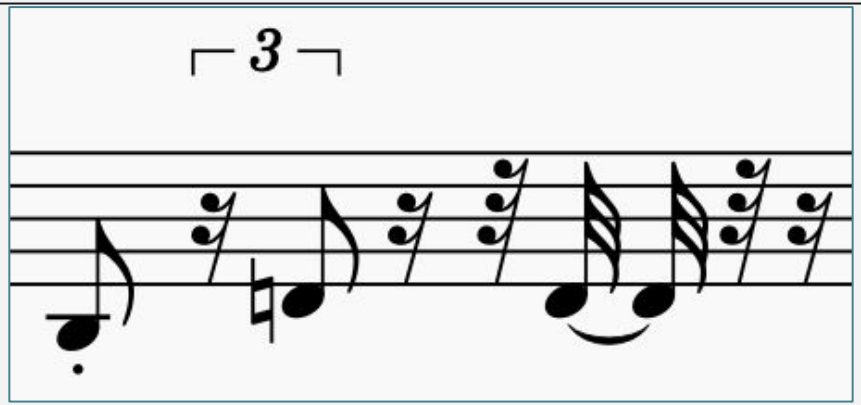


Musical Composer Identification and Generation Progress

Mildness Onyekwere & Shaun Thornton

Contents

Recap

Original Research Questions:

- How effectively can natural language processing–based models identify the composer of a given musical piece when given a symbolic/textual representation of it detailing the signatures, pitches, and their durations?
- Once the composer of musical pieces can be accurately identified, how well could a separate language processing–based model be trained to generate a novel piece in the style of a desired composer, using the first model as a guide?

Goal #1:

- Using the Maestro Dataset, create a classifier that can confidently predict the composer of a piece of piano music.

Goal #2:

- Build and train an LSTM to generate music in the style of a composer, and the use the first model to evaluate the accuracy of its production

Tokenization

MIDI data must be converted into a sequence of tokens, just as we would tokenize textual data. The **pretty_midi** library was used to parse and extract note information from MIDI files. For the RNN approach, we used a custom (and highly unstructured) tokenization format, rather than **MusicXML** / **ABCNotation**.

Musical Features	Token Representation Example
Pitch	p50 *(50th key)*
Duration	d0_25 *(quarter note)*
Rests	r0_50 *(half note rest)*
Inter-onset interval	iSIMUL *(denotes a chord)*

```
'p43', 'd0_1', 'iSIMUL', 'int+8', 'p70', 'd0_1',
```

Tokenization sequence snippet

Model #1: Composer Identification

This model attempts to predict whether a given piece was written by a specific composer

- This differs slightly from our original plan of having the model make predictions around a set of 3-5 composers
- Binary classification was ultimately more accurate and better suited for the task

sklearn Pipeline:

- **Tokenization***
- **TfidfVectorizer**
  - Converts our token sequences into (sparse) vectors
- **K-Nearest Neighbors Classifier** *(k=7)*
  - Training data is balanced by duration (equal playtime representation for positive and negative samples)
  - Each training sample is labeled depending on whether it is associated with the target composer (1) or not (0)

Model #1: Composer Identification

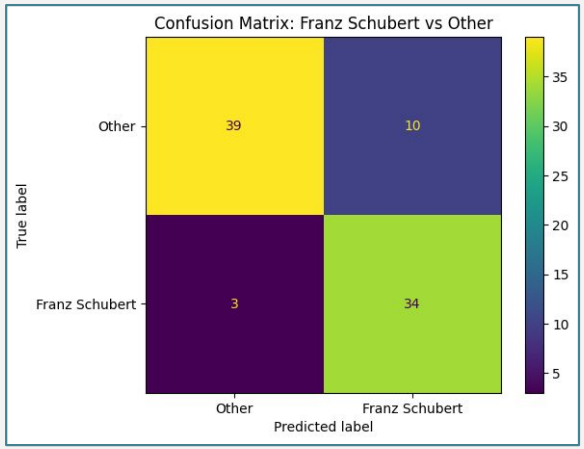
Pipeline →

Train →

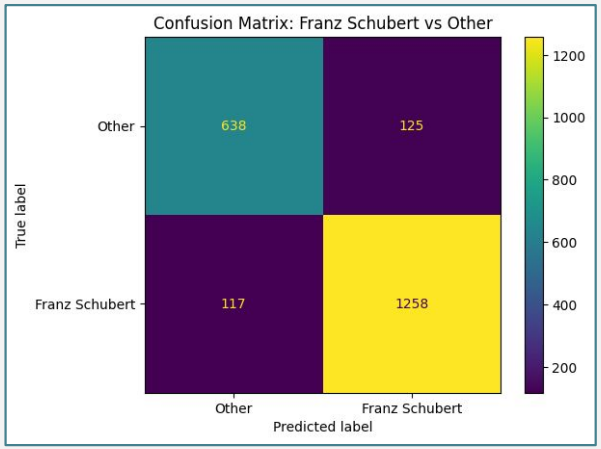
Predict →

```
pipeline = Pipeline([
    ("tfidf", TfidfVectorizer(
        analyzer="word",
        token_pattern=r"\S+",
        min_df=2,
    )),
    ("knn", KNeighborsClassifier(
        n_neighbors=7,
        metric="cosine",
        weights="distance",
    )),
])

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)
```

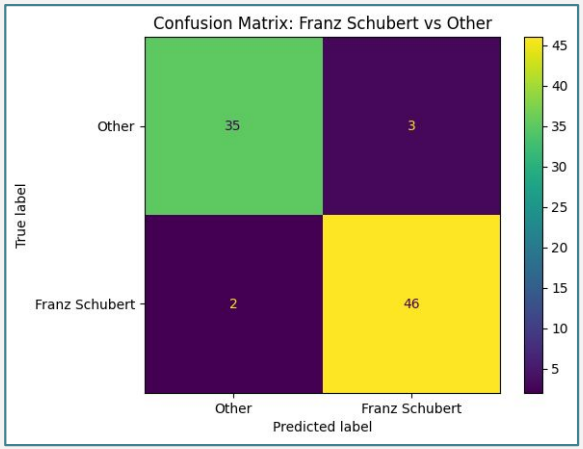
Model #1: Composer Identification



Track-level prediction (direct)
Accuracy (F1) = ~82%



Window-level prediction (30s)
Accuracy (F1) = ~88%



Track-level prediction (window voting)
Accuracy (F1) > 95%

Model #2: Music Generation

This model attempts to generate music in the style of a target composer. It is paired with an instance of model #1 that has also been trained on the same composer.

The training data

- Collect all MIDI pieces associated with the target composer
- Split each piece into 30 second windows
- **Tokenize** each window

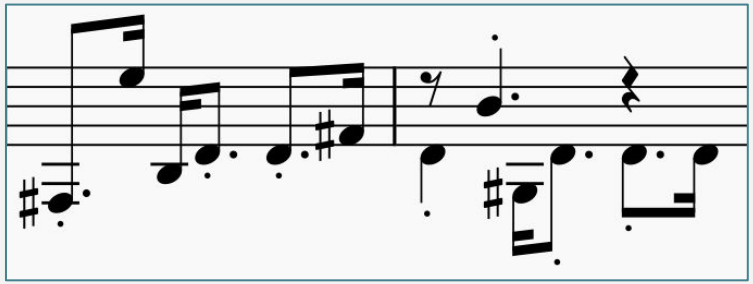
The vocabulary

- Contains all tokens seen throughout the data (p*, d*, etc.)
- Includes beginning / end of sequence tokens (<**BOS**>, <**EOS**>)
- Represents the set of tokens that our model can generate

The model

- Long short-term memory (**LSTM**) Recurrent neural network (**RNN**) implementation (built with **PyTorch**)
- To generate a piece, start the model off with <BOS> and have it repeatedly generate tokens until it produces a <EOS>

Model #2: Demo #1

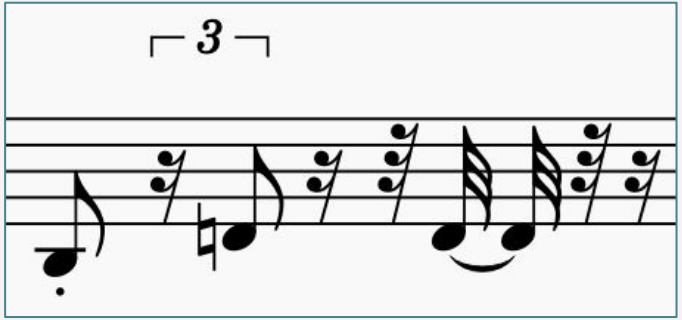


- Feature set
  - Pitch
  - Duration

- Comments
  - Hardly better than a random note generator
  - Painful to listen to

- Composer identification score *
  - ~70%

Model #2: Demo #2

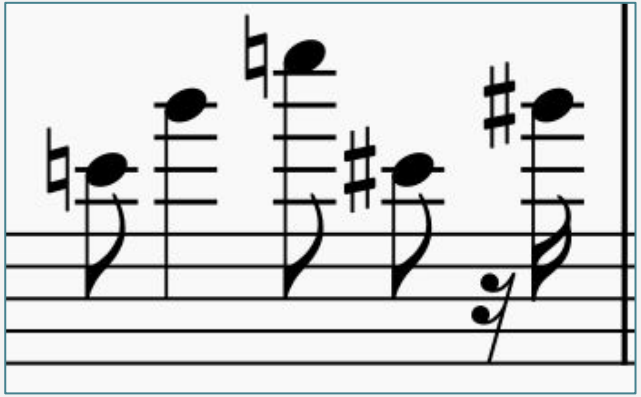


- Feature set
  - Pitch
  - Duration
  - Rests

- Comments
  - Still hardly better than a random note generator
  - The inclusion of rests makes the output sound more natural

- Composer identification score *
  - ~74%

Model #2: Demo #3

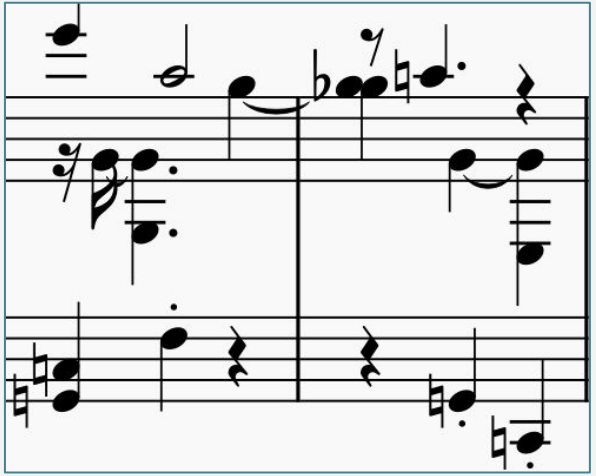


- Feature set
  - Pitch
  - Duration
  - Rests
  - Inter-onset interval

- Comments
  - The model now generates overlapping notes (!)
  - Notes jump around wildly and unnaturally

- Composer identification score *
  - ~83%

Model #2: Demo #4



- Feature set
  - Pitch
  - Duration
  - Rests
  - Inter-onset interval
  - Pitch intervals

- Comments
  - Notes are kept closer together
  - The model clearly still lacks an understanding of melody

- Composer identification score *
  - ~90%

Transformer Shenanigans

- From our background research (and practical work done so far) we know that RNN can only understand so much when generating consecutive notes and misses much of the musical context of overarching phrases
- From Model #2's implementation, we see that the model's accuracy improves when more features (rests, inter-onset interval, pitch intervals, etc...) are present
- We are developing another version of Model #2 using an attention based transformer model to see if generated music results sound more authentic
    - **New Goal**: Create "ScoreGPT"!
        - This would be Decoder-Only

Steps before Score GPT

- For Model #2, we've initially strayed away from using MusicXML or ABC Notation due to the simplicity of the RNN, but in order to provide the transformer based model context of the musical structure, this type of structured token created from MIDI data is needed. ABC Notation was selected over MusicXML due to its succinctness.
- Before creating and training this new Model #2 we would need to...
    1) Create a tool that processes MIDI into ABC Notation and vice versa
    2) Create tokenizer and parser of raw ABC Notation text to create context trees
    3) Create a final vocabulary that GPT can understand
    4) Create a tool that can transform the GPT output back into raw ABC Notation string

ABC Notation vs. MusicXML

```
V:1
"^intro"!mp! D3 E- | E D3 | D3 E- | E/D/ G/>F/ E2 |
w: ooh _|_ _|ooh _|_ _ hoo _ _|
 D3 E- | E4 | D3 E- | E/E/4F/4 E !^!E/ z/ !^!E/ z/ |
w: ooh _|_|ooh _|_ _ _ _ daht daht|
"^hook" F3 E- | E3 E | F2 G E- | E/D/4E/4 D2 D |
w: doo _|_ wah|doo _ _|_ _ _ _ wah|
 D3 E- | E3 F | z G2 E | F D E2 |
w: doo _|_ wah|doo doo|doo wah doo|
"^verse 1" D3 E- | E3 E | D3 E- | E2 E F |
w: doo doo|_ doo|doo doo|_ doo wah|
"^verse 2" D3 E- | E3 E | F2 G A | F D (E E) |
w: doo doo|_ oh|doo doo doo|doo doo woah *|
"^chorus 1" D3 E- | E3 E | D3 E- | E/D/4E/4 D2 D |
w: doo doo|_ doo|doo doo|_ _ _ _ wah|
 D3 E- | E3 E | F2 G A | F D E (D/E/) |
w: doo doo|_ wah|doo doo doo|doo wah doo wah *|
"^chorus but diff" D3/2 F2 E/ | E D3 | D3/2 F2 F/ | G/>A/ G G F |
w: doo doo doo|doo doo|doo doo doo|doo _ _ doo wah|
 E E E F/G/4A/4 | G F3 | F2 G2 | ^^G B c B |
w: doo doo doo doo _ _|_ doo|doo doo|doo woah la- ah|
[K:A]"^key change!!!" A2 A G- | G/F/4G/4 F2 G | A2 A G- | G/F/4E/4
F/4G/4 F F/ G |
w: doo doo doo|_ _ _ _ wah|doo doo doo|_ _ _ _ _ _ doo wah|
 F/E/ G G G- | G2 G G | A2 B c | G F/F/ A/B3/4 z/4 F/ |
w: round and round doo doo|_ doo doo|doo doo wah|* * * * * the|
```

ABC Notation

(Dense, but more english-like than MIDI)

```
<note>
  <pitch>
    <step>E</step>
    <alter>1</alter>
    <octave>4</octave>
    </pitch>
  <duration>2</duration>
  <tie type="stop"/>
  <voice>1</voice>
  <type>eighth</type>
  <stem>up</stem>
  <beam number="1">begin</beam>
  <notations>
    <tied type="stop"/>
    </notations>
  </note>
```

(Very readable, but also verbose)

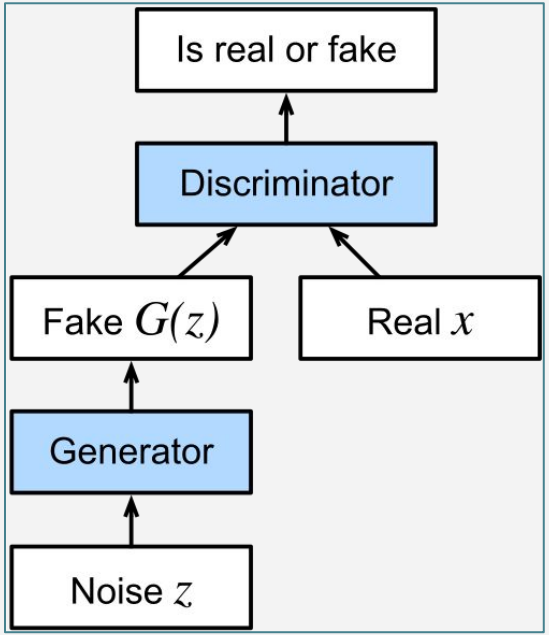
(This entire snippet represents a single note)

Future Idea: Generative Adversarial Network

Our existing dual-architecture model could form an effective Generative adversarial network (GAN)

- Model #1 would be the *discriminator*, which determines whether a given piece/window is written by
  - A specific composer
  - A human
- Model #2 would attempt to generate a piece/window that's *convincingly real*
  - Convince discriminator that it's from Beethoven
  - Convince discriminator that it's from a human

Under this strategy, the two models would effectively train off of each other and continually improve together. However, this would require a more thoughtful feature set than what we currently have.



Questions?