algorithme KNN fonctionne en recherchant les k voisins les plus proches d'un échantillon inconnu dans l'espace de caractéristiques. Ensuite, la classe majoritaire parmi ces k voisins est assignée à l'échantillon inconnu. Ainsi, comme évoqué dans la data visualisation, on soupçonne cet algorithme d'être moins performant pour nos données notamment lorsqu'il s'agit de différencier du rap et du rock.

Avec cette difficulté il est d'autant plus important d'effectuer une cross validation afin de connaître le nombre k de voisins optimal pour nos données.

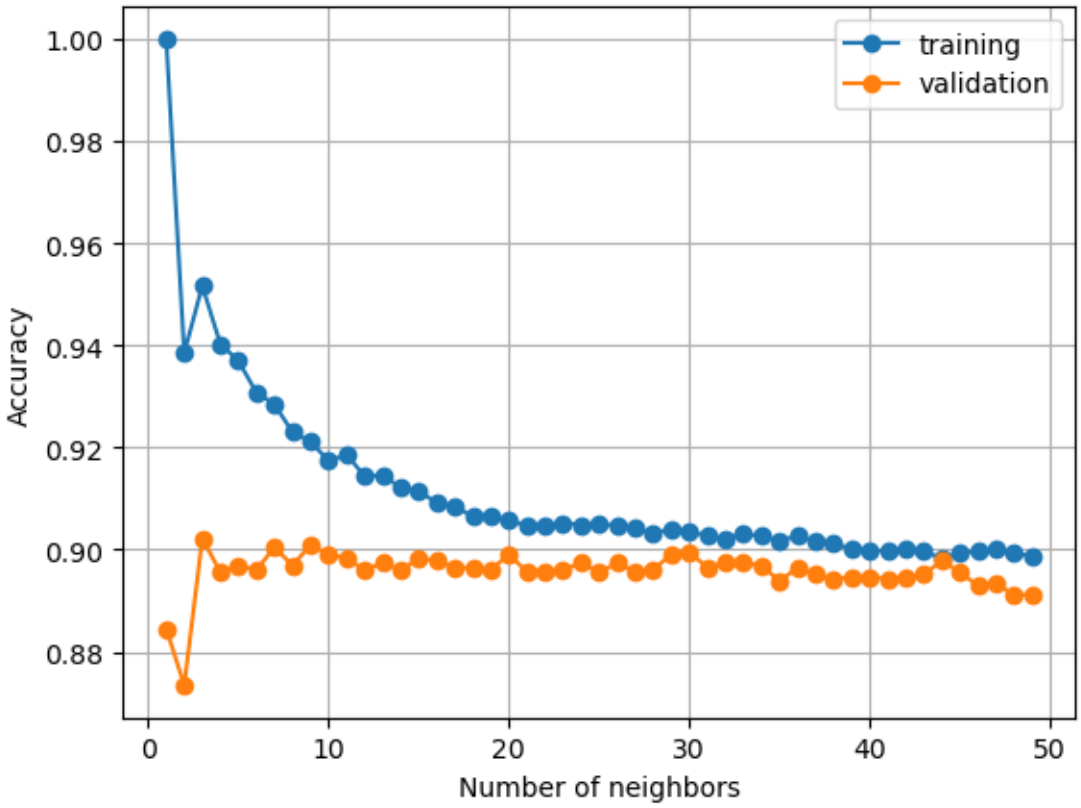
Entrainement/Cross Validation :

[29]:
```
knn = KNeighborsClassifier()
```

[30]:
```
cv = KFold(n_splits=5, shuffle=True)
nb_neighbors = np.arange(1,50,1)
```

[31]:
```
train_score, val_score = validation_curve(knn, X_train,
                                          y_train,
                                          param_name='n_neighbors',
                                          param_range = nb_neighbors,
                                          cv = cv)
```

[32]:
```
plt.plot(nb_neighbors,train_score.mean(axis=1),'o-',label='training')
plt.plot(nb_neighbors,val_score.mean(axis=1),'o-',label='validation')
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.grid()
plt.legend()
plt.show()
```



Le nombre optimal de voisin se situe alors entre 3 et 10. On décide de le fixer à 5 pour notre entraînement.

[33]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn = knn.fit(X_train,y_train)
```

[34]:
```
print('Train score :',knn.score(X_train,y_train))
print('Test score :',knn.score(X_test,y_test))
```

```
Train score : 0.9451327433628318
Test score : 0.9204946996466431
```

Prediction K-NN : On obtient les prédictions suivantes.

[35]:
```
csv_files = glob.glob('data/artists/*.csv')
```

[36]:
```
knn_predict = []
knn_tab_y_vrai = np.array([])
knn_tab_y_pred = np.array([])

for name in csv_files:
```

```
    data_artist = normalisation(pd.read_csv(f'{name}'))

    y_vrai = data_artist['genre']
    X_artist = data_artist.drop(['genre'],axis=1)

    knn_tab_y_pred = np.concatenate((knn_tab_y_pred, knn.predict(X_artist)))
    knn_predict.append(knn.score(X_artist, y_vrai))

    print(round(knn.score(X_artist, y_vrai),2), name[13:-4])

print('\n La moyenne du score des prédictions est de : ', round(np.
  ↪mean(knn_predict),3))
knn_mean = np.mean(knn_predict)
```

```
0.9 acdc
0.84 bach
0.91 drake
0.86 rollingstones
0.36 deadmau5
1.0 brahms
0.18 daftpunk
0.91 cardib
0.12 aphextwins
0.87 kendricklamar
0.91 travisscott
0.88 beatles
0.94 drpeacock
0.14 kraftwerk
0.26 chemicalbrothers
0.14 majorlazer
1.0 chopin
0.73 queen
0.94 jcole
0.94 ledzepplin
0.93 mozart
0.97 beethoven

 La moyenne du score des prédictions est de :  0.714
```

1.3.6 4 - Réseau de neurones

Enfin, la dernière approche que nous avons tenu à tester est un réseau de neurones. Le nombre de couches et le nombre de neurones ont été fixés de manière empirique, tandis que la fonction d'actictivation de sortie 'softmax' est particulièrement adaptée pour les problèmes de classification non-binaire.

Entrainement On utilise un encodage one-hot pour convertir le genre en donnée numérique.

[37]:
```
y_train_neuro = y_train.replace(['rock', 'rap', 'classique', 'electro'],␣
↪[0,1,2,3])
y_test_neuro = y_test.replace(['rock', 'rap', 'classique', 'electro'],␣
↪[0,1,2,3])
```

[38]:
```
y_train_neuro = keras.utils.to_categorical(y_train_neuro)  #One hot encodage␣
↪des solutions
y_test_neuro = keras.utils.to_categorical(y_test_neuro)  #One hot encodage des␣
↪solutions
```

[39]:
```
model = keras.Sequential()

model.add(keras.layers.Dense(16, activation='sigmoid', input_dim=14))
model.add(keras.layers.Dense(8, activation='sigmoid'))
model.add(keras.layers.Dense(4, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[40]:
```
model.summary()
```

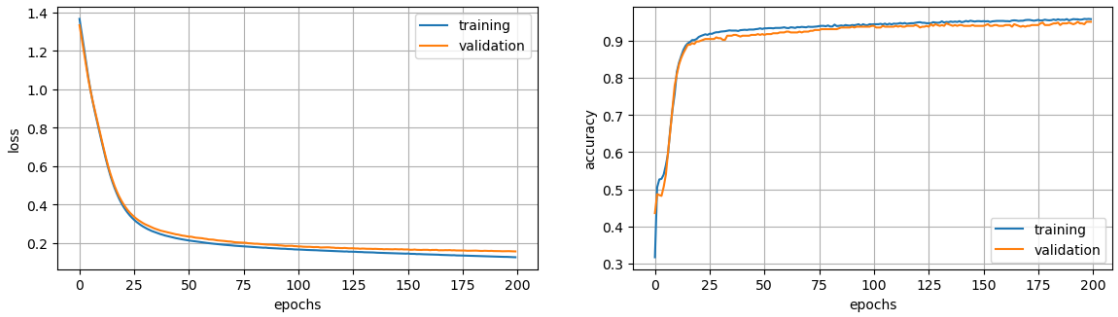
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                240

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 4)                 36

=================================================================
Total params: 412
Trainable params: 412
Non-trainable params: 0
_________________________________________________________________
```

[41]:
```
history = model.fit(X_train, y_train_neuro,
          batch_size=32,
          epochs=200,
          verbose=0,
          shuffle=True,
          validation_split=0.2) # % of data being used for validation_loss␣
↪evaluation
```

Après l'entraînement du réseau de neurones, les courbes de loss (ou courbes de perte) permettent

d'évaluer la performance du modèle au fil des epochs. Ces courbes représentent l'évolution de la fonction de coût (loss function) pendant l'apprentissage, qui mesure l'écart entre les prédictions du modèle et les valeurs réelles. L'analyse de ces courbes permet d'identifier les éventuels problèmes de sur-apprentissage (overfitting) ou de sous-apprentissage (underfitting) du modèle, ainsi que de sélectionner le nombre optimal d'epochs pour l'entraînement.

[42]:
```
plt.figure(figsize=(14,3.5))

plt.subplot(1,2,1)
plt.plot(history.history['loss'], label='training')
plt.plot(history.history['val_loss'], label='validation')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.grid()
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], label='training')
plt.plot(history.history['val_accuracy'], label='validation')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.grid()
plt.show()
```



[43]:
```
evaluation_train_set = model.evaluate(X_train, y_train_neuro)
```

```
71/71 [==============================] - 0s 837us/step - loss: 0.1300 -
accuracy: 0.9575
```

[44]:
```
evaluation_test_set = model.evaluate(X_test,y_test_neuro)
```

```
18/18 [==============================] - 0s 857us/step - loss: 0.1249 -
accuracy: 0.9505
```

Prédiction du réseau de neurones On obtient les prédictions suivantes.

[45]:
```
csv_files = glob.glob('data/artists/*.csv')
```

[46]:
```
neural_predict = []
neural_tab_y_vrai = np.array([])
neural_tab_y_pred = np.array([])

for name in csv_files:

    data_artist = normalisation(pd.read_csv(f'{name}'))
    y_artist = data_artist['genre'].replace(['rock', 'rap', 'classique',␣
 ↪'electro'], [0,1,2,3])

    neural_y_pred = model.predict(data_artist.drop(['genre'],axis=1),verbose=0)
    neural_y_pred = np.argmax(neural_y_pred,axis=1)

    neural_tab_y_vrai = np.concatenate((neural_tab_y_vrai, y_artist))
    neural_tab_y_pred = np.concatenate((neural_tab_y_pred, neural_y_pred))

    neural_predict.append(sklearn.metrics.accuracy_score(neural_y_pred,␣
 ↪y_artist))
    print(round(sklearn.metrics.accuracy_score(neural_y_pred, y_artist),2),␣
 ↪name[13:-4])

print('\n La moyenne du score des prédictions est de : ', round(np.
 ↪mean(neural_predict),3))
neural_mean = np.mean(neural_predict)
```

```
0.86 acdc
0.85 bach
0.91 drake
0.9 rollingstones
0.4 deadmau5
1.0 brahms
0.07 daftpunk
0.95 cardib
0.12 aphextwins
0.92 kendricklamar
0.95 travisscott
0.96 beatles
0.98 drpeacock
0.02 kraftwerk
0.18 chemicalbrothers
0.1 majorlazer
1.0 chopin
0.84 queen
0.92 jcole
```

```
1.0 ledzepplin
0.9 mozart
1.0 beethoven

 La moyenne du score des prédictions est de :  0.719
```

[47]:
```
#model.save('neural_music')
```

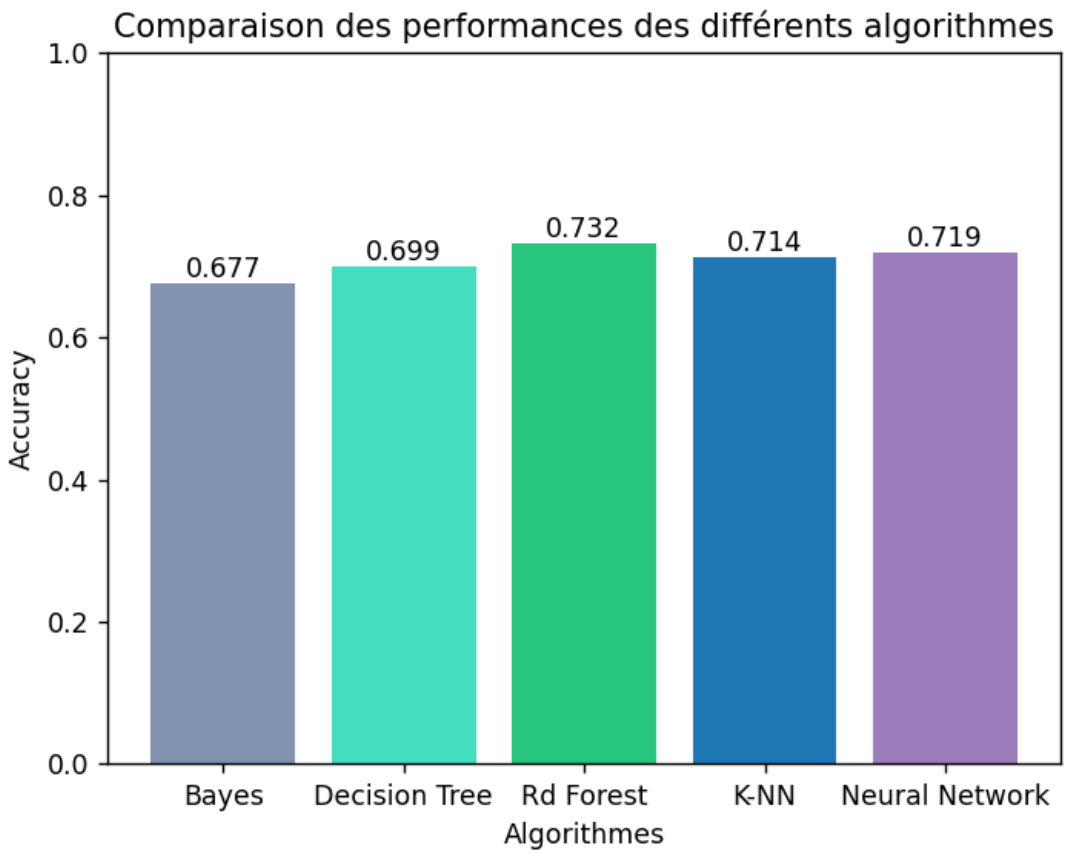
1.4 IV. Comparaison des modèles et discussion

Maintenant que nous avons présenté nos différents modèles de classification de genres musicaux, nous allons les comparer et analyser leurs performances respectives. Cette étape est essentielle pour déterminer le modèle le plus adapté à notre problématique, et ainsi en tirer des conclusions pertinentes.

Nous pouvons dans un premier temps évaluer le score et la performance global de chaque algorithme sur nos artistes.

[48]:
```
accuracies = [bayes_mean,tree_mean,randf_mean,knn_mean,neural_mean]
graph = ['Decision Tree','Rd Forest','K-NN','Neural Network']
colors = ['#8191b0','#42debf', '#28c67c', '#1f77b4', '#9c7cba']
```

[49]:
```
plt.figure(dpi = 125)
plt.bar(['Bayes'] + graph, accuracies,color=colors)
plt.ylim(0, 1)
plt.title("Comparaison des performances des différents algorithmes")
for i, score in enumerate(accuracies):
    plt.text(i, score, str(round(score,3)), ha='center', va='bottom')
plt.xlabel("Algorithmes")
plt.ylabel("Accuracy")
plt.show()
```



Nous voyons que les algorithmes ont des scores qui tournent aux alentours de 70%. La comparaison du score global nous permet d'avoir une vision d'ensemble de la performance de chaque modèle, en considérant l'ensemble des genres musicaux. Cette analyse nous permet d'identifier les modèles les plus performants de manière générale, mais ne nous donne pas d'informations détaillées sur la performance de chaque modèle pour chaque genre musical. Les scores globaux n'étant pas très élevés, nous pouvons nous demander si certains genres ne sont pas plus durs à prédire que d'autres.

Ainsi, la comparaison du score global nous donne une première indication sur la capacité de chaque modèle à prédire les genres musicaux, mais pour une analyse plus fine, il est nécessaire de zoomer sur les scores par genre, afin de déterminer la performance de chaque modèle pour chaque genre musical en particulier. Cela peut nous aider à identifier les modèles qui ont des difficultés à prédire certains genres, et à orienter nos efforts d'optimisation en conséquence.

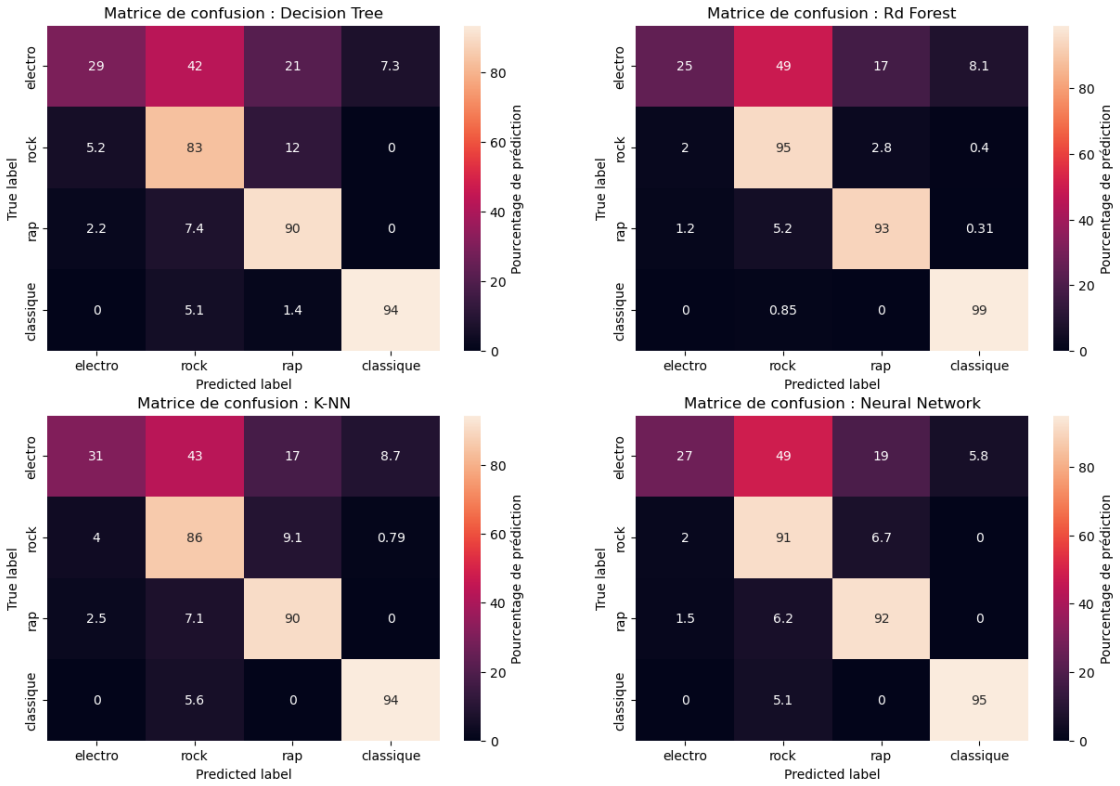
De plus, nous avons vu dans la data visualisation que les faibles distances euclidiennes entre le rock, le rap, et l'électro risquaient de nous poser problème.

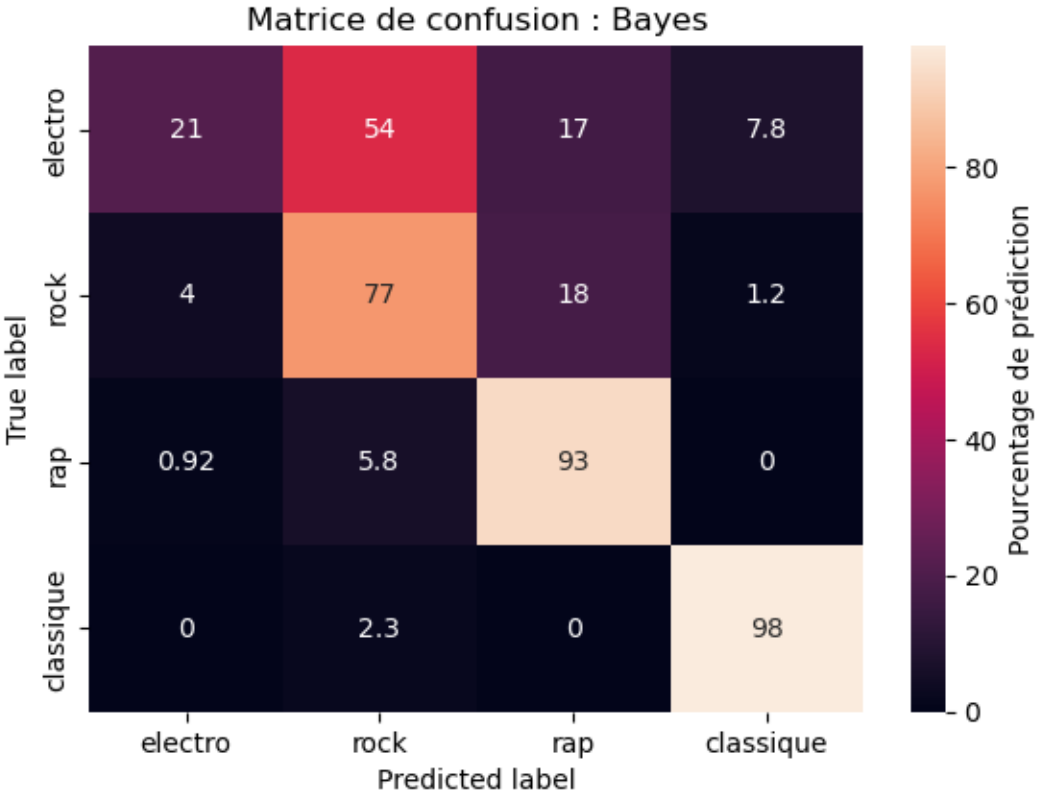
Dans une seconde partie, il est alors pertinent de zoomer sur la performance des algorithmes en fonction du genre. Pour cela, nous avons tracer les matrices de confusion pour chaque méthode.

[50]:
```
neural_tab_y_pred = pd.DataFrame(neural_tab_y_pred).replace([0,1,2,3],['rock',␣
↪'rap', 'classique', 'electro'])
```

[51]:
```
name = [tree_tab_y_pred, randf_tab_y_pred, knn_tab_y_pred, neural_tab_y_pred]
```

[54]:
```
plt.figure(figsize=(15,10))
labels = ['electro','rock','rap','classique']

for i,algo in enumerate(name):

    y_pred = algo

    conf = confusion_matrix(tab_y_vrai, y_pred,
                                labels = labels)
    conf = conf / conf.sum(axis= 1)[:, np.newaxis]*100 # Affichage en␣
 ↪pourcentages

    plt.subplot(2,2,i+1)
    sns.heatmap(conf, annot = True,
                xticklabels=labels,
                yticklabels=labels,
                cbar_kws = {'label': 'Pourcentage de prédiction'})

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title(f'Matrice de confusion : {graph[i]}')

conf = confusion_matrix(tab_y_vrai, bayes_tab_y_pred,labels = labels)
conf = conf / conf.sum(axis= 1)[:, np.newaxis]*100 # Affichage en pourcentages

plt.figure(figsize=(6.5,4.5))
sns.heatmap(conf, annot = True,
            xticklabels=labels,
            yticklabels=labels,
            cbar_kws = {'label': 'Pourcentage de prédiction'})
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title('Matrice de confusion : Bayes')
plt.show()
```


Matrice de confusion : Decision Tree
29 42 21 7.3
5.2 83 12 0
2.2 7.4 90 0
0 5.1 1.4 94
Matrice de confusion : Rd Forest
25 49 17 8.1
2 95 2.8 0.4
1.2 5.2 93 0.31
0 0.85 0 99
Matrice de confusion : K-NN
31 43 17 8.7
4 86 9.1 0.79
2.5 7.1 90 0
0 5.6 0 94
Matrice de confusion : Neural Network
27 49 19 5.8
2 91 6.7 0
1.5 6.2 92 0
0 5.1 0 95
electro
rock
rap
classique
True label
Predicted label
Pourcentage de prédiction
0
20
40
60
80


Matrice de confusion : Bayes
21 54 17 7.8
4 77 18 1.2
0.92 5.8 93 0
0 2.3 0 98
electro
rock
rap
classique
True label
Predicted label
Pourcentage de prédiction
0
20
40
60
80

Nous voyons que tous les algorithmes commettent, sur certaines chansons, la confusion entre rock et rap. Néanmoins, pour l'algorithme de RandomForest cette erreur est moins importante. Ainsi, comme il s'agit d'un algorithme moins affecté que les autres par les distances entre les echantillons il est normal qu'il obtienne de meilleur résultat que les autres sur ces deux genres.

Ensuite, nous observons la présence d'une erreur systématique dans nos algorithmes. Les perfomances sont très mauvaises pour l'électro. Contrairement à la confusion rock/rap cette erreur de par son importance, ne peut être seulement due à une proximité entre les caractéristiques acoustiques. En effet, on peut supposer un biais dans nos données. L'électro comportant beaucoup de sous-genres et étant lui même un genre qui se distingue par des caractéristique sonores très particulière nous pouvons nous interroger sur le choix des musiques d'entraînelent pour ce genre. Il est possible qu'il ne soit pas représentatif du genre et assez diversifié.

Une piste d'amélioration future pourrait être de collecter plus de données pour le genre 'électro', en veillant à la diversité et à la représentativité des musiques d'entraînement.

Pour améliorer la distinction rap/rock, une piste serait de combiner différents algorithmes de classification, qui ont montré des performances satisfaisantes sur ces deux genres musicaux en particulier. Par exemple, on pourrait utiliser un modèle de réseau de neurones en combinaison avec un modèle de Random Forest.

Une autre piste serait de travailler sur les caractéristiques des chansons utilisées pour l'entraînement des algorithmes. Il est possible que certaines caractéristiques acoustiques soient plus discriminantes pour le rock et le rap que pour les autres par exemple. Il serait donc intéressant de mener une analyse plus fine de ces caractéristiques, afin de les identifier et de les utiliser pour entraîner nos modèles en leur augmentant leur poids lorsqu'on a identifié que le morceau passé avait de grandes chances d'être du rock ou du rap.