1 Individual project

1.1 Describe the task including the details of the input data (data source, data size, the original number of features and instances, feature types, whether missing values exist, etc.) and the expected output of the system

For this individual project I decided to re-use the Free Music Archive (FMA) data set from [Ben+16][1] from Assignment 2 and perform a music genre classification based on the audio features present in the data set. The FMA data set in [Ben+16] consists of meta data and audio features about Creative Commons licensed audio from 106,574 tracks from 16,341 artists and 14,854 albums, arranged in a hierarchical taxonomy of 161 genres grouped together into 16 top level genres. As part of the FMA a set of 518 pre-computed audio features for each track is available in the features.csv file. The CSV file containing the features is ca. 950MB in size The data set therefore fulfills the assignment criteria of > 30 features and > 20 MB of size. The features were computed with the librosa Python library[2] and the code for the feature extraction is available on the internet[3]. The feature set consist of the following audio signal processing features [Ben+16]:

- 12 dimensional pitch chromagram [Got06; Bro91], computed from a waveform with the help of the Constant-Q Transformation (CQT). Figure 1 shows a spectogram generated by the CQT (covering 7 octaves). This spectogram is turned into a chromatogram broken down by the 12 pitch classes (C, C#, D, D#, E, F, F#, G, G#, A, A#, and B) as can be seen in Figure 2. Each element of the chroma vector corresponds to the sum of magnitude at frequencies of its pitch class in each of six octaves. [4][5]

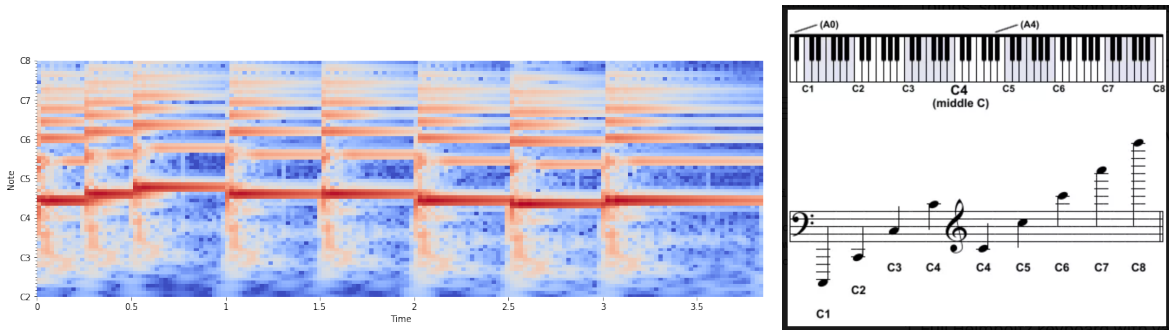

Figure 1: CQT based spectogram and visualisation of octave naming,
Source: `https://musicinformationretrieval.com/` and `https://www.liveabout.com/pitch-notation-and-octave-naming-2701389`

- 12 dimensional pitch chromagram [Got06], like above but computed from a waveform with the help of short-time Fourier transformation (STFT) instead of a CQT.[6][7][8]

- 12 dimensional Chroma Energy Normalized (CENS) pitch chromagram [Mül07; ME11], same as the (CQT) but with the following additional steps after the CQT:

[1] `https://os.unil.cloud.switch.ch/fma/fma_metadata.zip`
[2] `https://github.com/librosa/librosa`
[3] `https://github.com/mdeff/fma/blob/master/features.py`
[4] `https://librosa.org/doc/main/generated/librosa.feature.chroma_cqt.html`
[5] `https://colab.research.google.com/github/stevetjoa/musicinformationretrieval.com/blob/gh-pages/chroma.ipynb`
[6] `https://librosa.org/doc/main/generated/librosa.feature.chroma_stft.html`
[7] `https://colab.research.google.com/github/stevetjoa/musicinformationretrieval.com/blob/gh-pages/stft.ipynb`
[8] `https://musicinformationretrieval.com/chroma.html`

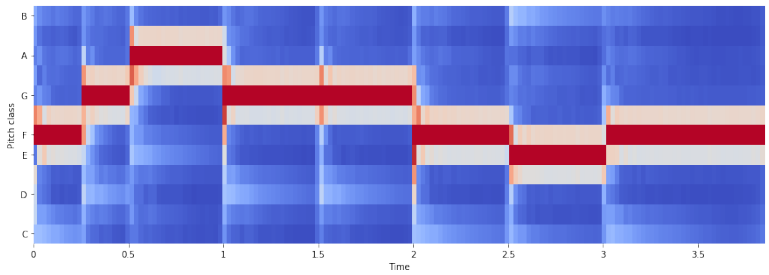

Figure 2: CQT based chromagram, Source: `https://musicinformationretrieval.com/`

  - L-1 normalization of each chroma vector
  - Quantization of amplitude based on "log-like" amplitude thresholds
  - Smoothing with sliding window. Default window length = 41 frames

  The main idea of CENS features is to take statistics over large windows smooths local deviations in tempo, articulation, and musical ornaments such as trills and arpeggiated chords.[9] CENS features are robust to dynamics, timbre and articulation.[10]

- 20 dimensional Mel Frequency Cepstral Coefficients (MFCCs) [RJ93], represents the short-term power spectrum of a waveform, based on a linear cosine transform of the log of a Fourier transformation expressed on a nonlinear mel scale of frequency [11][12][13].

- 6 dimensional tonnetz (harmonic network) [HSG06], which computes tonal centroid features by projecting chroma features onto a 6-dimensional basis representing the perfect fifth, minor third, and major third each as two-dimensional coordinates. [14]

- 1 dimensional root-mean-square (RMS) energy from the audio frame. [15][16]

- 1 dimensional zero-crossing rate in the audio time series of an audio frame. [17] [18]

- 1 dimensional spectral centroid. The magnitude spectrogram for an audio frame is treated as frequency histogram and the mean (centroid) of the distribution is extracted. [19] [20]

- 1 dimensional spectral bandwidth to measure the deviation from the spectral centroid (as measure of frequency dispersion). [21] [22]

- 7 dimensional spectral contrast [Jia+02], where a spectrogram is divided into 7 sub-bands and for each one the mean energy in the top quantile is compared to the one from the bottom quantile

[9] `https://musicinformationretrieval.com/chroma.html`
[10] `https://librosa.org/doc/main/generated/librosa.feature.chroma_cens.html`
[11] https://en.wikipedia.org/wiki/Mel-frequency_cepstrum
[12] `https://librosa.org/doc/main/generated/librosa.feature.mfcc.html`
[13] `https://musicinformationretrieval.com/mfcc.ipynb`
[14] `https://librosa.org/doc/main/generated/librosa.feature.tonnetz.html`
[15] `https://librosa.org/doc/main/generated/librosa.feature.rms.html`
[16] `https://colab.research.google.com/github/stevetjoa/musicinformationretrieval.com/blob/gh-pages/energy.ipynb`
[17] `https://librosa.org/doc/main/generated/librosa.feature.zero_crossing_rate.html`
[18] `https://colab.research.google.com/github/stevetjoa/musicinformationretrieval.com/blob/gh-pages/zcr.ipynb`
[19] `https://librosa.org/doc/main/generated/librosa.feature.spectral_centroid.html`
[20] `https://colab.research.google.com/github/stevetjoa/musicinformationretrieval.com/blob/gh-pages/spectral_features.ipynb`
[21] `https://librosa.org/doc/main/generated/librosa.feature.spectral_bandwidth.html`
[22] `https://colab.research.google.com/github/stevetjoa/musicinformationretrieval.com/blob/gh-pages/spectral_features.ipynb`

to calculate the energy contrast. [23] [24]

- 1 dimensional spectral rolloff, defined here as the 85-percentile frequency of a spectrogram (0.85 of the energy of the spectrum is at or below this frequency).[25][26]

All of the above dimensions are repeatedly calculated over many time slices of an audio file, leading to a distribution of values for each dimension. To arrive at scalar feature values for each of these distribution the kurtosis, max, mean, median, min, skew and standard deviation were calculated to constitute the actual features in the data set and capture some aspects of the distributions in form of scalar descriptors. The feature columns of the data set have no missing values, but over half of the class labels are missing (c.f. Table 1). An initial analysis of the features revealed that some features have very little variance in their values or in other terms very little dispersion. This can be seen in Table 4 in the annex and concerned mainly the maximum of the STFT and CQT chromagrams. It is highly likely that these features would not carry much predictive value and we will revisit this subject when analysing the results of the classification. The expected output of the classification program are class labels and/or class probabilities predicted based on the the trained model and input data. I used gradient boosted decision trees for this project. The main idea of gradient boosted decision trees is illustrated in Figure 3, they are an ensemble of "simple" decision trees, that get trained in a sequential fashion to focus on the wrongly classified instances of the previous iteration. Collectively these trees become a very strong classifier.

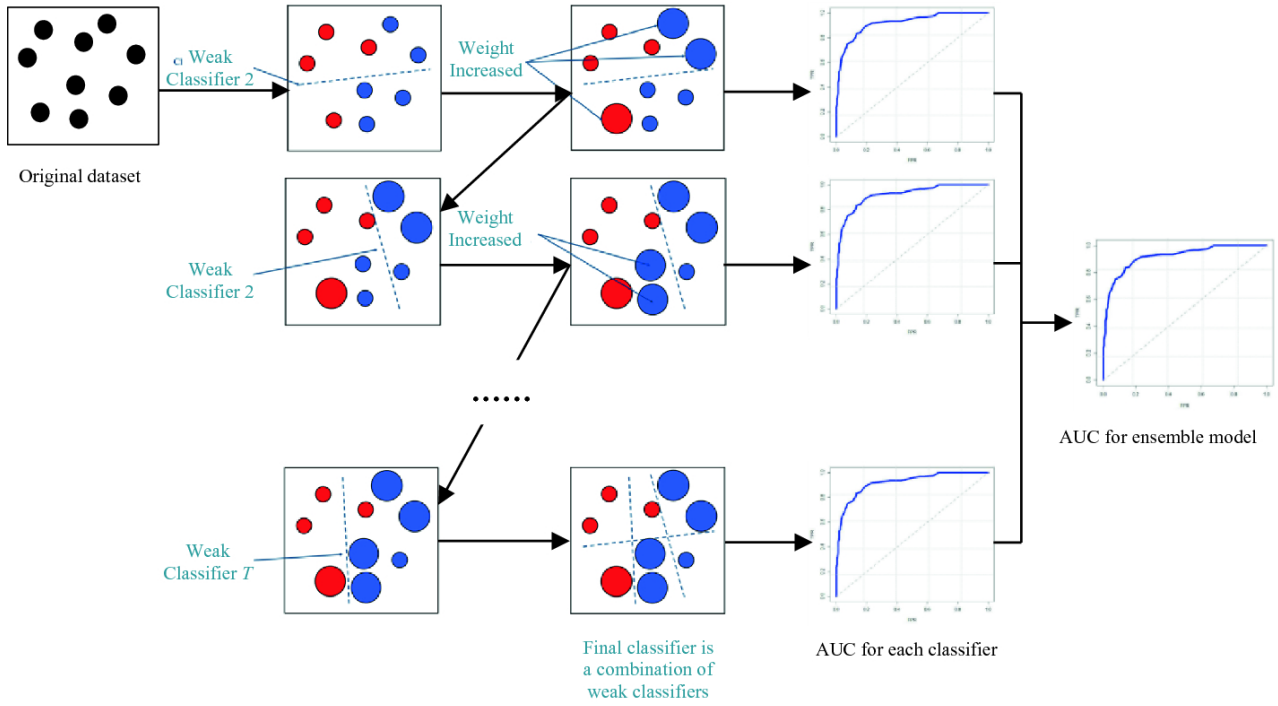

Figure 3: Schematic explanation of gradient boosted decision trees,
Source: `https://datascience.eu/machine-learning/gradient-boosting-what-you-need-to-know/`

1.2 Describe all the pre-processing steps that are applied to the downloaded data file(s) to obtain the data set that is used as input of your program, which must be at least 20MB and have at least 30 features.

[23] `https://colab.research.google.com/github/stevetjoa/musicinformationretrieval.com/blob/gh-pages/spectral_features.ipynb`
[24] `https://librosa.org/doc/main/generated/librosa.feature.spectral_contrast.html`
[25] `https://librosa.org/doc/main/generated/librosa.feature.spectral_rolloff.html`
[26] `https://colab.research.google.com/github/stevetjoa/musicinformationretrieval.com/blob/gh-pages/spectral_features.ipynb`

The data set can be downloaded from `https://os.unil.cloud.switch.ch/fma/fma_metadata.zip`. The zip file contains several csv files. For the purpose of this assignment I joined the features.csv table with the tracks.csv table on the track ID column. The features table contains all the audio features of the tracks, while the tracks table contains a lot of descriptive meta-data (artist, album, song title, genre). I only make use of the genre as a class label and don't use any of the other descriptive meta-data fields for predictive purposes. It is fairly obvious, that e.g. the artist name would carry a lot of predictive information about the genre of a song (if present in the training and test data set). But I decided to restrict myself to audio features for this classification task to make things more challenging and interesting. A pre-wrangled version of the data can be downloaded from `https://github.com/econdatatech/AIML427/blob/main/mfa_wrangled.bz2`. The data set was split 70%:30% into training and test data for the purpose of these experiments, but the above data set is the un-split set.

Genre	count
NaN	56976
Rock	14182
Experimental	10608
Electronic	9372
Hip-Hop	3552
Folk	2803
Pop	2332
Instrumental	2079
International	1389
Classical	1230
Jazz	571
Old-Time/Historic	554
Spoken	423
Country	194
Soul-RnB	175
Blues	110
Easy Listening	24

Table 1: Class distribution of the FMA data set

With regards to the Genre information I found a considerable amount of missing class labels and we can also see that the data set is highly imbalanced with regards to the different classes. I decided to remove all instances with a missing class label. The Spark-ML GBTClassifier is not able to perform multi class classification. So restricted the data set further to only include the Genres "Rock" and "Experimental", leading to 24790 instances with 518 features (which is still very well within the minimum limits for this assignment). As comparison I ran a multi-class classification with Microsoft's LightGBM library [Ke+17] on my home computer. For that purpose I restricted the classification task to the 3 biggest genres: Electronic, Experimental and Rock. This led to a data set containing 34162 instances.

1.3 Describe how to install and run the program step by step in a Readme.txt file. (This can be skipped in the presentation.)

1.3.1 Introduction

I did 3 different pieces of programming for this assignment.

- the bare knuckle python script to run things on the Hadoop cluster. This code is contained in the GBM.py script located at `https://github.com/econdatatech/AIML427/blob/main/GBM.py`. Unfortunately the Spark-ML GBTClassifier is not able to perform multi class classification. I therefore ran Microsoft's LightGBM library on my home computer with 3 classes as comparison and to make things more interesting (c.f. item number 3 further down)

- a Jupyter notebook that contains much of the same code as the GBM.py but is designed to run in Google Colab and mainly served as the development version and testing platform. This notebook is available from `https://github.com/econdatatech/AIML427/blob/main/AIML_427_Assignment_3_Individual_Part_PySpark_Colab.ipynb`

- I felt that the whole approach via Hadoop was pretty cumbersome and a lot of time and effort went into the "plumbing" just to make things work on the cluster rather than to solve a data mining problem. Furthermore I felt that many of the algorithms of the Spark-ML library weren't up to the state of the art. I was curious to find out how a desktop approach competes with the cluster and used Microsoft's LightGBM library [Ke+17] on my local laptop as comparison. This notebook is available from `https://github.com/econdatatech/AIML427/blob/main/Assignement%203%20AIML%20427%20LightGBM.ipynb`. This is also the notebook I used to generate some more graphics and investigations for this report, for which I used 3 classes instead of a binary classification problem. I felt it just made things a bit more interesting.

1.3.2 Reproducibility

All three pieces of code need the original data which can be downloaded from `https://os.unil.cloud.switch.ch/fma/fma_metadata.zip`. One has to unzip it and place the tracks.csv and features.csv files contained in the zip file in the directory referenced in the various pieces of code.

- for 2) in same folder as the code file in the google Colab runtime environment. It's quicker to first upload the data to google drive and then drag it into Colab from there than to upload it directly to Colab. That said, there is a section in the Notebook (that is currently commented out) that allows to download a pre-wrangled data set from `https://github.com/econdatatech/AIML427/blob/main/mfa_wrangled.bz2`
- for 3) the dataset also needs to be sitting in the working directory of the Notebook in the subfolder fma_metadata. Again there is an option to download a pre-wrangled dataset from `https://github.com/econdatatech/AIML427/blob/main/mfa_wrangled.bz2` and this code cell is currently not commented out.
- for 1) the directory would be */user/* $< USERNAME >$ */input* on the Hadoop file system. That said, the tracks.csv file of the data set is corrupted in the sense that it contains rather arbitrary line breaks and mixes quoted string with unquotes strings with commas and line breaks inside these strings. Any attempt to turn this file into a clean data frame with just spark code failed. The read_csv function from the pandas package had no issue with the file thought. Unfortunately using pandas on the Hadoop cluster failed as well and I also didn't manage to pass on the pandas package as part of my spark submit command. After considerable efforts I decided to use the pre-wrangled data set from `https://github.com/econdatatech/AIML427/blob/main/mfa_wrangled.bz2` on the cluster. The wrangling part is visible in `https://github.com/econdatatech/AIML427/blob/main/AIML_427_Assignment_3_Individual_Part_PySpark_Colab.ipynb`.

In terms of reproducability of the execution, the notebooks 2) and 3) are rather straight forward. We just need a Jupyter notebook environment and should be good to go. I have not tried to run 3) in Colabs though. Compared to the Jupyter notebooks the execution for 1) is a bit more involved. We need to upload the data to barretts.ecs.vuw.ac.nz. Either from the home computer via scp or from barrets with wget from the internet[27]. Once the data and the GBM.py is on barretts inside the home directory one needs to create an input and output directory, put the data into the Hadoop file system, execute the code, wait and finally download the result to the home directory. For all of this one first needs to logon to the cluster to start off with. When using ssh from OSX I didn't end up on a cshell, so I had to amend certain parts of the instructions contained in the lab tutorial. My approach is outlined below.

```
ssh <USERNAME>@barretts.ecs.vuw.ac.nz

ssh co246a-1

export HADOOP_VERSION=2.8.0
export HADOOP_PREFIX=/local/Hadoop/hadoop-$HADOOP_VERSION
export SPARK_HOME=/local/spark
export PATH=${PATH}:$HADOOP_PREFIX/bin:$SPARK_HOME/bin
export HADOOP_CONF_DIR=$HADOOP_PREFIX/etc/hadoop
export YARN_CONF_DIR=$HADOOP_PREFIX/etc/hadoop
export PYSPARK_PYTHON=python3.7
export LD_LIBRARY_PATH=$HADOOP_PREFIX/lib/native:$JAVA_HOME/jre/lib/amd64/server
need java8
```

[27] e.g. `wget -Omfa_wrangled.bz2https://github.com/econdatatech/AIML427/blob/main/mfa_wrangled.bz2?raw=true`

```
hadoop fs -mkdir /user/<USERNAME>/output
hadoop fs -mkdir /user/<USERNAME>/input

hadoop fs -put mfa_wrangled.bz2 /user/<USERNAME>/input

spark-submit --master yarn --deploy-mode cluster GBM.py \
/user/<USERNAME>/input /user/<USERNAME>/output

hadoop fs -copyToLocal /user/<USERNAME>/output
```

Due to the size of the ensemble models in terms of number of decision trees, there was no sensible way to turn them into a graphical or string representation that would fit inside this report. That said, at least the models of my LightGBM experiments have been stored in the pickle format and put on GitHub:

- No pre-processing: `https://github.com/econdatatech/AIML427/blob/main/modellg.pkl`
- Scaled features: `https://github.com/econdatatech/AIML427/blob/main/modellgsc.pkl`
- No pre-processing: `https://github.com/econdatatech/AIML427/blob/main/modellgpca.pkl`

1.4 Compare and discuss the results (including the training and test accuracy, the running time, the model, etc. depending on the program) of the program with and without normalising/scaling data.

As described in section 1.2 and section 1.3, I performed three different experiments:

- a classification with LightGBM [Ke+17] on my home laptop. Using 100 trees (default setting) to classify 3 Genres (Rock, Experimental and Electronic) across 34162 instances.
- a classification with Spark-ML GBDT on Google Colabs. Using 20 trees (default setting) to classify 2 Genres (Rock and Experimental) across 24790 instances
- a classification with Spark-ML GBDT on the ECS Hadoop cluster. Using 20 trees (default setting) to classify 2 Genres (Rock and Experimental) across 24790 instances

As per the assignment I ran each experiment on unscaled raw data, on scaled data and finally on the components of a PCA. I did not perform any explicit feature selection as decision trees perform implicit feature selection. Particularly if we talk about an ensemble of trees as one doesn't rely on a singular decision tree to make a call about feature importance. Below results in Table 2 are expressed in terms of balanced accuracy for the LightGBM experiment as the class distribution was slightly imbalanced in the source data set. The balanced accuracy is the average of the recall scores of each class[28]. The LightGBM default settings re-weight the class instances during the training according to their distribution in the training data set. Therefore no separate correction step for the class imbalanced was performed (e.g. under-sampling or over-sampling). The Spark-ML accuracy is reported in unbalanced accuracy and I'm not sure a weight rebalancing happens inside the algorithm according to the class distribution in the input data (as is the case for LightGBM).
We can see from Table 2 there is a gap between the training performance and the test performance. This is to be expected. What is interesting though is that the gap is smaller for the Spark-ML experiment with 20 trees. This indicates that the 100 trees in LightGBM over-fit the training data, which not only leads to a larger gap, but also to an overall poorer test set performance. That said, we can also see, that the cross-validation score for the LightGBM is in both cases a rather good indication of the test score. I ran 3 rounds of 10-fold stratified cross validation to estimate the score. So one could invest time to tweak the default parameters of the algorithm (hyper-parameter optimization) to

[28] `https://scikit-learn.org/stable/modules/model_evaluation.html#balanced-accuracy-score`

Algorithm	Pre-proc.	Training	X-val. mean (std)	Test	Run time
LightGBM	None	0.929	0.807 (0.009)	0.800	0.299 min.
LightGBM	Scaled/Normalized	0.931	0.806 (0.009)	0.800	0.275 min.
Spark-ML GBDT (Colab)	None	0.868	not done	0.831	15.42 min.
Spark-ML GBDT (Colab)	Scaled/Normalized	0.867	not done	0.826	17.33 min.
Spark-ML GBDT (Hadoop)	None	0.868	not done	0.829	3.28 min.
Spark-ML GBDT (Hadoop)	Scaled/Normalized	0.868	not done	0.829	5.34 min.

Table 2: Classification result of different scenarios expressed in balanced accuracy

narrow that gap. I did not perform any cross validation experiments in Colab or the cluster, as the run-time would have been rather large.
The overall results between raw and scaled features are almost identical across all three experiments. The run time is longer for Spark-ML as seemingly the standardization of the features takes a measurable amount of time. The test error is identical between raw and scaled features (with the exception of the Colab results). In a way the results are not surprising, as decision trees are not a similarity based method (like KNN) where it might be important that the features are brought to the same scale. The tree doesn't really care on what scale it splits a continuous feature into two leafs. Figure 4 provides

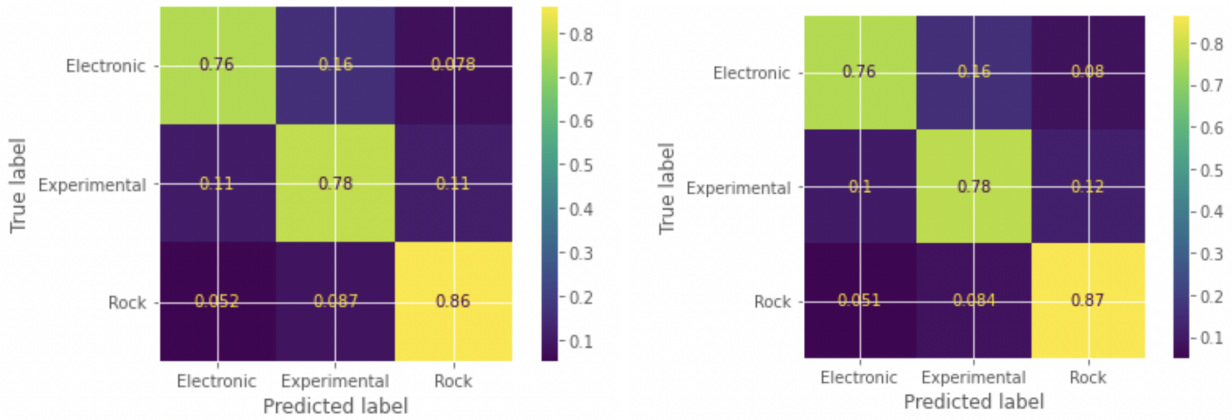

Figure 4: Confusion matrices for raw features (left) vs. scaled/normalized (right)

the confusion matrices for the test set of the raw features vs. the scaled/normalized features for the LightGBM experiment. Within each matrix the accuracy for the Electronic and Experimental genre is pretty similar while the Rock genre has a far better class separation from the rest. Between the two matrices there isn't too much difference. The scaled features lead to slightly better test accuracy for the rock genre, but with class balancing of the accuracy and rounding this advantage is not visible anymore in the test score in Table 2.
In terms of feature importance, we can be see in Table 5 as well as in Figure 10 that the mfcc and spectral contrast features dominate the field in terms of absolute feature importance across the ensemble (measured in split gains). The results of Table 5 and Figure 10 were restricted to the top 50 important features. When calculating the average feature importance by feature class over the top 50 important features the mfcc interestingly remains number one, but now the spectral centroid takes the second place. In terms of least important features one can have a look at Table 7. Chroma cqt and chroma stft dominate the field. In these two groups we find the exact features that were already identified during the exploratory analysis to have extremely low dispersion or variance (c.f. Table 4). In retrospect it would have been justified to exclude these features. Given that decision trees perform implicit feature selection there was no negative impact on the classification accuracy. There might have been some impact on the run time performance, but it was worth it in terms of empirically testing if decision trees indeed ignore "worthless" features (which they did).

I quickly investigated if there was any notable difference in the feature importance of the scaled features, but the results were almost identical between the two data sets.

1.5 Compare and discuss the results (including the training and test accuracy, the running time, the model, etc. depending on the program) of the program with and without transforming data using PCA.

Figure 5 and 6 show the results of the PCA that was performed on the 518 scaled/normalized features. We can see in the scree plot that the magnitude of the component eigenvalues diminishes very quickly after the first ca. 20-30 eigenvalues. That said, in figure 6 we can see, that we need nevertheless almost the first 100 components to explain 80% of the variance.

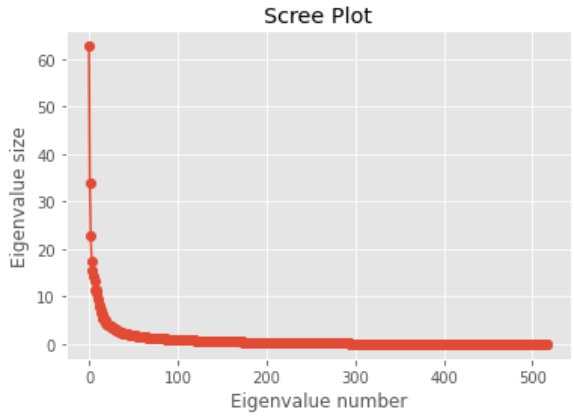

Figure 5: Scree plot of PCA results

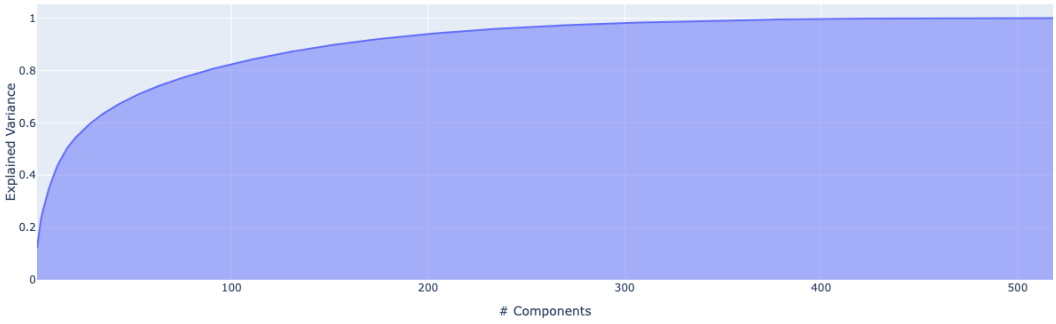

Figure 6: PCA results - Cumulative explained variance

I performed all classification experiments with all 518 principle components. The results are shown in Table 3. Again there are signs of over-fitting for the LightGBM and again the Spark-ML experiment leads to better test scrores. That said, again the cross-validation score in the LightGBM case is a very good indicator of the final test score and could be used to guide hyper-parameter optimization of one wanted to invest the time.
In terms of performance comparison between PCA and non-PCA, there is a minor drop in training performance with PCA processed input data across all three experiments. A slightly more sizable drop can be witnessed for the test score. The drop is the most pronounced in case of the LightGBM experiment. The results are again not too surprising. Given the implicit feature selection capabilities of decision trees, the PCA isn't providing much advantage compared to what the decision trees can do themselves when it comes to focusing on the right types of features. The problem with the PCA is, that one has to scale and perform the PCA on the training and test data separately and in my mind

Algorithm	Pre-proc.	Training	X-val. mean (std)	Test	Run time
LightGBM	None	0.929	0.807 (0.009)	0.800	0.299 minutes
LightGBM	PCA components	0.921	0.772 (0.007)	0.773	0.280 minutes
Spark-ML GBDT (Colab)	None	0.868	not done	0.831	15.42 min.
Spark-ML GBDT (Colab)	PCA	0.849	not done	0.807	20.31 min.
Spark-ML GBDT (Hadoop)	None	0.868	not done	0.829	3.28 min.
Spark-ML GBDT (Hadoop)	PCA	0.850	not done	0.812	3.92 min.

Table 3: Classification result of no PCA vs PCA expressed in balanced accuracy

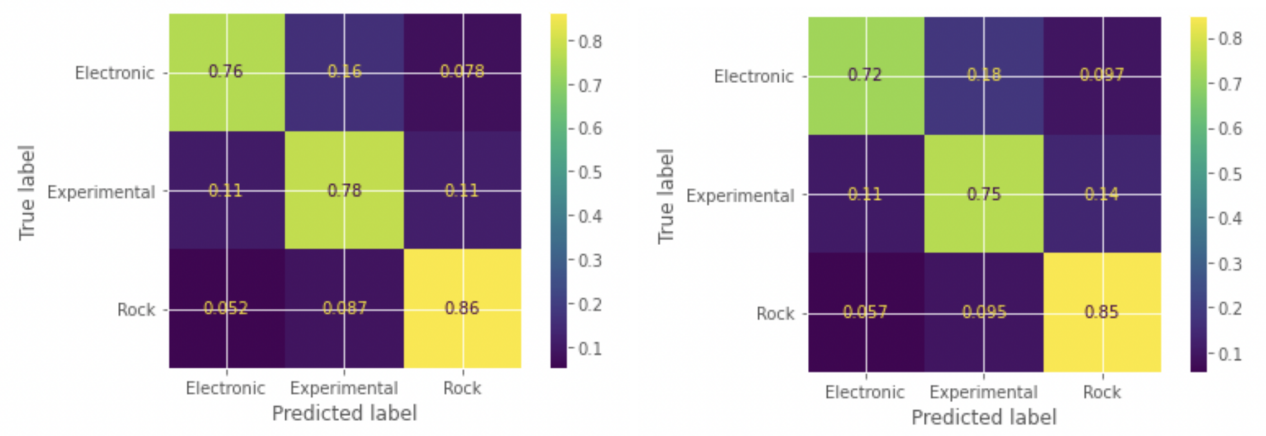

Figure 7: Confusion matrices for raw features (left) vs. principal components (right)

should use the scaler and the PCA from the training data and apply them to the test data. That might introduce additional problems when there are difference in the nature of the training and test data. Last but not least the runtime for the Spark-ML algorithms was extended due to the time it took to scale the features and perform the PCA. Figure 7 provides the confusion matrices from the LightGBM experiment for the test set of the raw features vs. the principal components. Within each matrix the accuracy for the Electronic and Experimental genre is pretty similar (though the gap is wider in the PCA version), while the Rock genre has a far better class separation from the rest (in both cases). Between the two matrices there are some differences in the per class accuracies. The raw features lead to slightly better test accuracy for the Rock genre, and even better performance for the Electronic and Experimental genre. This difference is also visible in the final test score in Table 3. One interesting finding from performing the PCA was that the empirical feature importance of the principal components doesn't follow the natural ordering of the components in terms magnitude of the eigenvalue as can be seen in Figure 8. I found that intriguing. It's not inexplicable in the sense that variance in the input space doesn't have to be oriented along an axis that separates the classes very well. That said, I still find it a valuable empirical lesson in the sense that there is frequently a tendency to only feed the top X principal components into an algorithm (as a means of feature selection). As we can see from the empirical results in Figure 8, that can be a misguided approach (depending on the data set and the classification or regression algorithm at hand).

1.6 Describe the program using UML class diagrams and/or pseudo-code

As there isn't much going on in my program apart from a split in training and testing data and feature scaling or PCA, I decided to include the pseudo code of the LightGBM algorithm. The normal Gradient boosted Decision Tree algorithm would neither include the EFB technique in step 1 nor the GOSS in step 5 which would be replaced by a plain vaniall resampling based on weights.

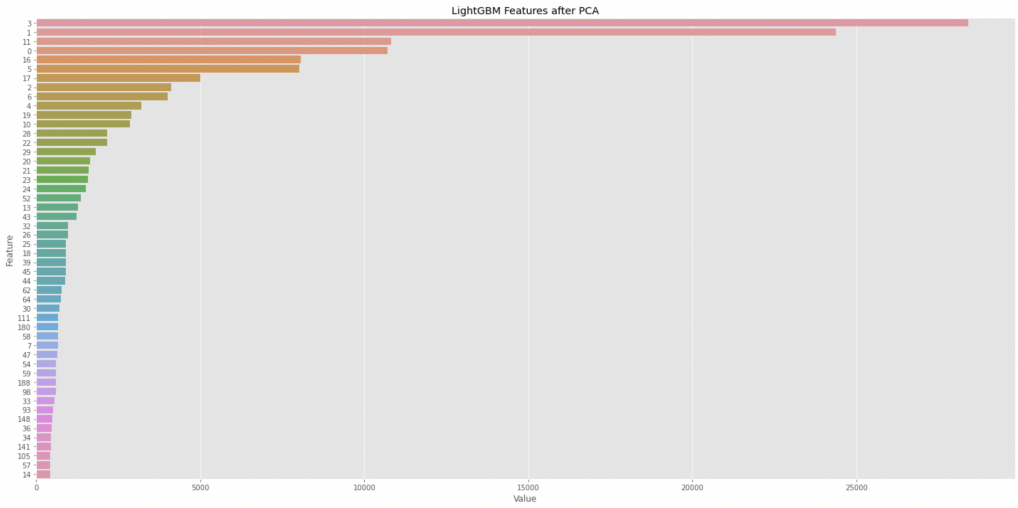

Figure 8: Feature importance plot of top 50 PCA components

The LightGBM algorithm
Input:
Training data: D = {(χ1, y1), (χ2, y2), ..., (χN, yN)}, χi ∈χ, χ ⊆ R, yi∈{−1,+1}; loss function: L(y, θ (χ));
Iterations:
M; Big gradient data sampling ratio: a; slight gradient data sampling ratio: b;
1: Combine features that are mutually exclusive (i.e., features never simultaneously accept nonzero values) of χi, i = {1, ...,N} by the exclusive feature bundling (EFB) technique;
2: Set $\theta_0(\chi) = arg\ min_c \sum_i^N L(y_i, c)$;
3: For m = 1 to M do
4: Calculate gradient absolute values:

$$r_i = \left| \frac{\partial L(y_i, \theta(x_i))}{\partial \theta(x_i)} \right|_{\theta(x)=\theta_{m-1}(x)}, i = \{1, \ldots, N\}$$

5: Resample data set using gradient-based one-side sampling (GOSS) process:
$topN = a \times len(D)$; $randN = b \times len(D)$;
$sorted = GetSortedIndices(abs(r))$;
$A = sorted[\ 1 : topN]$; $B = RandomPick(sorted[\ topN : len(D)]\ , randN)$;
$\acute{D} = A + B$;
6: Calculate information gains:

$$V_j(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_l} r_i + \frac{1-a}{b}\sum_{x_i \in B_l} r_i\right)^2}{n_l^j(d)} + \frac{\left(\sum_{x_i \in A_r} r_i + \frac{1-a}{b}\sum_{x_i \in B_r} r_i\right)^2}{n_r^j(d)} \right)$$

7: Develop a new decision tree $\theta_m(x)'$ on set D'
8: Update $\theta_m(\chi) = \theta_{m-1}(\chi) + \theta_m(\chi)$
9: End for
10: Return $\tilde{\theta}(x) = \theta_M(x)$

Figure 9: Pseudo code of the LightGBM algorithm. Taken from [Dun+21]

References

[Ben+16] Kirell Benzi et al. “FMA: A Dataset For Music Analysis”. In: *CoRR* (2016). URL: http://arxiv.org/abs/1612.01840.

[Bro91] Judith C. Brown. “Calculation of a constant Q spectral transform”. In: *The Journal of the Acoustical Society of America* 89.1 (1991), pp. 425–434.

[Dun+21] Nachiket Dunbray et al. “A Novel Prediction Model for Diabetes Detection Using Gridsearch and A Voting Classifier between Lightgbm and KNN”. In: Oct. 2021, pp. 1–7.

[Got06] M. Goto. “A chorus section detection method for musical audio signals and its application to a music listening station”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 14.5 (2006), pp. 1783–1794.

[HSG06] Christopher Harte, Mark Sandler, and Martin Gasser. “Detecting Harmonic Change in Musical Audio”. In: *Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia*. AMCMM ’06. Santa Barbara, California, USA: Association for Computing Machinery, 2006, pp. 21–26.

[Jia+02] Dan-Ning Jiang et al. “Music type classification by spectral contrast feature”. In: *Proceedings. IEEE International Conference on Multimedia and Expo*. Vol. 1. 2002, 113–116 vol.1.

[Ke+17] Guolin Ke et al. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[ME11] Meinard Müller and Sebastian Ewert. “Chroma Toolbox: MATLAB implementations for extracting variants of chroma-based audio features”. In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR)*. Miami, USA, 2011.

[Mül07] Meinard Müller. *Information Retrieval for Music and Motion*. Springer Verlag, 2007. ISBN: 3540740473.

[RJ93] L. R. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ: Prentice Hall, 1993.

Annex

feature	mean	std	min	25%	50%	75%	max
chroma_stft.max.12	0.999908	0.003189	0.796108	1.0	1.0	1.0	1.0
chroma_stft.max.10	0.999928	0.003767	0.513524	1.0	1.0	1.0	1.0
chroma_stft.max.01	0.999869	0.004130	0.665060	1.0	1.0	1.0	1.0
chroma_stft.max.05	0.999901	0.004192	0.680290	1.0	1.0	1.0	1.0
chroma_stft.max.03	0.999884	0.004487	0.548158	1.0	1.0	1.0	1.0
chroma_stft.max.04	0.999880	0.004519	0.492757	1.0	1.0	1.0	1.0
chroma_stft.max.06	0.999853	0.004936	0.669812	1.0	1.0	1.0	1.0
chroma_stft.max.02	0.999785	0.005263	0.721503	1.0	1.0	1.0	1.0
chroma_stft.max.11	0.999836	0.005317	0.582979	1.0	1.0	1.0	1.0
chroma_stft.max.08	0.999818	0.006276	0.533016	1.0	1.0	1.0	1.0
chroma_stft.max.09	0.999763	0.006653	0.584318	1.0	1.0	1.0	1.0
chroma_stft.max.07	0.999722	0.007608	0.486357	1.0	1.0	1.0	1.0
chroma_cqt.max.01	0.999771	0.007794	0.495821	1.0	1.0	1.0	1.0
chroma_cqt.max.03	0.999662	0.009210	0.327677	1.0	1.0	1.0	1.0
chroma_cqt.max.04	0.999626	0.009355	0.311660	1.0	1.0	1.0	1.0
chroma_cqt.max.05	0.999553	0.010041	0.362643	1.0	1.0	1.0	1.0
chroma_cqt.max.06	0.999543	0.010411	0.426659	1.0	1.0	1.0	1.0
chroma_cqt.max.08	0.999580	0.010416	0.369523	1.0	1.0	1.0	1.0
chroma_cqt.max.02	0.999592	0.010975	0.351773	1.0	1.0	1.0	1.0
chroma_cqt.max.07	0.999481	0.011604	0.340675	1.0	1.0	1.0	1.0
chroma_cqt.max.09	0.999468	0.012030	0.377115	1.0	1.0	1.0	1.0
chroma_cqt.max.12	0.999361	0.013233	0.332006	1.0	1.0	1.0	1.0
chroma_cqt.max.10	0.999334	0.013911	0.401990	1.0	1.0	1.0	1.0
chroma_cqt.max.11	0.999300	0.014124	0.292726	1.0	1.0	1.0	1.0

Table 4: Feature analysis - lack of dispersion in some feature

feature category	sum of importance value
mfcc	76177.32
spectral_contrast	21682.06
rmse	7174.21
tonnetz	6213.95
zcr	3062.85
spectral_centroid	2767.37
chroma_stft	2228.38
chroma_cqt	1012.18
spectral_bandwidth	939.92
chroma_cens	895.37

Table 5: Sum of feature importance over the top 50 important features grouped by feature category

feature category	average of importance value
mfcc	3627.49
spectral_centroid	2767.37
rmse	1793.55
spectral_contrast	1667.85
tonnetz	1553.49
zcr	1531.42
chroma_stft	1114.19
chroma_cqt	1012.18
spectral_bandwidth	939.92
chroma_cens	895.37

Table 6: Average of feature importance over the top 50 important features grouped by feature category

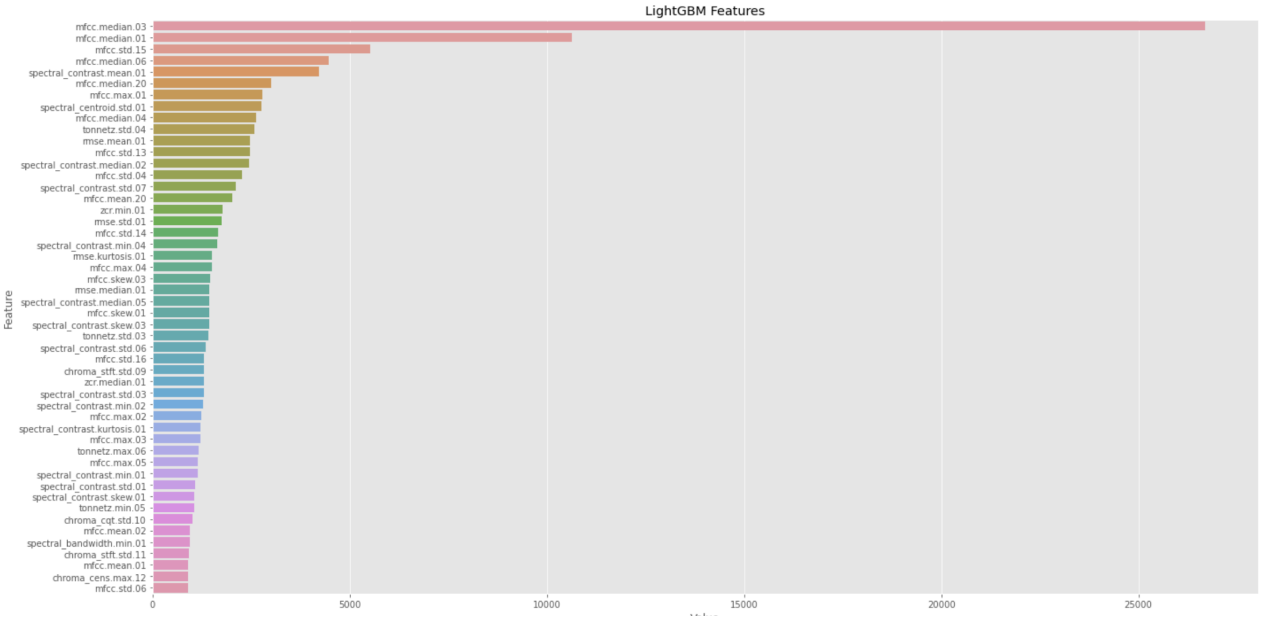

Figure 10: Feature importance plot of top 50 features - plain vanilla LightGBM

feature category	count in bottom 50
chroma_cqt	18
chroma_stft	17
chroma_cens	13
mfcc	1
tonnetz	1

Table 7: Count in the list of the 50 least important features grouped by feature category