Python - Data Analysis - Machine Learning - API

Year Prediction with a subset of the Million Song Dataset

Théophane DELBAUFFE
ESILV - A4 - DIA4

Summary

01 Dataset & Context

02 Objectives & Problems

03 Pre-processing, Metrics & Models

04 API with Streamlit

Dataset & Context

1. Million Song Dataset (MSD)

Collaboration of LabROSA & Echo Nest

Contains basis **metadata** (artist, title, ...), **artist information** (origin, similar artists), **audio features** (pitches and timbres per segment, beats, key, tempo, ...), lyrics, snippets

2. Echo Nest API

Free public web API
Takes a digital audio file and generates a **JSON file** describing the music's structure and musical content : meta data, rhythm, pitch, **timbre** , key, loudness, duration, segments (divisions of the song), beats, ...

3. Subset we use

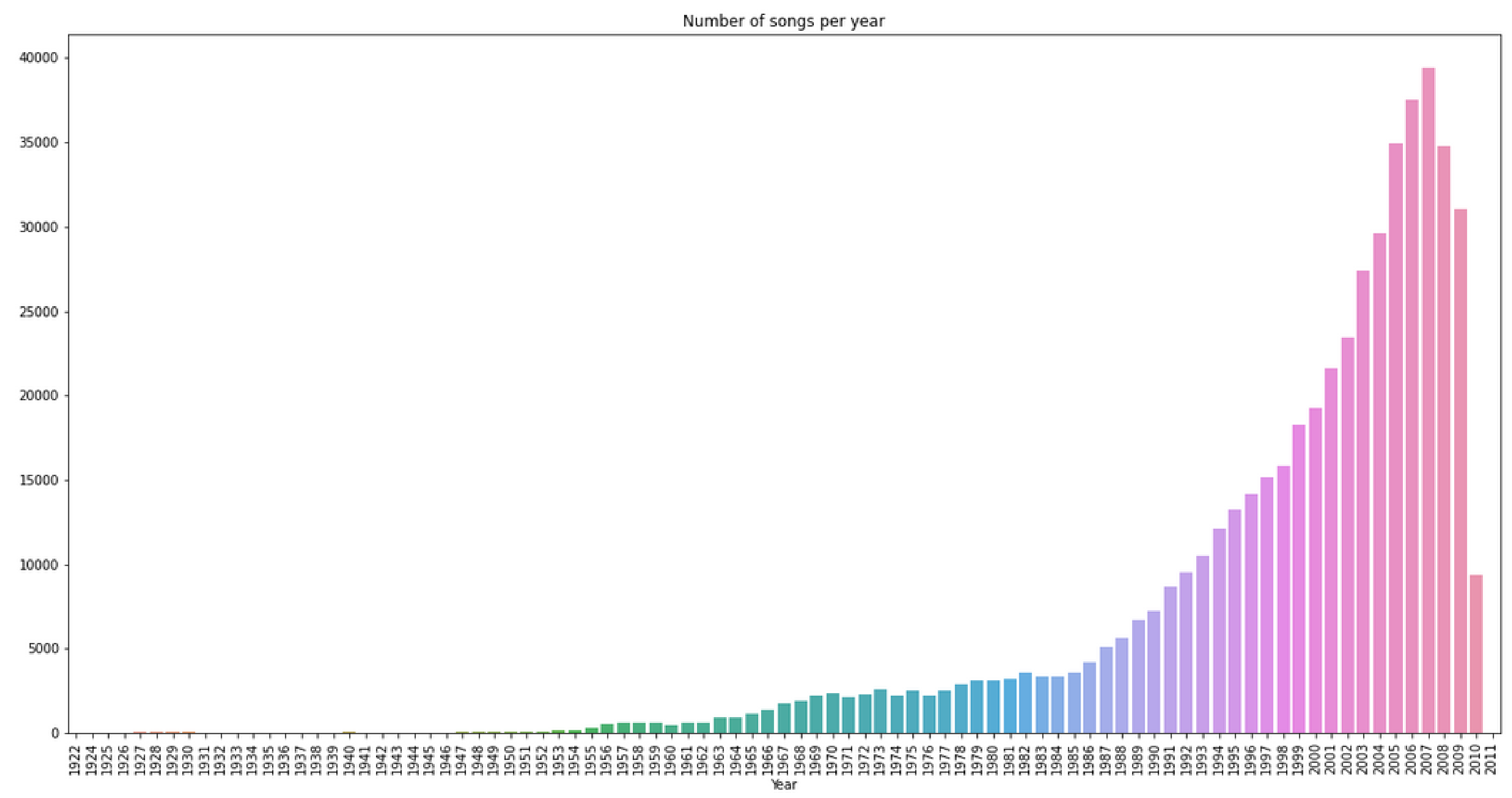
515 345 musics - 28 223 artists
Data represented by a target variable (the **release year**) and **90 predictors** (12 timbre **averages** et 78 timbre **covariances**)
The predictors are **computed from timbre features** of the Echo Nest API.
Each **timbre feature is a 12-dimensional** vector. A music is composed of hundreds of timbre features.
Average predictors are the means of each dimension while **Covariance predictors are the elements of the upper triangular covariance matrix.**
Target range : from **1922 to 2011**

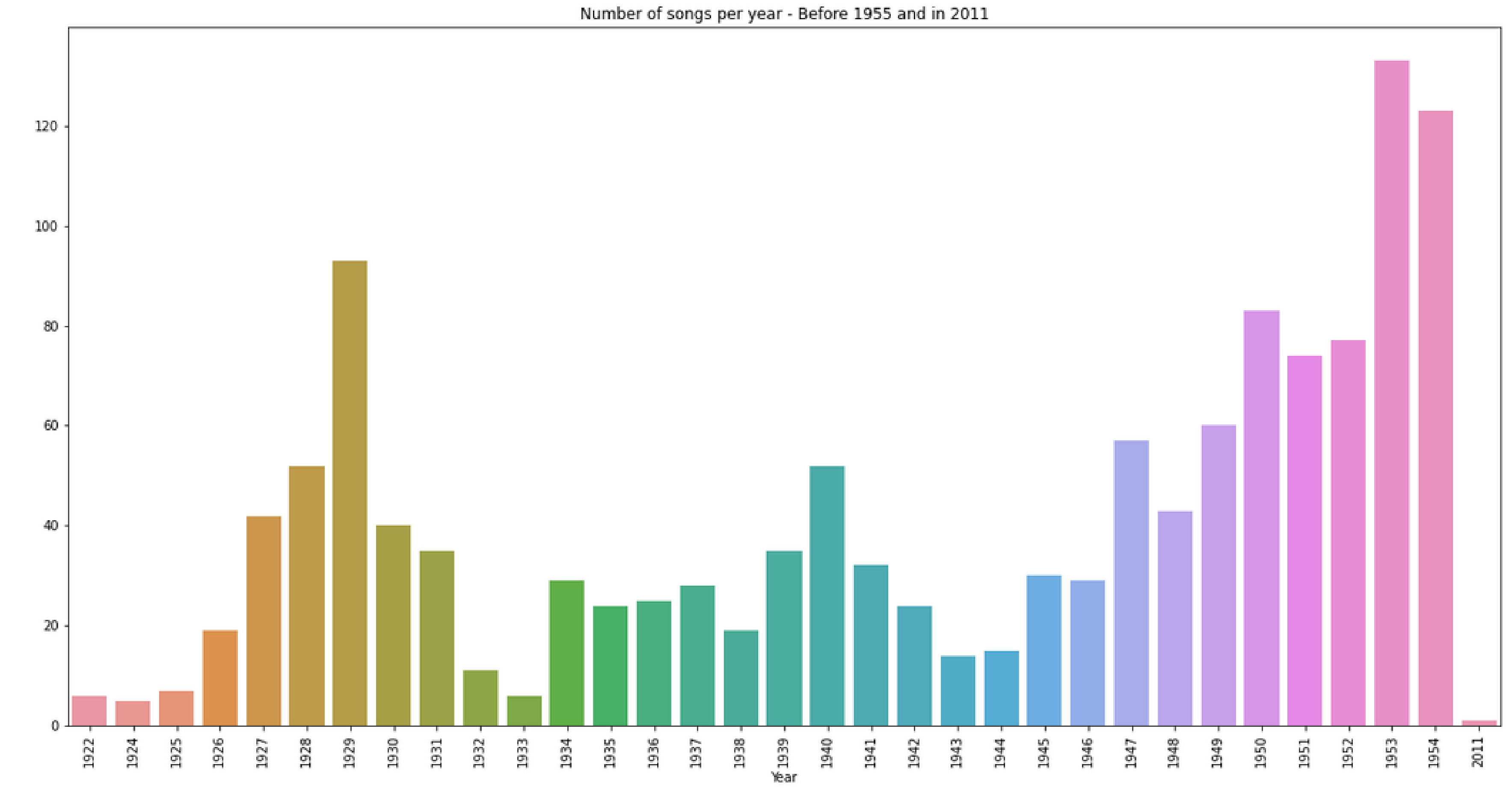
Exploring data

	Year	Avg1	Avg2	Avg3	Avg4	...	Cov74	Cov75	Cov76	Cov77	Cov78
0	2001	49.94357	21.47114	73.07750	8.74861	...	-23.08793	68.40795	-1.82223	-27.46348	2.26327
1	2001	48.73215	18.42930	70.32679	12.94636	...	-32.22788	70.49388	12.04941	58.43453	26.92061
2	2001	50.95714	31.85602	55.81851	13.41693	...	43.20130	-115.00698	-0.05859	39.67068	-0.66345
3	2001	48.24750	-1.89837	36.29772	2.58776	...	82.58061	-72.08993	9.90558	199.62971	18.85382
4	2001	50.97020	42.20998	67.09964	8.46791	...	-7.50035	51.76631	7.88713	55.66926	28.74903
5	2001	50.54767	0.31568	92.35066	22.38696	...	6.09352	35.18381	5.00283	-11.02257	0.02263
6	2001	50.57546	33.17843	50.53517	11.55217	...	35.46251	11.57736	4.50056	-4.62739	1.40192
7	2001	48.26892	8.97526	75.23158	24.04945	...	56.37650	-18.29975	-0.30633	3.98364	-3.72556
8	2001	49.75468	33.99581	56.73846	2.89581	...	-15.67150	-26.36257	5.48708	-9.13495	6.08680
9	2007	45.17809	46.34234	-40.65357	-2.47909	...	3.26926	-298.49845	11.49326	-89.21804	-15.09719
10	2008	39.13076	-23.01763	-36.20583	1.67519	...	44.60282	158.00425	-2.59543	109.19723	23.36143
11	2002	37.66498	-34.05910	-17.36060	-26.77781	...	-47.74886	-170.92864	-5.19009	8.83617	-7.16056
12	2004	26.51957	-148.15762	-13.30095	-7.25851	...	-137.72740	115.28414	23.00230	-164.02536	51.54138
13	2003	37.68491	-26.84185	-27.10566	-14.95883	...	-89.08971	-891.58937	14.11648	-1030.99180	99.28967
14	1999	39.11695	-8.29767	-51.37966	-4.42668	...	-98.76636	-122.81061	-2.14942	-211.48202	-12.81569

Issue

Imbalanced data





Objectives

First objective

Explore the dataset, make visualizations, find links between variables

Second objective

Find a way to face imbalanced data

Thrid objective

Thanks to the 90 features, fit a model to predict the release year of a song

Fourth objective

Build an API to deploy the model

Problems

First problem

Timbre is a mix of abstract variables of a segment of a song, such as loudness, brightness, flatness, attack, ... We can easily think that we need more data about the music itslef to predict its release year.

Second problem

Many artist copy the characteristics of older musics. For example, an 2010s' artist migth want to make a music with a 1980s' New wave style.

Conclusion

Fitting a model that will perfectly predict new songs' release year seems pretty hard. A solution to this will be to look at the absolute difference between predicted and real year rather than accuracy to evaluate our model.

Moreover, a classification task with imbalanced data may be difficult.

Pre-processing

Split

To split the dataset, a rule is given by its creator :

- **Train** set : first **463,715** examples
- **Test** set : last **51,630** examples

Suppression

All years **from 1922 to 1952, including 2011**, were **deleted** from the train and test sets.

There was not enough data to train and test the model well, and these labels were **too little represented**, comparing to others

Scaling & PCA

- **Standardization** of my sets after fitting my scaler on the train set
- I fitted PCA on my train set and I kept 90% of the information with **55 components** (against 90 predictors)

New variable

I added the '**Decade**' variable to make it easier computing other metrics.

Metrics

Accuracy

My first metric was accuracy : nb_correct / nb_wrong

Decade Accuracy

As the accuracy was very low for every models (around 8%), I decided to also compute accuracy for the Decade. If the predicted release year is in the same decade as the real release year, then the prediction is considered as right.

Problem

Accuracies were **not relevant enough** to compare my models because a **Dummy Classifier made better** results than actual ones. Indeed, this is due to imbalanced data (1990s and 2000s are **over-represented.**

Average absolute difference (in years)

Example :

Predictions = [2000, 1980] - Reality = [2002, 1976]

avg_abs_diff = ((2002 - 2000) + (1980 - 1976)) / 2

avg_abs_diff = 3 years

Models

Here is a list of different models I tried :

- Classification
  - K-NN
  - Logistic Regression
  - Linear Discriminant Analysis
  - Quadratic Discriminant Analysis
  - Random Forest Classifier
- Regression
  - Linear Regression

My final one K-NN
with k_neighbors = 3 & weights = 'distance'

As you can see, I tried to see this problem as a Regression one. Indeed, years can be seen as **series of integers** instead of labels.

I thought that I could round the predicted result. It showed good performance but I noticed that a few predictions were nonsense (year 3150 for example).

So I left it out.

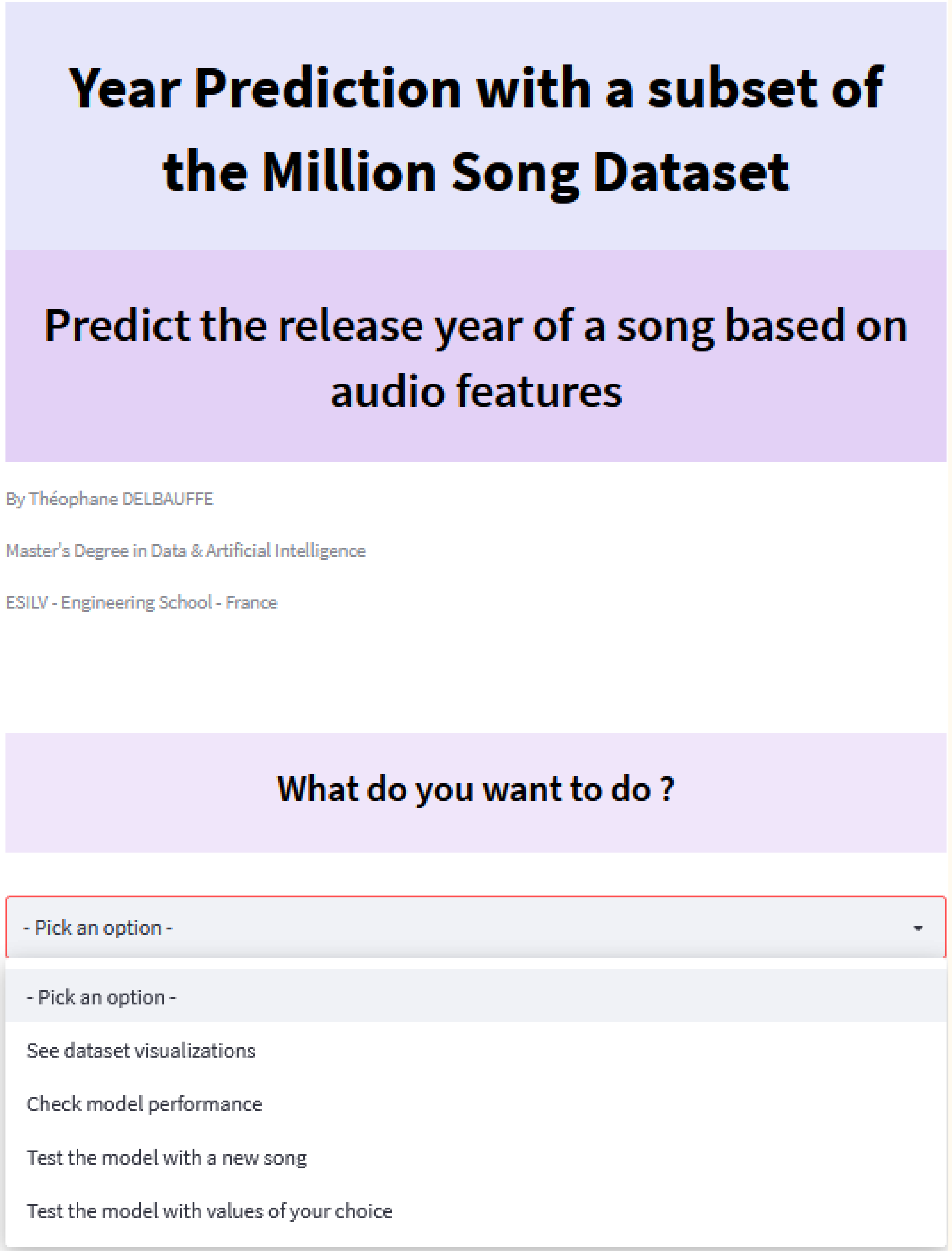
API with Streamlit

Download the folder and then,

in Anaconda Prompt or else :

streamlit run .\streamlit_app\main.py

Main window of the API



Each option leads to a new display

Have a look !