Here we will discuss the work of 2 students: Wissem HAMIDOU and Benjamin LAZARD. We decided to stop, mostly because we had to take care of other tasks, but we had a few ideas that may have improved our final score, that we could not try out. We will attempt to describe shortly the logical process that led us to an estimator which were able to predict the test set with an accuracy of nearly 90%. The reasoning is simple but it took us a lot of time: both thinking of new ways to classify the data and waiting for the computer to display the results of simulations, that ran some tasks for up to 18h in a row.

1. **Understanding the dataset**

The first step was to understand what we were dealing with. This time the basic files were not signal processed, as opposed to the previous challenge in SD210. After many tests, we opted for the teacher's recommendation, i.e. a late fusion with MFCC features.

With a simple classifier (the default Multi-Layer Perceptron), we first tried many other feature extraction techniques. We explored the Librosa library, and tried to visualize the different analyses (MFCC, melspectra, chroma), for several FFT windows, hop\_length, and number of MFCC values. We eventually picked the MFCC as it seemed not to collect too much data (space in memory), and yet to have a lot of variance (whereas Mel spectrum implied a lot of features which did not change a lot from a file to another, and chroma\_cens seemed not detailed enough). Simulations confirmed this visual choice.

Then we had to tackle the fusion problem. Indeed, this kind of analysis results in a frequency analysis for each time frame of the track, instead of labeling directly the files. Eventually we sticked to the late fusion model, implying that we would classify each timeframe, and then decide the label of the file based on a majority rule over its timeframes. But prior to deciding this, we tried several types of fusions trying to keep as much information as possible for each file. Concatenating all the timeframes revealed a terrible idea, concatenating only a few happened to be only slightly better. We also performed a few analyses that could have been the result of MLP processing such as computing mean, median, and variances of frequencies for each file over time… But it was always worse than raw late fusion based training. We even tried to perform some sort of late fusion that would take into account several signal analysis techniques for each timeframe (such as MFCC + chroma), without success.

We thought of applying outliers detection, but given the confusion matrixes we got whilst training on the training set, and validating on the validation set, it seemed irrelevant to us. Indeed, most confusions are "understandable" such as tram and train being confused, which is probably due to the nature of the files and not to a labeling fault.

1. **Dimensionality reduction/ separating space**

Surprisingly, whatever model we used, the principal component analysis was not always helpful. It has 2 main benefits: project the input data into a space where it is more separable, and then potentially remove useless features. The new space happened not to be necessarily better when using non-linear boundaries. If SVM with linear kernel were positively impacted, MLP did not show significant changes. However, when we tried to maximize the number of features per file, it often led to about 10% dimensionality reduction, without any variance loss which we appreciated as it reduced the computation time.

We eventually created a function to automatize the process of standardization, projection, and shuffling, on all useful sets, as well as tools to save the results…

We also tried LDA which is like a supervised PCA, in order to transform the original space of features before applying another classifier… But it did not lead to great results.

We later realized it was preferable to train on both the training set and the validation set to predict the test set. It could have led to overfitting or not brought any significant information, but we noticed it was useful.

1. **Multi-Layer Perceptron**

We tried a lot of things with this estimator which led to the best results many times over… But we found it operates like a black box, and we were surprised to see some carefully planned approaches fail while some crazy attempts worked beyond our hopes. For example, we achieved up to 90% success on the test set, with the default MLP after a PCA operated without standardization, and without shuffling, whereas a more serious approach with standardization + pca + shuffling + gridsearch on hyper parameters (many different layers types, regularization α or tolerance factors) would barely make us reach 80% success.

We carefully read the documentation, but despite the "warm start" attribute supposed to reduce the overfitting, and the ability to train the neural network several times over the same dataset, we were blindfolded, and had to try hidden layers shapes almost randomly picked. Of course, we used many grid searches to select the best afterwards.

1. **Other classifiers**

Basically, we tried them all. But some of them like SVM cost so much time, that we could not tune them perfectly over all hyper-parameters since it forbad us to try more promising methods. Although it looked promising (test score of up to 68%).

In the new feature space (post PCA/LDA), we expected a lot from clusterisation techniques such as K-nearest neighbors or Gausssian Mixture Models, but not only was it very time-consuming, it also did not perform well.

The trees and ensemble methods relying on it also led to poor results compared to MLP. Bagging of weak MLP allowed us to reach the 75% test score though.

Logistic regression was also quite efficient (~70%).

Each time the default parameters yielded non neglectable scores we adapted the regularization parameters the best we could.

1. **Ideas for further improvement**

We did not have time to try everything, but we read some papers that made use of Convolutional Neural Networks. The idea would be to use Tensorflow® or a similar library to analyze each file as a matrix, instead of being compelled to use the late fusion. Usually an image recognition tool, it could operate on the set (timeframes/frequencies) as a picture reader, thus detecting patterns such as jumps from a frequency to another, otherwise harder to spot with a late fusion, where each time frame is classified individually.

**References:**

We did not use as much documentation as the last time, mostly because we now are familiar with sklearn, and because of the time constraint.

But we would like to quote in particular:

* <https://librosa.github.io/librosa/feature.html#spectral-features> we tried to explore the full audio library, at least the feature extraction part, and searched the web for the "most relevant" features for audio scenery recognition… which are quite known to be MFCC.
* <https://www.researchgate.net/publication/250008991_Features_for_Audio_Classification> This short paper, inspired us into keeping MFCC.
* <http://scikit-learn.org/stable/modules/neural_networks_supervised.html> To use at best MLP features, and understand their role. We basically used the entire sklearn library documentation center.
* <https://aqibsaeed.github.io/2016-09-24-urban-sound-classification-part-2/> <http://blog.christianperone.com/2015/08/convolutional-neural-networks-and-feature-extraction-with-python/> 2 papers about the use of convolutional networks for audio files based machine learning.
* <http://scikit-learn.org/stable/modules/outlier_detection.html> We eventually did not use it
* <https://www.kaggle.com/aniruddhaachar/data-visualization-and-machine-learning-algorithms> An example which made us think about trying LDA and study confusion matrixes.