University of Pisa

Master's Degree in Artificial Intelligence and Data Engineering

Data Mining and Machine Learning Project

Urban Sound Recognition system

Candidates:
Massimo Valentino Caroti
Simone Landi

Academic year 2022-2023

INDEX

1 INTRODUCTION

The aim of our project was to create an application that allows users to determine the urban sounds around them. The application is able to automatically classify environmental noises according to their category. The classification part uses a machine learning algorithm to classify the correct category, selecting it from the most frequent categories.

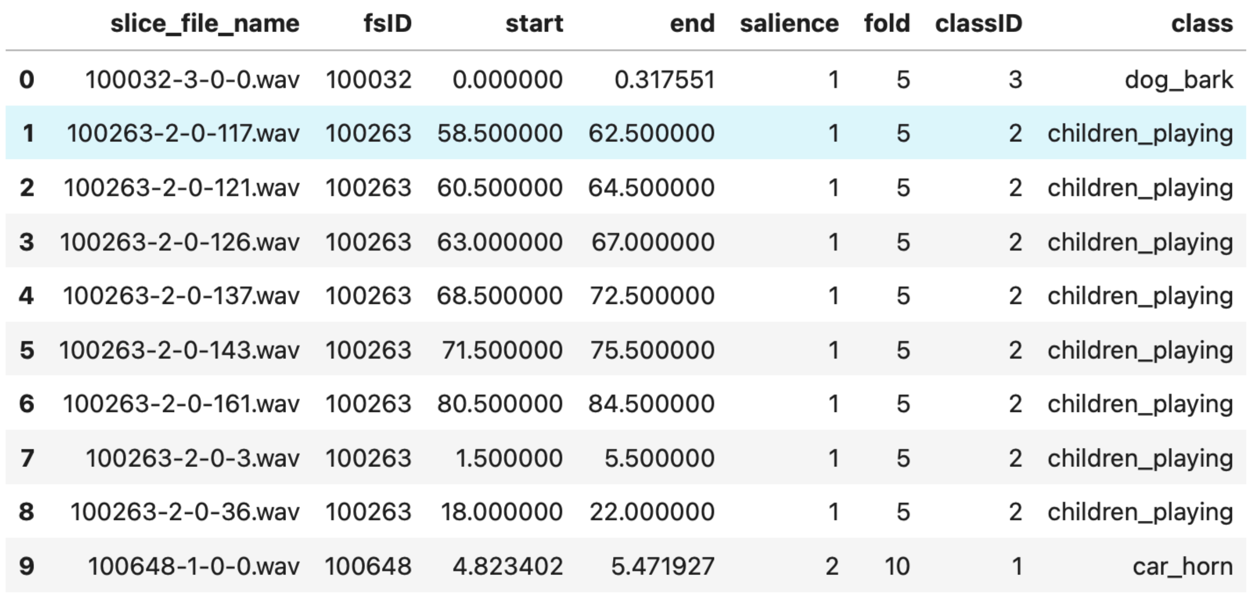
Dataset

To create the ambient sounds classifier, we used a set of 8732 labelled sound files obtained from Freesound (https://freesound.org/), a community-built sound sample database. We extracted the information from the audio files using the Python library Librosa. This library allows MFCC (Mel Frequency Cepstral Coefficient) features to be extracted. For our purpose, we downloaded 6 GB of audio file data in total (9k entities). The dataset used was downloaded from Kaggle (https://www.kaggle.com/datasets/chrisfilo/urbansound8k).

The dataset consists of 10 categories:

- Air conditioner
- Car horn
- Children playing
- Dog bark
- Drilling
- Engine idling
- Gunshot
- Jackhammer
- Siren
- Street music

Within the dataset there is a csv file where the category for each audio file is defined. The structure of the csv file is shown in figure 1.



	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.000000	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.500000	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.500000	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.000000	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.500000	72.500000	1	5	2	children_playing
5	100263-2-0-143.wav	100263	71.500000	75.500000	1	5	2	children_playing
6	100263-2-0-161.wav	100263	80.500000	84.500000	1	5	2	children_playing
7	100263-2-0-3.wav	100263	1.500000	5.500000	1	5	2	children_playing
8	100263-2-0-36.wav	100263	18.000000	22.000000	1	5	2	children_playing
9	100648-1-0-0.wav	100648	4.823402	5.471927	2	10	1	car_horn

Figure 1: dataset

Requirements

Functional Requirements :

- Ambient sounds classifier must provide a way for Anonymous Users to register and become Standard Users
- Ambient sounds classifier must deny to the Anonymous Users the access to the app and to all functionalities designed for the Standard Users
- Ambient sounds classifier must provide a way to perform the login with credentials

Non-Functional Requirements :

- Ambient sounds classifier must embed at least one machine learning algorithm
- Ambient sounds classifier should be intuitive and easy to use

2 MACCHINE LEARNING

The classification of ambient noise makes it possible to determine possible risk events in the context of autonomous driving or wearable devices. A user during an outdoor running session might be alerted to the presence of sirens. The same situation could occur for car drivers who will be notified of the presence of emergency vehicles.
Accurately identifying the origin of ambient noises is not an easy task due to their numerous presence in different sources.This is why we have decided to keep the ten most frequent categories.

Analysis

The analysis of the dataset was performed in Python, using scikit-learn an open-source library that provides a wide range of tools for data analysis; all the pre-processing and algorithms evaluation steps are present in the Jupyter notebook files hosted on the GitHub repository.

Data Pre-Processing

During the preprocessing phase, we verified that the lengths of the audio samples were sufficient to be able to extract MFCC features.
In Figure 2 is shown the distribution of the genre considered for classification.

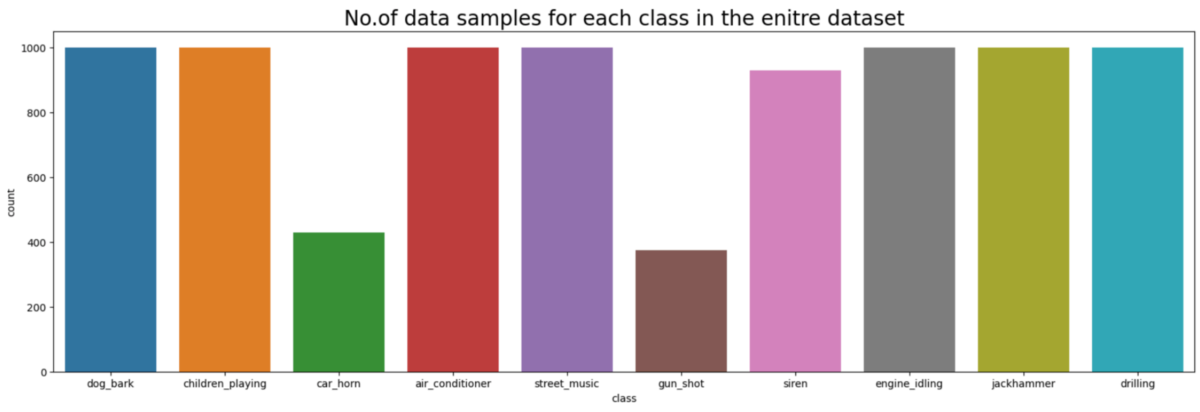


Figure 2: Class distribution

In order to classify audio, it is necessary to transform it into a numeric vector; to do this, we used the Librosa library's feature.mfcc method.
The following steps are performed to obtain the 40 features:

- Spectrograms, using the Short-Time-Fourier-Transform (STFT)
- The Mel spectrogram, from applying Mel scale filterbanks to the STFT
- Mel Frequency Cepstral Coefficients, from applying the DCT transform on the mel-spectrogram

Feature Selection

For testing the different algorithms we also applied a feature selection technique for search through all possible combinations of features in the data, a subset of attributes that works better for prediction. We use the SelectKBest method, implemented in scikit-learn library, for extracting the k highest scoring features, testing with the scoring function f_classif.

Rebalancing

The dataset analyzed is not balanced so we tried to apply three different techniques to approach the imbalanced dataset:

A) **Oversampling**: it consists in duplicating samples in the minority classes by picking them with replacement randomly.
B) **Undersampling**: it undersample the majority classes by randomly picking samples without replacement.
C) **Smote**: in generates synthetic example from the existing sample of the minority classes.

In this case the performances of the models are lower or equal than those without rebalancing. Balancing the genre of each noise will give each genre a probability of being classified 1/n_genres, so the classifier will forget the actual distribution of features in the original sample. On the other hand, without rebalancing, frequent cases will have fewer misclassifications, in which case balancing the dataset means losing information about the frequencies of appearance, and would mean training the model on a different distribution than the one that will appear in the distribution of the new samples to be classified.

Model's evalutation

The following classifiers were tested for genre's classification:

1) AdaBoost Classifier
2) Gaussian NB
3) K-Neighbors Classifier
4) Logistic Regression
5) Random Forest Classifier
6) Support Vector Classification

Initially we evaluated the performance of the previous algorithms without applying feature selection techniques, while in a second step we evaluate a second time the algorithms applying feature selection techniques using different score functions and a different number of selected features to find the best parameters.

To perform these evaluations, we used the GridSearchCV method implemented in scikit-learn that allow us to specify different parameters and find the ones that get a higher score.
For each evaluated algorithm we compared the best result obtained. In the following table are presented with different scoring metrics and the computational time needed for training the model.

classifier	accuracy	precision	recall	f1	time
AdaBoostClassifier	0.837	0.850	0.837	0.839	4.088
GaussianNB	0.489	0.512	0.489	0.488	0.008
KNeighborsClassifier	0.880	0.887	0.880	0.879	0.008
LogisticRegression	0.590	0.594	0.590	0.583	3.764
RandomForestClassifier	0.896	0.900	0.896	0.896	3.089
SVC	0.928	0.929	0.928	0.928	1.748

Figure 3: 10-fold cross validation

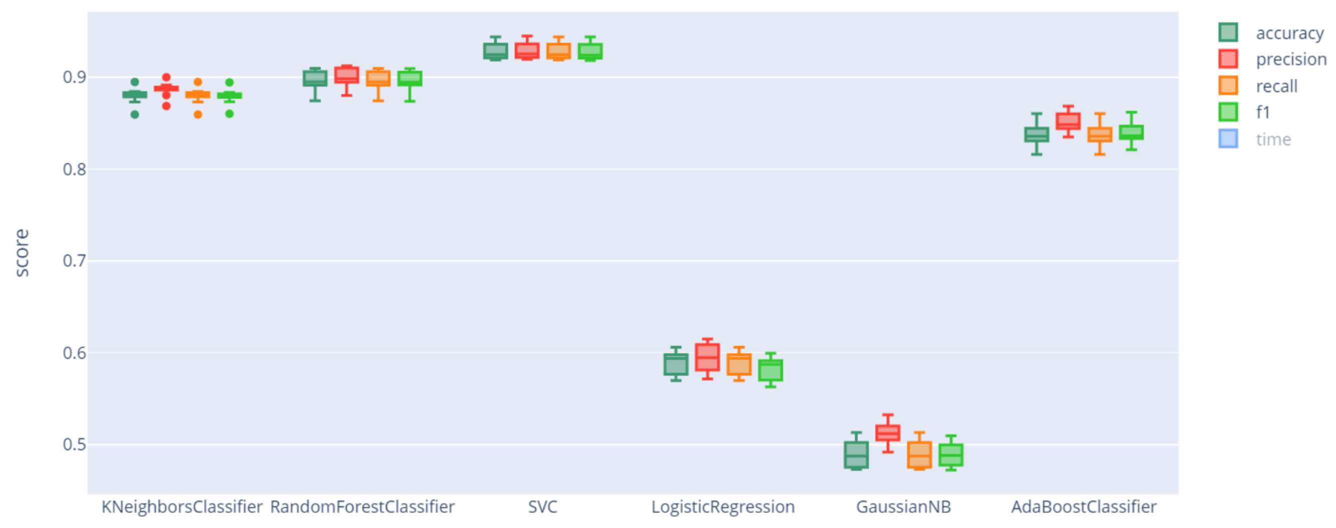


Figure 4: Results

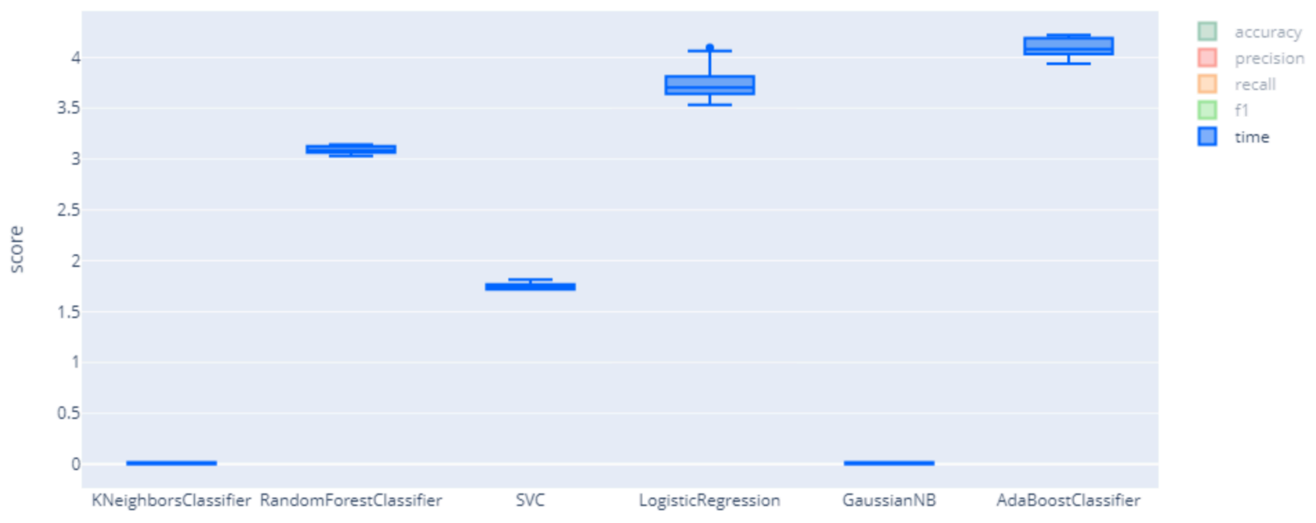


Figure 5: Time result

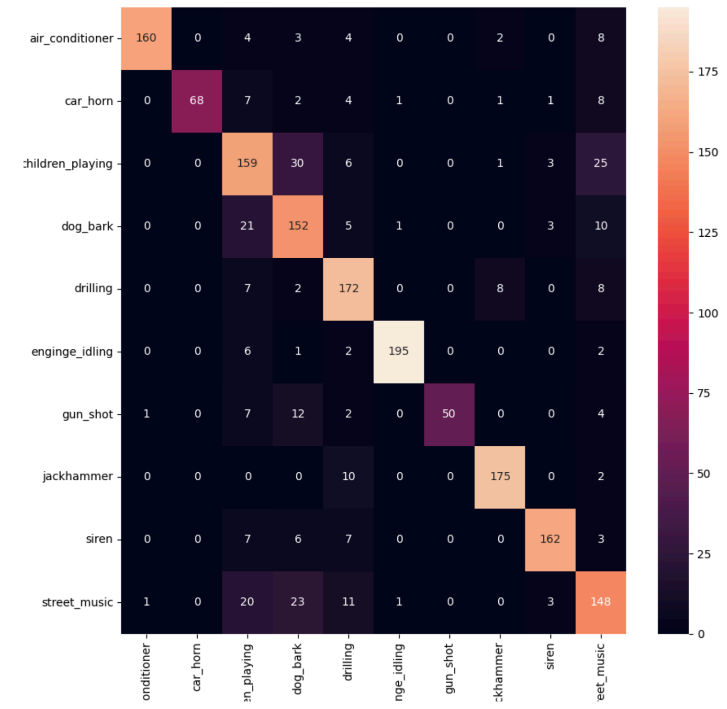
```
T_Test between KNeighborsClassifier & RandomForestClassifier : T Value = -3.5608456965019624, P Value = 0.002467424657633586
T_Test between KNeighborsClassifier & SVC : T Value = -12.482071575581564, P Value = 2.6838945792280996e-10
T_Test between RandomForestClassifier & SVC : T Value = -7.069511586759577, P Value = 1.9931633953332466e-06
```

Figure 6: T-Test

Below, is shown the confusion matrix for each method evaluated with the best parameters:

ADABOOST CLASSIFIER

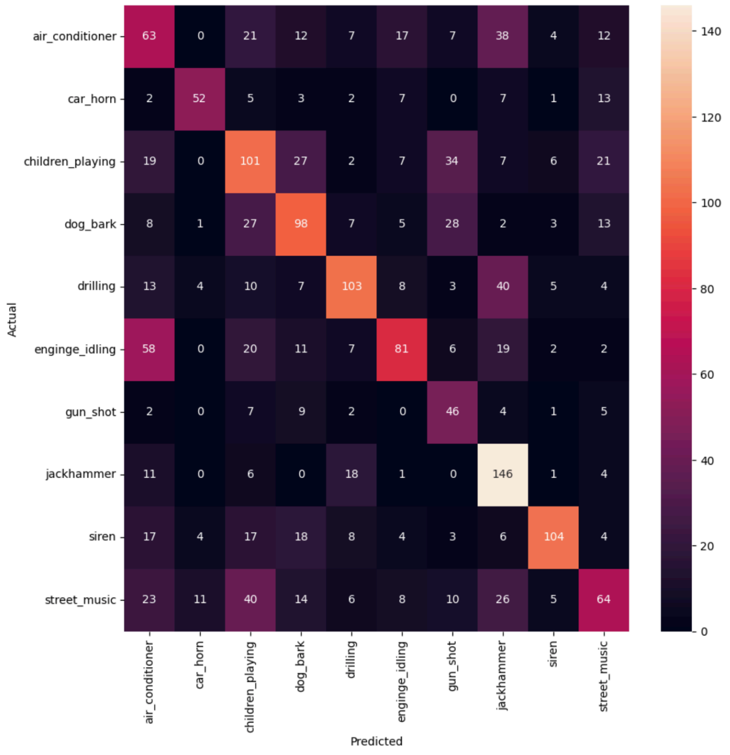
The best result is obtained applying features selection technique, limit the number of features to 37. F1-score: **0.839**



	onditioner	car_horn	en_playing	dog_bark	drilling	nge_idling	gun_shot	ckhammer	siren	eet_music
air_conditioner	160	0	4	3	4	0	0	2	0	8
car_horn	0	68	7	2	4	1	0	1	1	8
:hildren_playing	0	0	159	30	6	0	0	1	3	25
dog_bark	0	0	21	152	5	1	0	0	3	10
drilling	0	0	7	2	172	0	0	8	0	8
enginge_idling	0	0	6	1	2	195	0	0	0	2
gun_shot	1	0	7	12	2	0	50	0	0	4
jackhammer	0	0	0	0	10	0	0	175	0	2
siren	0	0	7	6	7	0	0	0	162	3
street_music	1	0	20	23	11	1	0	0	3	148

GAUSSIAN NB

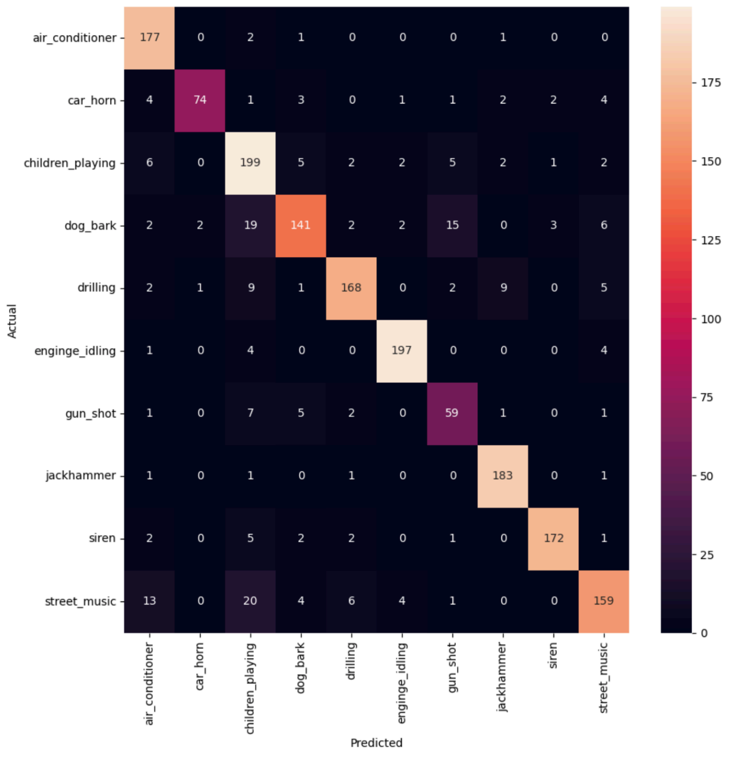
The best result is obtained applying features selection technique, limit the number of features to 23. F1-score: **0.488**



Actual \ Predicted	air_conditioner	car_horn	children_playing	dog_bark	drilling	enginge_idling	gun_shot	jackhammer	siren	street_music
air_conditioner	63	0	21	12	7	17	7	38	4	12
car_horn	2	52	5	3	2	7	0	7	1	13
children_playing	19	0	101	27	2	7	34	7	6	21
dog_bark	8	1	27	98	7	5	28	2	3	13
drilling	13	4	10	7	103	8	3	40	5	4
enginge_idling	58	0	20	11	7	81	6	19	2	2
gun_shot	2	0	7	9	2	0	46	4	1	5
jackhammer	11	0	6	0	18	1	0	146	1	4
siren	17	4	17	18	8	4	3	6	104	4
street_music	23	11	40	14	6	8	10	26	5	64

K-NEIGHBORS CLASSIFIER

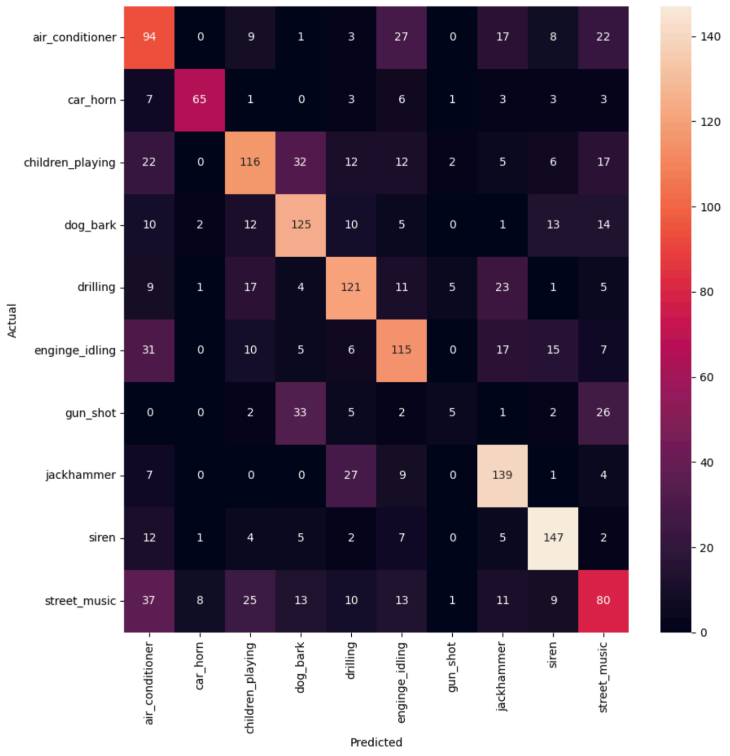
The best result is obtained applying features selection technique, limit the number of features to 38. F1-score: **0.879**



LOGISTIC REGRESSION

The best result is obtained without applying features selection technique.

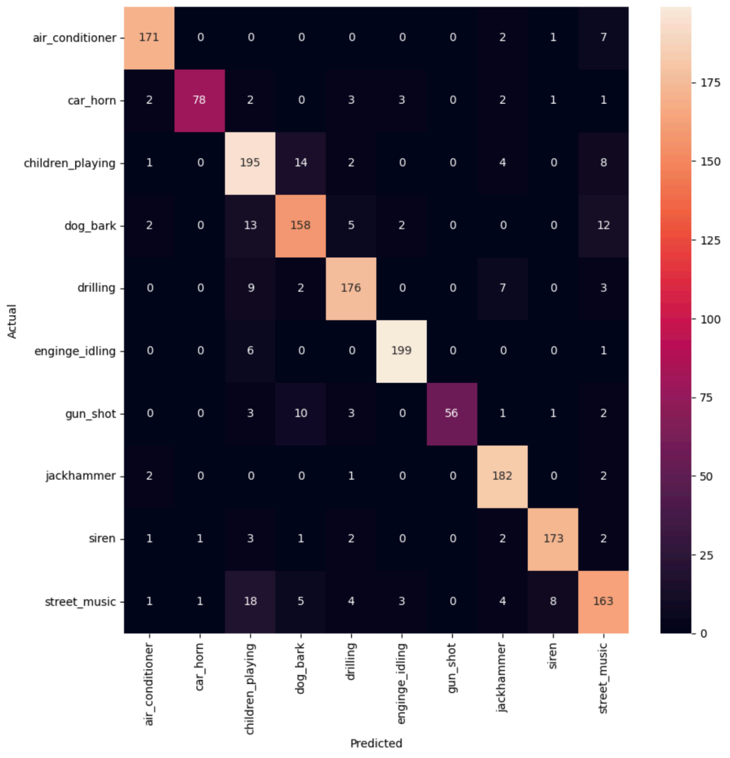
F1-score: **0.583**



RANDOM FOREST CLASSIFIER

The best result is obtained without applying features selection technique.

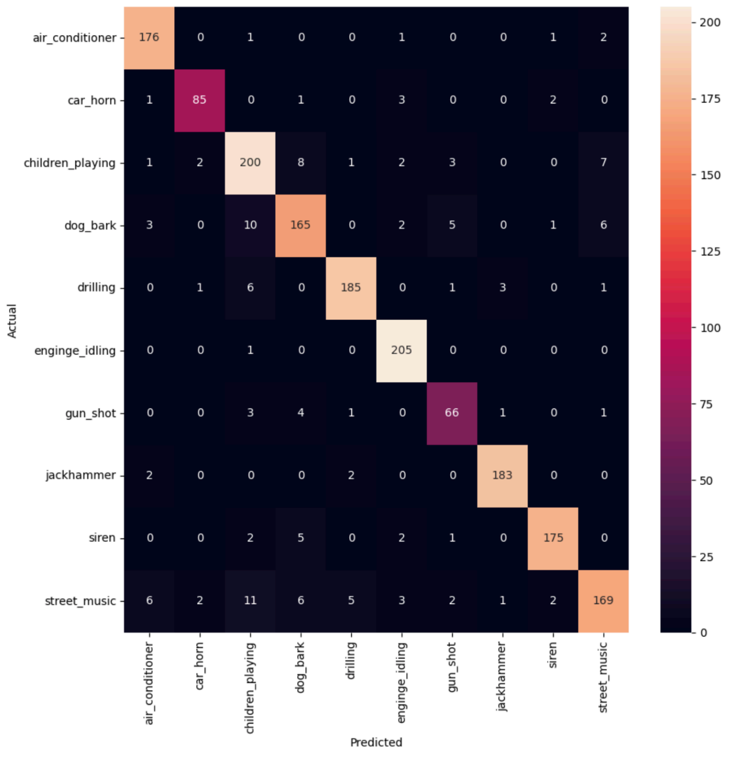
F1-score: **0.896**



	air_conditioner	car_horn	children_playing	dog_bark	drilling	enginge_idling	gun_shot	jackhammer	siren	street_music
air_conditioner	171	0	0	0	0	0	0	2	1	7
car_horn	2	78	2	0	3	3	0	2	1	1
children_playing	1	0	195	14	2	0	0	4	0	8
dog_bark	2	0	13	158	5	2	0	0	0	12
drilling	0	0	9	2	176	0	0	7	0	3
enginge_idling	0	0	6	0	0	199	0	0	0	1
gun_shot	0	0	3	10	3	0	56	1	1	2
jackhammer	2	0	0	0	1	0	0	182	0	2
siren	1	1	3	1	2	0	0	2	173	2
street_music	1	1	18	5	4	3	0	4	8	163

SUPPORT VECTOR CLASSIFICATION

The best result is obtained without applying features selection technique. F1-score: **0.928**



	air_conditioner	car_horn	children_playing	dog_bark	drilling	enginge_idling	gun_shot	jackhammer	siren	street_music
air_conditioner	176	0	1	0	0	1	0	0	1	2
car_horn	1	85	0	1	0	3	0	0	2	0
children_playing	1	2	200	8	1	2	3	0	0	7
dog_bark	3	0	10	165	0	2	5	0	1	6
drilling	0	1	6	0	185	0	1	3	0	1
enginge_idling	0	0	1	0	0	205	0	0	0	0
gun_shot	0	0	3	4	1	0	66	1	0	1
jackhammer	2	0	0	0	2	0	0	183	0	0
siren	0	0	2	5	0	2	1	0	175	0
street_music	6	2	11	6	5	3	2	1	2	169

Conclusions

From the T-test table we can see that there is no a statistical difference between the models selected.
We chose to implement in the application the model created with ***SVC***, because compared to **K-Neighbors** and **Random** Forest has a optimum trade-off between F1 and training time and compared to other models although in some cases it has a lower precision, we can attribute this difference to the fact that the other model predict very well the majority class but not the other in the same way (we can see it in the confusion matrix).

3 IMPLEMENTATION

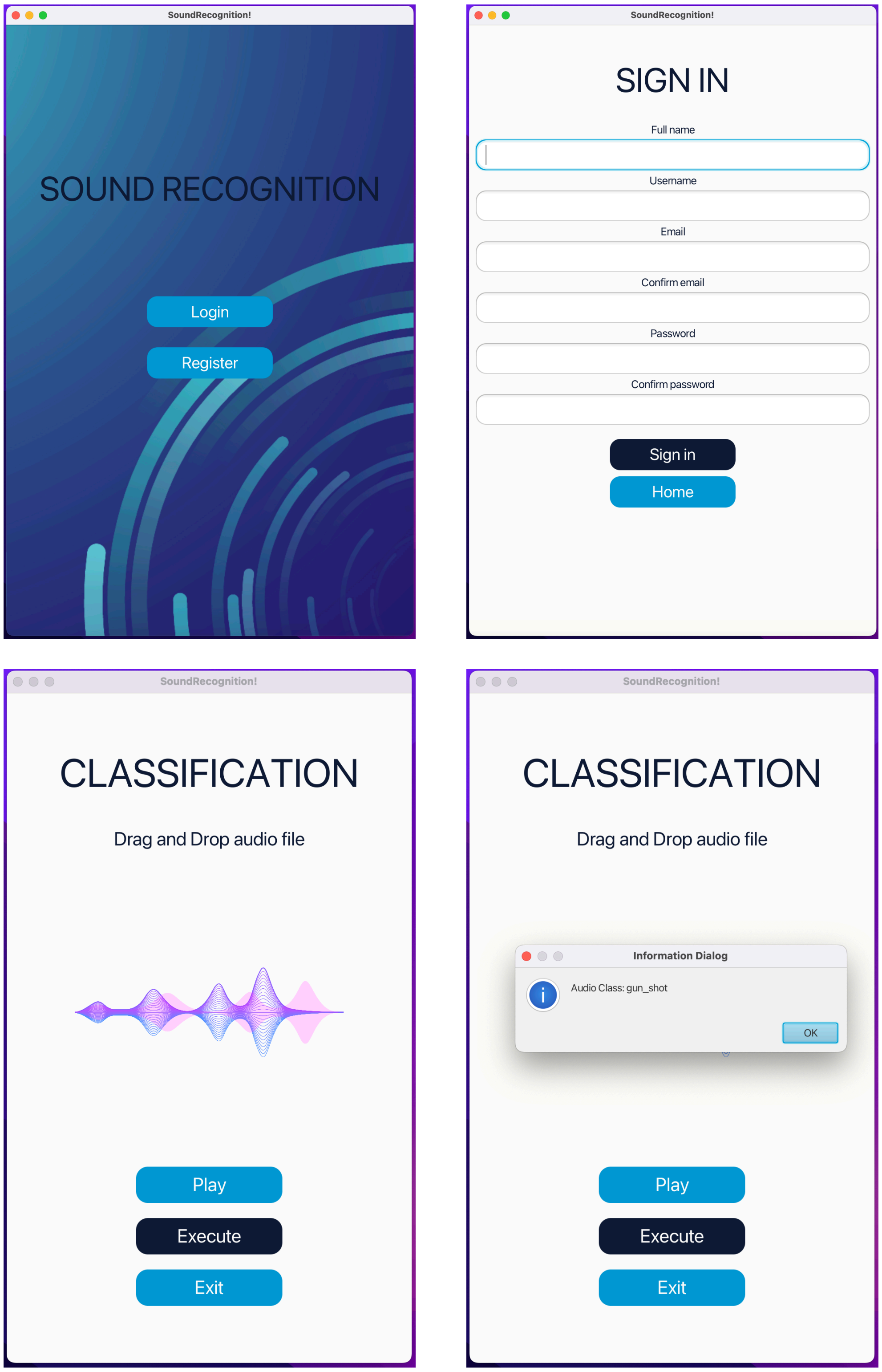
Modules

- **VoiceIDNotesFeatureExtractor**: contains the Python scripts needed to extract audio features from audios and the implementation of the server. The server implementation is single process and is defined in the main.py.
- **VoiceIDNotesApp**: is the actual application, developed following the MVC (Model, View, Controller) pattern. That guarantees to have a good separation and maintainability of the code.

Application

We have developed a Java application to test the model. The registration and login are the first part of the application. Once registered, users can access the classification function. They can provide an audio file to the application via the "Drag and Drop" function.
After inserting the audio file, users can listen to it or proceed with the classification. The file is saved locally and sent to the Python server running on the local host.

The Python server uses Librosa to extract features and the SVM model to assign a class to the audio file. The class is then sent to the Java application, which provides it to the user via an alert.



Audio Class: gun_shot

Figure 7: Application