Python for data analysis

Final Project

Diane DU PELOUX
Mohammad Reza ZOHRABI

ESILV - A5
2020-21

Plan

I. Introduction

   A. Présentation du dataset
   B. Création de notre Subset
   C. Problèmes rencontrés

II. Visualisation et Problématiques

   A. Problématiques
   B. Questions / Réponses
   C. Réduction dimensionnelle pour la visualisation (ACP et T-SNE)

III. Classification

IV. API

Introduction

Présentation du dataset (½)

Pour notre étude, nous avons pour sujet le dataset “Million Song”. Nous pourrons décrire ce jeu de données par la citation officielle suivante :

“Le dataset Million Song est une collection gratuite de fonctionnalités audio et de métadonnées pour un million de morceaux de musique populaire contemporaine.

Ses objectifs sont les suivants :

- Encourager la recherche sur des algorithmes de taille commerciale
- Fournir un ensemble de données de référence pour l’évaluation de la recherche
- Comme raccourci alternatif à la création d’un grand ensemble de données avec des API (par ex. The Echo Nest)
- Pour aider les nouveaux chercheurs à démarrer dans le domaine MIR”

Le cœur de l’ensemble de données est l’analyse des caractéristiques et les métadonnées d’un million de chansons, fournies par The Echo Nest. L’ensemble de données n’inclut pas d’audio, seulement les fonctionnalités dérivées.

http://millionsongdataset.com/

Présentation du dataset (2/2)

Dans notre dataset nous pouvons retrouver par champs :

1. [valeur 1] L'année cible allant de 1922 à 2011
2. [valeur 2 à 14] TimbreAverage(1 à 12)
3. [valeur 15 à 91] TimbreCovariance(1 à 78)

Ces fonctionnalités ont été extraites des fonctionnalités 'timbre' de l'API Echo Nest. Les auteurs ont pris la moyenne et la covariance sur tous les «segments» et chaque segment a été décrit par un vecteur de timbre à 12 dimensions. Le timbre des signaux audio n'a pas été clairement défini mathématiquement. On a émis l'hypothèse que la structure temps-fréquence des signaux audio contribue à la propriété du timbre. Dans certains articles, la matrice de covariance à partir des signaux de sortie de banque de filtres multi-bandes d'un signal audio est réalisée et le filtre Common Spatial Pattern (CSP) est appliqué pour caractériser le timbre des signaux audio. Les résultats des simulations confirment que la matrice de covariance des signaux audio multibandes et du filtre CSP peut être utilisée comme caractéristique potentielle de la classification des timbres.

Création de notre subset

Pour notre subset, nous allons utiliser le dataset mis à disposition par Million Song et lui ajouter les headers correspondant. Nous avons décidé de conserver toutes les données malgré leur nombre important, nos ordinateurs ne rencontrant aucun problème de compilation chronophage..

Pour certaines de nos visualisations, nous utiliserons seulement la Moyenne de chaque timbre pour observer la corrélation des timbres avec l'année de sortie d'une musique.

Problèmes rencontrés

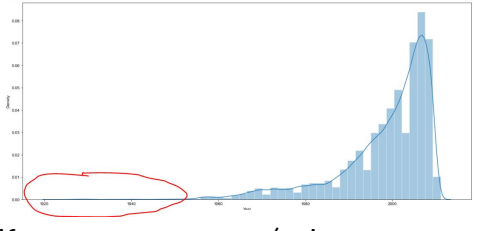
Ci-dessous la liste des problèmes rencontrées et leur solution dans la mise en place d'un dataset "propre" :

- La non présence de headers que nous avons dû recréer à la main :

```
# ajout des headers en fonction des données.
header_tab =["Year","TimbreAvg1","TimbreAvg2","TimbreAvg3","TimbreAvg4","TimbreAvg5","TimbreAvg6","TimbreAvg7"

#music = pd.read_csv("/content/drive/MyDrive/FinalProject_PythonAnalysis/YearPredictionMSD.csv", header=None, names= header_tab)
#si vous n'utilisez pas drive
music = pd.read_csv("YearPredictionMSD.csv", header=None, names= header_tab)
```

- Le manque de données récoltées pour les musiques remontant avant les années 50. Nous les avons tout simplement enlevées du dataset.



- L'échelle des timbres allant jusque dans les négatifs, que nous avons ré-ajuster pour que nos données soient indiquées entre 0 et 1. Cela nous permet de préparer les données qui n'ont pas forcément les mêmes échelles.

```
music.iloc[:,1:] = (music.iloc[:,1:]-music.iloc[:,1:].min())/(music.iloc[:,1:].max() - music.iloc[:,1:].min())
music.iloc[:,1:].describe()
```

	TimbreAvg1	TimbreAvg2	TimbreAvg3	TimbreAvg4	TimbreAvg5	TimbreAvg6	TimbreAvg7	TimbreAvg8	TimbreAvg9	TimbreAvg
count	515345.000000	515345.000000	515345.000000	515345.000000	515345.000000	515345.000000	515345.000000	515345.000000	515345.000000	515345.0000
mean	0.691420	0.469220	0.496370	0.317065	0.395025	0.291364	0.515292	0.354893	0.477336	0.4267
std	0.100755	0.071524	0.056533	0.033315	0.051486	0.051839	0.040408	0.039970	0.038797	0.0640
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	0.634423	0.431296	0.464117	0.297366	0.363241	0.255425	0.492028	0.331428	0.455263	0.3842
50%	0.705890	0.479105	0.499284	0.313357	0.396254	0.284665	0.516247	0.355178	0.477685	0.4257
75%	0.755261	0.517524	0.530202	0.332624	0.427220	0.320143	0.539971	0.378515	0.500192	0.4685
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000

8 rows × 90 columns

Visualisation et Problématiques

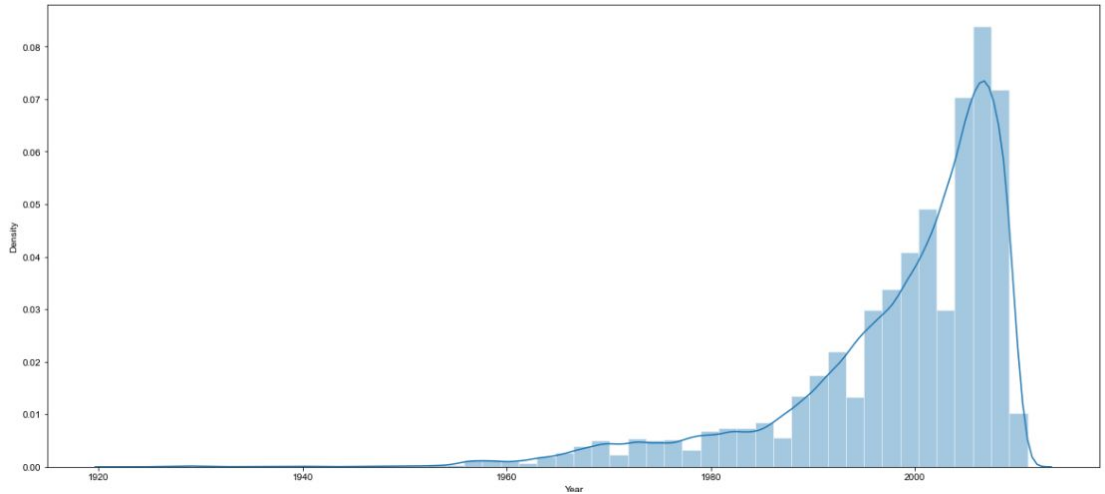
Problématiques

Pour en connaître plus sur notre dataset nous nous poserons les questions suivantes et y répondons par différents moyens de visualisation :

1. Quelle est la répartition du nombre de musiques par année (à quelle point notre base de données sera efficiente pour nos futurs modèles?) ?
2. Comment sont répartis les différents timbres moyens ? A quoi correspondent-ils?
3. Comment est réparti un timbre moyen sur plusieurs années (100 ans) ?
4. Quelle est la corrélation entre chaque timbre moyen ?
5. Estimation de la corrélation entre timbre et décennie qui nous permettra de nous diriger vers notre problémtique finale : Réussir à définir l'année (décennie) de sortie d'une chanson en fonction des timbres dont elle est constituée.

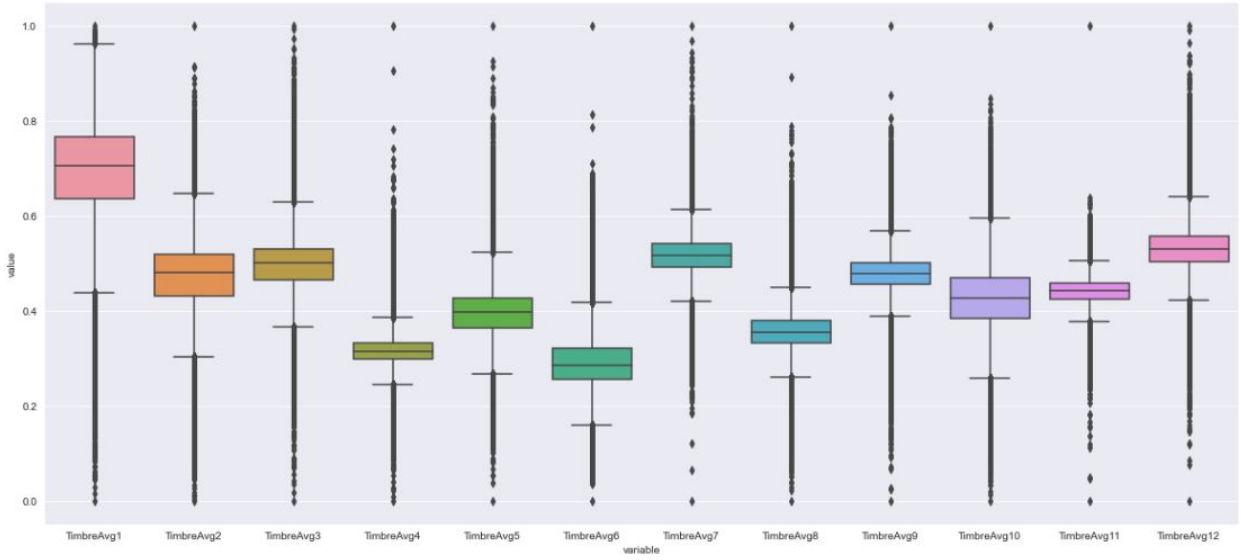
Questions et Réponses

Quelle est la répartition du nombre de musiques par année (à quelle point notre base de données sera efficiente pour nos futurs modèles?)?



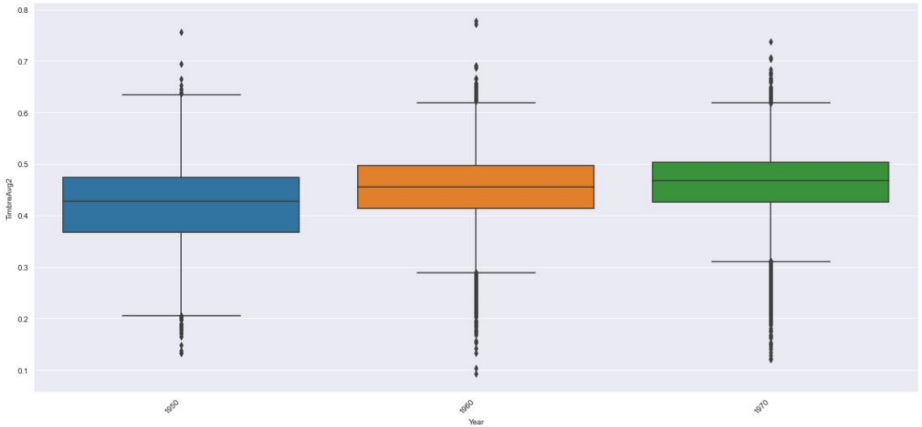
Réponse : Nous observons que la répartition des musiques récoltées en fonction de leur date de sortie n'est pas uniforme dû à la plus grande production musicale aujourd'hui et sûrement au manque de données récupérées par MSD. Nous verrons si ces différences amènent à des modèles moins efficients.

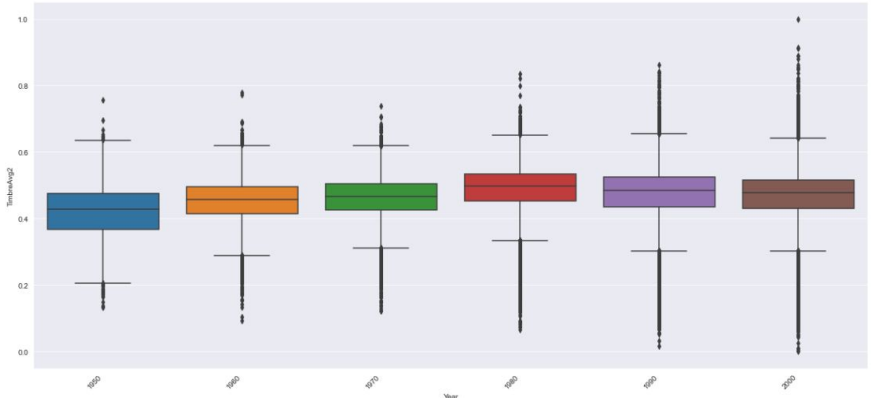
Comment sont répartis les différents timbres moyens ? A quoi correspondent-ils?



Réponse : Le timbre distingue différents types de production sonore, comme les voix de chœur et les instruments de musique, comme les instruments à cordes, les instruments à vent et les instruments à percussion. Sur le graphique ci-dessus, nous pouvons voir que les timbres utilisés ont des valeurs moyenne proches et sont donc tout aussi présents au travers des années. Leur présence plus ou moins élevée par musique nous permettra sûrement de la caractériser et de lui donner une appartenance à une décennie. Une musique Pop n'aura sûrement pas les même timbres qu'une musique Jazz, ce qui nous permettra de les différencier. Et même deux musiques de genre Jazz auront quelques différences en fonction de leur décennie de sortie grâce à l'évolution des instruments utilisés.

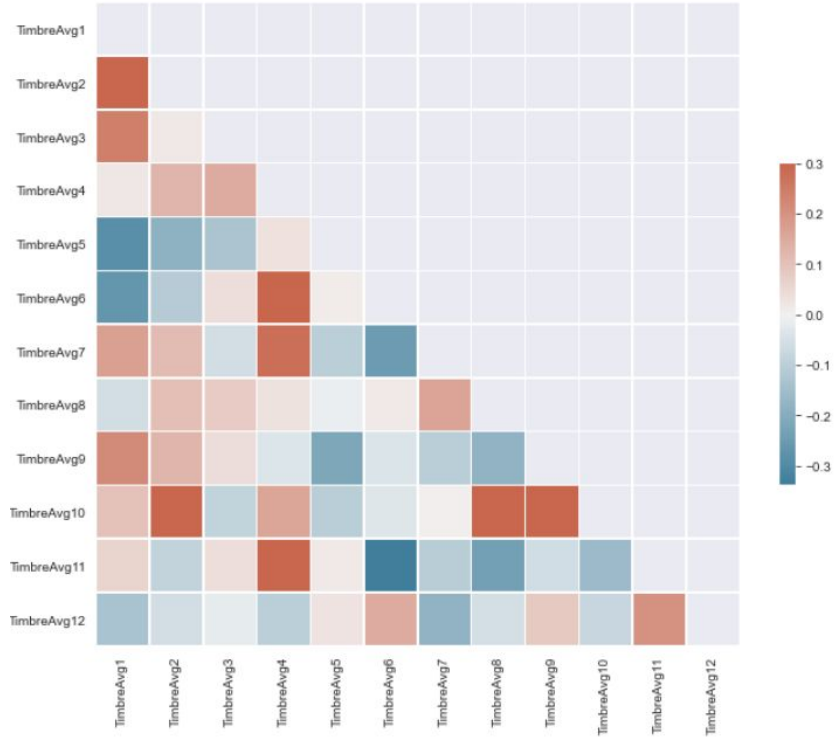
Comment est réparti un timbre moyen sur plusieurs années (100 ans) ?





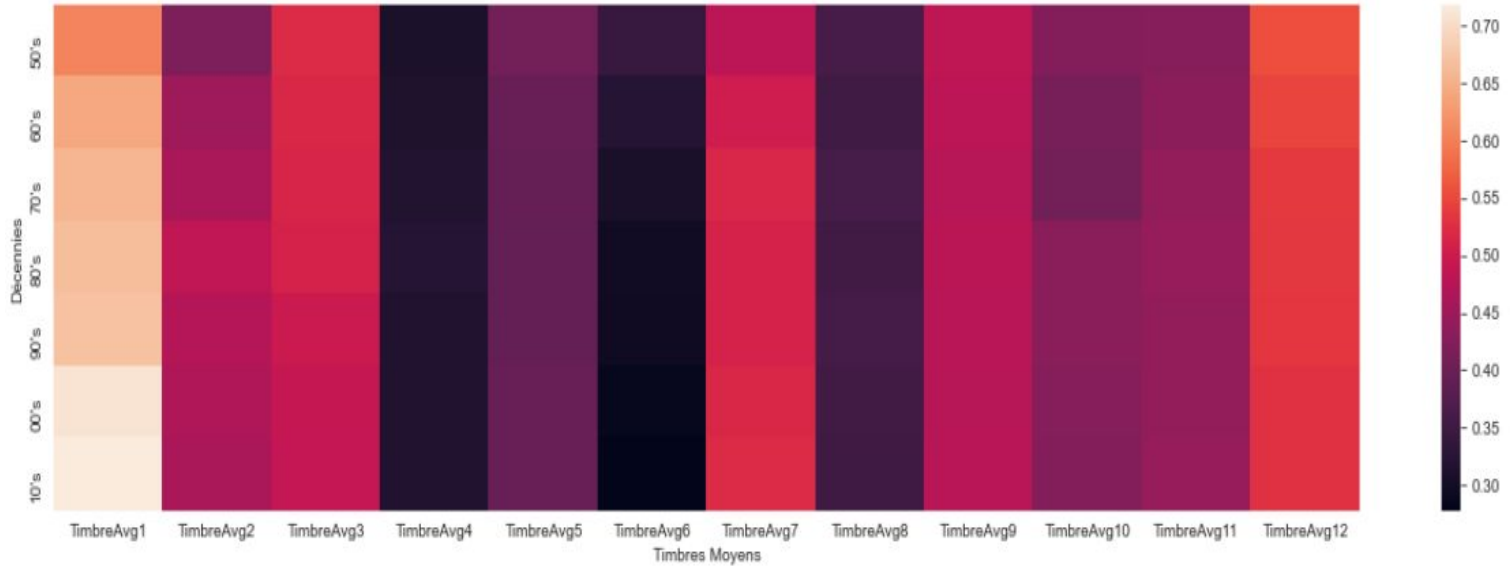
Réponse : La répartition du timbre moyen 2 confirme l'hypothèse que nous avons suggéré lors de la réponse à la question précédente. On observe qu'il est beaucoup plus présent dans les dernières sorties musicales et moins avant 1970. Sa présence est un bon indicateur, mais nous allons devoir vérifier cela pour tous les timbres avec une grille de corrélation.

Quelle est la corrélation entre chaque timbre moyen ?



Réponse : Nous observons qu'il y a effectivement une corrélation entre les différents timbres, et que celle-ci sont disparates. Nous pouvons faire l'hypothèse qu'une certaine association de timbres sera caractéristique d'une décennie.

Estimation de la corrélation entre timbre et décennie qui nous permettra de nous diriger vers notre problémtique finale : Réussir à définir l'année (décennie) de sortie d'une chanson en fonction des timbres dont elle est constituée.



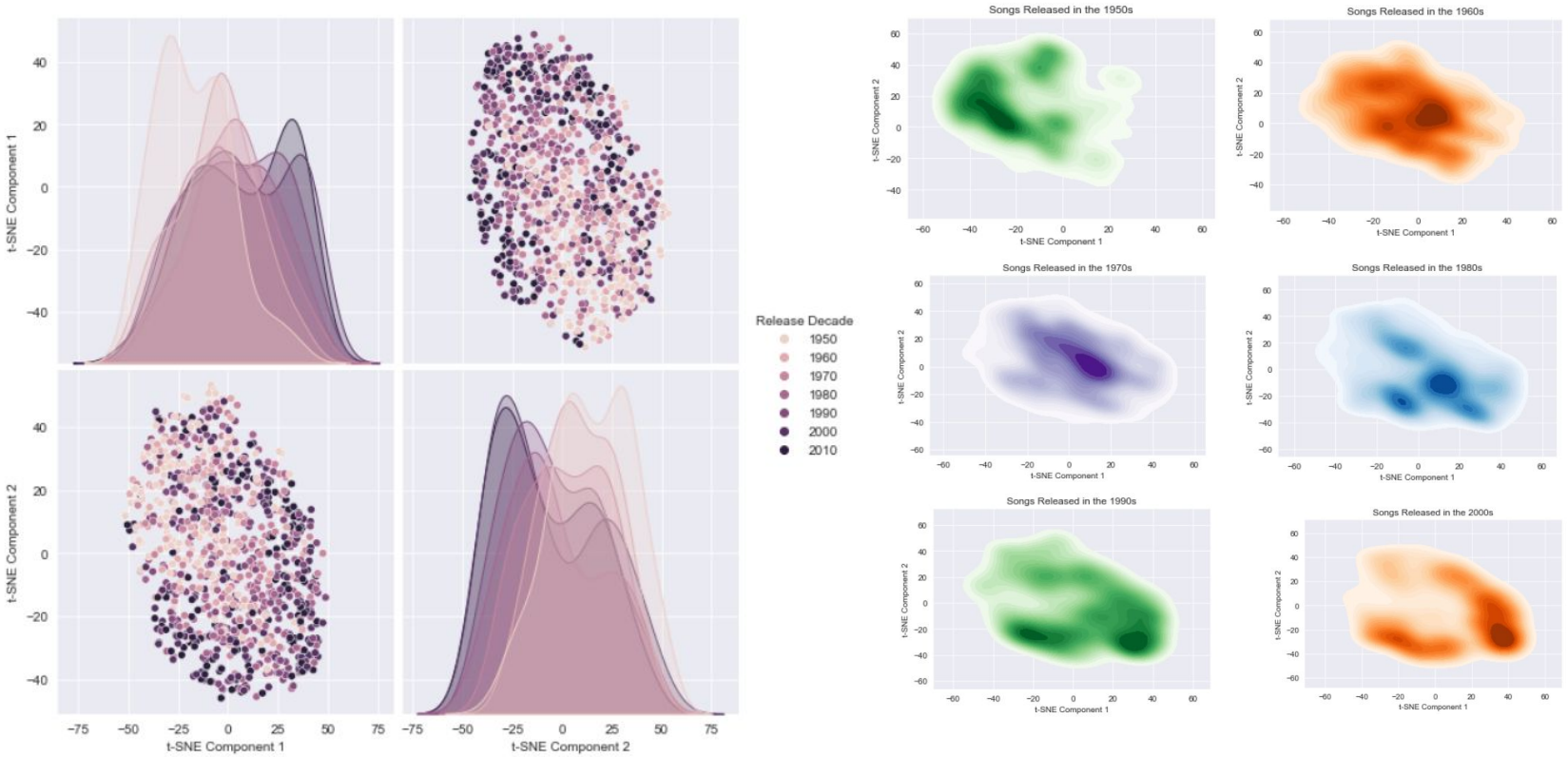
Réponse : Nous voyons de fortes corrélations entre la majorité des timbres et les années. Une alliance entre plusieurs timbres pourront donc effectivement déterminer la décennie d'appartenance d'une musique.

Réduction dimensionnelle pour la visualisation (ACP et T-SNE)

1. L’ACP constitue la base de l’analyse de données multivariées fondée sur des méthodes de projection. L’utilisation la plus importante de l’ACP est de représenter un tableau de données multivariées en tant que petit ensemble de variables (indices récapitulatifs) afin d’observer les tendances et les valeurs inattendues. Cet aperçu peut révéler les relations entre les observations et les variables, ce qui nous permettra de confirmer les hypothèses que nous avons énoncés lors de notre parcours du questionnaire autour de MSD.

2. t-SNE (t-distributed stochastic neighborhood embedding) est également une technique non supervisée de réduction dimensionnelle non linéaire et de visualisation des données. Il intègre les points d’une dimension supérieure à une dimension inférieure en essayant de préserver le voisinage de ce point.

   Contrairement à l’ACP, elle tente de préserver la structure locale des données en minimisant la divergence Kullback-Leibler (divergence KL) entre les deux distributions en ce qui concerne l’emplacement des points de la carte. Ainsi l’ACP permet de visualiser la structure générale des données, et informe sur les variables les plus discriminantes (ce que nous n'observons pas particulièrement sur le graphique ci-dessus). La représentation t-SNE permet, elle, d’identifier les groupes d’observations proches, et allie un début de classification automatique à la réduction de dimensionnalité.



Les graphiques que nous avons produits sont significatifs et montrent que par une réduction dimensionnelle, les composantes principales différencient les chansons par leur année de sortie mais pas avec une grande différence. Quelques détails montrent des ressemblance entre année de sortie, ce qui nous empêche d'identifier avec assurance l'année de sortie d'une musique.

Classification

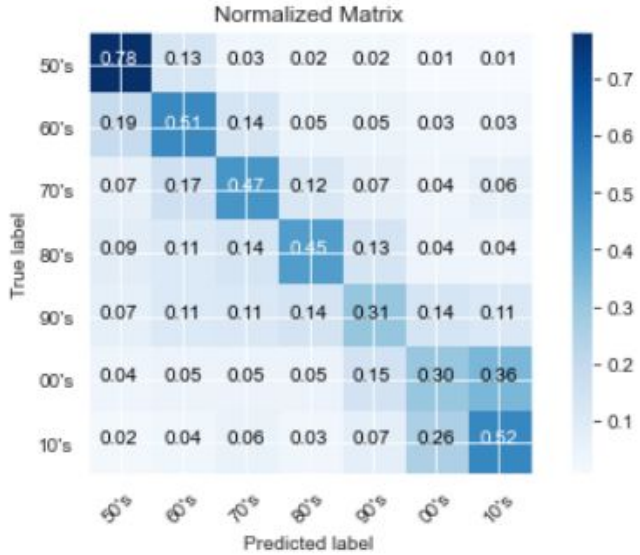
```
tst = music_test
X_test = tst.iloc[:,1:].values
y_test = tst.iloc[:,0].values
expected = y_test
predicted = clf.predict(X_test)
print("Classification  %s:\n%s\n"
      % (clf, metrics.classification_report(expected, predicted)))
cnf_matrix = metrics.confusion_matrix(expected, predicted)
```

```
Classification  SVC(C=10, gamma=5):
              precision    recall  f1-score   support

        1950       0.62      0.78      0.69       956
        1960       0.47      0.51      0.49       977
        1970       0.46      0.47      0.47       908
        1980       0.52      0.45      0.48       948
        1990       0.39      0.31      0.35       940
        2000       0.36      0.30      0.33       882
        2010       0.46      0.52      0.49       904

    accuracy                           0.48      6515
   macro avg       0.47      0.48      0.47      6515
weighted avg       0.47      0.48      0.47      6515
```

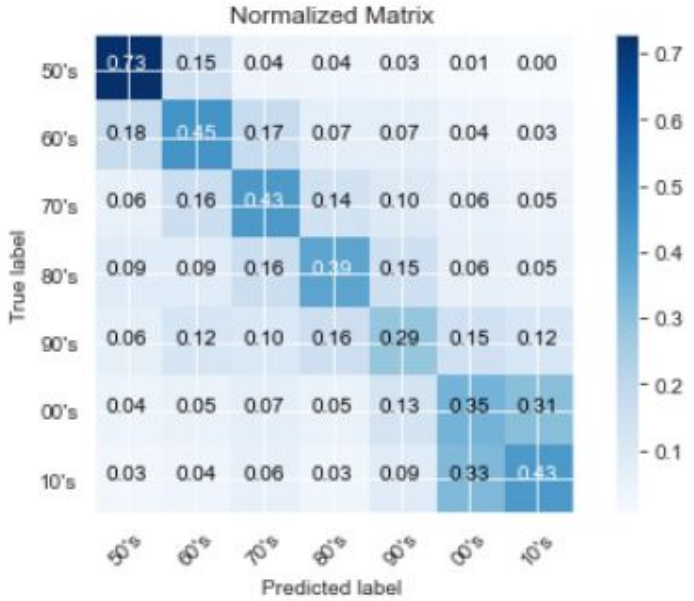
Après avoir séparé en deux notre dataset nous allons maintenant ajuster le modèle SVC correspondant. L'objectif d'un SVC linéaire est de s'adapter aux données, en renvoyant le meilleur ajustement qui divisera nos données.



Nous sommes donc arrivés au bout de notre analyse avec le modèle suivant comprenant les meilleurs hyperparamètres d'après GridSearch.

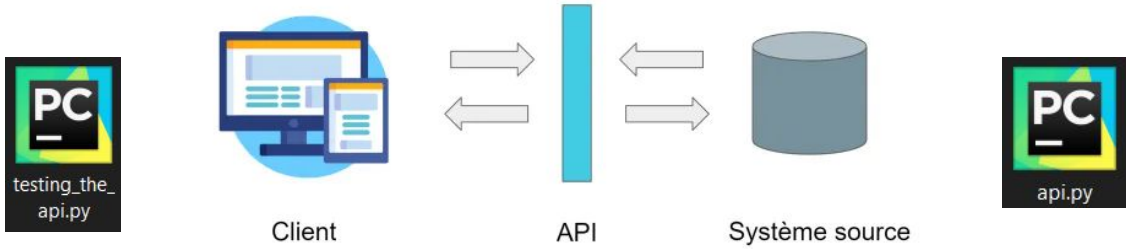
En conclusion, nous pouvons voir que celui-ci a seulement des chances moyennes de connaître l'année de sortie d'une chanson, si celle-ci est datée des années 90 ou 2000. Celles-ci sont grandement méprises entre elles et avec les années 2010. Les autres décennies sont plus reconnaissables mais n'atteignent pas les résultats escomptés.

Nous pourrions reprocher cela à un manque de données ou des timbres mal choisies pour ce dataset. A moins que les timbres utilisés depuis les années 50 ne se ressemblent trop malgré l'évolution musicale sur ces 70 dernières années.
Nous allons maintenant comparer ce premier modèle avec le modèle XGBClassifier. XGBoost (comme eXtreme Gradient Boosting) est une implémentation open source optimisée de l'algorithme d'arbres de boosting de gradient. Le Boosting de Gradient est un algorithme d'apprentissage supervisé dont le principe et de combiner les résultats d'un ensemble de modèles plus simple et plus faibles afin de fournir une meilleur prédiction.
Nous reproduisons donc les étapes précédentes pour estimer l'efficacité de ce modèle.



Nous observons que pour certains cas, le modèle XGBboost est plus efficace mais comme en témoignent l'accuracy nous opterons pour le premier modèle dans l'ensemble. Pour pouvoir créer notre API, nous exportons les modèles sous forme de pickles, afin de les tester en direct. Dans l'ensemble nous concluerons que le dataset n'est pas assez pourvu pour définir l'année de sortie d'une musique ou que les variables explicatives ne sont pas liées à la variable de prédiction.

API
(Application Programming Interface)




```python
import requests
import pandas as pd
import pprint


def scale_func(music):
    music.iloc[:, 1:] = (music.iloc[:, 1:] - music.iloc[:, 1:].min()) / (
            music.iloc[:, 1:].max() - music.iloc[:, 1:].min())
    return music


def random_sample(filename="YearPredictionMSD.csv", num_samples=10):
    base_for_samples = pd.read_csv(filename, header=None)

    print("Shape of the original dataset : ", base_for_samples.shape, end='\n\n\n')

    base_for_samples = base_for_samples[base_for_samples[0] > 1950]
    print("Shape of the dataset after getting cleaned : ", base_for_samples.shape, end='\n\n\n')

    samples = base_for_samples.sample(num_samples, axis=0)
    print("What our sample looks like : ", samples, end='\n\n\n', sep='\n')
    return scale_func(samples)


data = random_sample(num_samples=5)
data = data.drop(axis=1, columns=0)

print(data, end="\n\n\n")
req = data.to_csv(index=0)

resp = requests.post("http://127.0.0.1:5000/predictor", req).json()
pprint.pprint(resp)
```

```python
import io
import pickle
import pandas as pd
from flask import Flask, request
from joblib import load

app = Flask(__name__)


@app.route('/predictor', methods=['POST'])
def predictor():
    """model = pickle.load(open("final_prediction_clf.pickle", "rb"))"""
    model = load('filename.joblib')
    req = io.StringIO(request.data.decode('utf-8'))
    df = pd.read_csv(req, sep=',', header=0)
    prediction = model.predict(df)
    response = pd.DataFrame(prediction, index=df.index, columns=["The predicted decade of production with CLF"])
    json_response = response.to_json(orient='table', indent=2)
    return json_response, 201


@app.route("/")
def greeting():
    return "Hello" \
           " let's predict the year of the production of your desired song!"


if __name__ == '__main__':
    app.run(debug=True)
```

Nous avons configuré notre API en localhost et avons créé un fichier python pour jouer le rôle de client. Le client envoie une requête par le protocole HTTP vers notre API, et reçoit une réponse avec le même protocole.

```
D:\Programmes\Python38\python.exe C:\Users\mrzoh\PycharmProjects\pythonProjectForDataScience\api.py
 * Serving Flask app "api" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 111-677-676
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [10/Jan/2021 19:51:56] "POST /predictor HTTP/1.1" 201 -
```

Nous avons mis en service notre API, et avons effectué une requête. Comme montré, la réponse 201 qui montre la réussite de notre requête a été émis.

```
C:\Users\mrzoh\PycharmProjects\pythonProjectForDataScience>python testing_the_api.py
Shape of the original dataset :  (515345, 91)

Shape of the dataset after getting cleaned :  (514430, 91)

what our sample looks like :
            0         1         2         3         4         5         6   ...        84
442402   2008  49.13098  23.93454  -6.48962   8.84494  -0.76086 -19.32630  ...  15.86351
40920    1995  50.59764  68.14194  14.93025  -7.67738 -22.03601 -18.72819  ...   1.52611
201025   2008  44.56681  -3.46929  15.20972  21.88126 -15.78027  12.59838  ...  89.80787
445785   2000  44.80464  56.90822 -13.62047   2.40020  -4.07550  -8.74534  ...  11.00546
317453   1997  37.93568 -37.05720  45.78891  21.86665 -38.54182   6.34607  ...  20.62712 -

[5 rows x 91 columns]


               1         2         3         4         5         6         7   ...
442402   0.884168  0.579774  0.120029  0.558968  1.000000  0.000000  0.940063  ...  0.1624
40920    1.000000  1.000000  0.480576  0.000000  0.436882  0.018735  0.000000  ...  0.0000
201025   0.523705  0.319279  0.485280  1.000000  0.602461  1.000000  0.861393  ...  1.0000
445785   0.542488  0.893215  0.000000  0.340935  0.912267  0.331435  0.248504  ...  0.1073
317453   0.000000  0.000000  1.000000  0.999506  0.000000  0.804154  1.000000  ...  0.2163

[5 rows x 90 columns]


{'data': [{'The predicted decade of production with CLF': 2000, 'index': 0},
          {'The predicted decade of production with CLF': 2000, 'index': 1},
          {'The predicted decade of production with CLF': 2000, 'index': 2},
          {'The predicted decade of production with CLF': 2000, 'index': 3},
          {'The predicted decade of production with CLF': 2000, 'index': 4}],
 'schema': {'fields': [{'name': 'index', 'type': 'integer'},
                       {'name': 'The predicted decade of production with CLF',
                        'type': 'integer'}],
            'pandas_version': '0.20.0',
            'primaryKey': ['index']}}
```

Chez le client nous préparons un paquet de 5 samples, nous enlevons la 1ère colonne qui est l'année, et le mettons à l'échelle, comme nous avions fait pour entraîner nos modèles. Une fois la requête envoyée, nous recevons la réponse, sous format json.
Nous avons aperçu que la réponse était souvent 2000 comme année de production, nous avons partagé le problème, avec nos camarades et notre professeur, finalement nous avons conclu qu'il manquait des données significatives dans le dataset. Nous avons fait beaucoup de recherches, mais aucune des astuces n'a pu nous aider.

Mode de fonctionnement d’API

Au tout début de projet, nous souhaitions interconnecter notre API avec The Echo Nest (l’API qui a permit la construction de ce dataset), afin de le rendre plus facile à utiliser. Mais nous n’avons pas pu la retrouver en ligne (a été acquis par Spotify et mis hors service). Nous avons donc décidé d’envoyer quelques “samples” choisis à partir de notre dataset d’une manière aléatoire, de côté du client. Nous avons utilisé Flask pour concevoir notre API.

Pour pouvoir utiliser les modèles que nous avions préparé, nous avons trouvé deux façons de faire joblib et pickle, les deux ont montré la une performance identique, mais comme expliqué ici, ayant un soucis de persistance de modèle nous choisi joblib.