Guess it in a few seconds

Statistical Learning - Group 19

Valerio Antonini, Maria Luisa Croci, Daniele Sanna, Alice Schirinà, Giuseppe Stefanelli

19/7/2019

The main goal of the work is to classify music's genres with the help of supervised (NN, Random Forest, SVR) and unsupervised learning techniques (such as clustering).

All the project has been implemented using **Python** as programming language.

1) Collecting data

We collected 100 songs for 5 music genres (*Classic, Pop, Metal, Reggae and Electronic*), for a total of 500 songs. In order to have an higher accuracy in our results, we decided to record around 60 seconds for each song using 3 sensors (*PitchSensor, DecibelSource e AmbientLightSensor*) from *Science Journal* app (*Fig. 1*). Those features measure respectively the tone (*Hz*), the intensity (*dB*) and the light (*lux*). For the latter we used another app available on *PlayStore*, called **Zeus**^{TM} **Music Strobe Light** that works as a strobo emitting flashes with the rithm of the music.

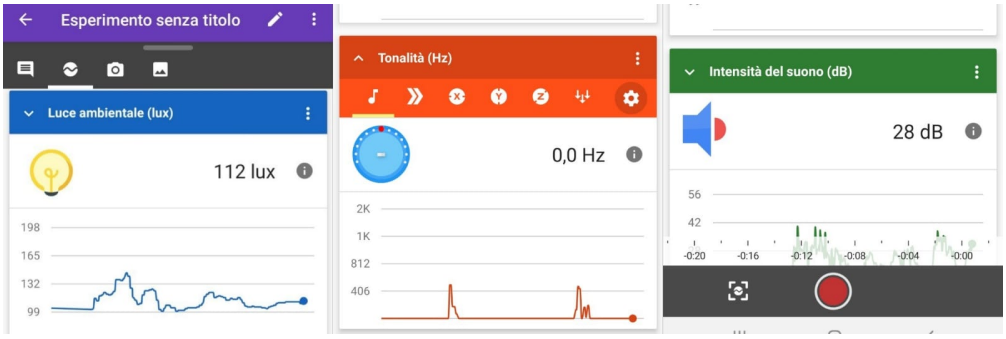


Figure 1: Science Journal's interface with the three sensors

In order to get the measures more accurate as possible we recorded in quite and not so bright room. Finally, we extract the csv files for each song, each of them composed by thousands of rows and three columns for the features mentioned before, as follow in *Fig. 2*:

relative_time	DecibelSource	PitchSensor	AmbientLightSensor
0		309.43384806315106	
2		309.43384806315106	
31	74.12328914397617	309.43384806315106	
43			152.0
69			152.0
72	74.34376272881852	309.43384806315106	
77		438.62727864583337	

Figure 2: csv example of a song

---

2) Data imputing

Afterwards we inspected the datasets to get what the rate of missing values were. We used *KNN* (*K Nearest Neighbors*) algorithm (*Fig. 3*) that creates a basic mean impute and then uses the resulting complete list to construct a KDTree. Then, it uses the resulting KDTree to compute nearest neighbours (NN). After it finds the k-NNs, it takes the weighted average of them.

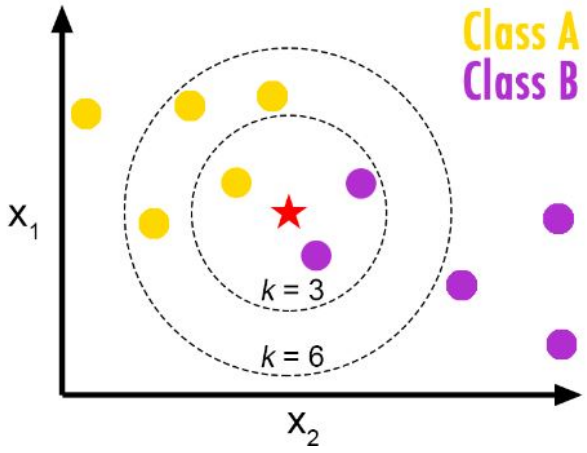


Figure 3: KNN Algorithm

From our features, we have *PitchSensor* and *AmbientLightSensor* that don't admit the creation of new features, so we used a *signal processing* of the feature in dB in order to obtain a valid spectogram. At the beginning, the data about this last are as in *Fig. 4*:

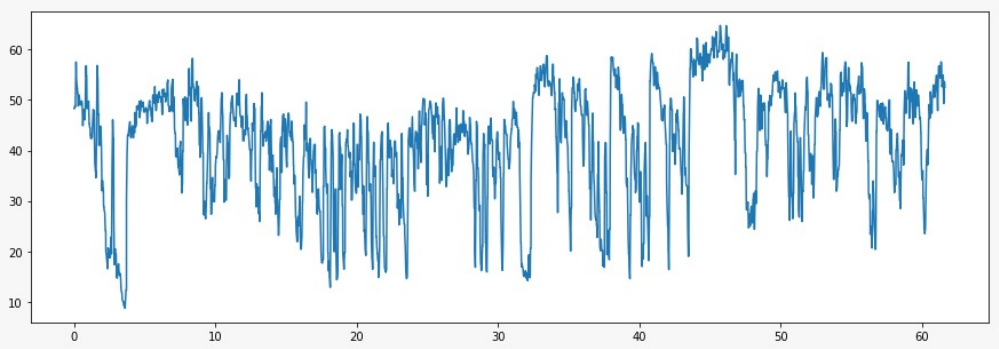


Figure 4: Signal recorded from smartphone's sensor

For this reason we converted our data from dB scale in a linear transormation:

$$L_p = 20 \log_{10} \left(\frac{p}{p_0} \right) db$$

where p_0 is the pressure sound of an acoustic wave (the reference sound pressure is typically the threshold of human hearing, remember that it's 2 x 10-5 Pa) and p the one recorded, obtaining the real shape of the wave recorded from the microphone (without amplifying in dB).

Data obtained like this are as follow in *Fig. 5*:

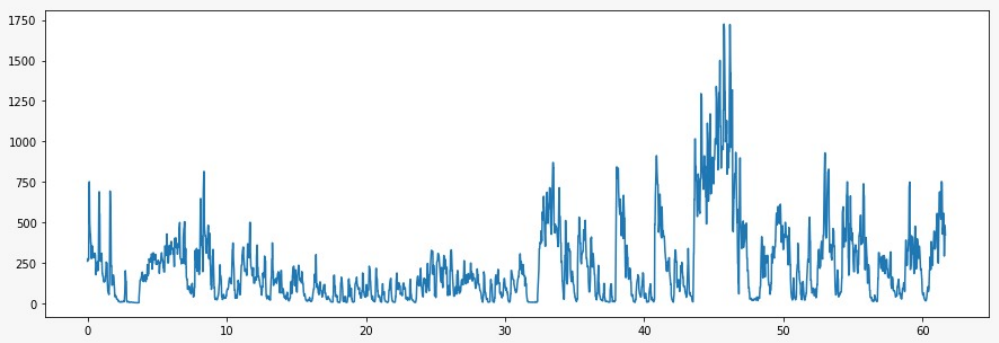


Figure 5: Signal converted from dB scale to linear

For our goal, the spectogram taken from data of that wave shape is not useful yet. In *Fig. 6* we can observe an example of spectogram with a lot of noise and from which is quite hard and impossible differenciate the genres: infact, looking at different songs and genres in this shape, the outcome is always the same.

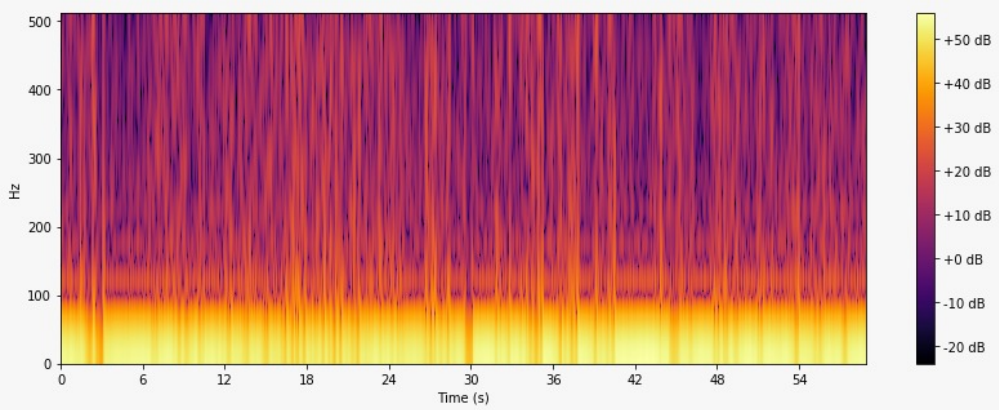


Figure 6: Spectrogram with noise

For having a "readable" spectogram (for our classification), we applied a low-pass filter (that take only low frequencies) in order to have a smoother signal. The example is as follow in *Fig. 7*:

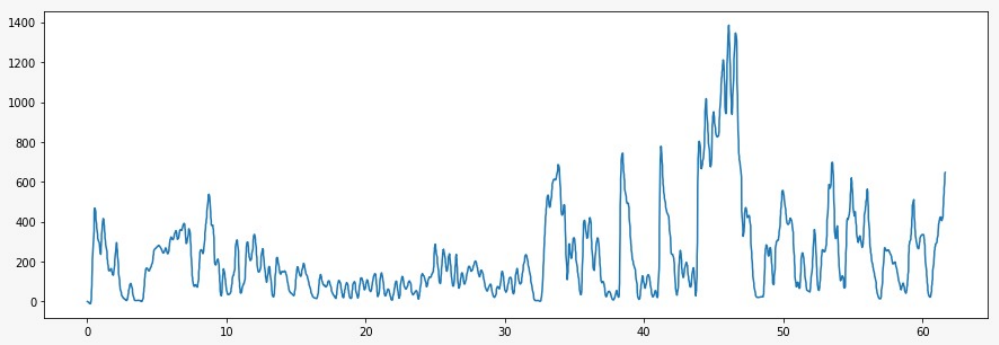


Figure 7: Signal filtered

Obviously our outcome is not close to the ideal one, obtained from a *mp3* file, as showed in *Fig. 8*. Infact, the sample signal of the *mp3* allows to get around 4 millions sample instead of the 4 thousands taken with *SJ*' app using the same signal processing techniques.

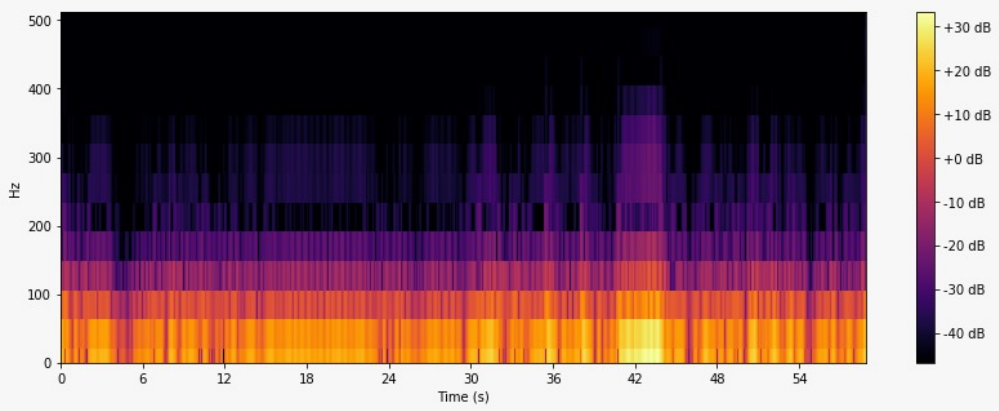


Figure 8: Spectogram without noise

Although our spectogram is poor and quite rudimental, we can observed slight differences between genres (just looking at the figures) but still useful for the aims of the classification.

---

3) Modeling

In order to compare different results, we decided to use different approaches, applying either Neural Network models and classic machine learning algorithms. For each of these methods, we will specify the kind of dataset in input, the model description and the result we obtained.

Figure 9: Neural Network

3.1) Neural Network

In our first implementation we used a classical **Neural Network**. For this model we created a dataset composed by some of the features coming out from the spectrogram of the sound intensity sensor:

- 20 MFCCs (Mel-frequency cepstral coefficients) that concisely describe the overall shape of a spectral envelope;
- Zero Crossing Rate that is the rate of sign-changes along a signal, i.e., the rate at which the signal changes from positive to negative or back;
- Chroma features in which the entire spectrum is projected onto 12 bins representing the 12 distinct semitones (or chroma) of the musical octave;
- Spectral centroids that indicates where the "centre of mass" for a sound is located and is calculated as the weighted mean of the frequencies present in the sound. So spectral centroid for Jazz song will lie somewhere near the middle of its spectrum while that for a Metal song would be towards its end;
- Spectral rolloff that is a measure of the shape of the signal. It represents the frequency below which a specified percentage of the total spectral energy, e.g. 85%, lies;
- Spectral bandwidth that is the Wavelength interval in which a radiated spectral quantity is not less than half its maximum value;
- RMS, the root-mean-square value for each frame, either from the audio samples y or from a spectrogram S;
- The averages values of the brightness and tonality sensor.

Once we splitted the dataset in train and test, we gave it as input to the NN (composed by an input layer) a tensor containing extracted features, three hidden layers and an output layer; we used **RELU** as activation function in input and hidden layers, **Softmax** in the output layer:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

and the sparse categorical cross entropy (CE) as loss function:

$$CE = -\sum_i^C t_i \log(f(s)_i)$$

Here the code used:

```
model = Sequential()

model.add(Dropout(0.2, input_shape=(28,)))
model.add(Dense(60, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(BatchNormalization())

model.add(BatchNormalization())
model.add(Dense(40, activation='relu'))
model.add(BatchNormalization())
```

```
model.add(Dropout(0.2))

model.add(BatchNormalization())
model.add(Dense(20, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(5, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',
optimizer='adadelta', metrics=['accuracy'])

history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=300, verbose=0,batch_size=300)
```

We got a very good score (88% on the test set) using this model.

3.2) Convolutional Neural Network

The second method consists in the implementation of a **Convolutional Neural Network** (CNN). The approach consistently changes since we used as input data the pictures of the spectrogram of each song. Once again we splitted the dataset in train set and test set, composed by 400 and 100 pictures respectively, converted into matrices of numerical values.

But how does it work?

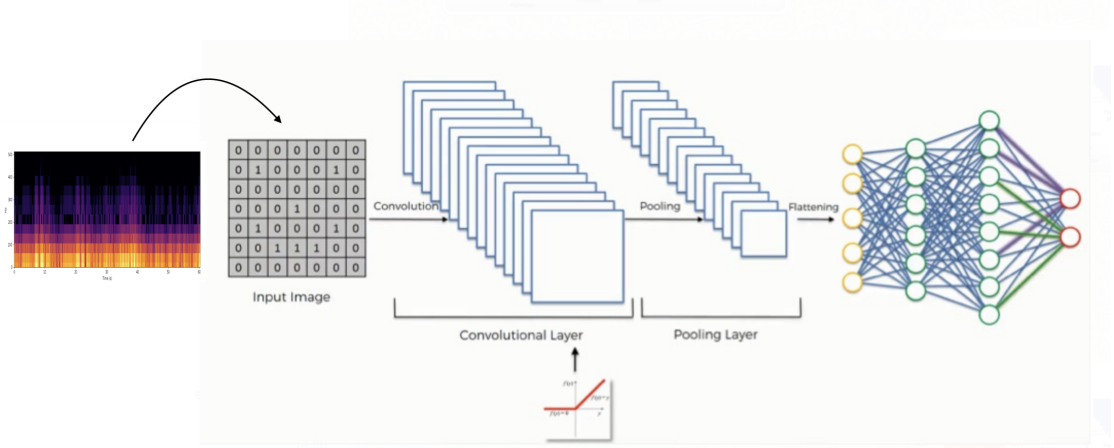


Figure 10: CNN

Every numerical matrices (so, the spectrograms) is scanned by a filter called kernel, i.e. a 5x5 matrix; the kernel starts to shift inside the image matrix, and in every shift step, it performs a matrix multiplication operation between kernel and the portion of the image over which the kernel is hovering.

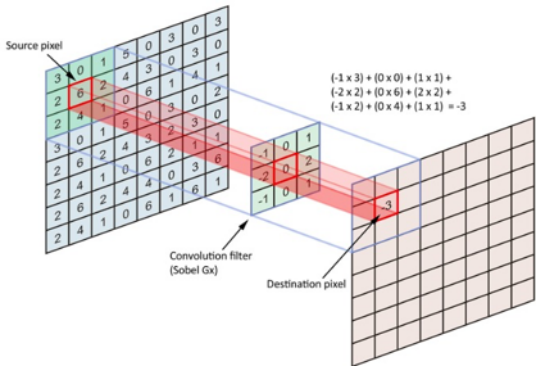


Figure 11: Convolution

The next step is the max pooling in which the filtered matrix is scanned again by a new 3x3 matrix, every time keeping the element with the highest value.

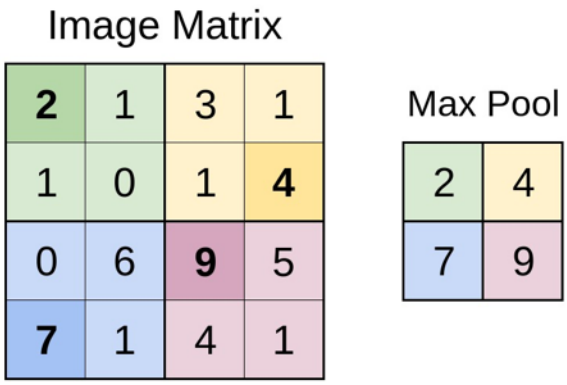


Figure 12: Max Pool

This procedure is repeated three times, reducing consistently the matrix dimension. Finally, the matrix given by the 3 convolutional layers will be converted in a 1D vector and sent to 2 layers of a classical NN in order to be classified. We used the same activation and loss function of the previous NN.

Here the code:

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(5,5),strides=(2, 2),
activation='linear', input_shape=(360, 1008,4),padding = 'valid'))
```

```
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D((3, 3),padding='valid'))
model.add(BatchNormalization())


model.add(Conv2D(64, kernel_size=(5,5),strides=(2,2),
activation='linear',padding = 'valid'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D((3, 3),padding='valid'))
model.add(BatchNormalization())


model.add(Conv2D(128, kernel_size=(5,5),strides=(2,2),
activation='linear',padding = 'valid'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D((2, 2),padding='valid'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(10, activation='linear'))
model.add(LeakyReLU(alpha=0.1))


model.add(Dense(5, activation='softmax'))

model.compile(optimizer='Adadelta', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

history1 = model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=50, verbose=1,batch_size=50)
```

Using this implementation, we got a score of 79% on the test set.

3.3) Recurrent Neural Network

The third implementation consists in a Recurrent Neural Network (RNN). Since this method requires a dataset composed by temporal observation, from every csv the column Decibel Source is selected and then transposed, obtaining a final dataset composed by 500 rows (one row for each song) and as many columns as there are selected observations. Again, the dataset is splitted between train set and test set.

Here there is a brief description of how RNN works: the decision a recurrent net reached at time step t-1 affects the decision it will reach one moment later at time step t. So recurrent networks have two sources of input, the present and the recent past, which combine to determine how they respond to new data, much as we do in life.

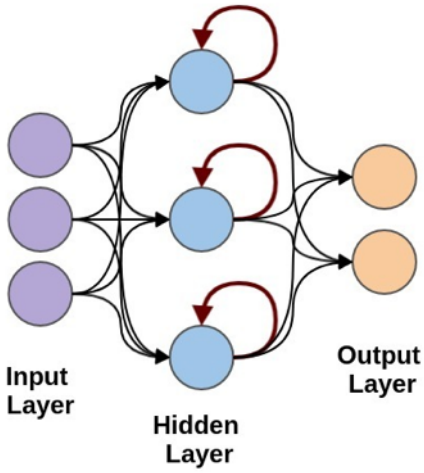


Figure 13: RNN

In order to have a better comprehension of the data, a LSTM model is used as recurrent layer, so we can have data with different dimensions incoming (like in this case).

Here the code:

```
model = Sequential()
model.add(LSTM(128,input_shape=(3000,1),return_sequences=True,recurrent_dropout=0.2))
model.add(BatchNormalization())

model.add(LSTM(64,recurrent_dropout=0.2,return_sequences=True))
model.add(BatchNormalization())

model.add(LSTM(32,recurrent_dropout=0.2))
model.add(BatchNormalization())

model.add(Dense(5, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy',
optimizer='adadelta', metrics=['accuracy'])

history2 = model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=40, batch_size=150,verbose=1)
```

The result obtained is 81% on the test set.

3.4) CRNN

The fourth implementation consists in a hybrid network between CNN and RNN: individually the two units work as described above but CRNN has advantage of convolutional neural networks (CNNs) for local feature extraction and of recurrent neural networks (RNN) for temporal summarization of the extracted features. Specifically, five convolutional layers are used to reduce the size of the images to a 1D vector, which is then given in input to the recurrent layer as time observation.

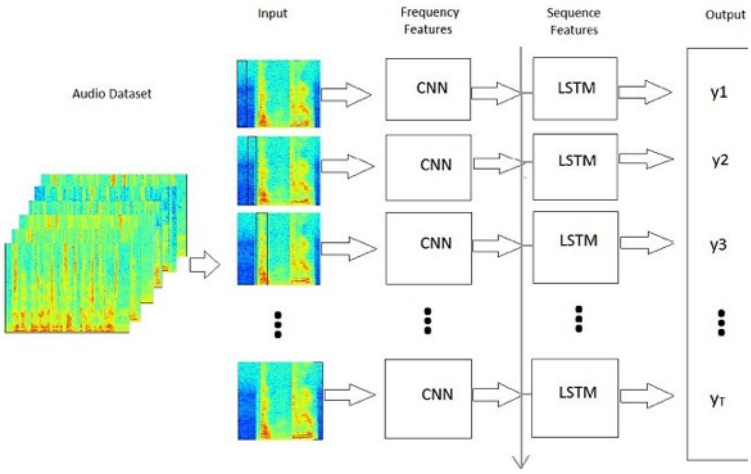


Figure 14: CRNN

The code is the following:

```
input_shape=(360, 1008, 4)

model = Sequential()

model.add(Conv2D(32, kernel_size=(5,5), activation='relu', input_shape= input_shape,
padding = 'valid',trainable=False))
model.add(BatchNormalization(momentum=0.9))
model.add(MaxPooling2D((3,3),strides=(3, 3),padding='valid'))

model.add(Conv2D(32, kernel_size=(4,4), activation='relu',padding = 'valid',
trainable=False))
model.add(BatchNormalization(momentum=0.9))
model.add(MaxPooling2D((3,3),strides=(2, 2),padding='valid'))

model.add(Conv2D(64, kernel_size=(4,4), activation='relu',padding = 'valid',
trainable=False))
model.add(BatchNormalization(momentum=0.9))
model.add(MaxPooling2D((3,3),strides=(2, 2),padding='valid'))

model.add(Conv2D(64, kernel_size=(4,4), activation='relu',padding = 'valid',
trainable=False))
model.add(BatchNormalization(momentum=0.9))
model.add(MaxPooling2D((4,4),strides=(4, 4),padding='valid'))
```

```
model.add(Conv2D(128, kernel_size=(5,5), activation='relu',padding = 'valid',
trainable=False))
model.add(BatchNormalization(momentum=0.9))
model.add(MaxPooling2D(1,strides=(2, 2),padding='valid'))


model.add(Reshape(target_shape=(8, 128)))
model.add(CuDNNLSTM(40, return_sequences=False, name='gru'))
model.add(Dense(5, activation='softmax', name='output'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adadelta',
metrics=['accuracy'])

history3 = model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=30, verbose=1,batch_size=50)
```

Once again, the prediction is obtained with the basic NN layer and the result is 83% on the test-set.

3.5) SVC

The fifth implementation method is SVC, a supervised learning method of the SVM (**Support Vector Machine**) class used for classification.

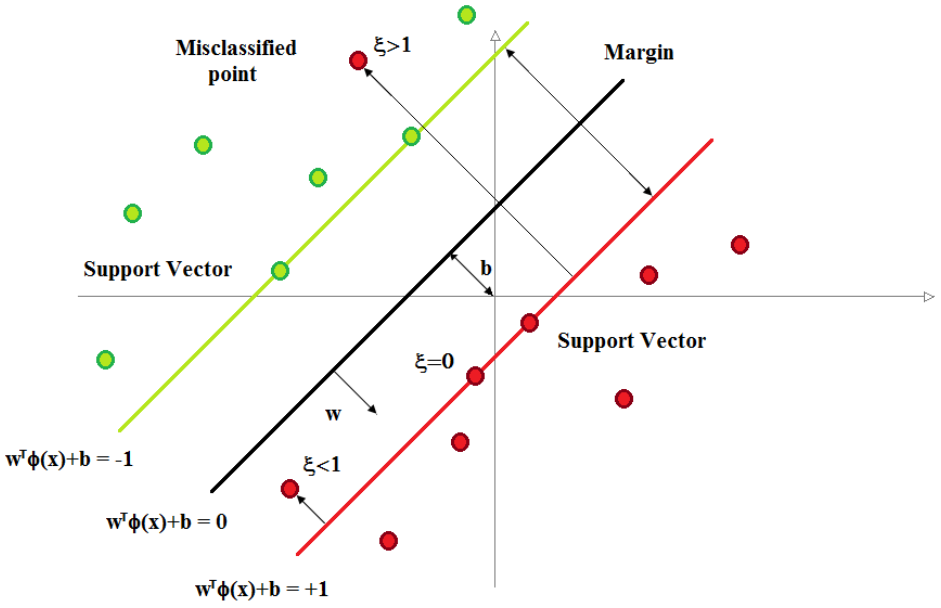


Figure 15: SVC classifier

In this case, using the dataset composed by the 28 features extracted from the spectrogram, splitting it into train and test and using the default parameters a score of 82% on the test was reached.

3.6) Random Forest Classifier

The second machine learning algorithm is Random Forest Classifier which is an ensemble learning method for classification.

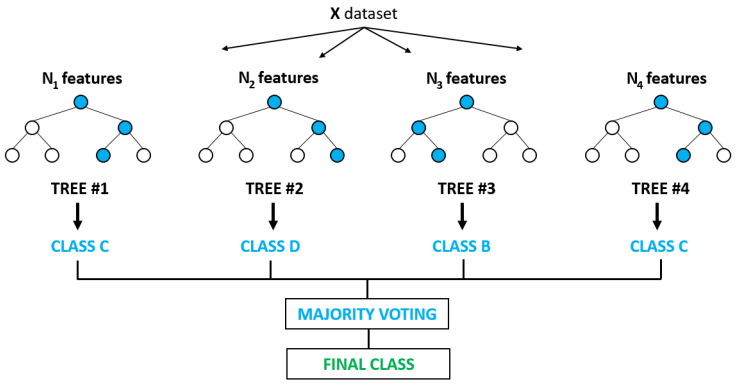


Figure 16: Random Forest Algorithm

Taking in input the dataset as above, this time a Principal Component Analysis has been performed: the PCA is a dimensionality-reduction method used to reduce the dimensionality of a large dataset, by transforming it into a smaller one that still has more information than the original one. To do this the sklearn package was used and, in particular, the explained variance was observed via the command `pca.explained_variance_ratio_`: it tells how much information (variance) each of the principal components represents. In this case the explained variance of five components is 0.99.

The figure shows the trend of the explained variance with respect to the number of components.

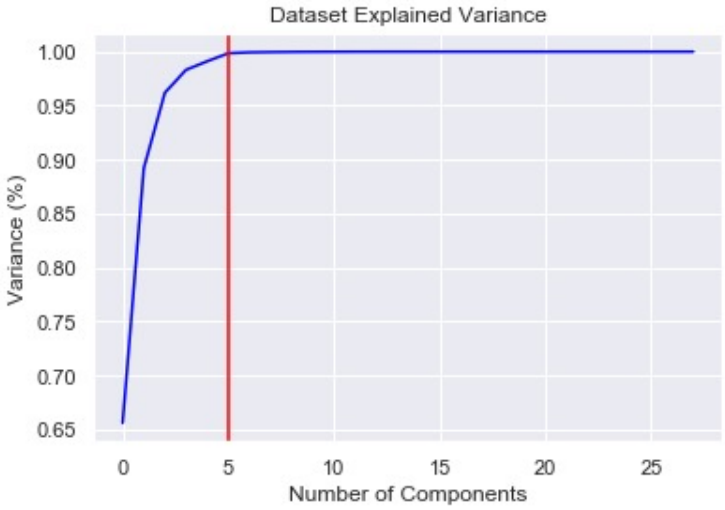
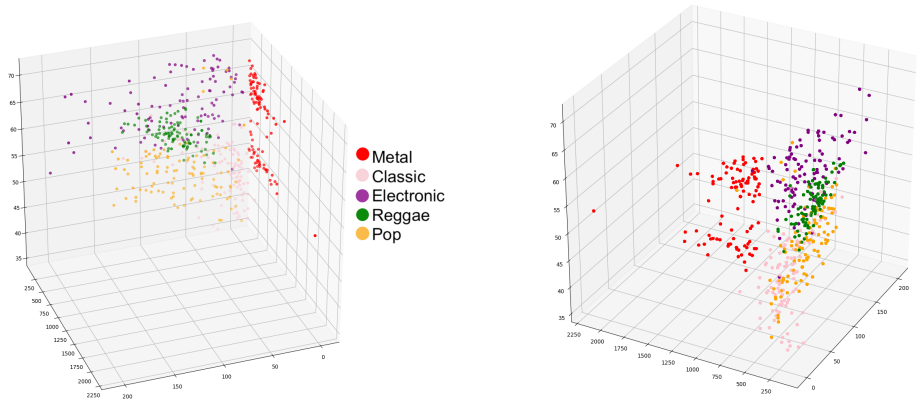


Figure 17

Then simply specifying the number of trees in the forest and the maximum depth of the tree, the result on test is 81%.

3.7) Spectral Clustering

Before to proceed with the clustering, let's take a look at the 3D-plot of each song summarized by the average of the 3 features.



(a) (b)

Figure 18: 3D-plot of songs differentiated for genre

This last method take into account is the Spectral Clustering, where data are considered as nodes of a graph and links are weigthed with the similirity between two nodes. At the beginning we tried with *K-Means* Algorithm but it didn't produce any satisfying outcome. In this algorithm we have K centroids adjusted in an iterative process to find our clusters BUT there are two main problems:

- It makes assumption on the shape of the data (a round sphere, a radial basis);
- It requires multiple restarts at times to find the local minima (i.e. the best clustering).

Spectral Clustering algorithm helps to solve these two problems. This algorithm relies on the power of graphs and the proximity between the data points in order to cluster them, makes it possible to avoid the sphere shape cluster that the K means algorithm forces us to assume. The stages of the algorithm are:

- Constructing a nearest neighbours graph (KNN graph) or radius based graph;
- Embed the data points in low dimensional space (spectral embedding) in which the clusters are more obvious with the use of eigenvectors of the graph Laplacian;
- Use the lowest eigen value in order to choose the eigenvector for the cluster.

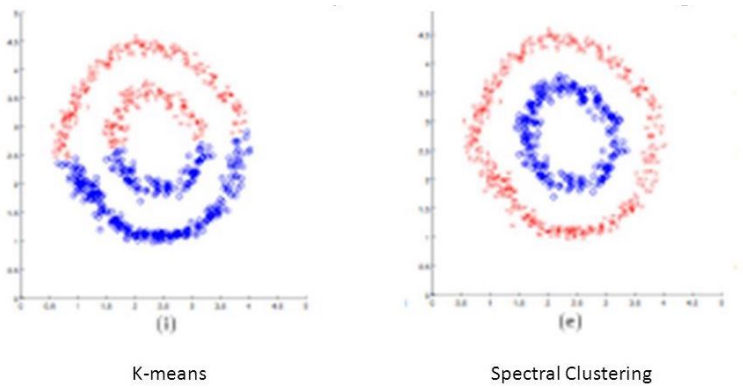


Figure 19: K-means vs Spectral Clustering

Also this time the dataset used is that with the PCA. In order to have an idea on the result you can look at the confusion matrix: as we can see the spectral clustering performed well in classic and metal (85% and 88% respectively), but gives bad result on the other genres.

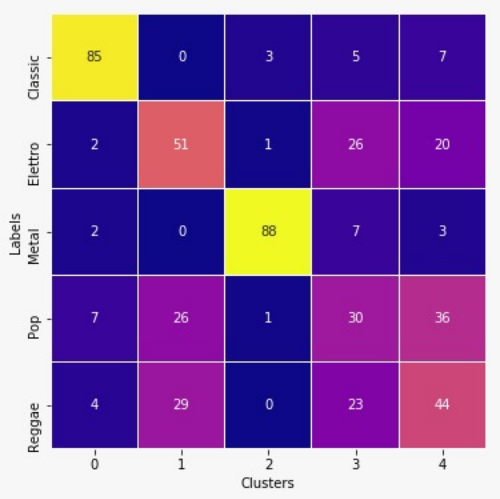


Figure 20: Confusion Matrix

4) Conclusions

Algorithm	***Accuracy***
NN	88%
CRNN	83%
SVC	82%
Random Forest	81%
RNN	81%
CNN	79%

From the analysis performed on our data, it's possible to deduce that in general neural network models outperformed the machine learning ones. The best result reached with standard classifiers is that from SVC (82%), while the absolute best result comes from basic neural network model. Specifically, the basis NN reaches a 88% of accuracy on the test-set: this result comes from a training of `n_epochs=1800` (the overfitting was controled with drop-out factor). In the other NN models the computing power isn't enough to train as in the basic one, so a model with a "right" number of parametes (meaning that tuning of trade off between the information) is created.

In the remaining NN models, since that the computing power isn't suffificient, the computational cost is reduced in order to be able to do training, reaching a good result.

Even with a reduced number of epochs, the score is better compared to the fully-connected models, performed with an higher number of them. A desired (possible) scenario is the one with a really high computing power (and, of course, more data!), pretending to reach (even) more accuracy on test-set. There isn't so much to say about SVC and Random Forest, except for the combined use of Random Forest with dimensionality reduction (PCA) that softly improves the accuracy on the test-set.

Finally, let's spend some words about clustering. Also the spectral clustering is combined with PCA in an unsupervised learning scenario (labels are removed). It's well known that, having a labeled dataset, supervised machine learning almost always outperforms unsupervised learning in classification tasks (however an attempt was done).

The intuition about the use of strobo light is based on the expectation of observing an higher flashes frequency in some genrer instead of others but, as you can see from the confusion matrix, this happens only in metal and classic songs.
