Brainstorming

McAuley goes over the assignment in Thu 4/24 LE

Resources

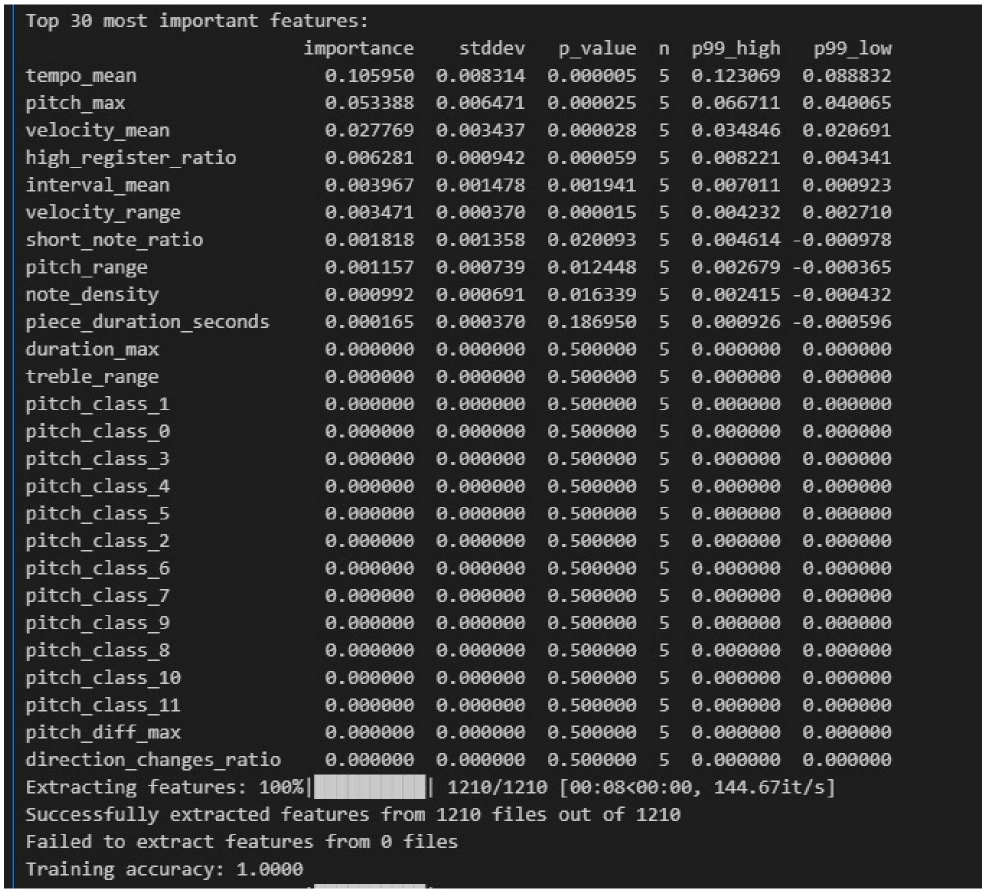
- Workbook 1
- Workbook 2, ml_basics

Writeup and .py files

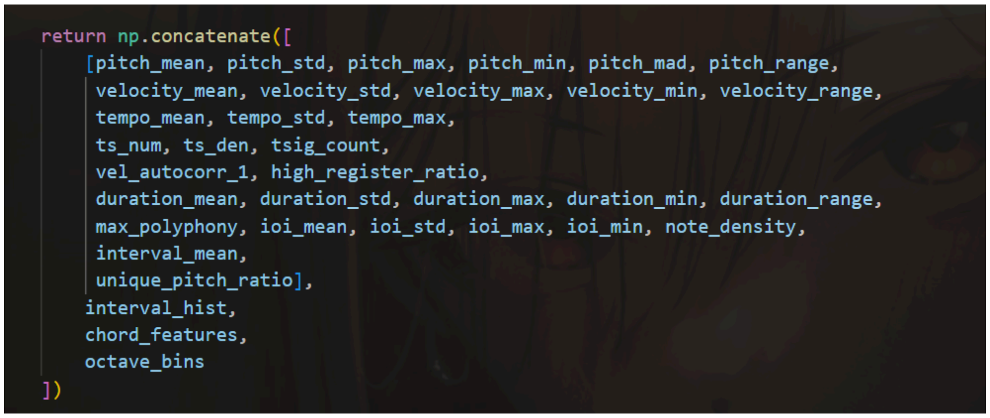
- Do it

Task 1: composer prediction (midi -> composer)

- Classification
- Feature vector: length, note frequencies, register (lots of high notes or lots of low notes), chord choices, chord progressions, pairwise probabilities (note A followed by note B)
- Logistic regression
- @178 Try the fastest model from HW2 as a starting point, and CNNs (as in the baseline) if you have times. Tasks 1 and 2 can be done with "traditional" classifiers.
- @303 Features: Mean, std, and range of pitches, Mean and std of note durations, Total note count, Note-density (notes / time span), Mean and std of pitch–intervals, Mean and std of inter-onset intervals, Mean and std of MIDI velocities, fraction of notes overlapping, Counts of notes in eight 16-semitone “octave” bins, Fraction of major, minor, diminished, augmented, and other chords.
    - I have also used the ensemble learners method, which included models like: SVM, XGBoost, LightGBM, CatBoost, ExtraTrees, and KNN.
    - You could perhaps do more here with tempo, maybe think of something more "bigram"-ish re notes and chords. Tasks 1 and 2 can also be approached with data augmentation strategies (or at least, pitch shifting), which may or may not help but are definitely present in some top solutions.
- @346 pitch, rhythmic, melodic, dynamic, harmonic/chordal, register/textural, and tempo features
- Discord: choose classifiers that do well with tabular data
    - like lightgbm, xgboost, tabpfn, etc.
    - autogluon
- Discord: I have 26 features with GBM and can get an accuracy of over 70% on public. I am curious how the first place got 92%💀
    - I have 12 different pitches. Although they seem to have little importance in the feature importance map, my accuracy drops significantly after I remove them. I don't know why, so i just keep them
- Discord: i got above 0.7 using chord histograms
- using xgboost



```
Top 30 most important features:
                        importance    stddev   p_value  n  p99_high   p99_low
tempo_mean                0.105950  0.008314  0.000005  5  0.123069  0.088832
pitch_max                 0.053388  0.006471  0.000025  5  0.066711  0.040065
velocity_mean             0.027769  0.003437  0.000028  5  0.034846  0.020691
high_register_ratio       0.006281  0.000942  0.000059  5  0.008221  0.004341
interval_mean             0.003967  0.001478  0.001941  5  0.007011  0.000923
velocity_range            0.003471  0.000370  0.000015  5  0.004232  0.002710
short_note_ratio          0.001818  0.001358  0.020093  5  0.004614 -0.000978
pitch_range               0.001157  0.000739  0.012448  5  0.002679 -0.000365
note_density              0.000992  0.000691  0.016339  5  0.002415 -0.000432
piece_duration_seconds    0.000165  0.000370  0.186950  5  0.000926 -0.000596
duration_max              0.000000  0.000000  0.500000  5  0.000000  0.000000
treble_range              0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_1             0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_0             0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_3             0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_4             0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_5             0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_2             0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_6             0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_7             0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_9             0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_8             0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_10            0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_class_11            0.000000  0.000000  0.500000  5  0.000000  0.000000
pitch_diff_max            0.000000  0.000000  0.500000  5  0.000000  0.000000
direction_changes_ratio   0.000000  0.000000  0.500000  5  0.000000  0.000000
Extracting features: 100%|██████████| 1210/1210 [00:08<00:00, 144.67it/s]
Successfully extracted features from 1210 files out of 1210
Failed to extract features from 0 files
Training accuracy: 1.0000
```

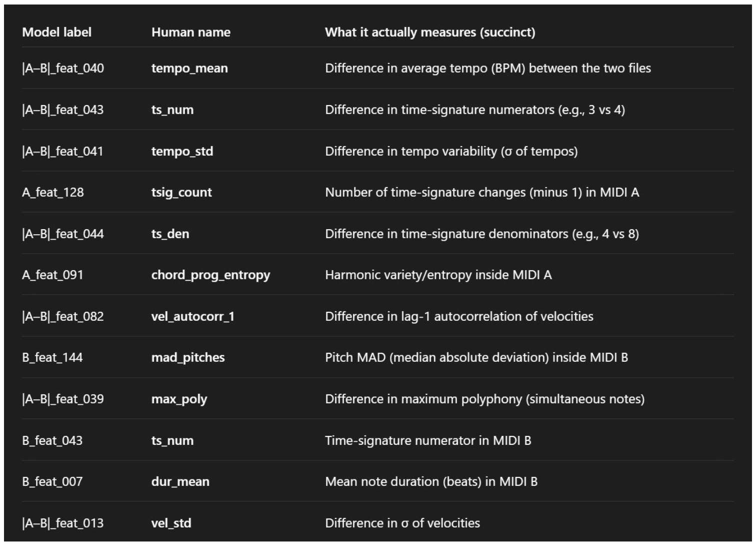


```python
return np.concatenate([
    [pitch_mean, pitch_std, pitch_max, pitch_min, pitch_mad, pitch_range,
     velocity_mean, velocity_std, velocity_max, velocity_min, velocity_range,
     tempo_mean, tempo_std, tempo_max,
     ts_num, ts_den, tsig_count,
     vel_autocorr_1, high_register_ratio,
     duration_mean, duration_std, duration_max, duration_min, duration_range,
     max_polyphony, ioi_mean, ioi_std, ioi_max, ioi_min, note_density,
     interval_mean,
     unique_pitch_ratio],
    interval_hist,
    chord_features,
    octave_bins
])
```

- Ideas
    - Exclude the last half second of audio? Sometimes it goes quiet during the last bit
    - Average volume and volume variation is probably important
    - Speed variation: rests, variation in the time between each note
    - ~~Number of notes?~~ Made accuracy worse

Task 2: sequence prediction ((bar 1 midi, bar 2 midi) -> next to each other Y/N?)

- About half is Y, about half is N

- Feature vector: differences in absolute values (e.g. average pitch (baseline)), same key/scale? (looking at the set of notes for both bars), note frequency (speed)
- Discord
    - 

Model label	Human name	What it actually measures (succinct)
|A–B|_feat_040	**tempo_mean**	Difference in average tempo (BPM) between the two files
|A–B|_feat_043	**ts_num**	Difference in time-signature numerators (e.g., 3 vs 4)
|A–B|_feat_041	**tempo_std**	Difference in tempo variability (σ of tempos)
A_feat_128	**tsig_count**	Number of time-signature changes (minus 1) in MIDI A
|A–B|_feat_044	**ts_den**	Difference in time-signature denominators (e.g., 4 vs 8)
A_feat_091	**chord_prog_entropy**	Harmonic variety/entropy inside MIDI A
|A–B|_feat_082	**vel_autocorr_1**	Difference in lag-1 autocorrelation of velocities
B_feat_144	**mad_pitches**	Pitch MAD (median absolute deviation) inside MIDI B
|A–B|_feat_039	**max_poly**	Difference in maximum polyphony (simultaneous notes)
B_feat_043	**ts_num**	Time-signature numerator in MIDI B
B_feat_007	**dur_mean**	Mean note duration (beats) in MIDI B
|A–B|_feat_013	**vel_std**	Difference in σ of velocities

    - the main trick is to keep your features minimal and focus on what should not differ between bars
    - so e.g. dont bother putting pitch mean in, because thats very weakly correlated with bar similairty
    - take fur elise as an example
    - some consecutive bars are increasing
    - then the part right after is constant
    - so pitch mean between bars is not predictive of similarity
    - similarly for velocity: think about how crescendos and decrescendos will affect this relationship
    - on the other hand, composers very rarely vary tempo and time signature between bars
    - they also tend not to vary intervals (usually)
    - a basic feature vector that takes the difference of tempos and time signatures should give you >80% immediately
    - then you should experiment with if these differences are signed (do pieces tend to speed up or slow down? or neither) and based off of that, decide if you want the signed difference or the abs difference
    - you should be able to hit 94% using just velocity, tempo, time signature, and interval
    - if you listen to the midis and try to compare them, youll realize that the features that discrminate composers are not actually very discriminatory for bar similarity
    - it basically comes down to like three or four key features that are not at all correlated to the composer but extreemly storngly correlated to bar placement
- Discord (which task is this for?)
    - you should listen to the 3rd test case, the 14th test case, and the 25th test case
    - they gave me the most insight into the important features

Task 3: tagging (audio -> tags)

- Multiclass, multilabel
- Data preprocessing: extract all features into memory rather than reading from the file each time if you're I/O bound
- Cheaper model like MFCC
- Data augmentation: shifting operations, adding noise, random cropping
- @288 Try out different versions of spectrogram representations
- Feature engineering alone didn't produce good results for this task.
- @340 You should have access to a GPU-enabled instance via DataHub for this class I suppose. Setting up this Audio Spectrogram Transformer from HuggingFace got me to about 0.46 on the public leaderboard. There are also more similar models on HuggingFace finetuned on the GTZAN dataset (example) that gave me really good results, and I used a default threshold of 0.5 for determining if a genre should be tagged or not for my final predictions.
    - My current submission doesn't use the above anymore, but it was the best out-of-the-box approach based on my experiments.
- Discord: RE AST
    - I eventually switched strategy to an ensemble of different finetuned models
    - But maybe search for music auto tagging
    - The AST model isn't a part of my current solution, but got me to about 0.47
    - epochs=10, batch_size=4, lr=1e-5
- Discord: I literally inrease the epochs of the baseline and it gotme to 36

HuggingFace docs: ASTConfig, ASTFeatureExtractor, ASTModel, ASTForAudioClassification

- its output is 527
- u gotta convert it into 10 labels

Misc Task 1 Notes

```
XGBoost
Task 1 training accuracy = 1.0
Task 1 validation accuracy = 0.8553719008264463
Task 1 training accuracy = 1.0
Task 1 validation accuracy = 0.8801652892561983
GBC
Task 1 training accuracy = 0.9865702479338843
Task 1 validation accuracy = 0.8223140495867769
LightGBM
Task 1 training accuracy = 1.0
Task 1 validation accuracy = 0.8801652892561983
```

38: 0.6934673366834171 <- I feel like this is overfitting to the public leaderboard

88: 0.6884422110552764

Task 3
dropout = 0.3

```
valuating
evaluating
evaluating
Task 3 training accuracy (exact match) = 0.9747222222222223
Task 3 training mAP = 0.9997261727930568
Task 3 validation mAP = 0.7695737084954082
```

no dropout

```
evaluating
evaluating
evaluating
Task 3 training accuracy (exact match) = 0.9869444444444444
Task 3 training mAP = 0.9997839422684139
Task 3 validation mAP = 0.7650571444951682
```

General guidelines:

- This assignment is tentatively due on **Tuesday, 5pm, week 8 (May 20);** as always, the final deadline is **as posted on gradescope**
- Note the slightly unusual deadline: I want to avoid a clash with the HW4 deadline, and I want the leaderboard to close while people are awake
- Also note that there is **no late period** for this assignment: please submit on time!
- **This assignment is worth 20% of the final grade** (noting grade scaling as posted on the course webpage)
- Assignment 1 must be completed individually

Data and baselines

Assignment 1 data and baselines:
https://drive.google.com/drive/folders/1U_Wb7Q1JhrljRiG9dQ6oyCSPvPCbynm9?usp=sharing
(data is also posted here in case drive gets rate-limited:
https://cseweb.ucsd.edu/classes/sp25/cse253-a/data/)

There are two files in the above folder:

student_files.tar.gz: A compressed folder containing:
- Training datasets for each of the three tasks below, *with* labels
- Testing datasets for each of the three tasks, *without* labels

baselines.ipynb: Working baselines for each of the three tasks

Note that testing labels are not provided: rather you have to make your own predictions on the test set and upload them to gradescope.

Also note that there is no stub provided: it's easiest to just edit **baselines.ipynb** directly to form the basis of your solution. You are not required to follow any of the functions outlined in that file if you don't want to.

Tasks

You are required to build classifiers for each of the following three tasks:

Task 1: Composer classification (symbolic, multiclass). For this task you are required to process a midi file and predict which composer wrote the piece of music. This task is evaluated based on **accuracy** (percentage of correct predictions).

Task 2: Next sequence prediction (symbolic, binary). For this task you are given two bars of music (really, two midi files) and you are required to guess whether the second bar would follow the first in a real piece of music (or whether they are unrelated). I.e., predict “True” if the two bars are neighbors and “False” otherwise. This task is evaluated based on **accuracy.**

Task 3: Music tagging (continuous, multilabel, multiclass). Predict a set of tags (e.g. “electronic”, “chill”) associated with an audio file. The input audio files are outputs of a generative model that synthesized music based on the given tags. Note that for this task an example can have multiple tags. This task is evaluated based on the **mean average precision** (mAP).

Baselines for the three tasks are fairly straightforward, mostly just to show you how to process the data and generate valid outputs that can be submitted to gradescope:

Task 1: Logistic regression; features are computed based on the average pitch and duration.

Task 2: No ML is used: this baseline just measures whether the two bars have a similar average pitch.

Task 3: CNN based on MelSpectrograms; this baseline also shows you how to build your own validation set using a sample of the training data.

Submission

Running the baseline code will produce a valid submission that you can submit to gradescope. It will generate three outputs:

predictions1.json, predictions2.json, predictions3.json

These contain the predictions for the three tasks on the test set. Although you are not given the test labels, the evaluation code in the baselines file shows you how your predictions will be compared to the test labels by the autograder.

The autograder will evaluate your test performance and print your results on a leaderboard. Note that there is both a “public” and “private” leaderboard, each of which contains a random half of the test set. The latter will only be visible after the deadline to prevent people from overfitting to the public half.

You are also required to submit two additional files:

assignment1.py: the code you used to train your model. The autograder does not currently run your code, though this should be runnable in the event that there is any doubt about your solution.
writeup.txt: 1-2 sentences describing your solution to each task. I just want this so that I can describe solutions after the assignment finishes.

Grading

The assignment is worth **20 marks.** The grade breakdown is as follows:

- Submit valid writeup.txt and assignment1.py files **(2 marks).** The autograder doesn't currently check these (and will just give out the marks) but could be modified e.g. if people are submitting empty files.
- Each of the three tasks is worth **6 marks** (3 for the public leaderboard and 3 for the private)

The **breakdown of the 3 marks per leaderboard & task is TBD** but planned guide is as follows (same for all three tasks):

- **0.5 marks:** Submit a valid solution (just run the baseline code and submit it!)
- **1 mark:** Submit a solution that is at least 5% better than the baseline (i.e., >= baseline performance * 1.05)
- **1.5-2.5:** Submit a solution that is "significantly" better than the baseline. Tentative thresholds are below (may adjust down or add smaller increments, won't adjust up)
  - Task 1: 0.4 (1.5); 0.5 (2.0); 0.6 (2.25); 0.65 (2.4); 0.7 (2.5)
  - Task 2: 0.7 (1.5); 0.8 (2.0); 0.85 (2.25); 0.9 (2.5)
  - Task 3: 0.28 (1.5); 0.3 (2); 0.35 (2.25); 0.4 (2.5)
- Submit a solution that is among the top 10% of submissions: **3 marks** (possibly computed "smoothly" rather than given to the top 10% in a binary fashion)

(the above three marks are computed for both the public and private leaderboards, and for each of the three tasks, for a total of 18 marks)

Note: baseline performance, task 1/2/3 (public): 0.251256 / 0.623775 / 0.270429

Getting started

We extracted features from midi files in **Homework 1** and built CNN classifiers for audio in **Homework 2:** you can use those (or the solution files, which I'll post) as the basis of your solutions.

Also please watch the **Thursday week 4 lecture** for discussion of this assignment and some tips about getting started!