Composers of classical music in .midi format

(Code is also at <https://github.com/c-j-lh/classical> Requirements: >= python 3.6)

Identifying the composer of a piece of classical music is part of many music training programmes and tests (e.g. ABRSM aural tests). While speech recognition is a popular topic in AI, less work has been done on the related audio-processing issue of AI in music.

Much research has been done on working with .mp3/.wav files that store the sound waves of recordings of the pieces. Some approaches to working audio files include plotting a spectrogram and running image recognition on it, or computing the MFCC (Mel-Frequency Cepstrum Coefficients) of timeframes from the audio file and classifying based on that. However, much less work has been done with .midi files, which only encode when a note is played and at what amplitude.

When working with midi files, different approaches have been tried. Some algorithms extract hand-crafted features from the midi-files, such as note range and frequency [c], inter-onset interval [4] or tempo and frequency [b] which we try. Other algorithms liken midi files to language, and use tools like tf-idf frequency and paradigms like sentiment analysis [5].

In this project, we work on a dataset of .midi files from http://www.kunstderfuge.com/.

First, we tried using the python library pretty\_midi to **extract features** such as tempo from the midi files, and classified the tracks based on the results. However, the results were appallingly bad, and we got an accuracy of 42.5%. Normalising the features improved the results to 75.0%, and tuning hyperparameters boosts performance to 76.0%. This reflects that simple metrics such as tempo and time signature are sufficient to reflect composers' styles.

(For chord frequency, refer to experiment-preprocessing.ipynb and experiment-classification.ipynb for now) Next, we classify the files using **chord frequency** and chord progression [1, 3]. In music theory, chords are groups of notes that are usually played at the same time, and chords are to notes as words are to letters. In fact, chord progressions like the imperfect cadence are very popular and often used to end a section. Hence, chords can reveal stylistic choices that the composer might have made for a piece. tf-idf is used on the frequency of each chord, similar to how n-grams are computed for words in NLP (Natural Language Processing). We find this approach successful in classifying the files, getting an accuracy of 55.61%.

Lastly, we **characterise** our model, and look at what chords and chord progressions our model has attributed to each composer. This information may prove useful for music students familiarising themselves with composers' styles.

**Main Objectives**

1. Classify the composers of classical pieces(in piano track matrices format)
   1. Using features (e.g., tempo, number of pitch classes used, polyphonic rate) [d] computed by pypianoroll
   2. Using chord frequency [a]
   3. Tune hyperparameters and optimize using GridSearchCV
2. Characterize

**Development Processes and Tools Used**

Sklearn

* Classification models like MLPClassifier, GradientBoostedClassifier, KNearestClassifier
* GridSearchCV and RandomizedSearchCV for hyperparameter tuning

Music libraries

* pypianoroll to get basic attributes and convert to .midi
* MuseScore to convert .midi to .mxl
* music21 to transpose and get chord frequencies during preprocessing

|  |
| --- |
|  |
| *Pipeline of classifier* |

**Application of concepts and techniques**

Preprocessing features before passing to models

K-fold cross-validation when tuning hyperparameters

In utilising chord frequency as a feature, we used 2 concepts: tf-idf and n-grams.

Tf-idf is a value for how frequently a term appears in a text, multiplied by how distinguishing the term is [6]. Most of the code for the classification we used was from [a].

An n-gram is “a contiguous sequence of n items” ([https://en.wikipedia.org/wiki/N-gram](https://en.wikipedia.org/wiki/N-gram#Applications)), a term most commonly used in NLP. A 1-gram is known as a unigram, a 2-gram a bigram, a 3-gram a trigram et cetera. In our investigation, we found that tetragrams were not helpful for the results, but found that adding trigrams improved results for the K-Neighbors Classifier model. Overall, we found {unigrams, bigrams} to be the best model - that is, the model performed best when it considered both every chord and every pair of chords. This runs contrary to intuition, since in music theory, most chord progressions are made of three or more chords (e.g., perfect, imperfect cadence; blues chords) [7]

Aside from chords, we also considered the durations separately, and combined them, as can be seen in tests 5 and 6 (table 1), and this improved results, showing that the length of each chord (i.e., patterns in the rhythm) also characterises a composer.

**Results and Findings**

1a) Preprocessing improves performance from 42.5% to 71%

1b) Adding chord frequency and duration to pypianoroll-derived features improves performance negligibly from 71.0% to 71.3%

2) Tuning hyperparameters for pypianoroll-derived features improves performance from 71% to 76%

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | SVC | LR | KNC | MLP | RFC | GBC | MNB |
| 1 *pypianoroll features*  default hyperparameters | 0.680 | 0.530 | 0.645 | 0.700 | **0.710** | 0.695 |  |
| 2) *pypianoroll features*  tuned hyperparameters | 0.685 | 0.540 | 0.715 | 0.720 | **0.760** | 0.730 |  |
| 3) *pypianoroll features + (1,2)-gram chord frequency*  default hyperparameters | 0.5011 | 0.3366 | 0.0963 | 0.5492 | **0.6561** | 0.5535 |  |
| *4) pypianoroll features + (1,2,3)-gram chord frequency*  default  hyperparameters | 0.4328 | 0.2989 | 0.1003 | 0.5116 | **0.6354** | 0.5076 |  |
| *5) chord frequency & (1)-gram duration*  author’s manually chosen hyperparameters | **0.7040** | 0.5513 | 0.5727 | 0.6736 | 0.5075 | 0.5912 | 0.5600 |
| *6) chord frequency & (1,2)-gram duration*  author’s manually chosen  hyperparameters | **0.7133** | 0.5476 | 0.5329 | 0.6461 | 0.5912 | 0.6301 | 0.5556 |
| *7) chord frequency & (1,2)-gram duration*  tuned  hyperparameters | **0.726** |  |  | 0.724 |  |  |  |

*Table 1. Table of results for 6 tests and 7 models (SVC: SupportVectorClassifier, LR: LogisticRegression, KNC: KNeighborsClassifier, MLP: Multi-Layer Perceptron, RFC: RandomForestClassifier, GBC: GradientBoostedClassifier, MNB: Multinomial Naive Bayes’) with different features. (best values for each test in bold)  
Note MNB is not applicable for normalized pypianoroll features since it can only be applied to nonnegative data.*

Lastly, we attempt to characterise our model, and find that the 4 composers have their favourite chords.

|  |  |
| --- | --- |
| Bach  C# D E F# A 6.39%  C D E G B 4.89%  D E F# G B 4.32%  D F# G A B 3.80%  C# E G# A B 3.48% | Mozart  D D# F G A# 5.43%  C D F A A# 4.78%  C D E G B 4.36%  C D D# G A# 4.15%  C E F G A 3.38% |
| Beethoven  C D E G B 5.15%  C E F G A 3.62%  C D F A A# 2.76%  D D# F G A# 2.55%  D F# G A B 2.12% | Debussy  C D E G B 3.88%  C D E G A# 2.87%  C# D# F G# B 2.69%  D# E F# G# B 2.37%  D E F# G# B 2.08% |

*Table 2. Composers’ most frequently used chords and their frequency.*

We can observe Debussy uses his favourite chords least, hinting that he probably doesn’t rely on a few chords in his pieces and has the largest variance in chord choice. This makes sense given that Debussy is a 20th Century composer and music then *“challenged the previously accepted rules of music of earlier periods, such as the use of altered chords and extended chords”* [8]

**Limitations and Extensions**

Preprocessing the data was too time-consuming, and as such we could only work on a small dataset of 200 files. Given more time, we would have liked to expand our reach to other composers and a greater number of pieces.

The project mainly focused on the specific chords used in each piece. Aside from chords, other features like the harmonic bass and melody could have been extracted and worked upon similarly.

**Conclusion**

Given that music students are required to reliably identify a piece’s composer in music exams, we can see that classifying .midi files is a relatively easy task for humans. Indeed, even for machine learning, even with a small dataset of only 50 pieces per composer, the model manages to pick up clues from the piece’s length, the number of notes played at the same time, chord frequency and rhythm to classify the pieces correctly more than three-quarters of the time.

In the longer-term, chord frequency (that is, the frequency of a chord’s use in a given piece) and progression, along with rhythm, can be an interesting and relevant feature when handling music pieces.

# References

[1] A. Koh. Recognising Classical Composers using High-Level Music Features

[2] Wang, Xindi, and Syed Arefinul Haque. "Classical Music Clustering Based on Acoustic Features." arXiv preprint arXiv:1706.08928 (2017).

[3] <https://github.com/robert-d-schultz/music-classification/blob/master/preprocess.py>

[4] Lan, Janice, and Armon Saied. "Music Classification by Composer." (2012).

[5]Oramas, Sergio, et al. "Natural language processing for music knowledge discovery." Journal of New Music Research 47.4 (2018): 365-382.

[6] <https://en.wikipedia.org/wiki/Tf%E2%80%93idf>

[7] <https://lotusmusic.com/lm_chordprogressions.html>

[8] <https://en.wikipedia.org/wiki/20th-century_music>

**Code sources**

[a] <https://github.com/achimkoh/midi-classification>

[b]<https://github.com/sandershihacker/midi-classification-tutorial/blob/master/midi_classifier.ipynb>

[c] <https://github.com/dantasfiles/CatiMidi>

[d] <https://salu133445.github.io/pypianoroll/metrics.html>

**Work Distribution Matrix**

|  |  |  |
| --- | --- | --- |
| **Work Description** | **Chieu Le Heng** | **Oliver James Tan** |
| Ideation & Proposal | ✔ | ✔ |
| Data Fetching | ✔ | ✔ |
| Balancing Classes | ✔ |  |
| Classifier Design | ✔ | ✔ |
| Classifier Optimisation | ✔✔ | ✔ |
| Model Evaluation | ✔ | ✔ |
| Report | ✔ | ✔✔ |