

A Pilot Study on Identifying Music Styles of the Common Practice Period

Assignment 2 Huanye Liu

November 5, 2019

Introduction

The common practice period in western classical music has been specified by musicologist as the period of tonal music. The period lasted for about 300 years consisting mainly of three eras in western music history: the Baroque era from 1600 to 1760, the Classical era from 1730 to 1820 and the Romantic era from 1780 to 1910. Each musical era is characterized by certain music style one may perceive in music compositions from that era, and people usually consider those who can identify the distinctions between the compositional styles of different music eras must have had certain level of musical training. However, with the aid of computing technology, we can now find out some basic music elements that may underlie the musically educated listeners' recognition of different compositional styles of the three eras in the common practice period.

By far, we strongly believe that rhythmic changes can be a good feature or legisign that can be used to identify the music style from one particular era in the common practice era. In the Baroque era, due to the popularity among composers of the basso continuo which governs the basic rhythm of the composition, one can rarely find radical change of rhythm in most works from that era. In the Classical era, the decline of the basso continuo application and the emergence of style galant lead to the more frequent change of rhythmic pattern compared with the Baroque era. When romantic era came, rhythmic change even within one musical phrase can be common practice thanks to the innovation of music instruments to prolong the duration of the

pitch.

To facilitate data processing by the computer, we need to quantify the rhythmic change of a music composition to indicate the change of the note durations. Here we use the “normalized Pairwise Variability Index” or nPVI proposed in [1] which initially was applied to speech recognition:

$$nPVI = \frac{100}{m-1} \sum_{k=1}^{m-1} \left| \frac{d_k - d_{k+1}}{d_k + d_{k+1}} \right|$$

where m is number of notes in one part of the music composition and d_k is the duration of the k th note. Two more specifications are: 1) The duration of the first note is always 1, and following notes' duration values are the relative durations to the first note. 2) If the composition consists of several parts, we also use the maximum nPVI among all the parts as the nPVI for the whole composition.

we can see from the formula that the index is in essence an aggregation of local changes in note duration by taking the average of all the neighboring difference of note duration, and also that this index making use of neighboring differences (unigram model being the counterpart in natural language processing) as the minimum units for aggregation can match human experiences of perceiving music in that we usually only need to hear a short music segment to identify the music style.

Question 1

So the research question is whether rhythmic changes of music compositions can be an argumentative symbolic legisign of music styles in the common practice period. We consider it as an argumentative symbolic legisign here because although some rhythmic changes can be perceived locally such as within a musical phrase, but we need to take the average of the local changes to measure the general degree of rhythmic changes for the whole music composition, so it corresponds to a larger structure or more complete organization consisting of many smaller musical phrases.

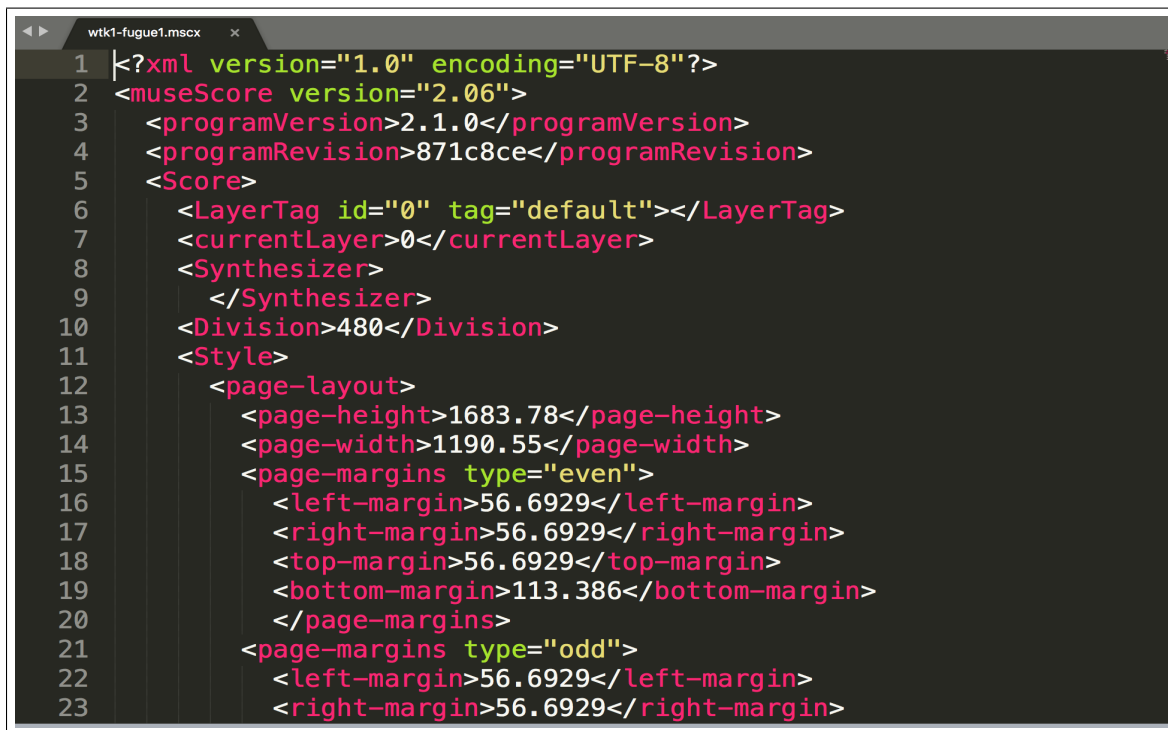
Question 2

The data I use for the project are midi files can be downloaded from the website <https://www.mutopiaproject.org> which has collected more than 677 midi files of compositions from the Baroque era, 612 midi files from the Classical era and 443 midi files from the Romantic era. However, we need to convert those downloaded midi files to MusicXML files which are formatted like XML files so that I can extract the information of note duration indicated by tags with ease. Specifically, I use the MuseScore application to convert midi files to MusicXML files on mac terminal for batch processing:

```
Huanyes-MacBook-Pro:schuman huanye$ for f in *.mid; do /Applications/MuseScore\ 2.app/Contents/MacOS/mscore -o "${f%.*}".mscx $f; done
initScoreFonts 0x7fbe7a73c060
[convert <02_Freisinn.mid> to <02_Freisinn.mscx>
[initScoreFonts 0x7fdafbd27ca0
convert <AnDieMusik.mid> to <AnDieMusik.mscx>
initScoreFonts 0x7fd356f46330
convert <Erlkoenig.mid> to <Erlkoenig.mscx>
initScoreFonts 0x7f9d1162b4b0
convert <Erlkoenig_alt.mid> to <Erlkoenig_alt.mscx>
initScoreFonts 0x7fb18167f420
convert <Ich_grolle_nicht.mid> to <Ich_grolle_nicht.mscx>
initScoreFonts 0x7fb499681d20
convert <Ich_will_meine_Seele_tauchen.mid> to <Ich_will_meine_Seele_tauchen.mscx>
initScoreFonts 0x7fd23a6572e0
convert <SchubertF-D120-TrostInThraenen.mid> to <SchubertF-D120-TrostInThraenen.mscx>
initScoreFonts 0x7f85b7f6b360
convert <SchubertF-D162_NaeheDesGeliebten.mid> to <SchubertF-D162_NaeheDesGeliebten.mscx>
initScoreFonts 0x7f839966fa30
convert <SchubertF-D163-D165_SaengersMorgenlied.mid> to <SchubertF-D163-D165_SaengersMorgenlied.mscx>
initScoreFonts 0x7fc6cec49370
```

Question 3

Next for each converted MusicXML file, I write python code to extract the note durations in sequence and use the formula presented in the section of Introduction to compute the normalized Pairwise Variability Index for each music composition from the particular musical era. For this pilot study, I only choose some selective compositions of four icons from the three musical eras in the common practice period: Bach from the Baroque era, Mozart from the Classical ear and Schubert+Schumann from the Romantic era, and I will use all compositions downloaded from the website later for the final project. All relevant python codes are attached to the end of this file. Below is a screenshot of the converted MusicMXL file:

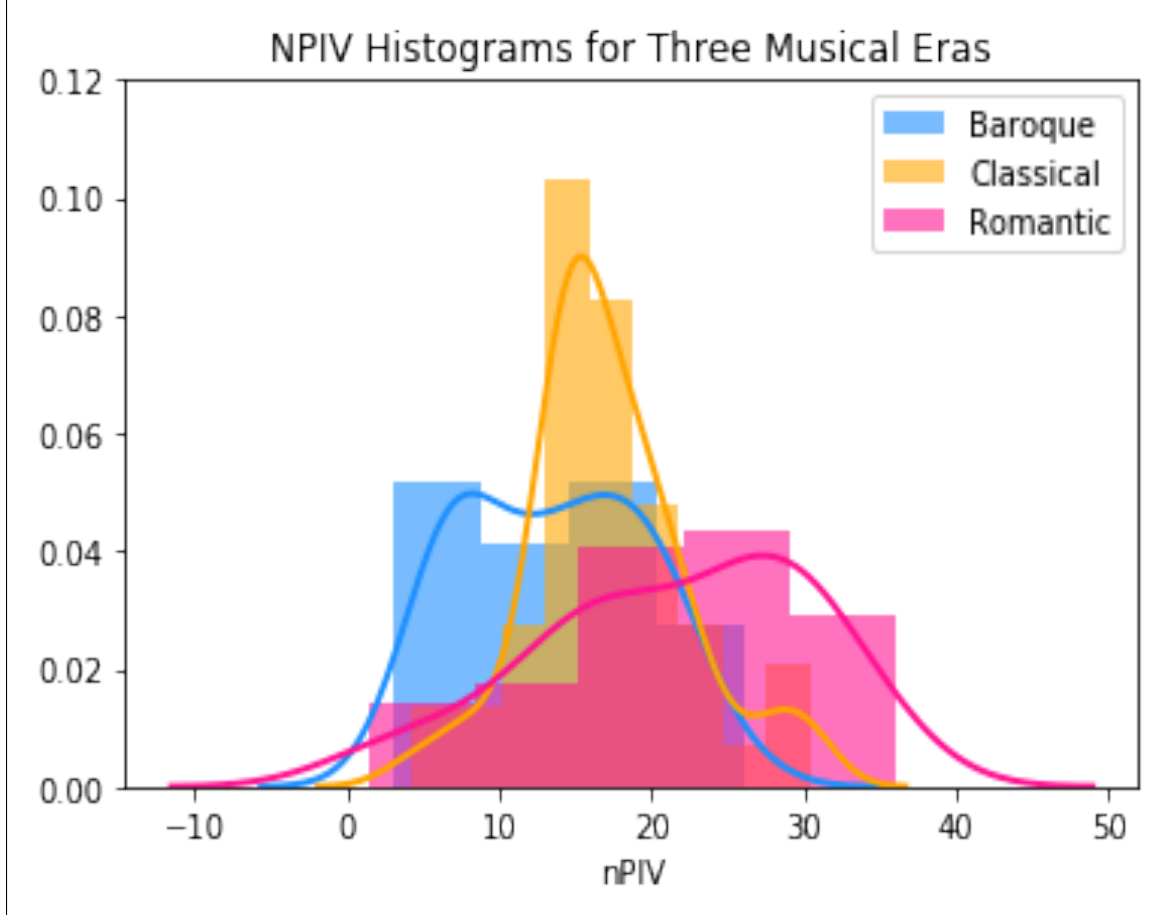
A screenshot of a text editor window showing a MusicXML file named 'wtk1-fugue1.mscx'. The code is XML-formatted with syntax highlighting. It includes a root element 'museScore' with version '2.06'. Inside, there are elements for 'programVersion' (2.1.0), 'programRevision' (871c8ce), 'Score', 'LayerTag' (id='0', tag='default'), 'currentLayer' (0), 'Synthesizer', 'Division' (480), 'Style', 'page-layout', 'page-height' (1683.78), 'page-width' (1190.55), 'page-margins' (type='even' with left, right, top, and bottom margins), and 'page-margins' (type='odd' with left and right margins).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <museScore version="2.06">
3   <programVersion>2.1.0</programVersion>
4   <programRevision>871c8ce</programRevision>
5   <Score>
6     <LayerTag id="0" tag="default"></LayerTag>
7     <currentLayer>0</currentLayer>
8     <Synthesizer>
9     </Synthesizer>
10    <Division>480</Division>
11    <Style>
12      <page-layout>
13        <page-height>1683.78</page-height>
14        <page-width>1190.55</page-width>
15        <page-margins type="even">
16          <left-margin>56.6929</left-margin>
17          <right-margin>56.6929</right-margin>
18          <top-margin>56.6929</top-margin>
19          <bottom-margin>113.386</bottom-margin>
20        </page-margins>
21        <page-margins type="odd">
22          <left-margin>56.6929</left-margin>
23          <right-margin>56.6929</right-margin>
```

Question 4

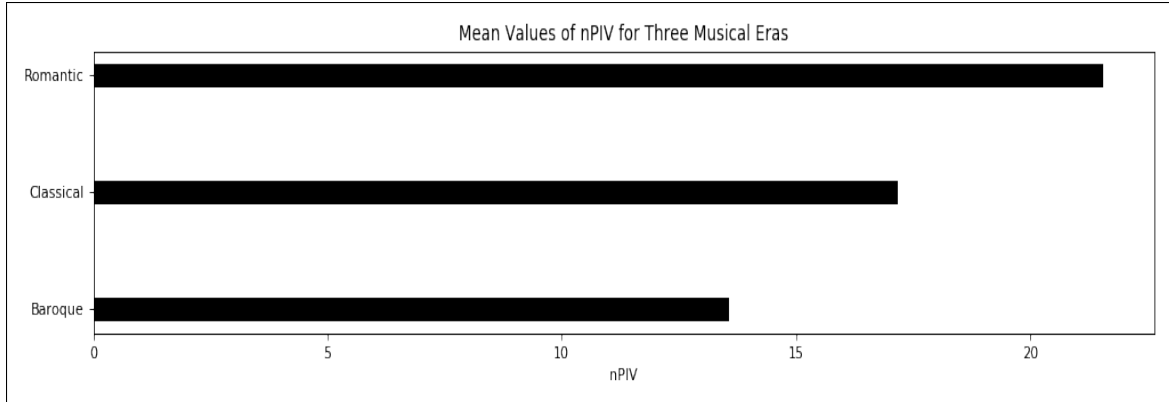
Next I collect computed nPIVs for all music compositions from three particular musical eras in three lists, and make one histograms for each list of nPIVs from that

particular music era and put the resulting three histograms of nNPV for three music era in the same plot for comparison:



We can see the graph shows some positional patterns: although there are overlap areas between different histograms partly due to the relatively small sample size for this pilot study and other reasons presented later, we can clearly detect the positional sequence of three histograms from the left to the right: the Baroque, the Classical and the Romantic in order. To confirm this, I also plot the means of nPIVs from the three music eras so that we can see the pattern more clearly.

This pattern of monotonically increasing nPIVs from the Baroque period to the Romantic period confirms the concept of "historical distance effect" proposed by musicologists[2]: historical proximity reflects the similarity in musical styles. So if we consider the rhythmic change is an appropriate argumentative symbolic legisign for



musical styles in the common practice period, then that distances between nPIV histograms align with the proximity in history cannot be a surprise. On the other hand, we can also see that histograms overlap, meaning the patterns of rhythmic change evolve gradually, and some composers were born and thus situated in the transitional period between two musical eras, such as Beethoven and Schubert across the Classical and Romantic eras. Actually in this pilot study we can see the nPIV histogram of the Romantic era covers a wide interval, even overlap with the Baroque, which may be resulted from the fact that half of the sample works from the Romantic era are composed by Schubert who were influenced heavily by the Classical compositional style. Another reason for the overlap may be that some composers' musical style can deviates from the style of the era of his living years. Brahms is one such example who lived in the Romantic era but composed with the Classical style. Of course, we need more data to see if the overlaps between nPIV histograms for different musical era will decrease or increase, but at least for now, I would say the rhythmic change is an appropriate argumentative symbolic legisign for musical styles in the common practice period.

Question 5

There are at least three aspects for future work: 1) I need to download all data for three musical periods from the website to answer the research question with more confidence. For example, I wonder if the three histograms become more separate of

more overlapped using the full data. Are there any other pattern from the histogram such as the spread or standard deviation of the histogram? 2) I need to use more statistical tools to make this research more rigorous. For example, how can I measure the difference between two histograms or distributed? Can I relate the "historical distance effect" mentioned above to some statistical measure such as the Kullback-Leibler divergence. In addition for the nPIV mean comparisons for the three musical era, we may need to use the t-test to confirm the difference is statistically significant. 3) I also want to explore some other possible legisigns in addition to the rhythmic change. Because intuitively, the rhythm only captures part of the musical features for a music composition, and I maybe I should consider other features related to the melody and texture of the music, which can be combined with the rhythm to characterize the tonal system in the common practice period. Extracting melodic features and texture can be more complex than the rhythmic change since it involves more music elements such as pitch, theme, range, register, imitation, harmony and tonality etc.. I would like to try at least some of these elements, but what I want to derive is something easy to interpret from its quantitative derivation but effective to identify the musical styles from the three musical era in the common practice period just like the rhythmic change does.

Bibliography

- [1]Patel, A. D., Daniele, J. R. (2003a). An empirical comparison of rhythm in language and music. *Cognition*, 87, B35–B45.
- [2]Gromko, J. E. (1993). Perceptual differences between expert and novice music listeners: A multidimensional scaling analysis. *Psychology of Music*, 21, 34–47.

Untitled

November 5, 2019

```
In [99]: import pandas as pd
```

```
In [79]: import xml.etree.ElementTree as et
```

```
In [126]: def toNumber(string,first):
    ref = ["128th","64th","32nd","16th",'eighth','quarter',"half","whole"]
    result = 1
    if string == first:
        return result
    else:
        first_core = first
        if '.' in first:
            first_core = first[:first.index(".")]

        index1 = ref.index(first_core)
        string_core = string
        if '.' in string:
            string_core = string[:string.index(".")]
        index2 = ref.index(string_core)

        if '.' in first:
            index1 = index1+0.5
        if '.' in string:
            index2 = index2+0.5
        result = 2**(index2-index1)

    return result

for part in score.findall("Staff"):
    first = None
    durations = []
    for measure in part.findall("Measure"):
        for chord in measure.findall("Chord"):

            dots = ""
            if chord.find("dots")!=None:
                dots = "."
            durations.append(chord.find("durationType").text+dots)
```



```

        if first==None:
            first = durations[0]
        if len(chord.findall("Note"))==1:

            if note.find("Tie")!=None:
                ID = note.find("Tie").attrib.get("id")
                durations.append(ID)
            if note.find("endSpanner")!=None:
                ID = note.find("endSpanner").attrib.get("id")
                durations.append(ID)

last = None
durations_update = []
for i in range(len(durations)):
    if durations[i].isdigit():
        if last == None or last != durations[i]:
            last = durations[i]
        else:
            duration1 = durations_update.pop(-1)
            duration2 = durations_update.pop(-1)
            duration = duration1+duration2
            durations_update.insert(len(durations_update),duration)
    else:
        durations_update.append(toNumber(durations[i],first))

durationLsts.append(durations_update)

```

In [88]: `def getDurations(filename):`

```

xtree = et.parse(filename)
xroot = xtree.getroot()
durationLsts = []
score = xroot.find("Score")
for part in score.findall("Staff"):
    first = None
    durations = []
    for measure in part.findall("Measure"):
        for chord in measure.findall("Chord"):
            dots = ""
            if chord.find("dots")!=None:
                dots = "."
            durations.append(chord.find("durationType").text+dots)
            if first==None:
                first = durations[0]
            if len(chord.findall("Note"))==1:
                if note.find("Tie")!=None:
                    ID = note.find("Tie").attrib.get("id")
                    durations.append(ID)
                if note.find("endSpanner")!=None:

```

```

        ID = note.find("endSpanner").attrib.get("id")
        durations.append(ID)

    last = None
    durations_update = []
    for i in range(len(durations)):
        if durations[i].isdigit():
            if last == None or last != durations[i]:
                last = durations[i]
            else:
                duration1 = durations_update.pop(-1)
                duration2 = durations_update.pop(-1)
                duration = duration1+duration2
                durations_update.insert(len(durations_update),duration)
        else:
            durations_update.append(toNumber(durations[i],first))
    durationLsts.append(durations_update)
    return durationLsts

In [89]: def computeNPIV(lst):
    m = len(lst)
    abs_sum = 0
    for i in range(m-1):
        abs_sum+=(abs(lst[i]-lst[i+1]))/(lst[i]+lst[i+1]))
    return 100*abs_sum/(m-1)
def nPIVMusic(Lsts):
    nPVIIs = []
    for lst in Lsts:
        nPVIIs.append(computeNPIV(lst))
    return max(nPVIIs)

#print(nPIVMusic(getDurations("../mozart/zufriedenheit-piano.mscx")))

In [118]: import os
import matplotlib.pyplot as plt
def getNPIVERa(directoryName):
    npivs = []
    count = 0
    for filename in os.listdir(directoryName):
        if filename.endswith(".mscx"):
            npivs.append(nPIVMusic(getDurations(directoryName+filename)))
    return npivs
baroque = getNPIVERa("../bach/")
classical = getNPIVERa("../mozart/")
romantic = getNPIVERa("../schuman/")
df = pd.DataFrame({"Baroque":baroque,"Classical":classical,"Romantic":romantic})

In [140]: import seaborn as sns
kwargs = dict(hist_kws={'alpha':.6}, kde_kws={'linewidth':2})

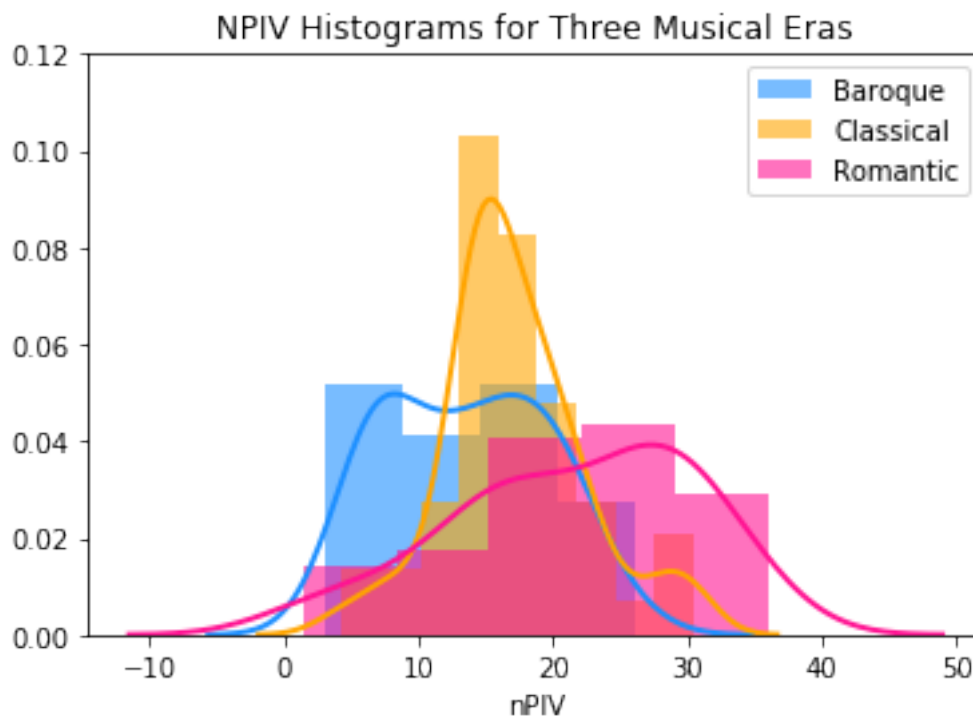
```

```

sns.distplot(df['Baroque'], color="dodgerblue", label="Baroque", **kwargs)
sns.distplot(df['Classical'], color="orange", label="Classical", **kwargs)
sns.distplot(df['Romantic'], color="deeppink", label="Romantic", **kwargs)
plt.legend()
plt.ylim(0,0.12)
plt.xlabel("nPIV")
plt.title("NPIV Histograms for Three Musical Eras")
plt.show()

```

/Users/huanye/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1633: FutureWarning:
return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval



```

In [142]: plt.figure(figsize=(15,3), dpi= 80)
plt.barh(["Baroque","Classical","Romantic"],df.mean(),0.2,facecolor='black')
plt.title("Mean Values of nPIV for Three Musical Eras")
plt.xlabel("nPIV")
plt.show()

```

