The goal was to classify songs as Bach or Mozart. My target metric was accuracy, because there isn't a higher cost for listening to Mozart when I think it's Bach rather than vice versa (for me at least.)

I was also interested in seeing how four strategies would compare with one another based on their validation performance.

Design

Data
To avoid biasing my results by performer, I collected midi files from various music sites, and was able to get the vast majority of Bach and Mozart's keyboard works. I only used piano midis so that the results wouldn't be instrument-based either. After splitting into 30-second snippets, there were 1,400 examples. Instead of just reading the midi files, I converted them to mp3s audio format to make it a more realistic challenge.

Strategy
I followed four separate training strategies, in ascending order of abstraction:

- *Raw chromogram arrays:* 1400 frames x 12 pitch classes -- which I flattened into 1 dimension per example, so that there were 15,000 features each representing a note at a moment in time.
- *Audio features:* Audio feature extraction using Librosa
- *Musical features:* Music feature engineering based on musical analysis
- *Combination of audio and musical features*

Tools
I built three .py files to help out. They're included in my github.

- *MTheory*: Module I wrote for music theory computation and feature generation
- *IREP*: Python implementation of Cohen's Incremental Reduced Error Pruning
- *imlearn*: Implements CustomCV, CustomGridCV, etc. classes, which allow you to train/test-split and train/val-split along a particular feature (ex. 'Song') to avoid information leak

Other tools included Librosa and Flask.

Algorithms

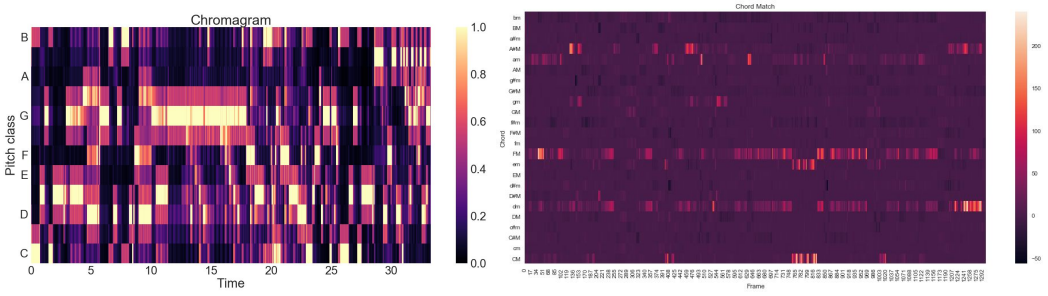
- *Strategy 1: Raw chromogram arrays:* The low-level pitch class arrays were a very girthy dataset -- about 10-to-1 features to examples. So I used logistic regression, SVM, and SVM with PCA. It barely improved off the baseline, and I think something convolutional like RNN might stand a better chance at capturing signal from it.

- *Strategy 2: Audio features:* I extracted selected audio features using Librosa and ran a couple iterations of grid searches split on song, using KNN, SVM (rbg), logistic, tree, and and random forest. The first iteration used log spaces, the second added on to the edges of the grid in cases where the best model fell at a parameter extreme, and the third was a linear space between the best model's neighboring parameters on its previous run. They all did pretty well accuracy-wise, and none of the learning curves looked like they overfit.

- *Strategy 3: Musical features:* MTheory basically computes domain-related distance functions to produce successively higher-abstraction musical features.

  It begins with respective pitch class intensities over time. The most important steps are note-to-note distance (calculated by circle of fifths steps), notes to chord (the shortest distances between each note being played and a note in the chord) for finding how well a possible chord "fits" the actual notes being played. From chords we can derive things like key (based on what harmonies are used in a key), how distant the estimated key is from the estimated key of the song, and so forth. Tonal clarity, mean and var polyphony, use of intervals, and dissonance/consonance/perfect interval classes are some of the other features it can generate.

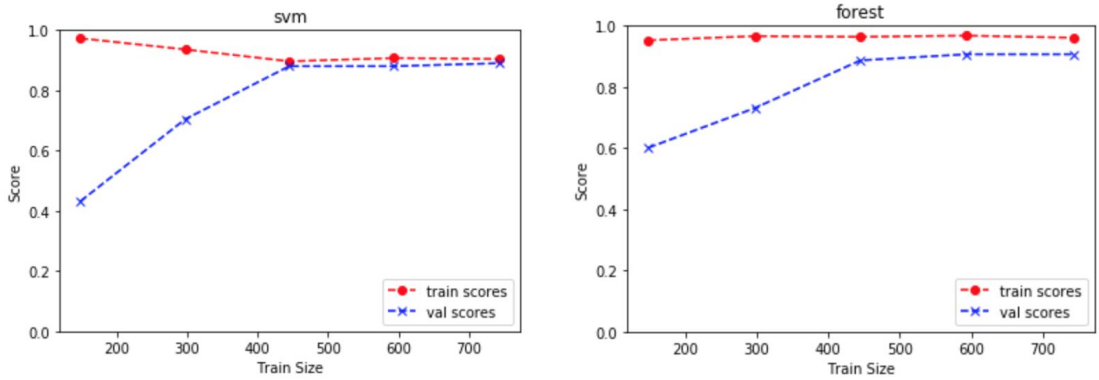
  I implemented MTheory by making Sample, Note, and Chord classes, with most of the relations encoded canonically in the key of C, and transpositions on those canonical relations to describe other notes, chords, and keys. I also implemented a canonical dataframe so you can translate between chords, harmonic relations, and custom domain-specific distance functions, with transpositions allowing it to process any combination. I made a demonstration Jupyter notebook, and I'm planning on writing some documentation so it's easy for my classmates to use in the future if they want.

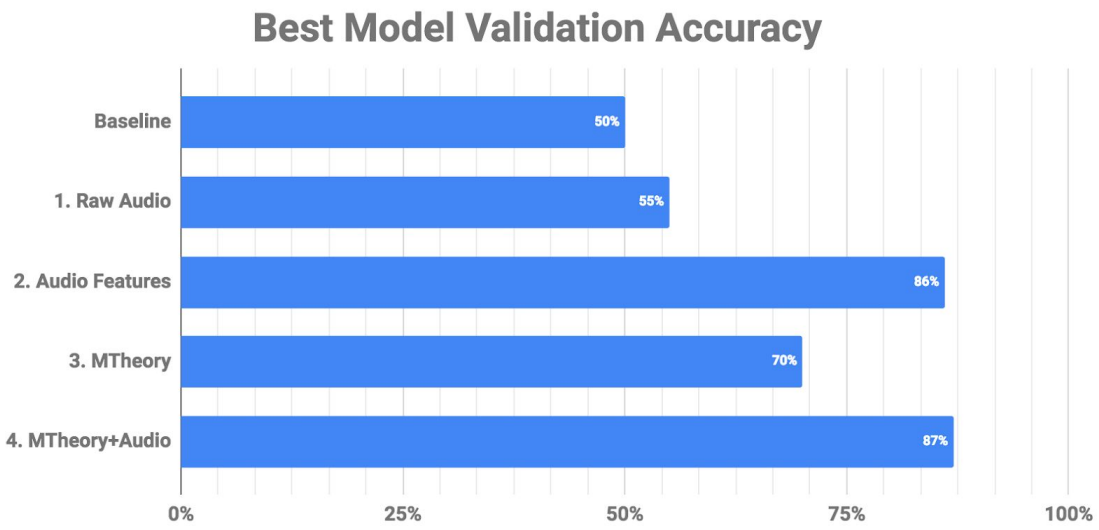


Notes to chords

I followed the same training procedure as with audio features. Though it didn't perform as well as the audio features did, it was still a significant improvement off the baseline, again with non-overfitting-looking learning curves, and it helped improve performance in strategy 4.

- *Strategy 4: Combination of audio and musical features:* Last, I merged the previous two dataframes to combine their features. I applied the same training procedure, and I also trained my implementation of IREP, to see how it would do. The top validation score was only slightly higher, but several of the models improved dramatically. The winners were SVM, random forest, and IREP, with scores of 87%, 86%, and 86%, respectively. I chose random forest as my final model because the scores were similar enough, but it had more of a gap between the training and validation scores on my learning curve and wasn't overfitting, and seemed to have potentially higher upside than SVM were I to train it on the whole test set. My final test score on it was also 86%.





Model	Validation Accuracy	Test Accuracy
Forest	**86%**	**86%**
SVM	**87%**	
IREP	**86%**	
Logistic	83%	
Tree	82%	
KNN	70%	

Finally, I built a flask app that allows the user to input an audio file, and it plays the audio while it extracts librosa data, performs the musical analysis, and applies my pickled model to make a prediction, display a picture of the predicted composer and style change, and in english say whether it was pretty sure or only sort of confident about its prediction.