Classical Music Period Classification by Spotify Musical Features

Cai, Ding, Peng

Section 1

1.1 Abstract

Music classification has been one of the active subfields of Music Information Retrieval[1]. Specifically, several attempts have been made to classify Western classical music into different commonly-understood historical periods based on audio characteristics. For example, in 2020, Zheng and Xu[2] extracted Mel-frequency Cepstral Coefficients and chroma features from audio signals to train the SVM and KNN classifiers in classifying classical music into four periods. Instead of hand-extracting a few audio features. We aim to find whether Spotify's thorough audio features for each piece of music provided through their Web API can be used to accurately classify songs into the classical periods. We used three traditional classification models, including Multinomial Logit, random forest, as well as Naive Bayes, and were able to build well-calibrated models that achieved high accuracy and AUC scores. We also used classification prediction intervals as a metric to quantify the uncertainty beyond point predictions, which offered us insights to how closely related are the audio features from different periods that hint towards the evolution of classical music.

1.2 Introduction

Spotify provides access to their music catalog through the WEB API, including songs, audio books, and movie soundtracks. For each track, **audio features** have been extracted from cutting-edge audio processing techniques and supervised machine learning algorithms. These audio features encapsulates each track's tempo, rhythm, loudness, and other characteristics.

With Spotify's audio features, we can begin a more-specific genre classification task. The goal of our project is to find whether audio features can be used to accurately classify songs into one of the five classical music periods. In fact, there are more than five periods of classical music, but we combined several similar neighbouring periods when there are too few instances in these categories. The five periods used in our study are shown in figure 1.2 below. These five periods have many musical traits and styles that should make them distinct from each other. To a trained ear, it is easy to classify which of these five periods the music came

---

[1] https://www.ismir.net/

[2] Zheng, Y., & Xu, W. (2020). Western classical music classification by exploiting Mel frequency cepstral coefficients and chroma features. Applied Sciences, 10(6), 1968.

from upon listening. For example, Baroque period music has consistent tempo and ornamented melodies, and many of them are written for the purpose of court dances; Romantic period music is much more free-flowing in tempo, and the melodies are more passionate and expressive; whereas much Modern music is avant-garde, experimental, and atonal. However, instead of getting the opinion of a trained ear, our conjecture was that Spotify's audio features of songs captured some of these subjective terms used to describe classical music, and these quantified features could be used to train and obtain reasonable classification results.

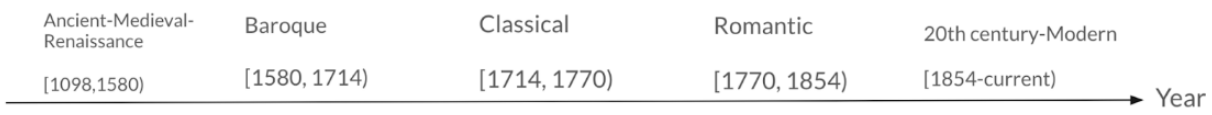


Figure 1.2 Five Periods of Classical Music

Since our response variable is how audio features change over time, we hope the models would offer hints towards the evolution of classical music.

Section 2

2.1 Dataset

2.1.1 Combining Two Datasets

First, we chose the Spotify music dataset on Kaggle[3], where the author pulled about 10,000 songs per genre (Classical, Pop, Rock, etc.) from the Spotify API. We used the portion of the Kaggle dataset where genre is equal to Classical, the 9000 Classical song instances would be sufficient for this project. Originally, there are 14 features besides metadata.

Furthermore, it is also important to subdivide the dataset to more specific historical periods than the broad "umbrella" term – "Classical Music". Therefore, in order to categorize the composers by historical period, we obtained the other dataset from the Classical Music Navigator[4] website. From the website, we generated a composer birth year csv with 500 main composers of classical music. We joined the Spotify Classical dataset (9000 songs) with the Classical Composer Birth Year csv based on the composer name. The final csv file contains 7662 songs with a period column that indicates one of the five historical periods. We chose the boundary years between historical periods manually based on common understanding in music history[5]. In some instances, we chose the cutting point particular to the birth of epoch-making composers presented in the combined dataset. For example, Beethoven transitioned the Classical period into the Romantic period, thus we chose his birth year 1770 as an approximate threshold.

[3] https://www.kaggle.com/datasets/zaheenhamidani/ultimate-spotify-tracks-db?resource=download
[4] https://people.wku.edu/charles.smith/music/index2.htm
[5] https://en.wikipedia.org/wiki/Dates_of_classical_music_eras

2.1.2 Feature Description

Feature explanations are as the following in table 2.1, more details can be found on https://developer.spotify.com/documentation/web-api/reference/get-audio-features:

Variable	Type	Range	Description
Acousticness	Numeric	[0 , 1]	Confidence measure of whether the track is acoustic
Valence	Numeric	[0 , 1]	Musical positiveness conveyed by a track
Danceability	Numeric	[0 , 1]	How suitable a track is for dancing
Energy	Numeric	[0 , 1]	Perceptual measure of intensity and activity
Instrumentalness	Numeric	[0 , 1]	Predicts whether a track contains no vocals
Liveness	Numeric	[0 , 1]	Detects the presence of an audience in the recording
Loudness	Numeric	[-60 , 0]	The overall loudness of a track in decibels (dB)
Speechiness	Numeric	[0 , 1]	Detects the presence of spoken words in a track.
Tempo	Numeric	[34.208 , 212.923]	Overall estimated tempo of a track in beats per minute
Popularity	Numeric	[0 , 68]	How relatively popular the artist is based on total plays
Duration	Numeric	[15509 , 3391040]	Duration of the track in milliseconds
Key	Categorical	{Musical Keys}	The key the track is in
Mode	Categorical	{1 , 0}	Modality (Major is represented by 1 and Minor is 0)
Time_Signature	Categorical	[¾ , 7/4]	Notational convention to specify how many beats are in each bar (or measure)

Table 2.1 Data Description Table

2.2 Exploratory Analysis

2.2.1 Data Transformations

For the purpose of classifying musical periods, seven heavily skewed features were transformed as in table 2. Three of them were binned based on quintiles and the rest four were taken log. Before-and-After transformation histograms of logged feature distribution, and corresponding frequency tables of binned feature distribution against Period are in table 3. After taking logs, previously skewed features have normal or smooth distributions. After binning, previously skewed features appear to have informative distributions against Period Class. Boxplots of transformed feature distributions by Period are included in appendix A.

After the above transformations, two pairs of features appear to be highly correlated as in Figure 2.2.6: LogEnergy and Loudness have a correlation of 0.84; LogValence and Danceability have a correlation of 0.62. Even though these two pairs of features are not collinearity, LogEnergy and Danceability are plotted against Loudness and LogValence, respectively, by Period as in Figure 2.2.8 and 2.2.9. Figures show that LogEnergy and Loudness

indeed correlate across different periods, but are no longer strong enough to concern multicollinearity, but we decide to move forward since 0.84 does not pose a multicollinearity problem. However, LogValence and Danceability are no longer strongly correlated across different periods, which means that both features provide different information on predicting Period.

Variable	Skewness	Transformation
Duration_ms	Right-skewed	Taken log
Energy	Right-skewed	Taken log
Liveness	Right-skewed	Taken log
Valence	Right-skewed	Added a constant of 0.01, then taken log
Speechiness	Right-skewed	'Non-Speech' < 0.042, 'Music-and-Speech' < 0.063, 'Speech' < 1
Acousticness	Left-skewed	'Low' < 0.7, 'Medium' < 0.9, 'High' < 1
Instrumentalness	Left-skewed	'Vocal' < 0.339, 'Vocal-like' < 0.552, 'Instrumental' < 1

Figure2.2.1, Data Transformation Table

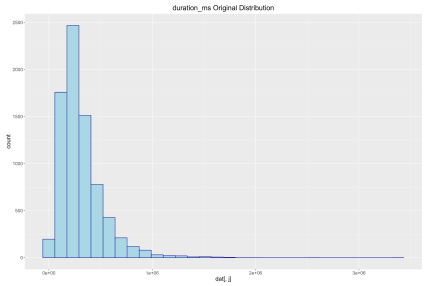


Figure 2.2.2 Duration_ms original distribution

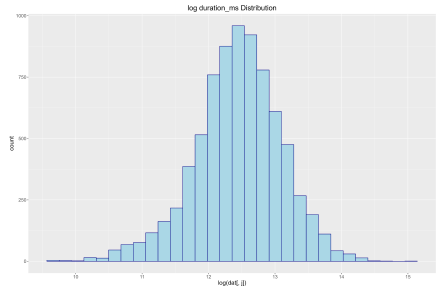


Figure 2.2.3 Duration_ms distribution after transformed

By taking log, Duration is transformed into a smooth and symmetric distribution with one central mode.

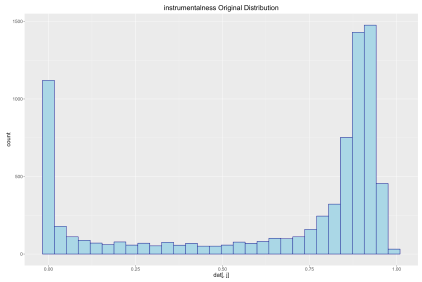


Figure 2.2.4 Instrumentalness Original Distribution

	Instrumental	Vocal	Vocal-like
ancient_medieval_renaissance	14	38	4
baroque	640	530	56
classical	618	328	61
romantic	2545	610	154
twentiethcentury_modern	1548	408	108

Figure 2.2.5 Instrumentalness Binned Distribution by Period

After binning, the majority of Ancient music falls in being "Vocal", while Baroque and Classical music are evenly distributed between being "Instrumental" and "Vocal". The majority of Romantic and Modern music falls in being "Instrumental".

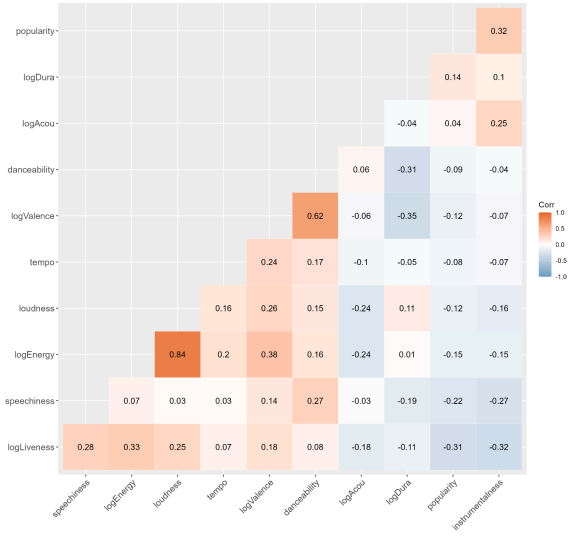


Figure 2.2.6 Correlation Plot of Transformed Data

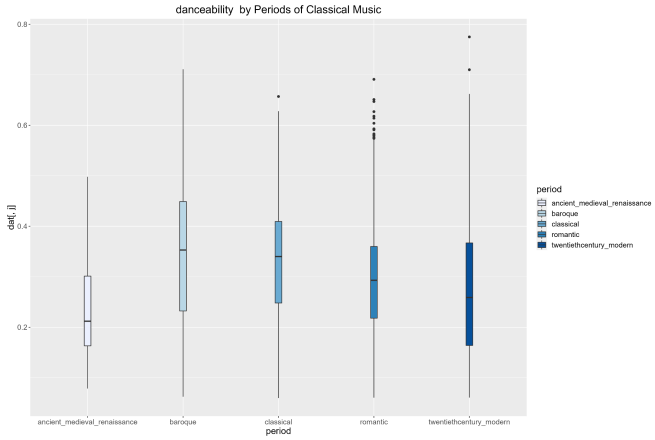


Figure 2.2.7 Danceability Boxplot

This danceability boxplot by period in Figure 2.2.7 is an example of why these audio features should provide insight into classifying songs into different periods. It shows that danceability's value varies across different periods, with ancient songs being the least danceable on average, and baroque songs being the most danceable on average, and the mean danceability for the next three periods decreases gradually. As we explained in our first presentation, many baroque songs are written for the purpose of court dances, so they are more regular in rhythm and more suitable for dancing in general. In contrast, twentieth-century and modern songs are more experimental and less consistent in rhythm.

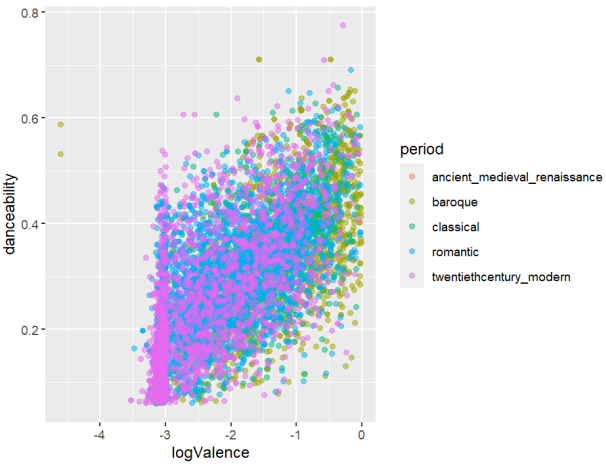


Figure 2.2.8 LogEnergny against Loudness by Period

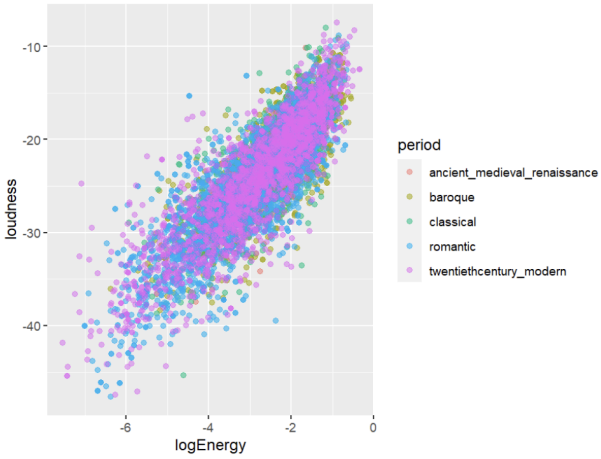


Figure 2.2.9 Danceability against LogValence by Period

2.3 Feature Selection

Feature selection is a crucial step as it helps in identifying the most relevant features that contribute to improving the accuracy of the model while reducing the dimensionality of the problem. In this project, we have used a Random Forest model to classify the periods of a set of

data based on 14 features. However, given the high number of features, we have applied feature selection techniques to identify the most relevant features.

Before diving into feature selection specifics, we would like to define the micro and macro AUC metric that we used extensively in this report. The difference between the two are: Micro AUC is area under one ROC curve where the overall True Positive Rate (TPR) and False Positive Rate (FPR) are calculated by pooling all corresponding labels in the entire dataset, whereas the Macro AUC first computes the TPR and FPR within each class separately, then averages the ROC curves across all classes. Therefore, the Micro AUC score gives equal weight to all instances regardless of classes, whereas the Macro AUC gives equal weight to all classes and averages the performance across classes. For the purpose of feature selection in section 2, we chose to use Macro AUC score as model-comparison criteria, since there is class imbalance in our dataset. We would like to give equal attention to the minority class like the Ancient period, while Micro AUC would ignore class imbalance and overlook the minority periods.

However, our interest extends beyond just the ancient period; since the Macro AUC pays equal attention to the ancient period with only 56 records as other periods with around 2000 records, we also used more specific AUC as well as misclassification rate **for each period** when we compare across model results in Section 4.

2.3.1 Backwards Elimination

The initial model was trained with 14 features, and the aim was to reduce the number of features while minimizing the impact on the model's performance. A backward elimination method under the greedy algorithm was applied to iteratively remove the least important feature from the model, and the resulting macro AUC of the new model was monitored to ensure that the removal of a feature did not significantly affect the model's performance.

We identified the least important feature using the MeanDecreaseGini ranking, which ranks the features based on their contribution to the homogeneity of the subsets created by splitting the data. We chose Gini over MeanDecreaseAccuracy because it can handle imbalanced classes and is less sensitive to the number of classes.

The algorithm starts with all features included in the model, and removes the least important feature at each iteration. The performance of the model is evaluated after each iteration, and the feature subset that produces the macro AUC that is decreased within the pre-set threshold of the current best model is selected as the final set of features. In other words, we remove the least important feature at each step as long as the difference between the max macro AUC so far and the current macro AUC is within 0.02.

2.3.2 Cross-Validation

To ensure the reliability of our feature selection process, we employed 3-fold cross-validation. We divided the dataset into 3 equal parts, and for each iteration, one part was used for validation while the remaining parts were used for training. We repeated this process 3 times, with each part used once for validation. The cross-validation results were averaged to obtain a more robust estimate of the model's performance.

As shown in the 3 MeanDecreaseGini feature importance score plot, the selected features from 3 folds are exactly the same but with different rankings for a few features.

However, the overall feature importance trend is highly similar. This confirms the reliability of our feature selection process, and we can confidently use the selected features in our final model.

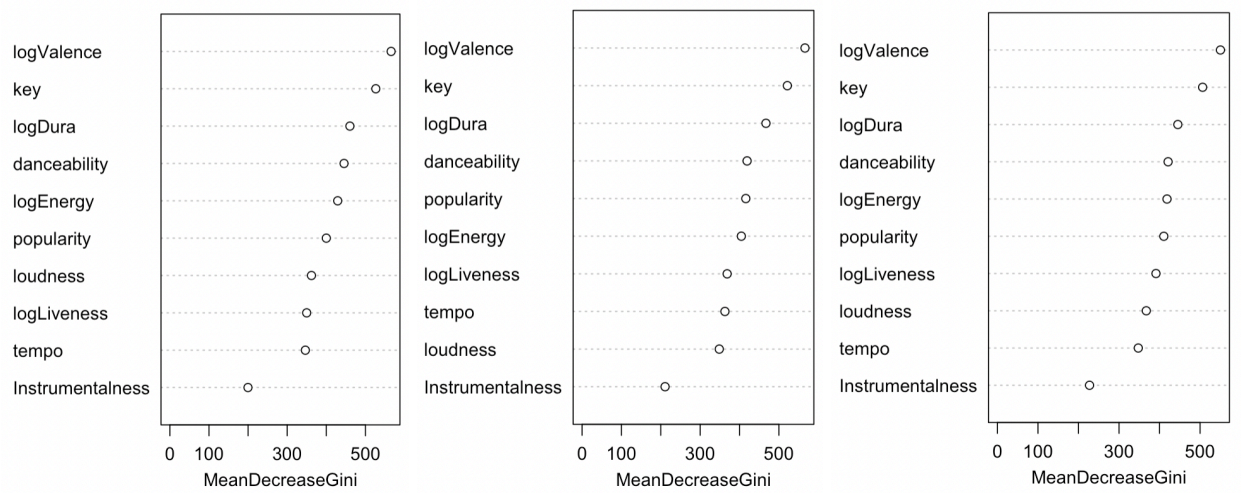


Figure 2.3 Feature Importance Ranking Plot over 3 folds

2.3.3 Resulting Features

The algorithm successfully reduced the total features from 14 to 10 while maintaining the resulting AUC within a threshold of 0.02 difference from the max AUC. The reduction of features from 14 to 10 brings significant benefits in terms of the model's complexity, memory usage, and processing time, especially when dealing with large datasets. Furthermore, a model with fewer features is less prone to overfitting, which can lead to better generalization performance. Despite reducing the 4 features, the resulting macro AUC score only decreased within 0.02. This indicates that the removed features (Acousticness, Speechiness, Mode, and Time_signature) did not significantly impact the model's accuracy, and the remaining features were sufficient to predict the target variable.

Section 3

3.1 Methods and Results

3.1.1 Method 1: Weighted Multinomial Logit Classification

Multinomial Logit Classification is a method used to model relationships between multiple categorical variables based on log ratios, explaining how much each independent variable contributes to the probability of each category of the dependent variable. The package we used for such a model is the “vglm” function from library (VGAM), where models are not given explicit baselines. Our base class is the ancient-medieval-renaissance period. In our case, the dependent variable, period, has five musical periods, and hence the output of multinomial

logistic regression has five sets of coefficients, which represent the effect of each musical feature on the likelihood of each period. These coefficients are later used to make predictions about the probability of each period based on the values of the 10 selected Spotify musical features.

We fit a 3-fold cross-validated weighted multinomial logit regression model on the data, with selected features and weights proportional to the reciprocal of the period (class) sample size due to the unbalanced classes. Based on summary statistics of the model in Figure 2.5, LogValence and LogDura appear to have the most predictive power in this method, since all estimated coefficients associated with these two features are statistically significant. LogEnergy tends to discriminate between Baroque and Ancient music, while LogLiveness tends to discriminate against Ancient music, since corresponding estimated coefficients have significant p-values. Tempo appears to discriminate Ancient and Romantic music.

```
Call:
vglm(formula = period ~ ., family = multinomial(), data = train,
    weights = wt)

Coefficients:
                      Estimate Std. Error z value Pr(>|z|)
(Intercept):1         4.495817   1.430729   3.142 0.001676 **
(Intercept):2         6.711830   1.234295   5.438 5.39e-08 ***
(Intercept):3        -8.410900   1.213429  -6.932 4.16e-12 ***
(Intercept):4        -2.659115   1.141626  -2.329 0.019847 *
logValence:1          0.571258   0.095298   5.994 2.04e-09 ***
logValence:2          1.722609   0.089967  19.147  < 2e-16 ***
logValence:3          1.106925   0.084985  13.025  < 2e-16 ***
logValence:4          0.199017   0.076683   2.595 0.009450 **
logDura:1            -0.583331   0.104450  -5.585 2.34e-08 ***
logDura:2            -0.396573   0.091463  -4.336 1.45e-05 ***
logDura:3             0.811059   0.088861   9.127  < 2e-16 ***
logDura:4             0.162189   0.082991   1.954 0.050666 .
logEnergy:1           1.365779   0.111173  12.285  < 2e-16 ***
logEnergy:2          -0.064929   0.086854  -0.748 0.454720
logEnergy:3          -0.039930   0.084515  -0.472 0.636597
logEnergy:4           0.086399   0.075795   1.140 0.254324
danceability:1       -4.599519   0.657339  -6.997 2.61e-12 ***
danceability:2       -2.966367   0.547309  -5.420 5.96e-08 ***
danceability:3        0.313126   0.534419   0.586 0.557930
danceability:4        0.912537   0.483519   1.887 0.059123 .
popularity:1          0.053475   0.004766  11.220  < 2e-16 ***
popularity:2          0.022712   0.003855   5.891 3.83e-09 ***
popularity:3         -0.008039   0.003607  -2.229 0.025822 *
popularity:4          0.005032   0.003458   1.455 0.145610
logLiveness:1        -1.003183   0.115089  -8.717  < 2e-16 ***
logLiveness:2         0.050942   0.095363   0.534 0.593212
logLiveness:3         0.066272   0.088843   0.746 0.455696
logLiveness:4         0.004181   0.090030   0.046 0.962963
loudness:1           -0.171180   0.019140  -8.944  < 2e-16 ***
loudness:2            0.022468   0.015672   1.434 0.151687
loudness:3            0.012032   0.015436   0.779 0.435697
loudness:4           -0.038157   0.014321  -2.664 0.007714 **
tempo:1              -0.004369   0.002026  -2.156 0.031045 *
tempo:2               0.001543   0.001792   0.861 0.389168
tempo:3               0.001696   0.001732   0.979 0.327654
tempo:4              -0.002487   0.001642  -1.515 0.129839
```

Figure 2.5 Example of part of Multinomial Logistic Output

Overall, results show that the model is consistent across all 3 folds in different out-of-sample criteria, including Macro AUC, 50% and 80% prediction intervals. Moreover, by applying weights to adjust for imbalanced data, minority periods like Ancient Medieval Renaissance are predicted accurately. The method is well-calibrated across folds and periods, especially at 50% intervals. However, the accuracy of classifying minority periods is at the cost of misclassifying some observations in majority periods. 50% prediction intervals for majority periods are worse than minorities'. Fortunately, such trade-off does not persist as prediction intervals get wider to 80%.

The method is calibrated and consistent across all folds and periods. The underperformance in fold 3 compared to the other two folds is out of chance due to sampling variability, which persists in all methods attempted.

Multinomial	Fold 1	Fold 2	Fold 3	Average

Macro AUC	0.744	0.740	0.731	0.738
Micro AUC	0.748	0.746	0.745	0.746
1 - Misclassification Rate (Coverage)				
50% for Ancient	17/23=0.739	12/16=0.75	10/17=0.588	0.692
80% for Ancient	19/23=0.826	13/16=0.813	14/17=0.824	0.821
50% for Baroque	260/406=0.640	270/430=0.628	2537/390=0.649	0.639
80% for Baroque	327/406=0.805	366/430=0.851	321/390=0.823	0.827
50% for Classical	242/334=0.725	216/308=0.701	252/365=0.690	0.705
80% for Classical	291/334=0.871	256/308=0.831	313/365=0.858	0.853
50% for Romantic	645/1065=0.606	682/1141=0.598	640/1103=0.580	0.595
80% for Romantic	880/1065=0.826	1008/1141=0.883	970/1103=0.880	0.863
50% for Modern	446/728=0.613	376/659=0.571	388/679=0.571	0.585
80% for Modern	654/728=0.898	547/659=0.830	569/679=0.838	0.856
Overall 50%	1609/2554=0.630	1556/2554=0.609	1543/2554=0.604	0.614
Overall 80%	2169/2554=0.849	2190/2554=0.858	2187/2554=0.856	0.854

Figure 3.1.1 Multinomial Coverage Table

3.1.2 Method 2: Random Forest

Random Forest is a tree-based ensemble learning method used to build a model by creating multiple decision trees and combining their outputs to improve the overall accuracy and reduce overfitting. This method was chosen for its ability to handle unbalanced classes and its ability to capture non-linear relationships between the selected features and the dependent variable Period.

Same as the previous method, we fit a 3-fold cross-validated weighted multinomial logit regression model on the data, with selected features and weights proportional to the reciprocal of the period (class) sample size to adjust for the class imbalance in the target variable, Period. The results showed that the model was consistent across all three folds, with an average Macro AUC of 0.863.

Overall, the Random Forest method shows promising results in classifying the music period based on the selected musical features. The method is less prone to overfitting and well-calibrated for the majority classes except the “Ancient” class. The trade-off of sacrificing the classification coverage of the "Ancient" class to improve other classes is evident in the results, which may be acceptable since all other classes are performing better.

Random Forest	**Fold 1**	**Fold 2**	**Fold 3**	**Average**

Macro AUC	0.867	0.868	0.855	0.863
Micro AUC	0.905	0.906	0.897	0.903
1 - Misclassification Rate (Coverage)				
50% for Ancient	3/23=0.130	3/16=0.188	1/17=0.059	0.126
80% for Ancient	4/23=0.174	5/16=0.313	5/17=0.235	0.241
50% for Baroque	332/406=0.818	351/430=0.816	312/390=0.8	0.811
80% for Baroque	373/406=0.919	396/430=0.921	351/390=0.9	0.913
50% for Classical	252/334=0.754	232/308=0.753	272/365=0.745	0.749
80% for Classical	300/334=0.898	273/308=0.886	329/365=0.901	0.895
50% for Romantic	900/1065=0.845	956/1141=0.838	915/1103=0.830	0.838
80% for Romantic	1044/1065=0.980	1122/1141=0.983	1074/1103=0.974	0.979
50% for Modern	592/728=0.813	507/659=0.769	537/679=0.791	0.791
80% for Modern	709/728=0.974	640/659=0.971	649/679=0.956	0.967
Overall 50%	2079/2554=0.814	2049/2554=0.802	2037/2554=0.798	0.805
Overall 80%	2430/2554=0.951	2436/2554=0.954	2408/2554=0.943	0.949

Figure 3.1.2 Random Forest Coverage Table

3.1.3 Method 3: Naive Bayes

The Naive Bayes method calculates the probability of belonging to each period for a given input, and classifies the input to the period with the highest probability. The Gaussian kernel is used with bandwidth obtained from bw.nrd0() for each feature given a class. The assumption of independence between the features is what makes the calculation of the likelihood term in Naive Bayes simple (the likelihood is simply the product of PDFs or PMFs over all features for a given period). Since logEnergy and loudness have a linear correlation of 0.84 which is not too collinear, we decide to proceed with Naive Bayes without dropping either variables.

The unbalanced dataset can be addressed using equally weighted prior probabilities. As shown in the stat447-discriminant-analysis lecture slides, the infrequent class, in our case the ancient-medieval-renaissance period, would have “higher posterior predicted probability when the infrequent class is clearly larger, which leads to more interpretable comparisons”. We can see that Naive Bayes’ AUC for the first period is 0.863, even larger than the macro AUC across different classes, averaged over three folds.

Naive Bayes	**Fold 1**	**Fold 2**	**Fold 3**	**Average**
Macro AUC	0.770	0.778	0.749	0.765

Micro AUC	0.781	0.781	0.776	0.779
1 - Misclassification Rate (Coverage)				
50% for Ancient	16/23 = 0.696	12/16 = 0.75	10/17 = 0.588	0.678
80% for Ancient	18/23 = 0.783	13/16 = 0.813	12/17 = 0.706	0.767
50% for Baroque	242/406 = 0.596	282/430 = 0.656	242/390 = 0.621	0.624
80% for Baroque	299/406 = 0.736	341/430 = 0.793	298/390 = 0.764	0.764
50% for Classical	214/334 = 0.641	185/308 = 0.601	215/365 = 0.589	0.610
80% for Classical	271/334 = 0.811	234/308 = 0.760	277/365 = 0.759	0.777
50% for Romantic	620/1065 = 0.582	621/1141 = 0.544	611/1103 = 0.554	0.560
80% for Romantic	856/1065 = 0.804	912/1141 = 0.799	890/1103 = 0.807	0.803
50% for Modern	344/728 = 0.473	300/659 = 0.455	296/679 = 0.436	0.455
80% for Modern	585/728 = 0.804	503/659 = 0.763	516/679 = 0.760	0.776
Overall 50%	1436/2554 = 0.562	1400/2554 = 0.548	1374/2554 = 0.538	0.549
Overall 80%	2106/2554 = 0.825	2003/2554 = 0.784	1996/2554 = 0.782	0.797

Figure 3.1.2 Naive Bayes Coverage Table

3.2 Method Comparison and Summary

With the greedy cross-validated feature selection procedure, 10 features are selected based on macro AUC, and three classification methods are cross-validated in 3 folds. Out of all 14 features, 10 features were selected by the random forest model, including “logValence”, “key”, “logDura”, “danceability”, “logEnergy”, “popularity”, “loudness”, “logLiveness”, “tempo”, and “Instrumentals” in decreasing order of importance. We used these 10 selected features to fit all models. As expected, these Spotify audio features can accurately classify genres to as high of a model prediction accuracy as 0.64, macro AUC of 0.863, and generally well-calibrated prediction intervals.

Looking at table 4.2.1, in terms of point prediction, Random forest performs the best, followed by Naive Bayes, then Multinomial Logistic. In terms of model interpretability and simplicity, multinomial logistic models have the advantage of log-ratio interpretation with (5 classes - 1)×(10 features + 1) = 44 parameters in our case, while Naive Bayes can be interpreted on the modular level given the independence assumption. Random forest is the least interpretable model with the most complexity.

Besides showing the misclassification rate by period in the tables in Section 3.2, the result of specific AUC scores by period is also shown in figure 3.3 below. We chose fold 2 for illustration purposes, but the conclusions are drawn across all three folds: For class A, the Naive Bayes method performs the best since it has the highest average AUC across all three folds of 0.862, followed by RF with an average AUC of 0.863, and Logit with an average AUC of 0.834. For all other periods, random forest was able to achieve much higher AUC scores than the other

two models, and multinomial logit performs slightly better than naive bayes. However, when looking at the accuracy score from table 4.2.1, naive bayes method performs slightly better than multinomial logit. This is because accuracy only looks at true positive rate, whereas AUC uses both true positive rate and false positive rate across different thresholds. Instead of choosing a better model, it would be a better idea to look at specific classes.

```
$auc_objects$Fold2_RF
$auc_objects$Fold2_RF$new
$auc_objects$Fold2_RF$new$A
[1] 0.8781028

$auc_objects$Fold2_RF$new$B
[1] 0.9204627

$auc_objects$Fold2_RF$new$C
[1] 0.8775384

$auc_objects$Fold2_RF$new$R
[1] 0.8360479

$auc_objects$Fold2_RF$new$M
[1] 0.82678

$auc_objects$Fold2_RF$new$macro
[1] 0.8677758

$auc_objects$Fold2_RF$new$micro
[1] 0.9064961
```

```
$auc_objects$Fold2_Logit
$auc_objects$Fold2_Logit$new
$auc_objects$Fold2_Logit$new$A
[1] 0.8307723

$auc_objects$Fold2_Logit$new$B
[1] 0.8109348

$auc_objects$Fold2_Logit$new$C
[1] 0.7550147

$auc_objects$Fold2_Logit$new$R
[1] 0.6816676

$auc_objects$Fold2_Logit$new$M
[1] 0.6197365

$auc_objects$Fold2_Logit$new$macro
[1] 0.7396248

$auc_objects$Fold2_Logit$new$micro
[1] 0.7460616
```

```
$auc_objects$Fold2_NB
$auc_objects$Fold2_NB$new
$auc_objects$Fold2_NB$new$A
[1] 0.9107072

$auc_objects$Fold2_NB$new$B
[1] 0.8224445

$auc_objects$Fold2_NB$new$C
[1] 0.772071

$auc_objects$Fold2_NB$new$R
[1] 0.7095966

$auc_objects$Fold2_NB$new$M
[1] 0.6734126

$auc_objects$Fold2_NB$new$macro
[1] 0.7776455

$auc_objects$Fold2_NB$new$micro
[1] 0.7810075
```

Figure 3.2: Fold 2 AUC score by class for all three method

Looking at prediction intervals, all three methods demonstrated satisfying out-of-sample coverage, but differ in details. Despite being useless at classifying the Ancient period, weighted RF provides near-perfect out-of-sample classifications at 80% intervals across folds. The Weighted Multinomial Logit model does a very good job accounting for imbalanced periods, yet at the cost of less accurately classifying the majorities. Moreover, it is the only method which maintains calibration across all folds and periods at both prediction intervals. The Naive Bayes method demonstrates the same pattern as the Weighted Multinomial Logit model. In terms of misclassification rate, the Naive Bayes method performs worse than the Weighted Multinomial Logit model at both prediction intervals across all folds and periods. Moreover, it becomes uncalibrated at the 50% prediction interval for classifying Modern music. Nonetheless, it successfully accounts for our imbalanced periods and relatively accurately classifies the minority musical periods, which makes itself expectation-meeting.

Overall, the confidence level for 50% classification PI for all methods seems to underestimate the actual coverage, with exception to the ancient period for RF. Besides the ancient period, both 50% and 80% PI for random forest also have much larger coverage than expected. Naive Bayes have less coverage than expected for the 80% PI in some classes.

To summarize, if AUC is criteria or focus is on periods excluding Ancient, random forest is the best method. If periods are equally concerned and model interpretations are required, weighted multinomial logistic is the best choice. If the independence assumption truly holds and computational efficiency is prioritized, the Naive Bayes method is the best choice.

Lastly, music from the Ancient period turns out to be the most difficult to classify given its very small class size of 56. Comparing all three methods, Baroque and Classical period music appear to be the easiest to classify, mainly due to their mid-sized class size (unaffected by

weights) as in the Weighted Multinomial Logit model and unique feature distributions as in Naive Bayes.

Section 4

4.1 Summary

See Section 3.2.

4.2 Discussion

4.2.1 Voting Classifier:

Since certain methods have advantage in making predictions for certain classes, we decide to employ a common ensemble learning technique, Voting Classifier[6]. Voting Classifier aggregates the results from all input models and predicts the final response based on the majority vote. Here, we use “hard” voting as opposed to “soft” voting. “Hard” voting lets each model input a predicted period and vote on the most common letter across our three models. On the other hand, “soft” voting is inputting the predicted probabilities for all classes then averaging the probabilities across different models, and finally taking the highest probability between the five classes probabilities for each instance’s predicted period. We use “hard” voting since predicted probabilities for minority class may be too small for less well-behaved models, and using “soft” voting would wrongfully drag down the average predicted probability. Our hope was that the Voting Classifier would be more robust, and it would improve accuracy, and eliminate the wrong prediction instances from certain classifiers by taking advantage of the majority vote.

Below table summarizes the point prediction accuracy for each of our three models, as well as that of the Voting Classifier. Voting Classifier has higher accuracy than Multinomial Logit and Naive Bayes but lower accuracy than RF. Although the Voting Classifier did not perform better than the best model, RF, this behaviour is as expected. For the Voting Classifier to outperform the best classifier, most of the incorrect predictions by the best classifier would have to be outvoted by the other two less accurate models that happen to make the correct predictions for these instances. Unfortunately, this scenario did not arise. In the future, we could implement the prediction intervals for the Voting Classifier and see if the Voting model performs better when taking into consideration the confidence of the point estimates.

Accuracy	RF	Multi Logit	NB	**Voting**
Fold 1	0.643	0.405	0.458	**0.5**
Fold 2	0.646	0.386	0.442	**0.481**

[6] https://www.kaggle.com/code/saurabhshahane/voting-classifier

Fold 3	0.619	0.386	0.44	**0.475**
Average	0.636	0.392	0.447	**0.485**

Table 4.2.1: Accuracy across different classifiers, including Voting Classifier

4.2.2 Undersampling:

The macro AUC for the infrequent ancient period is lowest for random forest, and the 50% category prediction intervals are not calibrated in random forest albeit applying weighting inversely proportional to the varying class sizes. Undersampling could still be applied in the future to enhance prediction for the minority class.

4.2.3 Music Evolution:

Through the prediction interval table, we can also discover hints of the evolution of classical music. One piece of evidence for the characteristics of music evolved is that in all three methods, closer periods are more easily misclassified, and cases of misclassification generally decrease as periods get further apart in time. For example, as shown in Figure 4.2.3, 108 observations from the Romantic period were misclassified as Classical music. The number of misclassifications decreases as the time gap between musical periods increases: 81 and 84 observations were misclassified as Baroque and Modern music, respectively. Only 45 observations were misclassified as Ancient period which is the farthest from the Romantic period.

	A	AB	AC	AM	AR	B	BA	BC	BM	BMC	BR	C	CA	CAR	CB	CBR	CM
ancient_medieval_renaissance	13	0	0	1	0	1	0	2	0	0	0	2	1	0	1	0	0
baroque	33	2	3	5	5	168	5	24	1	1	10	27	3	0	20	0	4
classical	20	2	6	4	3	34	1	10	2	0	1	134	7	0	24	0	9
romantic	45	0	5	15	20	81	3	22	5	0	12	108	11	0	34	1	12
twentiethcentury_modern	49	3	6	18	15	57	5	18	12	0	5	51	8	1	25	0	9

	CMB	CR	CRB	M	MA	MB	MBR	MC	MR	R	RA	RB	RC	RM	RMA
ancient_medieval_renaissance	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
baroque	0	9	1	13	3	3	0	4	9	24	1	7	4	17	0
classical	0	16	0	3	2	2	1	3	5	21	1	2	4	17	0
romantic	2	37	0	84	8	2	0	8	51	366	12	16	31	74	0
twentiethcentury_modern	0	15	0	163	15	8	0	15	55	103	7	6	8	49	0

Figure 4.2.3: Naive Bayes Fold 1 50% PI

Given the 50% and 80% PI tables for any method on any fold, if one starts looking from the row for the period of interest, the number radiating out to both before and after the current period decreases as the time gap increases. The few counterexamples to this pattern should not affect our conclusion that closer periods indeed have more similar Spotify audio features.

4.2.4 Applications:

As one of the major music streaming APP nowadays, Spotify has over 500 million users listening to over 100 million tracks on their platform[7]. Over this huge classical music catalog, not all tracks have complete metadata information such as year of composition. Even worse, some tracks have year metadata based on the year an artist recorded this classical piece (though we excluded these instances by only including artists whose names are known to be classical

[7] https://newsroom.spotify.com/company-info/

composers). Therefore, using Spotify's audio features, we can distinguish the composition period from the recording year.

Better automatic period classification, as part of the larger "genre tagging" task, is also a vital step towards developing an accurate and personalized music recommendation system. Apple Music also recently launched Apple Music Classical APP on March 28th, 2023, dedicated to streaming classical music exclusively. The point of entry to their 5 million tracks catalog, besides the content in the front page, is its powerful search engine for music by composers, periods, genres, soloists, amongst other choices. The 8 specific periods displayed are Medieval, Renaissance, Baroque, Classical, Romantic, Early 20th Century, Late 20th Century, and 21st Century, which are quite similar labels to our 5 periods. One future step of the project could be to further subdivide the periods into notable composers, though the number of notable composers in any period would far exceed 5 categories.

What's more, we could return the index number of some tracks and its associated information so that developers or users can see what type of songs are easily misclassified. In recent years, the use of deep learning methods to classify musical periods has also been more prevalent[8]. Therefore, it is our hope that users can begin their classical music experience and navigate through the breath-taking classical music repertoire with ease.

Section 5

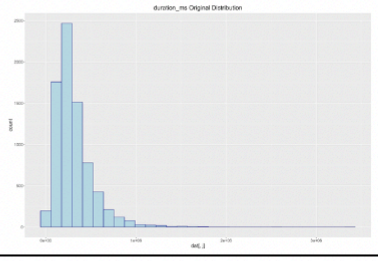
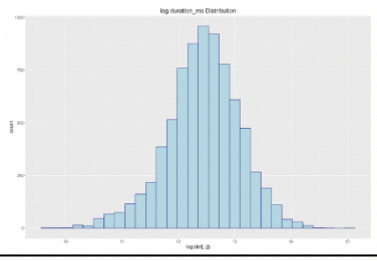
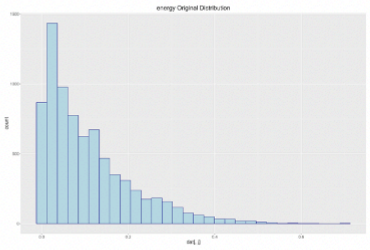
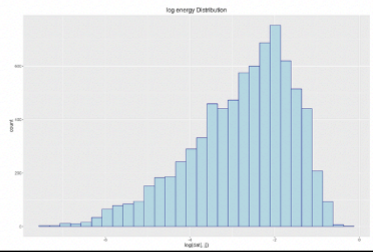
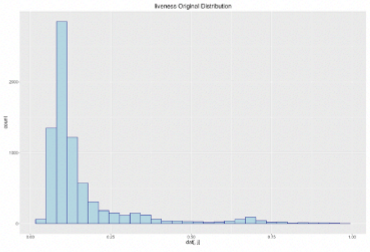
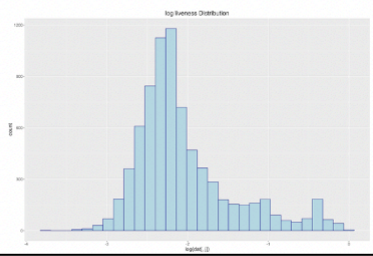
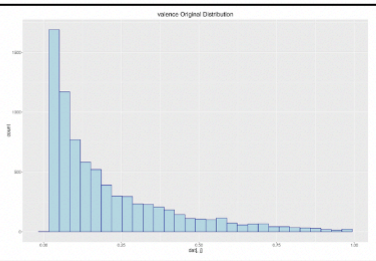
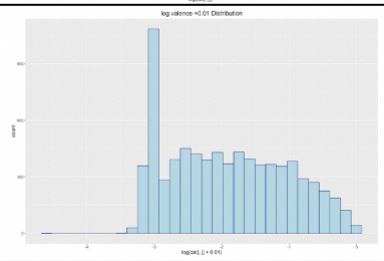
Contributions

Team Member	Code	Report Sections
Yufei Cai	Feature Selection; Random Forest; Code cleaning and documentation; Code refactorization and optimization	2.3, 3.1.2
Meiying Ding	Data Collection; Data Transformation; Data Wrangling and EDA (Joint); Naive Bayes	1, 2.1, 3.1.3, 3.2 (Joint), 4.1 (Joint), 4.2
Yulong Peng	Data Transformation; Data Wrangling and EDA (Joint); Multinomial Logistic	2.1.2, 2.2, 3.1.1, 3.2 (Joint), 4.1 (Joint)

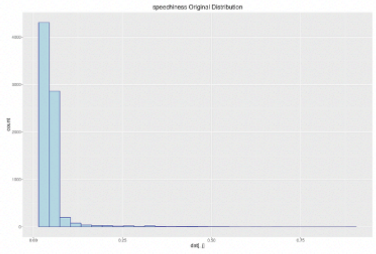
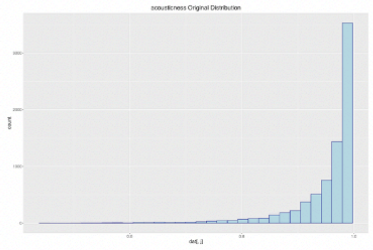
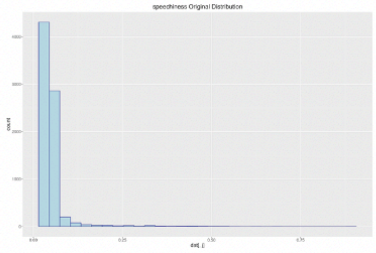
Appendix

Secondary Plot

[8] Choi, K., Fazekas, G., & Cho, K. (2018). Transfer learning for music classification and regression tasks. IEEE Transactions on Audio, Speech, and Language Processing, 26(11), 2180-2193.

Log Transform	Origin	Transformed
Duration_ms		
Energy		
Liveness		
Valence		

Appendix Table 1: Log Transform Result

Binned	Origin	Transformed
Speechiness		
Acousticness		
Instrumentalness		

	music-and-speech	non-speech	speech
ancient_medieval_renaissance	42	14	0
baroque	631	422	173
classical	462	442	103
romantic	1594	1463	252
twentiethcentury_modern	994	843	227

	high	low	medium
ancient_medieval_renaissance	53	1	2
baroque	924	15	287
classical	929	6	72
romantic	3064	14	231
twentiethcentury_modern	1702	54	308

	Instrumental	Vocal	Vocal-like
ancient_medieval_renaissance	14	38	4
baroque	640	530	56
classical	618	328	61
romantic	2545	610	154
twentiethcentury_modern	1548	408	108

Appendix Table 2: Binned Result

```
$auc_objects$Fold1_RF
$auc_objects$Fold1_RF$new
$auc_objects$Fold1_RF$new$A
[1] 0.8731727

$auc_objects$Fold1_RF$new$B
[1] 0.9130088

$auc_objects$Fold1_RF$new$C
[1] 0.8793642

$auc_objects$Fold1_RF$new$R
[1] 0.8204233

$auc_objects$Fold1_RF$new$M
[1] 0.8492097

$auc_objects$Fold1_RF$new$macro
[1] 0.867027

$auc_objects$Fold1_RF$new$micro
[1] 0.9045423
```

```
$auc_objects$Fold1_Logit
$auc_objects$Fold1_Logit$new
$auc_objects$Fold1_Logit$new$A
[1] 0.8536238

$auc_objects$Fold1_Logit$new$B
[1] 0.7883069

$auc_objects$Fold1_Logit$new$C
[1] 0.7807695

$auc_objects$Fold1_Logit$new$R
[1] 0.6416494

$auc_objects$Fold1_Logit$new$M
[1] 0.6562449

$auc_objects$Fold1_Logit$new$macro
[1] 0.7441184

$auc_objects$Fold1_Logit$new$micro
[1] 0.7482691
```

```
$auc_objects$Fold1_NB
$auc_objects$Fold1_NB$new
$auc_objects$Fold1_NB$new$A
[1] 0.8612509

$auc_objects$Fold1_NB$new$B
[1] 0.7954656

$auc_objects$Fold1_NB$new$C
[1] 0.7936397

$auc_objects$Fold1_NB$new$R
[1] 0.7001409

$auc_objects$Fold1_NB$new$M
[1] 0.698167

$auc_objects$Fold1_NB$new$macro
[1] 0.7697313

$auc_objects$Fold1_NB$new$micro
[1] 0.7809425
```

Appendix Fig 1: Fold 1 AUC score by class for all three method

```
$auc_objects$Fold3_RF
$auc_objects$Fold3_RF$new
$auc_objects$Fold3_RF$new$A
[1] 0.8584711

$auc_objects$Fold3_RF$new$B
[1] 0.8924309

$auc_objects$Fold3_RF$new$C
[1] 0.8755872

$auc_objects$Fold3_RF$new$R
[1] 0.8124668

$auc_objects$Fold3_RF$new$M
[1] 0.8364112

$auc_objects$Fold3_RF$new$macro
[1] 0.8550665

$auc_objects$Fold3_RF$new$micro
[1] 0.8965604
```

```
$auc_objects$Fold3_Logit
$auc_objects$Fold3_Logit$new
$auc_objects$Fold3_Logit$new$A
[1] 0.8163649

$auc_objects$Fold3_Logit$new$B
[1] 0.7865551

$auc_objects$Fold3_Logit$new$C
[1] 0.7626889

$auc_objects$Fold3_Logit$new$R
[1] 0.6587291

$auc_objects$Fold3_Logit$new$M
[1] 0.6305241

$auc_objects$Fold3_Logit$new$macro
[1] 0.7309721

$auc_objects$Fold3_Logit$new$micro
[1] 0.7451673
```

```
$auc_objects$Fold3_NB
$auc_objects$Fold3_NB$new
$auc_objects$Fold3_NB$new$A
[1] 0.8167358

$auc_objects$Fold3_NB$new$B
[1] 0.7972795

$auc_objects$Fold3_NB$new$C
[1] 0.7671133

$auc_objects$Fold3_NB$new$R
[1] 0.6917516

$auc_objects$Fold3_NB$new$M
[1] 0.6727344

$auc_objects$Fold3_NB$new$macro
[1] 0.7491218

$auc_objects$Fold3_NB$new$micro
[1] 0.7757099
```

Appendix Fig 2: Fold 3 AUC score by class for all three method

R Code

```
###### STAT 447 Project: Classical Music Historical Period Classification ######

### Source Import ###
source("./01-data-wrangling.R")
source("./02-data-transform.R")
source("./03-feature-selection.R")
source("./04a-random-forest.R")
source("./04b-multinomial-logit.R")
source("./04c-naive-bayes.R")
source("./04d-vote.R")
source("./05-cross-validation.R")
source("./06-method-comparision.R")
source("./functions.R")

### Set Seed ###
seed = 447

### Data Loading ###
dat <- read.csv("data/spotifyclassical_composerbirthperiod.csv")

### Data Wrangling ###
data_wrangle(dat)

### Data Transformation ###
set.seed(seed)
data_transform(dat)
load('data/transformed_data.RData')

### Feature Selection ###
n = nrow(dat)
iperm = sample(n)
crossValidate_feature(Kfold=3, iperm, dat, seed)
load('output/select_features_fold1.RData')

### Method & Cross Validation ###
cv_result = crossValidate(Kfold=3, iperm, dat, nperfMeas=6, seed, selected_features)

### Method Comparison ###
method_compare(cv_result)


 ##### Data Wrangling #####

 data_wrangle <- function(dat) {
   library(dplyr)
   library(ggplot2)
   library(ggcorrplot)
   library(gridExtra)
   library(magrittr)  # for pipe %>%
   library(tibble)

   ## EDA and Plots ##
   # Factorize categorical features
   dat$period <- as.factor(dat$period)
   dat$time_signature <- as.factor(dat$time_signature)
   dat$mode <- as.factor(dat$mode)
   dat$key <- as.factor(dat$key)

   # Select out features to be used for classification
   dat = dat[,c(6:19,23)]
   names(dat)

   # Balanced dataset?
   print(table(dat$period))
   # Period and Key
   print(table(dat$period,dat$key))
   # Period and
   print(table(dat$period,dat$mode))
   # Period and Signature
   print(table(dat$period,dat$time_signature))

   # Summary Statistics and plots for initial data analysis
   summary(dat)
   varn = names(dat)

   # Plot histograms of original distribution of features
   for (j in c(1:6,8,9,11,12,14))
   { print(ggplot(dat, aes(x = dat[,j])) +
             # geom_violin(trim = FALSE) +
             theme(text = element_text(size=15)) +
             geom_histogram(color = "darkblue",fill = "lightblue") +
             scale_fill_brewer() +
             ggtitle(paste(varn[j],"Original Distribution"))+
             theme(plot.title = element_text(hjust = 0.5)))
   }
 }
```

```
##### Transformations #####

data_transform <- function(dat) {
  dato = dat
  varn = names(dat)
  # Factorize categorical features
  dat$period <- as.factor(dat$period)
  dat$time_signature <- as.factor(dat$time_signature)
  dat$mode <- as.factor(dat$mode)
  dat$key <- as.factor(dat$key)

  # Select out features to be used for classification
  dat = dat[,c(6:19,23)]
  names(dat)
  # Transform Acousticness, Duration_ms, energy, liveness, speechiness, valence
  for (j in c(1:6,8,9,11,12,14))
  { if(j == 4|j == 5|j == 8){
    print(ggplot(dat, aes(x = log(dat[,j]))) +
            theme(text = element_text(size=15)) +
            geom_histogram(color = "darkblue",fill = "lightblue") +
            scale_fill_brewer() +
            ggtitle(paste("log",varn[j],"Distribution"))+
            theme(plot.title = element_text(hjust = 0.5)))
  }
    else{
      if(j == 14){
        print(ggplot(dat, aes(x = log(dat[,j]+0.01))) +
                theme(text = element_text(size=15)) +
                geom_histogram(color = "darkblue",fill = "lightblue") +
                scale_fill_brewer() +
                ggtitle(paste("log",varn[j],"+0.01 Distribution"))+
                theme(plot.title = element_text(hjust = 0.5)))}
      else{
        print(ggplot(dat, aes(x = dat[,j])) +
                theme(text = element_text(size=15)) +
                geom_histogram(color = "darkblue",fill = "lightblue") +
                scale_fill_brewer() +
                ggtitle(paste(varn[j],"Distribution"))+
                theme(plot.title = element_text(hjust = 0.5)))
      }
    }
  }
```

```
LogTransform = function(dataset)
{ dataTransform = as_tibble(dataset) %>%
  dplyr::mutate( logDura=log(duration_ms),
                 logEnergy=log(energy),
                 logLiveness=log(liveness),
                 logValence=log(valence+0.01)) %>%
  dplyr::select(logDura,logEnergy,logLiveness,logValence,
                popularity,danceability,
                key,loudness,mode,
                tempo,time_signature,
                period,Speechiness,Acousticness,Instrumentalness
                # acousticness,instrumentalness,instrumentalness,
  )
dataTransform
}

# Find Bins for speechiness: 0.042<0.063<1
quantile(dat$speechiness,seq(0,1,0.1))
summary(dat$speechiness>0.25)
summary(dat$speechiness>0.2)
summary(dat$speechiness>0.1)
summary(dat$speechiness>0.063)

# Find Bins for speechiness: 0.339<0.552<1
quantile(dat$instrumentalness, seq(0,1,0.1))
summary(dat$instrumentalness<0.339)
summary(dat$instrumentalness<0.552)

# transform re-loaded data and plot
dato = dato %>% mutate(period = as.factor(period),
                       time_signature = as.factor(time_signature),
                       mode = as.factor(mode),
                       key = as.factor(key),
                       Speechiness = as.factor(case_when(speechiness < 0.042 ~ 'non-speech',
                                                         speechiness < 0.063 ~ 'music-and-speech',
                                                         speechiness < 1 ~ 'speech')),
                       Acousticness = as.factor(case_when(acousticness < 0.7 ~ 'low',
                                                          acousticness < 0.9 ~ 'medium',
                                                          acousticness < 1 ~ 'high')),
                       Instrumentalness = as.factor(case_when(instrumentalness < 0.339 ~ 'Vocal',
                                                              instrumentalness < 0.552 ~ 'Vocal-like',
                                                              instrumentalness < 1 ~ 'Instrumental')))

dat = as.data.frame(LogTransform(dato))
varn = names(dat)
nfeature = length(varn)
```

```
  dat = as.data.frame(LogTransform(dato))
  varn = names(dat)
  nfeature = length(varn)

  # Side-by-side boxplots
  for(j in 1:(nfeature-3))
  { print(ggplot(dat, aes(x = period, y = dat[,j], fill = period)) +
            # geom_violin(trim = FALSE) +
            theme(text = element_text(size=15)) +
            geom_boxplot(width = 0.07) +
            scale_fill_brewer() +
            ggtitle(paste(varn[j],"Boxplot by Different Periods of Classical Music"))+
            theme(plot.title = element_text(hjust = 0.5)))}

  # Correlation Matrix and its plot
  cordat = dat[,c(1:6,8,10)]
  corr = round(cor(cordat),2)
  ggcorrplot(corr, hc.order = TRUE, type = "lower", lab = TRUE,
             outline.col = "white",
             ggtheme = ggplot2::theme_gray,
             colors = c("#6D9EC1", "white", "#E46726"))

  # Period and Speechiness
  print(table(dat$period,dat$Speechiness))

  # Period and Instrumentalness
  print(table(dat$period,dat$Instrumentalness))

  # Plot logEnergny against loudness by period
  ggplot(dat,aes(logEnergy,loudness, col = period))+
    geom_point(alpha=0.5)

  # Plot logValence against danceability by period
  ggplot(dat,aes(logValence,danceability, col = period))+
    geom_point(alpha=0.5)

  save(file="data/transformed_data.RData",dat)
}
```

```
###### Feature Selection ######

## Analyse Current Model
#' @describe This function calculated the current rebuild method Macro AUC score for further comparision
#' @param RF The rebuilt random forest model
#' @param holdo_data current fold holdo set
#' @return current model Macro AUC score
analyse_curr_model <- function(RF, holdo_data) {
  outpredRF <- predict(RF, type="prob", newdata=holdo_data)
  curr_auc = calculateAUC(outpredRF, holdo_data)[[1]][[6]]

  return (curr_auc)
}

## Rebuild
#' @describe This function is to rebuild the model without the bottom features
#' @param train_data current fold train set
#' @param holdo_data current fold holdo set
#' @param wt weight to account for the class imbalanced
#' @return a vector that with the current rebuild model, current model auc score, and new feature ranking
rebuild <- function(train_data, holdo_data, wt) {
  # Build random forest model with updated data
  RF_model <- randomForest(period ~ ., data = train_data, importance = TRUE, proximity = TRUE, weight = wt)
  imp_scores <- round(importance(RF_model), 2)
  top_feats <- imp_scores[order(-imp_scores[,"MeanDecreaseGini"]), , drop = FALSE]
  top_feats_names <- rownames(top_feats)

  # Calculate AUC of new model
  curr_auc <- analyse_curr_model(RF_model, holdo_data)

  print(".................CURRENT FEATURES.................")
  print(top_feats_names)
  print(curr_auc)

  return(list(model = RF_model, auc = curr_auc, selected_feats = top_feats_names))
}
```

```
## Feature Selection
#' @describe This function applied backward elimintation under the greedy algorithm
#' @param train_data current fold train set
#' @param holdo_data current fold holdo set
#' @param wt weight to account for the class imbalanced
find_max_auc <- function(train_data, holdo_data, wt, k) {
  max_auc <- 0
  selected_features <- c()

  # Initial full model
  prev_model <- randomForest(period ~ ., data = train_data, importance = TRUE, proximity = TRUE, weights = wt)

  # Initial AUC calculation
  curr_auc <- analyse_curr_model(prev_model, holdo_data)
  prev_auc <- curr_auc
  max_auc <- curr_auc
  curr_features <- colnames(train_data)
  max_auc_features <- colnames(train_data)

  while (length(curr_features) > 1) {
    # Rebuild model without least important feature
    model_result <- rebuild(train_data, holdo_data, wt)
    curr_model <- model_result$model
    curr_auc <- model_result$auc
    model_features<- model_result$selected_feats
    curr_features <- model_features

    # Update selected features and max AUC if current AUC is better
    if ((max_auc - curr_auc) < 0.02) {
      if (curr_auc >= max_auc) {
        max_auc = curr_auc
      }
      print(".....AUC Better.....")
      max_auc_features <- curr_features
    } else {
      print(".....AUC Worse.....")
      curr_auc = prev_auc
      break # Stop if current AUC is worse
    }
    prev_auc = curr_auc
    prev_model = curr_model
    # Get least important feature
    bottom_feats_name <- tail(curr_features, 1)
    print(".....ATTEMPT REMOVING BOTTOM FEATURE.....")
    print(bottom_feats_name)
    train_data <- train_data[, !colnames(train_data) %in% bottom_feats_name]
    holdo_data <- holdo_data[, !colnames(holdo_data) %in% bottom_feats_name]
  }
```

```
  # Save the list to a file
  selected_features=max_auc_features
  file_name <- paste0("select_features_fold", k, ".RData")
  folder_path <- "output/"
  save(selected_features, file = paste0(folder_path, file_name))

  # Plot the importance score ranking
  varImpPlot(prev_model)
}

## Build Random Forest Model Entry Point
#' @describe This function is the entry point of the feature selection and model building
#' @param train_data current fold train set
#' @param holdo_data current fold holdo set
#' @param wt weight to account for the class imbalanced
#' @param k current k-fold index
#' @return a vector that with the model that built with final selected features, the model auc object, and the outpred
class result
select_features <- function(train_data, holdo_data, wt, k) {
  library(randomForest)

  # Assign Periods
  periods <- levels(train_data$period)
  train_data$period <- factor(train_data$period, levels = periods)
  if (!is.null(holdo_data)) {
    holdo_data$period <- factor(holdo_data$period, levels = periods)
  }

  # Find best set of features based on training and holdout datasets
  result <- find_max_auc(train_data, holdo_data, wt, k)
}

##### Random Forest Model #####

## Analyse Final Model
#' @describe This function calculated the final RF model PI and save the result into RData file
#' @param RF The final random forest model
#' @param holdo_data current fold holdo set
#' @param k current k-fold index, use for separating the output in each fold
#' @return a vector that with current model auc object and the final outpred class result
analyse_final_model <- function(RF, holdo_data, k) {
  outpredRF <- predict(RF, type="prob", newdata=holdo_data)

  # Calculate PI
  predintRF <- CategoryPredInterval(outpredRF)
  pred50 = table(holdo_data$period, predintRF$pred50)
  pred80 = table(holdo_data$period, predintRF$pred80)
  rf_tables <- list(pred50, pred80)

  # Get outpred class result
  outpred_class <- max.col(outpredRF)
  outpred_class_factor <- factor(outpred_class, levels = c(1, 2, 3, 4, 5),
                                 labels = c("A", "B", "C", "R", "M"))

  # Save the list to a file
  file_name <- paste0("rf_tables_fold", k, ".RData")
  folder_path <- "output/"
  save(rf_tables, file = paste0(folder_path, file_name))

  model_auc_object = calculateAUC(outpredRF, holdo_data)

  return (list(model_auc_object=model_auc_object, outpred_class_factor=outpred_class_factor))
}
```

```r
## Build Random Forest Model Entry Point
#' @describe This function is the entry point of the feature selection and model building
#' @param train_data current fold train set
#' @param holdo_data current fold holdo set
#' @param wt weight to account for the class imbalanced
#' @param k current k-fold index
#' @return a vector that with the model that built with final selected features, the model auc object, and the outpred
class result
build_rf <- function(train_data, holdo_data, wt, k, seed) {
  library(randomForest)

  # Assign Periods
  periods <- levels(train_data$period)
  train_data$period <- factor(train_data$period, levels = periods)
  if (!is.null(holdo_data)) {
    holdo_data$period <- factor(holdo_data$period, levels = periods)
  }

  # Find best set of features based on training and holdout datasets
  set.seed(seed)
  RF_model <- randomForest(period ~ ., data = train_data, importance = TRUE, proximity = TRUE, weight = wt)

  analyse_result = analyse_final_model(RF_model, holdo_data, k)
  model_auc_object = analyse_result[[1]]
  model_outpred_class = analyse_result[[2]]

  return (list(model = RF_model, model_auc_object = model_auc_object, model_outpred_class=model_outpred_class))
}

##### Weighted Multinomial Logit #####

## Build Multinomial Model Entry Point
#' @describe This function is the entry point of the model building
#' @param train_data current fold train set
#' @param holdo_data current fold holdo set
#' @param wt weight to account for the class imbalanced
#' @param k current k-fold index
#' @return logit model
build_multinom <- function(train, holdo, wt, k, seed) {
  library(VGAM)
  ## Fit full and selected models with weighted versions
  selected_weighted_model = analyse_logit(train, holdo, wt, k, seed)
  return (selected_weighted_model)
}

## Analyse Logit
#' @describe This function is the build the multi logit model and its PI,
#' saving the PI into RData file and calculate the outpred class result
#' @param train_data current fold train set
#' @param holdo_data current fold holdo set
#' @param wt weight to account for the class imbalanced
#' @param k current k-fold index
#' @return a vector with logit model, auc object and outpred class result
analyse_logit <- function(train, holdo, wt, k, seed) {
  # Fit multinomial logit model with weights
  set.seed(seed)
  mlogit = vglm(period~., multinomial(), data=train, weights = wt)
  outpred = predict(mlogit, type="response", newdata=holdo)
  predint = CategoryPredInterval(outpred)

  outpred_class <- max.col(outpred)
  outpred_class_factor <- factor(outpred_class, levels = c(1, 2, 3, 4, 5),
                                 labels = c("A", "B", "C", "R", "M"))

  # Save the list to a file
  pred50 = table(holdo$period, predint$pred50)
  pred80 = table(holdo$period, predint$pred80)
  logit_tables <- list(pred50, pred80)

  file_name <- paste0("logit_tables_fold", k, ".RData")
  folder_path <- "output/"
  save(logit_tables, file = paste0(folder_path, file_name))

  # Calculate AUC score
  model_auc_object <- calculateAUC(outpred, holdo)

  return (list(logit_model = mlogit, model_auc_object = model_auc_object, outpred_class_factor=outpred_class_factor))
}
```

```
##### Naive Bayes  #####

## Build Navie Bayes Model Entry Point
#' @describe This function is the entry point of the model building
#' @param train_data current fold train set
#' @param holdo_data current fold holdo set
#' @param k current k-fold index
#' @return navie bayes model
build_nb <- function(train, holdo, k, seed) {
  # df including all features, as well as period as the last column
  selected_df_train <- train
  selected_df_holdo <- holdo
  selected_nb_model <- analyse_full_nb(selected_df_train, selected_df_holdo, k, seed)

  return (selected_nb_model)
}

## Build Navie Bayes Model Entry Point
#' @describe This function is the build the Naive Bayes model and its PI,
#' saving the PI into RData file and calculate the outpred class result
#' @param train_data current fold train set
#' @param holdo_data current fold holdo set
#' @param k current k-fold index
#' @return a vector with navie bayes model, auc object and outpred class result
analyse_full_nb <- function(train, holdo, k, seed) {
  set.seed(seed)
  # Get training data for each period/class
  c1 = subset(train, period == "ancient_medieval_renaissance")
  c2 = subset(train, period == "baroque")
  c3 = subset(train, period == "classical")
  c4 = subset(train, period == "romantic")
  c5 = subset(train, period == "twentiethcentury_modern")

  # Get product of pdfs over all features for each class k
  c1pdfpmfprod <- prod_features(holdout = holdo, ck = c1) # or holdo[1:x,] if want to examine first x songs
  c2pdfpmfprod <- prod_features(holdout = holdo, ck = c2)
  c3pdfpmfprod <- prod_features(holdout = holdo, ck = c3)
  c4pdfpmfprod <- prod_features(holdout = holdo, ck = c4)
  c5pdfpmfprod <- prod_features(holdout = holdo, ck = c5)

  # Get posterior probabilities for class 1 to 5
  outpredPostc1 <- round(c1pdfpmfprod/(c1pdfpmfprod+c2pdfpmfprod+c3pdfpmfprod+c4pdfpmfprod+c5pdfpmfprod), 3)
  outpredPostc2 <-round(c2pdfpmfprod/(c1pdfpmfprod+c2pdfpmfprod+c3pdfpmfprod+c4pdfpmfprod+c5pdfpmfprod), 3)
  outpredPostc3 <-round(c3pdfpmfprod/(c1pdfpmfprod+c2pdfpmfprod+c3pdfpmfprod+c4pdfpmfprod+c5pdfpmfprod), 3)
  outpredPostc4 <-round(c4pdfpmfprod/(c1pdfpmfprod+c2pdfpmfprod+c3pdfpmfprod+c4pdfpmfprod+c5pdfpmfprod), 3)
  outpredPostc5 <-round(c5pdfpmfprod/(c1pdfpmfprod+c2pdfpmfprod+c3pdfpmfprod+c4pdfpmfprod+c5pdfpmfprod), 3)

  # Within each class see the summary (mean/median/quantiles) of predicted probabilities
  summary(outpredPostc1[holdo$period == "ancient_medieval_renaissance"])
  summary(outpredPostc2[holdo$period == "baroque"])
  summary(outpredPostc3[holdo$period == "classical"])
  summary(outpredPostc4[holdo$period == "romantic"])
  summary(outpredPostc5[holdo$period == "twentiethcentury_modern"])


  # Format the out of sample prediction to be the same as the output of predict() function (ex. predict(mlogitw,
type="response", newdata=holdo))
  outpredPost <- cbind(outpredPostc1, outpredPostc2, outpredPostc3, outpredPostc4, outpredPostc5)

  # Get outpred class result
  outpred_class <- max.col(outpredPost)
  outpred_class_factor <- factor(outpred_class, levels = c(1, 2, 3, 4, 5),
                                 labels = c("A", "B", "C", "R", "M"))

  # Prediction Interval #
  predintNaiveBayes = CategoryPredInterval(outpredPost)

  pred50 = table(holdo$period, predintNaiveBayes$pred50)
  pred80 = table(holdo$period, predintNaiveBayes$pred80)
  nb_tables <- list(pred50, pred80)

  # Save the list to a file
  file_name <- paste0("nb_tables_fold", k, ".RData")
  folder_path <- "output/"
  save(nb_tables, file = paste0(folder_path, file_name))

  model_auc_object <- calculateAUC(outpredPost, holdo)

  return (list(outpred = outpredPost, model_auc_object = model_auc_object, outpred_class_factor=outpred_class_factor))
}
```

```
##### Voting #####

##  Voting Classifier
#' @describe This function calculate the exact match classification accuracy of
#' each method and aggregates the results from all input models and predicts the
#' final response based on the majority vote.
#' @param acc_matrix matrix to record the accuacy accross different fold
#' @param rf_output random forest outpred class result
#' @param multi_output multinomial logit outpred class result
#' @param nb_output naive bayes outpred class result
#' @param holdo current fold holdo set
#' @param k current k-fold index
#' @param Kfold total number of folds
#' @return accuracy matrix of all methods
vote <- function(acc_matrix, rf_output, multi_output, nb_output, holdo, k, Kfold) {
  # Create a matrix with the classification results from each method
  all_outputs <- matrix(c(rf_output, multi_output, nb_output), nrow=length(rf_output), ncol=3)
  # Take the mode (most common element) of each row
  majority_vote <- apply(all_outputs, 1, function(x) names(which.max(table(x))))

  holdo$period <- gsub("ancient_medieval_renaissance", "A", holdo$period)
  holdo$period <- gsub("baroque", "B", holdo$period)
  holdo$period <- gsub("classical", "C", holdo$period)
  holdo$period <- gsub("romantic", "R", holdo$period)
  holdo$period <- gsub("twentiethcentury_modern", "M", holdo$period)

  holdo_class <- holdo$period
  total <- length(holdo_class)

  ### RF ###
  correct_rf <- sum(rf_output == holdo_class)
  accuracy_rf <- correct_rf / total

  ### Multi ###
  correct_multi <- sum(multi_output == holdo_class)
  accuracy_multi <- correct_multi / total

  ### NB ###
  correct_nb <- sum(nb_output == holdo_class)
  accuracy_nb <- correct_nb / total

  ### Vote ###
  correct_votes <- sum(majority_vote == holdo_class)
  accuracy_votes <- correct_votes / total

  acc_matrix[k,1] <- accuracy_rf
  acc_matrix[k,2] <- accuracy_multi
  acc_matrix[k,3] <- accuracy_nb
  acc_matrix[k,4] <- accuracy_votes

  return(acc_matrix)
}
```

```
##### Cross Validation #####

## Cross Validate on Models
#' @param Kfold number of folds for cross-validation
#' @param iperm permutation vector for rows
#' @param datafr assume dataframe has y x1 ... xp, can modify below to be more general
#' @param nperfMeas number of performance measures
crossValidate = function(Kfold, iperm, datafr, nperfMeas, seed, selected_features) {
  set.seed(seed)
  n = nrow(datafr);
  nhold = round(n/Kfold)
  pred = list() # save predictions in order to be able to compare them for different methods
  accuracy = matrix(0,Kfold,4)
  perfMat = matrix(0,Kfold,nperfMeas)  # for storing performance measures
  auc_objects = list() # for storing auc objects for each model in each fold
  for(k in 1:Kfold) {
    ilow = (k-1)*nhold+1; ihigh = k*nhold
    if(k==Kfold) {
      ihigh = n
    }
    ifold = iperm[ilow:ihigh]
    train = datafr[-ifold,]
    holdo = datafr[ifold,]
    wt = calculate_weight(train)

    train_selected = train[, c(selected_features, "period")]
    holdo_selected = holdo[, c(selected_features, "period")]

    ### Random Forest ###
    rf_result = build_rf(train_selected, holdo_selected, wt, k, seed)
    rf_model = rf_result[[1]]
    rf_auc_object = rf_result[[2]]
    perfMat[k,1] = rf_auc_object[[1]][[6]] # Macro AUC
    perfMat[k,2] = rf_auc_object[[1]][[7]] # Micro AUC
    rf_outpred_class = rf_result[[3]]
    auc_objects[[paste0("Fold", k, "_RF")]] <- rf_auc_object


    ### Multinomial Logit ###
    multi_model = build_multinom(train_selected, holdo_selected, wt, k, seed)
    multi_auc_object = multi_model[[2]]
    perfMat[k,3] = multi_auc_object[[1]][[6]] # Macro AUC
    perfMat[k,4] = multi_auc_object[[1]][[7]] # Micro AUC
    multi_outpred_class = multi_model[[3]]
    auc_objects[[paste0("Fold", k, "_Logit")]] <- multi_auc_object

    ### Naive Bayes ###
    nb_model = build_nb(train_selected, holdo_selected, k, seed)
    nb_auc_object = nb_model[[2]]
    perfMat[k,5] = nb_auc_object[[1]][[6]] # Macro AUC
    perfMat[k,6] = nb_auc_object[[1]][[7]] # Micro AUC
    nb_outpred_class = nb_model[[3]]
    auc_objects[[paste0("Fold", k, "_NB")]] <- nb_auc_object


## Cross Validate on Feature Selection Process
#' @param Kfold number of folds for cross-validation
#' @param iperm permutation vector for rows
#' @param datafr assume dataframe has y x1 ... xp, can modify below to be more general
crossValidate_feature = function(Kfold, iperm, datafr, seed) {
  set.seed(seed)
  n = nrow(datafr);
  nhold = round(n/Kfold)
  for(k in 1:Kfold) {
    ilow = (k-1)*nhold+1; ihigh = k*nhold
    if(k==Kfold) {
      ihigh = n
    }
    ifold = iperm[ilow:ihigh]
    train = datafr[-ifold,]
    holdo = datafr[ifold,]
    wt = calculate_weight(train)

    ### Select Features ###
    select_features(train, holdo, wt, k)
  }
}
```

```
#### Method Comparison ####

## Compare Method
#' @describe print AUC matrix and load the PI into global var
#' @param cv_result the result from the cross validation
method_compare <- function(cv_result) {
  # Macro AUC Matrix
  print(cv_result[[1]])
  # Class AUC Matrix
  classAUCMatrix = cv_result[[4]]

  # Macro AUC Matrix
  print(cv_result[[5]])

  # Random Forest PI
  load("output/rf_tables_fold1.RData")
  rf_tables_fold1 <- rf_tables
  load("output/rf_tables_fold2.RData")
  rf_tables_fold2 <- rf_tables
  load("output/rf_tables_fold3.RData")
  rf_tables_fold3 <- rf_tables

  # Multi Logit PI
  load("output/logit_tables_fold1.RData")
  logit_tables_fold1 <- logit_tables
  load("output/logit_tables_fold2.RData")
  logit_tables_fold2 <- logit_tables
  load("output/logit_tables_fold3.RData")
  logit_tables_fold3 <- logit_tables

  # Navie Bayes PI
  load("output/nb_tables_fold1.RData")
  nb_tables_fold1 <- nb_tables
  load("output/nb_tables_fold2.RData")
  nb_tables_fold2 <- nb_tables
  load("output/nb_tables_fold3.RData")
  nb_tables_fold3 <- nb_tables
}
```

```
##  Calculate AUC
#' @description Calculate the area under the ROC curve (AUC) for a multiclass classification model.
#' @param outpredMlogit a matrix of predicted probabilities for each class label for each observation in the holdout
set.
#' @param holdo a data frame containing the true class labels for each observation in the holdout set.
#' @return the average AUC across all class labels.
calculateAUC <- function(outpred, holdo) {
  library(multiROC)
  # Convert for input to function multi_roc
  A = (holdo$period=="ancient_medieval_renaissance")
  B = (holdo$period=="baroque")
  C = (holdo$period=="classical")
  R = (holdo$period=="romantic")
  M = (holdo$period=="twentiethcentury_modern")

  ### Without weights
  inputROC = data.frame(as.numeric(A), as.numeric(B), as.numeric(C),
                        as.numeric(R),as.numeric(M),
                        outpred)
  names(inputROC) = c("A_true", "B_true", "C_true","R_true","M_true",
                      "A_pred_new", "B_pred_new", "C_pred_new", "R_pred_new", "M_pred_new")

  rocObj = multi_roc(inputROC)
  names(rocObj)
  # Calculate average AUC across all class labels
  return(rocObj$AUC)
}

## Calculate Weight
#' @description Calculate the weight for logit and rf to use
#' @param outpredMlogit a matrix of predicted probabilities for each class label for each observation in the holdout
set.
#' @param train a data frame containing the training value
#' @return the weight
calculate_weight <- function(train) {
  ntrain = nrow(train)
  nvec = c(table(train$period))
  wt = rep(1,ntrain)
  wt[train$period=="ancient_medieval_renaissance"] = (ntrain/5)/nvec[1]
  wt[train$period=="baroque"] = (ntrain/5)/nvec[2]
  wt[train$period=="classical"] = (ntrain/5)/nvec[3]
  wt[train$period=="romantic"] = (ntrain/5)/nvec[4]
  wt[train$period=="twentiethcentury_modern"] = (ntrain/5)/nvec[5]
  return (wt)
}
```

```
## Function to automate getting out-of-sample density values
# The bandwidth is obtained from bw.nrd0 for input to this function.
# Function could be more flexible to compute bw within, as an option.
#' @describe Gaussian kernel probability density function
#' @param xstar vector of values for computing density values (e.g., holdout)
#' @param xtrain vector of values for a variable in training set
#' @param bw bandwidth for Gaussian kernel, if 0 apply  bw.nrd0()
#' @param iprint flag for intermediate calculations, default=F
#' @return vector density variables at xstar based on Gaussian kernel
PDF = function(xstar, xtrain, bw=0, iprint=F) {
  nn = length(xstar); kpdf = rep(0,nn)
  # modified function to compute bandwidth here
  if(bw<=0) bw = bw.nrd0(xtrain)
  if(iprint) { cat(length(xtrain), bw, "\n") }
  for(i in 1:nn) { kpdf[i] = mean( dnorm(xstar[i]-xtrain, 0, bw) ) }
  kpdf
}

## Function to get product of pdfs and pmfs for class k over all features in the input holdout data, numeric and
categorical
#' @describe Product of PDFs and PMFs for a certain class k
#' @param holdout holdout dataframe
#' @param ck training dataframe where period == class k
#' @return product of pdfs and pmfs for class k over all features in the input holdout data
# Generic function for all numeric and categorical features
prod_features <- function(holdout, ck) {
  nfeatures = ncol(holdout)-1 # assume last column is period variable
  classk_prob = NULL
  for(i in 1:nfeatures) {
    if(is.numeric(holdout[,i])){ # numeric features calls helper function PDF
      # get density value of holdout based on curve for c_k, for feature i
      ckpdfi = PDF(holdout[,i],ck[,i],bw=0)
      # append a list of ith feature's densities (array with length = nrow(holdout)) to classk_prob
      classk_prob = append(classk_prob, list(ckpdfi))
    } else if (is.factor(holdout[,i])) { # categorical features computes pmfs
      # get pmf of class ck for feature i
      ckpmfi = c(table(ck[,i]))/nrow(ck)
      cipmfi_holdo <- ckpmfi[holdout[,i]]
      # append a list of ith feature's densities (array with length = nrow(holdout)) to classk_prob
      classk_prob = append(classk_prob, list(cipmfi_holdo))
    } else{
      print("Feature is not numeric or factor, check input holdout set")
    }
  }
  # get product of pdf over all features for class k by first turning the nested list into matrix of size
nfeatures*nrow(holdout)
  ckpdfpmfprod <- apply(matrix(unlist(classk_prob), nfeatures, nrow(holdout), byrow = TRUE), 2, prod)
  return(ckpdfpmfprod)
}
```