
POLITECNICO DI MILANO
Music and Acoustic Engineering

---

Computer Music - Languages and Systems
Homework 1

# Audio Effects Recognition

Authors:

BADIANE DAVID GIUSEPPE

DE MARCO DAVIDE

LORENZO FRANCESCO

PICCIRILLO JACOPO

# Contents

# List of Symbols

| | |
|---|---|
| *Fs* | Sampling frequency |
| *winlength* | Window length for windowing operations |
| *hopsize* | Hop-size for windowing operations |
| *window* | Choose the kind of window |
| *classes* | Names of the classes to distinguish |
| *featuresnames* | Names of the calculated features |
| *kbest* | Number of features chosen by feature selection |
| *framefeats* | Number of features computed with frameanalysis.getframefeatures() generate_datasets is True |
| *do_plot* | Boolean defining whether to plot all train features or only selected ones |
| *kfold* | Number of K-Folds for cross-validation metrics. |
| *generate_datasets* | Boolean defining whether to generate test/train data-sets from a single database |
| *test_size* | Number between 0 and 1 defining the test set length with respect to the single database_length it plays a role is generate_datasets is true |
| *amplitude_scale* | Scales the amplitude of each audio file, setting the max value |

In the folder *AudioFX_Recognition/environment/modules/analysislab* there is the *user_interface.py* file. Here the user can choose the above options.

# Chapter 1

# Low level Software Explanation

## 1.1 Abstract

The following report analyses and explains the audioFX classification software that we implemented by using python. The software has been modularized in order to improve its fixability, flexibility and speed. Also, modularization allows the project to be easily expandable and updatable. The various modules are organised on three levels. Main level scripts can all be run singularly from terminal.

- Main level:

  *main.py*
  *maintrain.py*
  *main_traintest.py*
  *plotfeatures.py*
  *plotselected.py*
  *maintest.py*

- Modules level:

  *trainingloop.py*
  *featureselction.py*
  *savetraindata.py dataloader.py*
  *testloop.py metrics.py*
  *supportvectormachines.py*
  *confusionmatrix-py*

- AnalysisLab level:

  *features.py*
  *frameanalisys.py*
  *user_interface.py*

## 1.2 Features

The calculated features are:

- Flatness;
- Rolloff;
- Centroid;
- Spectral Bandwidth;
- Zero Crossing Rate;
- Tremolo feature (custom feature);
- Tremolo feature2 (another custom feature);
- Mel Frequency Cepstrum Coefficients .

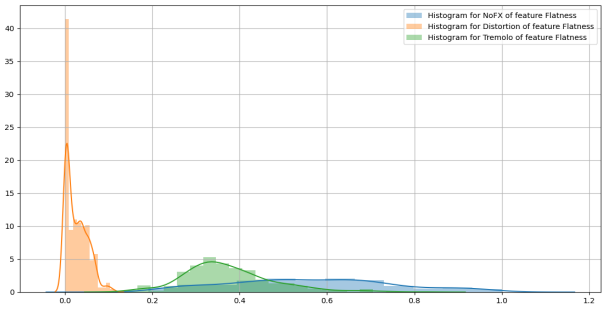
The first five features and MFFCs are implemented through *librosa* functions, while the other two features are custom.
The philosophy of computing this many features is that, since features are selected, the decision about features usage is left to the software. In fact, even if computing MFCCs may seem irrelevant to the task, some coefficients may prove themselves useful and be chosen.
The main problem of the classification between NoFX (no effects), Tremolo, Distortion classes is to distinguish between the NoFX and Tremolo classes. In order to achieve this distinction we designed two features that provide information on whether there is a sinusoidal amplitude modulation. Both tremolo features are fed with the locus of maxima (improperly called waveform) and return a scalar.
They perform the following steps:

0. Low pass filtering of the waveform implemented by the *butter_lowpass_filter* function for cutting high frequency noise;
1. Calculation of the number of relative maxima along the filtered waveform in order to feed them to the *linear* function by the name of *n_sections*;
2. The *linear* function generates a linearly interpolated version of the waveform over *n_sections* equally spaced samples;

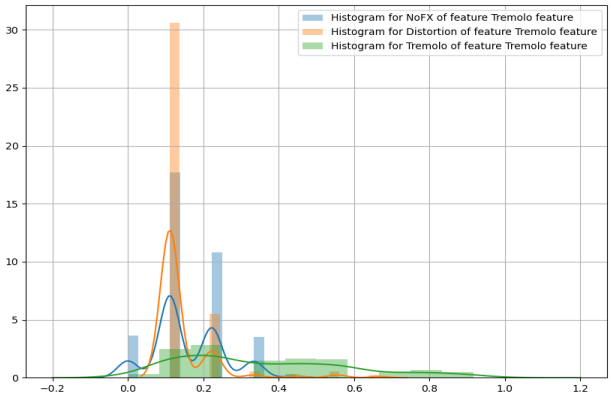
3. Subtract the linear waveform from the filtered waveform. The result of this subtraction will better isolate the periodic amplitude modulation component since the linear waveform sampling frequency is balanced with the number of relative maxima in the filtered waveform (*n_sections*);

4. Calculate the auto-correlation of the resultant signal;

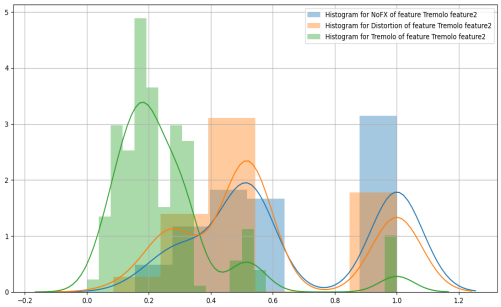
5. – The output of the first feature is the number of relative maxima of the auto-correlation ( this number usually increases for tremolo samples);
   – The output of the second feature is the average dynamics of the auto-correlation over the *n_sections*.




Histogram of Flatness feature: it's immediate to see that the Distortion class (orange) is clustered away from the others.




Histogram of Tremolo feature: the Tremolo class (green) has a wider distribution than the others.




Histogram of Tremolo feature: the Tremolo class (green) is quite clustered away form the NoFX class (blue).

## 1.3 Features extraction - Training/Test Loop

All the features are calculated on the whole audio, only the locus of maxima (improperly called waveform) is calculated at frame level. The architecture of feature calculation is the same for both train and test loop, the only things that change are the names of the objects and variables.

- We create an empty object indexable by the classes names:

  dict_train_features = 'NoFX': [], 'Distortion': [], 'Tremolo': [];

- We cycle over the classes, stored in the user interface. For each class we create a zeros NxM matrix, where N is the number of files for the class and M is the number of features to calculate;
- For each class, we cycle over every audio file of the class of the class and execute *features.getfeatures(audio)*, filling each time a tuple of the zeros matrix;
- Inside *features.getfeatures()* we also need to call *frameanalysis.getframefeatures()*, that, by windowing, calculates on frame level the locus of maxima and ,once the frame calculation is over, feeds it to the Tremolo_feature;
- After the cycle over audio files has finished, the zero matrix has been filled and is passed to *dict_train_features[class]*;
- This cycle is repeated for all the classes so that at the end we have: *dict_train_features = 'NoFX': [data], 'Distortion': [data], 'Tremolo': [data]*.

```
def getdicttrainfeatures(path):
    dict_train_features = {'NoFX': [], 'Distortion': [], 'Tremolo': []}
    fullpath = str(path) + '/environment/databases/train/{}'

    for c in user_interface.classes():  # loops over classes
        n_features = len(user_interface.featuresnames())
        train_root = fullpath.format(c)
        class_train_files = [f for f in os.listdir(train_root) if f.endswith('.wav')]
        n_train = len(class_train_files)
        train_features = np.zeros((n_train, n_features))

        for index, f in enumerate(class_train_files):  # loops over all the files of the class
            audio, fs = librosa.load(os.path.join(train_root, f), sr=None)
            train_features[index, :] = features.getfeatures(audio)

        dict_train_features[c] = train_features
    return dict_train_features
```

## 1.4 Feature Selection

Feature selection is achieved in the *featureselection.py* file by using the sklearn function *SelectKBest* which selects features according to the *k* highest scores of a given score function. Between all the possibilities, we used the chi2 scoring function that computes the distribution stats of non-negative features. The module provides and prints the selected matrix, the selected columns as well as the scores for each feature. The number of selected features can be chosen by the user from the *user_interface.kbest()* function.

## 1.5 Metrics Calculation

The function *compute_metrics* calculates the metrics for the binary case (when the possible classes are two). It compares the real labels with the predicted ones for each element of the test set. Therefore we have four possible cases: True Positive, True Negative, False Positive, False Negative. The true values are the ones in which the classification is correct. We use the definition of accuracy, precision, recall and F1 score and calculates the binary scores. In order to extend the calculation to the multiclass case we employ the function *get_metrics()*. Here we use *itertools.combination* function to cycle over all the possible couples of classes. In this loop we use support vector machines classification to obtain the predicted labels and feed them to *compute_ metrics* for each couple of classes. The metrics values are stored inside a tuple of a matrix for each couple. We then average each column of the matrix and obtain the actual average metrics.

## 1.6 Support Vector Machine

Inside the *supportvectormachine.py* file we have the function *get_predictions()*.
Multiclass classification is implemented by a majority voting algorithm, therefore we set a binary support vector machine for each couple of classes. also computed via itertools, and collect all
predicted labels. We then compare elementwise the three predicted labels column arrays by concatenating them in a matrix and put in *y_test_predicted_mv* the number that appears more times for each row ( it can 0, 1 and 2, representing respectively NoFX, Distortion and Tremolo classes).
We also provide cross validated scores using *sklearn.model_selection.cross_val_score()* function on each binary classifier.
Cross-validated metrics are obtained by splitting the datasets in k smaller sets; k-1 subsets are used for training while the remaining one is used for testing. This procedure is repeated k times, the scores reported by k-fold cross-validation are obtained by averaging the values computed in the loop. This approach can be computationally expensive, but doesn't waste too much data. The number of k-fold is set from the *user_interface*.

## 1.7 Confusion Matrix

The confusion matrix is is a specific table layout that allows visualization of the performance of an algorithm. Each row of the matrix represents the predicted classes while each column represents the real class or viceversa. The results in the diagonal are the ones in which the predicted class is the same as the real one (correct predictions). By inspecting the non diagonal elements one can understand how many bad predictions have been made.
In order to compute the confusion matrix we defined a function called *compute_cm_multiclass*, which expects as input the real labels and the predicted labels. In this function we fill a NxN matrix (where N is the number of classes) through two nested for loops. With the first for loop we get reference of the *i-th* row in which we are working. For each row we get pred_class, a portion of the predicted labels array that refers to files of just the i-th class. With the second for loop, while we are in a row, we iterate over the elements of pred_class. So for each element of *pred_class*, we add a 1 in the column selected by the *j-th* value of pred_class. At the end of this for loop we will have the confusion matrix.

# Chapter 2

# High level Software Explanation

Here we will analyze the structure and the overall architecture of the code, providing hints about the behavior of the software at an higher level of abstraction.

## 2.1 Main

The software follows two main algorithms, whether *generate_datasets* flag in the user interface is True or False.
If *datasets* is True the software will implement the following pipeline:

$$\text{maintrain.train()} \rightarrow \text{main\_traintest.traintest()} \rightarrow \text{maintrain.train()} \rightarrow \text{maintest.test}$$

otherwise the pipeline will look like this:

$$\text{maintrain.train()} \rightarrow \text{maintest.test()}$$

## 2.2 Main_traintest

This part of the code will be called and executed only if *user_interface.generate_datasets()* is set to True.
In this snippet we generate Train/Test data-sets from a single big database.
In order to do so, the features are calculated for each file of the database via the *trainingloop.py* function , which creates the dict_features object of this form:

dict_features = {'NoFX':[data], 'Distortion':[data], 'Tremolo':[data]}

The Main_traintest splits this data in two other objects of the same shape:

$$\text{dict\_features} \rightarrow \text{dict\_train\_feats \& dict\_test\_feats}$$

We recall just once that whenever we need to use a sklearn tool we need to format input data: sklearn expects a matrix of features,obtained by concatenating the three matrices relative to the classes and an array that labels each tuple of the matrix specifying by a number the class of each audio file .
We use the subsequent sklearn function in order to randomly pick train and test datasets:

X_train, X_test, y_train, y_test = train_test_split(X_mc, y_mc, test_size=test_size )

The test_size is specified in the user_interface. We now need to rebuild the objects with class clustered data since it allows more flexibility. This is done by declaring dict_train_feats and dict_test_feats objects, and righteously filling them class by class via a for cycle with the tuples of X_train and X_test.
We now need to store data through a function of the *Savetraindata.py* file:

savetraindata.save_datasets(dict_train_feats, dict_test_feats)

## 2.3 Savetraindata e Dataloader

### 2.3.1 Savetraindata

We find two functions in this module. They both store arrays and matrices as .dat files using the dump option of numpy elements: *savedata(dict_train_features, featurelst, feat_max, feat_min)* and *save_datasets(dict_train_feats, dict_test_feats)*. The first one is used to save the data after features selection. The second one is used to save test and train sets in the case in which generate_datasets is True.

### 2.3.2 Dataloader

This module defines many functions which retrieve the saved data via *numpy.load* algorithm:

- *Trainselected()* returns selected features;
- *featmax()* returns an array with the max of each feature;
- *featmin()* returns an array with the min of each feature;
- *dict_train_features(c)* returns the c class matrix of *dict_train_features*
- *dict_test_feats(c)* returns the c class matrix of d *dict_train_feats* (used if *generate_datasets* == true)
- *columns_selected()* returns the columns selected by the feature selection algorithm.

They are used in *Maintrain* and *Maintest* in order to retrieve datasets and information. Plus, they both have been constructed in order to avoid conflicts between the *dataset* mode and the traditional one.

## 2.4 Maintrain

Here we train our software and perform feature selection. We first get a hold of the path to the databases directory using the *Pathlib* built in library of Python 3. If *generate_datasets* is true, we get both train features and test features by calling *main_traintest.traintest()* and then get train features using *dataloader.dict_train_features(c)* function.
Otherwise we need to calculate train features via the training loop.
In any case we first need to label the files with an array and to extract max and min data. The data is then processed in sklearn format (as discussed in 2.2) in order to be fed to *featureselection.py* file functions. The function *savedata(dict_train_features, featurelst, feat_max, feat_min)* is then called in order to store the feature selection data. If requested by *user_interface.do_plot*, we plot all the Train features, otherwise only the selected ones.

## 2.5 Maintest

Here we do the testing and compute the metrics for our classification software. If *generate_datasets* is true, we get the test features via the *dataloader.dict_test_feats(c)*, otherwise we need to calculate them via the test loop. In any case we need to select the right number of columns (features) coherently to the feature selection; update *X_test* matrix and label the files with an y array. We process data in order to feed them to sklearn Support Vector Machine implementation using our *supportvectormachines.py* file function: Then we print the metrics and the confusion matrix, which allow the user to evaluate its classification.

y_test_predicted_mv = supportvectormachines.getpredictions( X_train_normalized_loaded_selected, y_train_selected, X_test_mc_normalized)

```
Confusion matrix:
[[ 97.   0.  19.]
 [  0. 111.   0.]
 [  6.   0. 106.]]
```

It-s easily seeable that miscalculations happen between the class 0 (NoFx) and class 2 (Tremolo), while Distortion is perfectly classified. K-fold cross validated scores are printed by the function *supportvectormachines.getpredictions()*.

# Chapter 3

# Software usage

Here we will briefly explain how to use the software and its different functionalities.

## 3.1 Database

The algorithm automatically retrieves all the *.wav* files placed in the */environment/databases* folders. Specifically inside this path we find two folders: *test* and *train*. Inside each one of them, we need to have three directories exactly called by the classes names, beware of upper-cases. If the Boolean *generate_datasets()* in *user_interface.py* is set to True, all the user has to do is to put a folder for each class of sounds in */environment/databases/train* and run *main.py* to get the results.
Otherwise, the user has to put the files to train the algorithm into *train* folder and files to be classified in *test* folder.

## 3.2 Analysislab

The sound analysis is implemented inside the */environment/modules/analysislab* folder. Once the database is set, the machine learning algorithm is ready to go and will implement classification based on the analysis made by the user from the modules inside this folder. Specifically here we find three modules: *features.py*, *frameanalysis.py* and *user_interface.py*. Inside the first one the key element is *featurearray*, the return value of the *getfeatures()* function. Descriptors are computed inside *getfeatures()* itself by high level libraries (*e.g. librosa*) or at a lower level via the *frameanalysis* module. *getfeatures()* will call in fact *getframefeatures()* from *frameanalysis*. More custom features can be computed here to be returned as an array.
Notice that all the features computed have to be singular scalar values and that *framefeats()* in *user_interface.py* has to return the number of features returned by *getframefeatures()* in order for the algorithm to work properly.
As said before, the user can set some audio analysis parameters such as the sampling frequency (*Fs()*), the type of window, the window length and the hop-size in the *user_interface* module by changing the return values of the corresponding functions.
Beware of the order of the elements inside the return value of *user_interface.featuresnames()*, it has to be the same one used in the building of *featurearray* inside *getfeatures()*; a different name order will cause a mismatch in plots and histograms showed (not affecting anyway the quality of the classification). Nevertheless, a different length between *featurearray* and the return value of *featuresnames()* will cause errors and malfunctioning.

## 3.3 Modular usage

All the *.py* modules we find inside the root folder are part of the overall algorithm and can also be used just by themselves, in order to ease the analysis and classification process for the user. In this section we will briefly explain what the execution of these modules by themselves will cause.
Before running *maintest.py*, *plotfeatures.py*, *plotselected.py* or *print_feature_sel.py*, a loop over the train database has to be already done, via *main.py*, *main_traintest.py* or *maintrain.py*.

### 3.3.1 main.py

Executing this will run the whole algorithm with the options selected in *user_interface*.

### 3.3.2 main_traintest.py

Executing this will loop over the *train* database splitting the data obtained in train data and test data respectively saving them as *".dat"* files.

### 3.3.3 maintrain.py

Executing this will loop over the train database, splitting (calling traintest) or not whether *generate_datasets()* is set. Afterwards, feature selection is performed and results are printed. Values and histograms of all the features or of only the selected ones will be plotted according to *do_plots()*.Beware of the fact that the new train data saved in *".dat"* format overrideany previous *dict_train_features* in the root directory ,even so the user may move .dat files at will, therefore he can reuse them.

### 3.3.4 maintest.py

Whether datasets is true or false, it may perform the loop over the test database, proceeding to classify test data and printing cross validation scores, overall metrics and the confusion matrix.

### 3.3.5 plotfeatures.py

Executing this will plot values and histograms of all the computed features.

### 3.3.6 plotselected.py

Executing this will plot only selected features values and histograms.

### 3.3.7 print_feature_sel.py

Executing this will print on the terminal the selected matrix along with selected features names and the scores assigned from the scoring function of the *feature selection* algorithm.

## 3.4 Conclusions

The software has been designed to let the user be as free as possible in .dat files usage and in the choice of Test/Train sets.
In fact we experimented a little bit with the IDMT database and understood that when training and test are computed over samples relative of the same instrument, overall metrics usually fluctuate between 90% and 95%, otherwise overall metrics will diminish more or less by a value between 5% and 8%.
Nevertheless, due to the fact that both the custom features are time based, the software displays increased precision and accuracy if the average frequency of the tremolo is high (10/8 Hz or so) regardless of the differences between train and test sets. Another thing we noticed is that database sounds did not have the same amplitude scaling: in order to check this, we calculated the max amplitude in the waveform of each audio and saw that the feature selection algorithm always chose this feature. This kind of feature may be helpful when the ranges of the audio files are not uniform between classes, Anyways, from a general point of view we thought it was better to normalize all audio files between the same range before extracting features (preprocessing).
In conclusion, this application can be used as the basic framework for a wider multiclass classification task since it's easily expandable. For example, in order to add a class one simply has to add the name of the class in the user_interface and to have coherent databases, the software will implement the whole algorithm considering the new class; our design focused on providing an audio analysis tool easy to adapt to every kind of classification via manipulation of just analysislab modules.