Summary of Composer Classifier Results

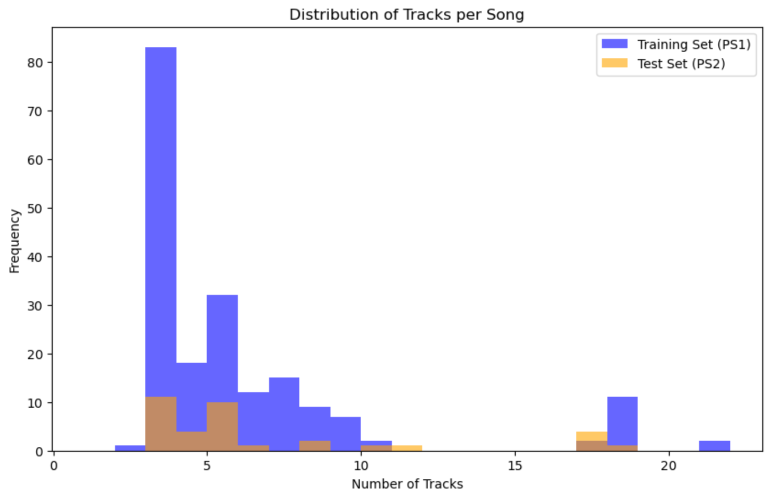
The client has asked us for a classification pipeline that can identify songs *not* written by Bach, Beethoven, Schubert, or Brahms. They claim that it will be used to label live audio, classifying chunks of a song as they are recorded. The training data they provided are multitrack MIDI files of songs written by these four composers, with *no* examples of songs not written by them. We are going to assume that the client has a live audio to MIDI converter, but that it does not create multiple tracks. Therefore, we will treat the training data as one track, as well as throwing out any metadata that would not be included in live audio at inference time.

The first thing to consider here is how we will classify the [not a known composer] when there is not any training data of that class. One option would be to go find or create that training data, but we are assuming the authors of this technical challenge were looking for an answer that worked around this limitation. Here are some strategies we could use:

1) Train four separate models: one that tells Bach from the others, one that tells Brahms from the others, etc. Then at inference time, if all four models output "other", then we label the song as being from an unknown composer.
2) Train a single model that classifies the training data by its composers, and can tell Bach, Beethoven, Schubert, and Brahms apart from each other. Then, at inference time with the live audio, if either the classification probabilities from this model are too low or an anomaly detecting function raises a flag, we label the song as being from an unknown composer. We have decided to use this approach.

The next choice is what type of model we will build. We could extract numeric features from the MIDI chunks such note/time frequencies, pitch frequencies, and many more, then use classical machine learning models on this tabular data. Alternatively, we could build time-series models such as a Hidden Markov Model, N-gram model, recurrent neural network, or more. Here, we have decided to use extracted numeric features and machine learning methods that are compatible with tabular data.

Now that we have decided on a broad strategy, it is time to perform an exploratory data analysis. We will look at the distribution of tracks per song, and class imbalance.



Because the training and test sets have similar distributions of tracks per song, we are going to assume that it is safe to combine tracks.

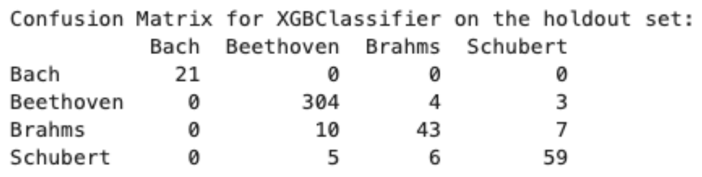
Class Imbalance:

Beethoven:	68%
Schubert:	13%
Brahms:	10%
Bach:	9%

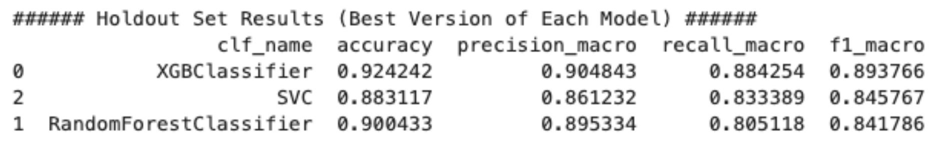
We have moderate class imbalance. It may cause an issue, and may not. We will correct for it by using the built-in class weighting tools in the imported libraries we will be using.

First, we build some helper tools, such as a MIDI file chunker, a feature extractor, and some plotting functions. To decide what features to extract, we googled featurization of music and selected the top recommendations.

We employed a standard model building process: split off a hold out set, perform k-fold cross validation on the remainder of the training set, perform a grid search of models and hyperparameters, check the best model for overfitting on the holdout set. We chose support vector classification, random forest, and gradient boosted trees in the form of XGBoost. The best performing model was XGBoost, and we proceeded without any further tuning or exploration of other models.



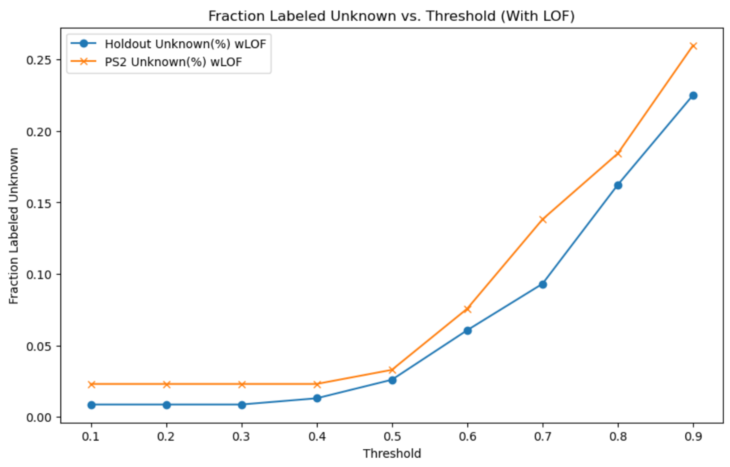
```
Confusion Matrix for XGBClassifier on the holdout set:
           Bach  Beethoven  Brahms  Schubert
Bach         21          0       0         0
Beethoven     0        304       4         3
Brahms        0         10      43         7
Schubert      0          5       6        59
```



```
###### Holdout Set Results (Best Version of Each Model) ######
                 clf_name  accuracy  precision_macro  recall_macro  f1_macro
0           XGBClassifier  0.924242         0.904843      0.884254  0.893766
2                     SVC  0.883117         0.861232      0.833389  0.845767
1  RandomForestClassifier  0.900433         0.895334      0.805118  0.841786
```

Before inference, we need to add the ability to label “unknown” other composers. We will look at the model’s prediction probability for a song, as well as whether songs raise a local outlier factor (LOF) flag. The client did not tell us how they feel about type 1 vs type 2 error, but we are assuming they are okay with some false positives in exchange for a higher detection rate of non-big4 composers. Because of this, we will label songs if they have a sufficiently low prediction probability AND/OR if they raise an LOF flag.

To use the prediction probabilities, we need to find a threshold that does not create too many false positives. Here, we plot threshold vs “unknown” composer label rate for both the hold out training set, and the true test set. Ideally, we want a value that creates a separation in the two rates. We plotted with LOF flag on to check if that had any contribution to labelling (it would show up as the y-intercept of each curve).



We see that LOF has labelled about 1/40 test samples as anomalies, while labelling fewer hold-out set samples as anomalies. This is indicative of predictive power. However, the method based on thresholding of model prediction probabilities does not have any threshold at which the test samples have a significantly higher proportion labelled "unknown" than the hold-out samples do. This indicates no predictive power here and indicates that if we set our threshold above 0.3, we will create false positives with little to no benefit to show for it. Moving forward, we set our threshold at 0.3 but will be relying on the LOF anomaly detector to do the heavy work.

We then run the test set through our model with these parameters and output the results. To get a better estimate of our performance, we ran multiple 30 second samples of each test set file through the pipeline:

```
PS2 Files labeled as UNKNOWN at threshold=0.3:, in at least one of their 30 second samples

   0.07186746659481313_adj.mid
   0.36321860283443286_adj.mid
   0.3334293069825759_adj.mid
```

These three files were identified as possibly being composed by "other" composers, but it is important to note that not all of their 30-second chunks were labelled this way.

For file 0.07186746659481313_adj.mid, 1/3 chunks were flagged
For file 0.3334293069825759_adj.mid, 3/14 chunks were flagged
For file 0.36321860283443286_adj.mid, 3/4 chunks were flagged

This disagreement between chunks is a consequence of an imperfect classifier, which has a measurable false positive rate (0.86% of hold-out training data falsely identified as having come from a fifth composer) and an unmeasurable false negative rate (no labelled training data from other composers).

The next steps towards making this pipeline more useful would be to collect lots of MIDI files generated by the client for composers that are not in the big four. After that, more advanced feature extraction and deep learning would be the recommended approaches to arrive at a powerful tool. Alternatively, because the big four composers only wrote a few a few thousand works combined, we could featurize and hash all segments of all their songs, then look at whether the client's incoming MIDI data match.