CS381 Proposal: Musical Vintage

Shlomi Helfgot, Ami Listokin, Zev Pinker

March 2025

1 Introduction

Only a fine-tuned ear can hear a song and know when it was written. But can a fine-tuned model do the same? Although genre lines often follow distinct tonal and rhythmic patterns that are simple to distinguish, fans of old music genres continue to compose in old styles today. Our goal is to train a model that predicts, with significant certainty, the decade in which a piece of music was composed based on how it sounds. Our data comes with raw track waveforms and was tagged by the data's purveyor using Librosa (518 features per track.)

We have experimented with a number of approaches distinguished by the model architecture and data cleaning strategy used in each. In this paper we will describe four of our model architectures. They are the single layer Multiclass Perceptron, Support Vector Machine with an RBF Kernel, the Kernel Ridge Regression Model, and the standard Neural Network. As we will discuss, much consideration was put into the proper organization of data and task formulations to correctly pair them with each model architecture. Due to the high-dimensionality of the data and the lack of data richness in most years (due to the lack of open source music created in those years), many of our models struggled to achieve an accuracy better than a random guess, or above 50%. Our best model achieved a significant accuracy of 75%.

2 Our Data

2.1 Summary

The Free Music Archive contains 106,574 tracks from 16,341 artists. Each track includes a .mp3 file. The data set also provides metadata on each track that includes the year published. It also provides features that characterize the musical content of the tracks, which we use to build lighter-weight predictive models that do not depend on the .mp3 files themselves.

2.2 Source: git source

2.3 Extracted Features: features were extracted from wave forms by Librosa

For each feature, the dataset provides the following statistical measures across seven relevant ranges: **Kurtosis, Max, Mean, Median, Min, Skew, and Std**.

- **Chroma_cens**: 12 ranges
- **Chroma_cqt**: 12 ranges
- **Chroma_stft**: 12 ranges
- **MFCC**: 20 ranges
- **RMSE**: 1 range
- **Spectral Bandwidth**: 1 range
- **Spectral Centroid**: 1 range
- **Spectral Contrast**: 7 ranges
- **Spectral Rolloff**: 1 range
- **Tonnetz**: 6 ranges
- **Zero Crossing Rate (ZCR)**: 1 range

This yields a total of **74** $\times$ **7**, i.e. **518** features extracted from audio, per track.

2.4 File structure

Features are found in the **fma_metadata** folder, in two files: **tracks.csv** for the track information, including year, recording engineer, studio, label, etc.; **features.csv** for the audio features extracted by Librosa. **features.csv** also contains a foreign key for the track id number.

3 Data Preprocessing Pipeline

As a first step, we preprocessed the data with **Pandas**. The main components of the preprocessing pipeline were a comprehensive cleaning of the data, as well as calling a join on the two tables, **features.csv** and **tracks.csv**.

features.csv requires the following steps for data cleaning:

- Multiline, hierarchical header (which for example, has "chroma_cens" as a general heading on line 1, for the 84 chroma_cens features on line 2) must be flattened into a one line header.

- The “track_id” foreign key in the file is to be used as the index column for the table.

tracks.csv is somewhat sparse; information is duplicated and/or scattered in various place. Hence, it requires the following steps for data cleaning:

- Check for the date created in ”date_released” column, ”album.2” column, or ”track.3” column - the proper datetime might be in any of these.
- Store in a new column, ’*tracks_df[“year”]*’.
- In the classification by decade case the following procedure is done: create a new column, ”decade”. This will enable multiclass classification while ensuring that the amount of classes remains low enough, for accuracy and efficiency.
- Use `pandas.dropna` to drop entries where the year is unknown.

We then *merged* the tables using the foreign key in **features.csv** (‘*track_id*’), in order to align the labels to the data.

4 Procedure and construction of models

4.1 Multiclass Perceptron: the non-linearity of data

Our first step was to consider whether our data was linearly separable or not. We evaluated the classifier’s performance for various values of `max_iter`, reporting the training and test accuracy, training time, and classification report.

Maximum Iterations	**Train Accuracy**	**Test Accuracy**	**Training Time (s)**
5	0.353581	0.346994	1.882972
10	0.335886	0.326607	2.887269
15	0.345452	0.336958	4.302975
20	0.381913	0.376059	3.652124
25	0.381913	0.376059	3.597393
30	0.381913	0.376059	4.502802
35	0.381913	0.376059	3.630650
40	0.381913	0.376059	3.575303
60	0.381913	0.376059	4.402979
100	0.381913	0.376059	3.507548

Table 1: Summary of Perceptron performance for various `max_iter` values

Table 2: Performance Metrics by max_iter

max_iter	**Training Accuracy**	**Test Accuracy**	**Training Time (s)**
5	0.3536	0.3470	1.88
10	0.3359	0.3266	2.89
15	0.3455	0.3370	4.30
20	0.3819	0.3761	3.65
25	0.3819	0.3761	3.60
30	0.3819	0.3761	4.50
35	0.3819	0.3761	3.63
40	0.3819	0.3761	3.58
60	0.3819	0.3761	4.40
100	0.3819	0.3761	3.51

Table 3: Precision Values by Class for Different max_iter

max_iter	**Precision by Class**					
	1940	**1960**	**1980**	**1990**	**2000**	**2010**
5	0.00	0.00	0.01	0.03	0.37	0.63
10	0.00	0.00	0.01	0.03	0.36	0.63
15	0.00	0.00	0.01	0.03	0.34	0.62
20	0.00	0.00	0.01	0.03	0.35	0.63
25	0.00	0.00	0.01	0.03	0.35	0.63
30	0.00	0.00	0.01	0.03	0.35	0.63
35	0.00	0.00	0.01	0.03	0.35	0.63
40	0.00	0.00	0.01	0.03	0.35	0.63
60	0.00	0.00	0.01	0.03	0.35	0.63
100	0.00	0.00	0.01	0.03	0.35	0.63

Table 4: Recall Values by Class for Different max_iter

max_iter	**Recall by Class**					
	1940	**1960**	**1980**	**1990**	**2000**	**2010**
5	0.00	0.00	0.20	0.12	0.30	0.39
10	0.00	0.00	0.20	0.24	0.23	0.39
15	1.00	0.00	0.18	0.10	0.16	0.46
20	0.50	0.00	0.18	0.12	0.21	0.49
25	0.50	0.00	0.18	0.12	0.21	0.49
30	0.50	0.00	0.18	0.12	0.21	0.49
35	0.50	0.00	0.18	0.12	0.21	0.49
40	0.50	0.00	0.18	0.12	0.21	0.49
60	0.50	0.00	0.18	0.12	0.21	0.49
100	0.50	0.00	0.18	0.12	0.21	0.49

Table 5: F1-Score Values by Class for Different max_iter

max_iter	**F1-Score by Class**					
	1940	**1960**	**1980**	**1990**	**2000**	**2010**
5	0.00	0.00	0.02	0.05	0.33	0.48
10	0.00	0.00	0.02	0.05	0.28	0.48
15	0.00	0.00	0.02	0.05	0.21	0.53
20	0.01	0.00	0.02	0.05	0.27	0.55
25	0.01	0.00	0.02	0.05	0.27	0.55
30	0.01	0.00	0.02	0.05	0.27	0.55
35	0.01	0.00	0.02	0.05	0.27	0.55
40	0.01	0.00	0.02	0.05	0.27	0.55
60	0.01	0.00	0.02	0.05	0.27	0.55
100	0.01	0.00	0.02	0.05	0.27	0.55

Table 6: Average Metrics for Different max_iter Values

max_iter	**Accuracy**	**Macro Avg**	**Weighted Avg**	**Convergence**
5	0.3470	0.15	0.41	No
10	0.3266	0.14	0.39	No
15	0.3370	0.14	0.40	Yes
20	0.3761	0.15	0.43	Yes
25	0.3761	0.15	0.43	Yes
30	0.3761	0.15	0.43	Yes
35	0.3761	0.15	0.43	Yes
40	0.3761	0.15	0.43	Yes
60	0.3761	0.15	0.43	Yes
100	0.3761	0.15	0.43	Yes

Table 7: Dataset Class Distribution (Support)

Class	**Support**
1940	2
1960	4
1980	56
1990	250
2000	3462
2010	5791
Total	9565

We trained a classifier with different values for the maximum number of iterations, max_iter, ranging from 5 to 60. For max_iter = 5, the training accuracy was 35.36%, and the test accuracy was 34.70%. Training took approximately 1.88 seconds. Despite low overall accuracy, some performance could be observed on the later classes (e.g., for the class labeled 2010, precision was 0.63, recall was 0.39, and the F1-score was 0.48). However, earlier classes had near-zero scores; between 5 and 10/15 the model did not improve much.

At `max_iter` = 20, both training and test accuracy saw noticeable improvement, reaching 38.19% and 37.61%, respectively. Training time dropped slightly to 3.65 seconds. Precision and recall improved modestly across most classes, with class 2010 reaching a recall of 0.49.

Interestingly, increasing `max_iter` beyond 20 (up to 60) resulted in no further improvement in either training or test accuracy, both of which plateaued at 38.19 percent and 37.61 percent, respectively. Training time fluctuated between 3.58 and 4.50 seconds, but the classification reports remained largely identical across these higher values. This implies that the model had likely converged around `max_iter` = 20.

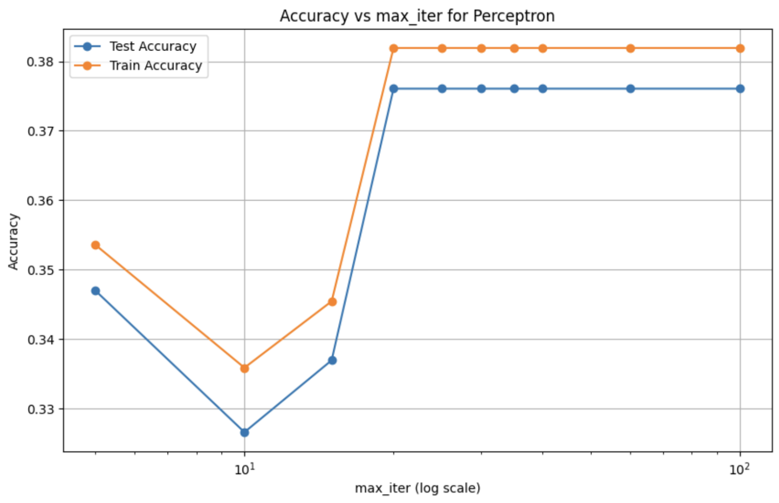

Figure 1: Perceptron Accuracy

In any event, owing to the dismal train and test scores, it is fairly safe to say that a different model is needed, in order to divide between these classes.

As a further visual aid to demonstrate the non-linearity of the dataset, we employed a PCA decomposition to find the 2 most important features, project the feature space onto a 2-dimensional space, and plotted the perceptron decision boundary and the data. It is clear that the data is not linearly separable:

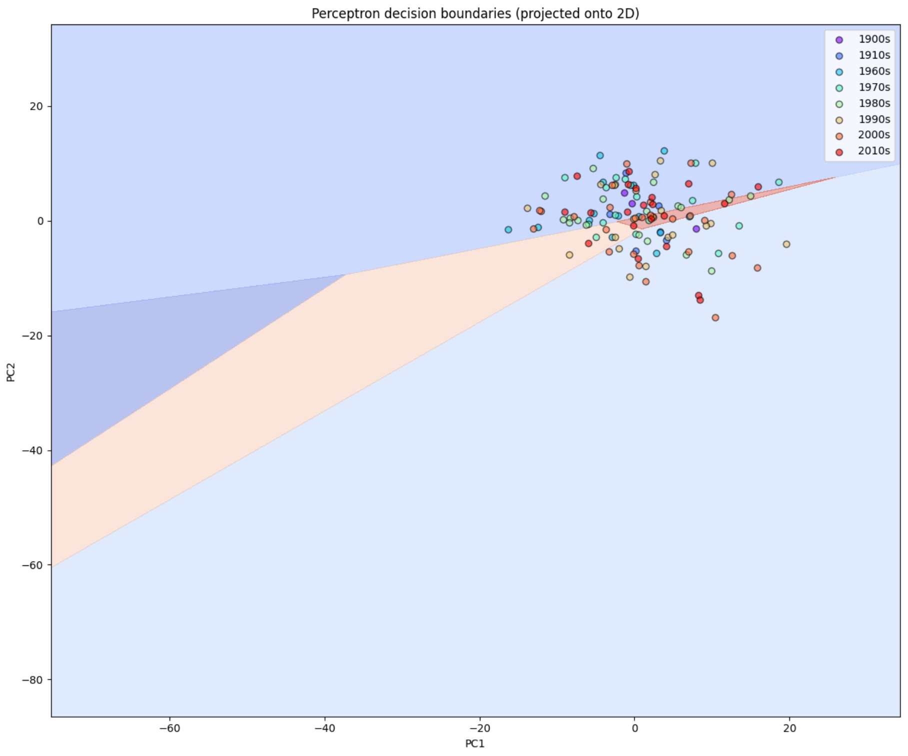

Figure 2: 2D projection of Perceptron Decision Boundaries

As such, we moved to developing non-linear approaches to finding a better decision boundary.

4.2 Kernel Ridge Regression: A more advanced model predicting values rather than classes

After determining the non-linearity of the data, we assessed the performance of various non-linear machine learning models for predicting song dates. We took two approaches to this task: predicting song release dates through multiclass classification and regression. Here, we discuss regression models.

4.2.1 Data Preprocessing Pipeline

We decided to construct a model solely based on songs from 2009 to 2013; this range was chosen due to the ample available of samples and relatively even distribution among those years. Expanding the regression data to encompass a larger number of years risks allowing some years to dominate others and gives small numbers of songs a disproportionate amount of influence in the model for years with fewer observations. However, restricting the regression data to songs between 2009 and 2013 also makes life more difficult for our regression. The regression must pick out trends within songs only within a five year period, a difficult task for a ridge regression model.

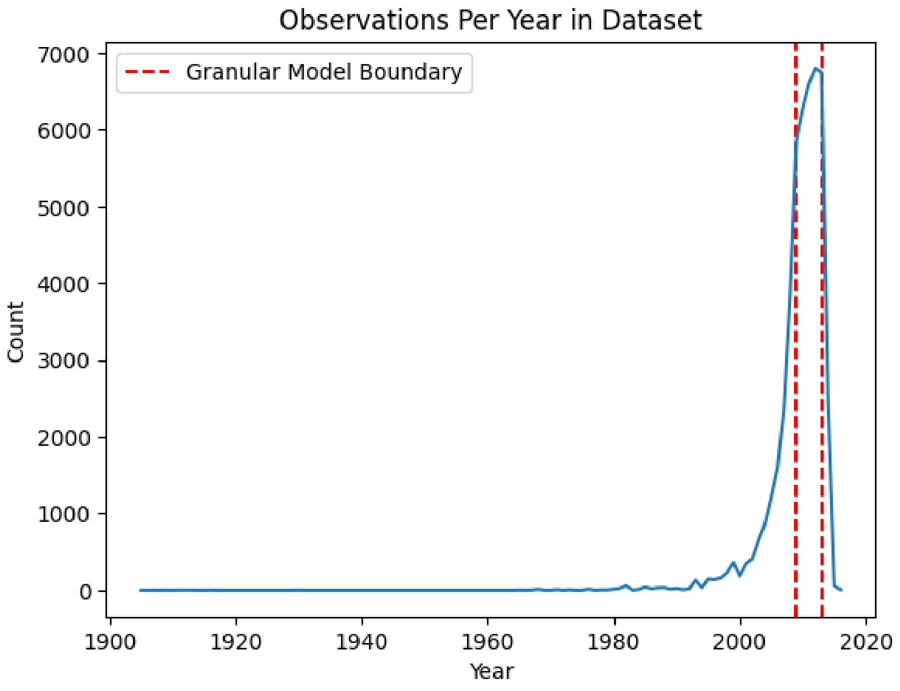

Figure 3: Observations per year with data for granular model marked

Since a regression predicts values rather than "buckets", we trained the regression to predict values over a less discrete, pseudo-continuous group of categories: Instead of using the year as a ground truth, we decided to use both year and month. This led to yet more data-cleaning difficulties, owing to the fact that observations per month varied wildly in the dataset. Hence, we select an equivalent amount of observations from every (month, year) combination. (With a hypothetical ideal dataset, we would not balance the data in this way. A hypothetical model trained on this dataset would accurately predict that more songs are published during more popular months for releasing an album. However, we had no assurance that our dataset accurately reflected these trends.)

4.2.2 Model Development

We began by standardizing our data using a `Standard Scaler`, and adding extra musical features described in more depth in section 4.3.2. Next, we used

SciKit's `PCA` toolkit to reduce dimensionality of data, since balancing the number of observations per month caused a significant loss in number of data points. Thus we cut down the number of features to keep our feature number and data point quantity balanced. We then mapped the reconstruction error for various component numbers in order to decide a reasonable number of components to preserve:

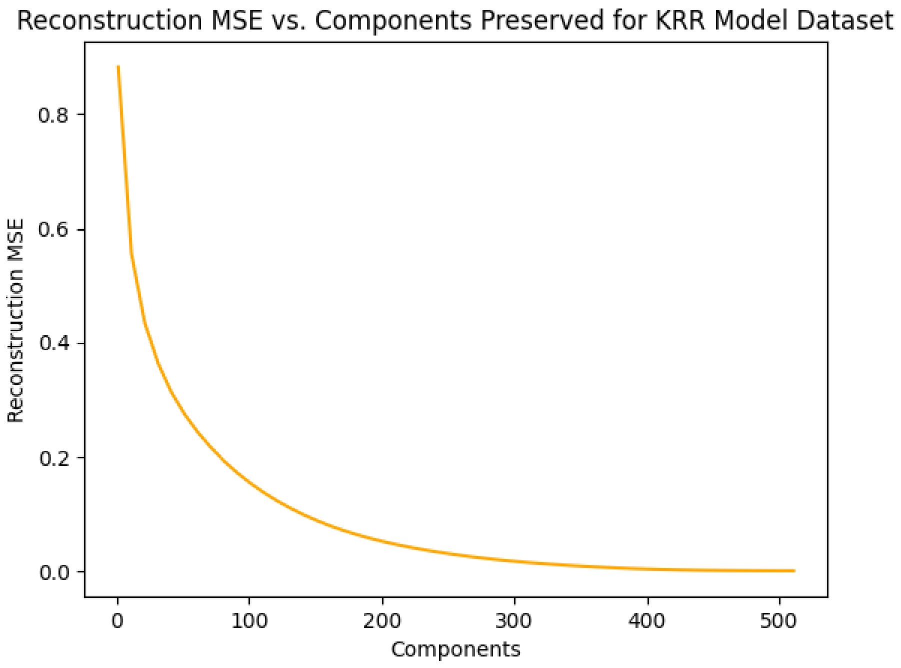

Figure 4: Reconstruction error for different numbers of preserved components for a month-balanced, feature-engineered, scaled dataset

We proceeded, preserving 300 components. Next, we tuned hyperparameters using `scikit`'s `ParamGrid` module. The table below summarizes the best results from the test.

α	γ	kernel	degree	Train MSE	Test MSE	Fit Time (s)
1	0.001	rbf	NA	222.39	284.22	12.68
0.1	0.001	rbf	NA	112.72	293.21	10.66
0.1	0.00001	rbf	NA	280.34	288.59	12.72
1.0	0.00001	polynomial	5	280.44	288.70	13.09
1.0	0.00001	polynomial	4	281.04	284.25	11.15

Our optimal mean squared error, 284.22, corresponds to an average distance from the correct result of $\approx \sqrt{284.22} = 16.86$ months.

Consider the distribution of guesses from two of our top models, visualized in figure 5 below. Each discrete integral number on the X-axis corresponds to a month-year combination between January 2009 and December 2013. (Negative values are meaningless outputs generated by the model due to the continuous nature of prediction.) For example, ”30” represents June 2011.

The figure demonstrates that lowering the γ variable leads to the model's guesses clumping around the center of the data labels. This suggests that models with low γ have not found a successful way to make informed decisions about the year and month a song is published. Instead, they repetitively guess values in the middle of the dataset to minimize its average distance from the correct values. Interestingly, our best model does not simply cluster its guesses around the center of the dataset. It achieves a marginally better test MSE by more evenly spreading its guesses. This suggests that the data do hold information about song year–just this model isn't strong or complex enough to fully access it. Our average distance from the correct result of ≈ 16.86 is not satisfactory. Always guessing the median month in the data yields an average distance of ≈ 15 (since there are 60 elements in the dataset). Our best model does worse than this.

Unfortunately, results are not much better for a wider set of data or for only predicting year, rather than both year and month. We applied a similar model but to song data from 2004 to 2014. We labeled the data points only with year rather than with both year and month. We balanced the dataset so that each year had the same number of observations. Performing KRR after a 300-component PCA, we received the following top models.

α	γ	kernel	degree	Train MSE	Test MSE	Fit Time (s)
1	0.001	rbf	NA	222.39	284.22	12.62
0.1	0.001	rbf	NA	112.72	293.21	10.66
0.1	0.00001	rbf	NA	280.34	288.59	12.72

The inconclusive results for using KRR to make granular time predictions on songs within a small time span suggested to us two options. One was to use a multi-layered model to make predictions within small time periods. We explore this question in section 4.4. The other was to use non-neural methods over wider time periods and for predicting less granular values. We explore this option in section 4.3.

4.3 SVM with various kernels, for non-linearity

In order to account for the non-linearity of the dataset, as demonstrated in the perceptron section, we attempt to find a better decision boundary using a **support vector machine** model.

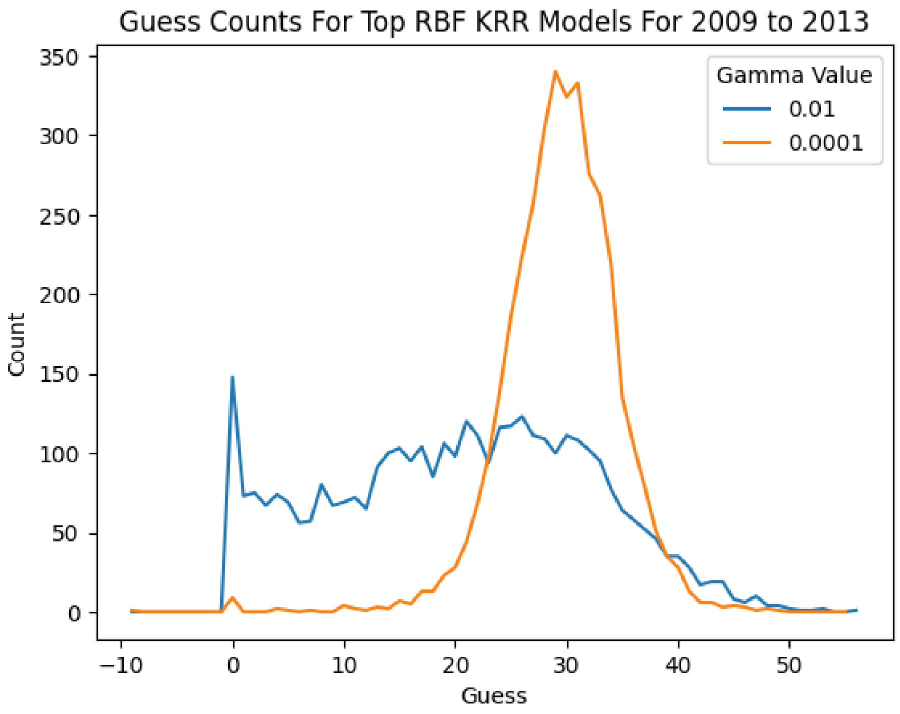

Figure 5: Distribution of predictions for two top models

4.3.1 Constraints

As noted in the introduction, the dataset exhibits high dimensionality and is extremely sparse – it is skewed towards the two classes C_{2010} and C_{2000}. Hence, in order to ensure that models run within the 24hr. technical limit prescribed by Google Colab Pro+[1], our SVM model was constrained using feature selection (k-highest), combining classes with sparse input data into more general classes, and a limit on maximum sample amount per class, in accordance with the following:

1. The 1950s/1960s were grouped together; the 1970s/1980s; all decade before 1950 were grouped together.

2. Classes with fewer than 2 samples were then removed.

[1]Per Google, "With compute units, your actively running notebook will continue running for up to 24 hours, even if you close your browser." See Colab signup page.

Decade	Original Count
2010	28,955
2000	17,312
1990	1,251
1980	246
1970	34
1960	17
1910	6
1900	3
1930	1

Table 8: Original class distribution. Decade 1930 (fewer than 2 samples) would have been marked for removal if not for grouping.

Decade	After Grouping
2010	28,955
2000	17,312
1990	1,251
1980	280
1960	17
1940	10

Table 9: Class distribution after grouping.

3. Maximum samples per class capped at 2,000, in an effort to allow for more efficient model construction.

Decade	Balanced Count
2000	2,000
2010	2,000
1990	1,251
1980	280
1960	17
1940	10

Table 10: Balanced class distribution (max 2000 samples per group).

4. Feature selection (k-highest) subpipeline implemented; this was used in later experiments:
    - Implemented `scikit`'s `StandardScaler` to scale input (X) values.
    - Set k-feature test amounts as $[60, 100, 150, 200]$, out of 518 features total. (60, as opposed to 50, in order to test 'best_k-50' and 'best_k+50' features once 'best_k' is found.)

- Use `scikit`'s `SelectKBest` with ANOVA (i.e. `f_classif` in SciKit; cf. documentation) to create training set and testing set.
- Create a simple one-vs-one `SVC` model with SciKit, with radial basis kernel (RBF), a gamma which scales with each step (instead of remaining static), uses 1 / (n_features * X.var()) as value of gamma, and with a regularization parameter $C = 1$.

4.3.2 Model Development

Beginning with a simple one-vs-one SVM model trained on various kernels, we progressively tweaked the implementation in an attempt to obtain a better model.

Baseline: Experiment 1 (simple SVM)

We began by splitting the dataset into training and test sets. Initial feature selection was then performed to identify the most predictive subset of features. Feature selection was tested across several values of k. This concluded successfully, after 43s (avg of 12.97s per iteration.)

The best initial number of features was found to be $k = 100$.

We then utilized a `scikit Pipeline` to perform a grid search for SVM on the following parameter grid, representing 16 possible combinations of hyperparameters:

- $C \in \{0.1, 1.0, 10.0, 100.0\}$
- $\gamma \in \{\text{scale}, 0.01, 0.1, 1.0\}$
- kernel $\in$ {rbf}
- class weight: `balanced`

The grid search completed successfully with all 16 parameter combinations tested. Below is a subset of the results:

C	γ	k	Train Score	Test Score	Fit Time (s)
10.0	0.01	100	0.9204	0.5513	17.32
10.0	scale	100	0.9204	0.5513	19.32
1.0	0.01	100	0.6756	0.5405	16.43
1.0	scale	100	0.6756	0.5405	18.70
100.0	scale	100	0.9987	0.5396	21.27

The best parameters found were:

- `feature_selection__k`: 100
- `svm__C`: 10.0
- `svm__gamma`: scale

- `svm__class_weight`: balanced

This configuration of hyperparameters achieved a best test score of **0.5513**.

The following is the classification report for **Experiment 1**. Note that the **f1-score** is a metric used for unbalanced datasets (among others.) [Its use was recommended to us by a review of the `sklearn` documentation, specifically the article here.] The F1-score is calculated as the harmonic mean of precision and recall: F1 = 2 * (precision * recall) / (precision + recall, where precision is the ability of the model not to label a negative sample as positive (a ratio), and recall is the ability of the model to find all the positive samples (ratio of true positives to all actual positives). Range is 0-1 (higher is better.) **Support**, on the other hand is just the number of actual occurrences of the class in the test dataset; i.e. it is the amount of test examples per class.

Class	**Precision**	**Recall**	**F1-score**	**Support**
1940	0.00	0.00	0.00	2
1960	0.00	0.00	0.00	4
1980	0.00	0.00	0.00	56
1990	1.00	0.01	0.02	250
2000	0.36	1.00	0.53	400
2010	1.00	0.01	0.02	400
Accuracy			0.37	1112
Macro avg	0.39	0.17	0.09	1112
Weighted avg	0.71	0.37	0.20	1112

And the following is the confusion matrix for the SVM model developed in **Experiment 1**.

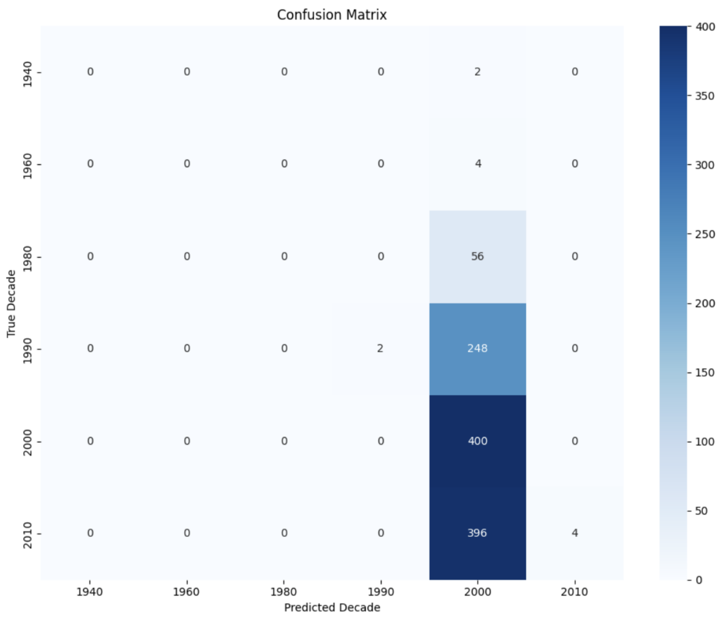

Figure 6: Confusion Matrix for SVM Experiment 1

As the classification report shows, even though the model achieved high training accuracy for some configurations, the performance on the test set remained weak. This is likely due to class imbalance and overlapping feature distributions.

Experiment 2: Testing over various kernels

The second experiment conducted was to test over various kernels, i.e. not only the RBF kernel originally selected. We added a polynomial kernel and sigmoid kernel using `scikit`'s built-in kernel function. [Linear kernel was excluded because we had already found that the data were not linearly separable.]:

'svm__kernel': ['rbf', 'poly', 'sigmoid']

When combined with the options enumerated in Experiment 1, this yielded 48 hyperparameter combinations to test over. Seeing as Experiment 1 yielded fairly abysmal results, we also decided to run the SVM on the optimal k-features as well as k+50 and k-50 features. Hence, testing commenced on an extended grid search of 108 parameter combinations. The best result was observed with:

- `feature_selection__k`: 50
- `svm__C`: 10.0
- `svm__gamma`: 0.1

- `svm__class_weight`: balanced
- `svm__kernel`: rbf

This configuration achieved a test accuracy of **0.5504**.

Below is a summary of some of the top-performant parameter sets from the grid search:

C	γ	k	Train Score	Test Score	Fit Time (s)
100.0	0.1	50	0.9991	0.5504	17.94
10.0	0.1	50	0.9991	0.5504	16.56
10.0	0.01	150	0.9933	0.5459	27.07
10.0	0.01	100	0.9370	0.5450	18.19
10.0	scale	100	0.9370	0.5450	18.53

Upon selecting the best model, we made predictions on the test set. The **classification report** follows:

Decade	Precision	Recall	F1-score	Support
1940	0.00	1.00	0.01	2
1960	0.00	0.25	0.01	4
1980	0.02	0.02	0.02	56
1990	0.23	0.10	0.14	250
2000	0.36	0.02	0.04	400
2010	0.65	0.06	0.10	400
Accuracy			0.05	1112
Macro avg	0.21	0.24	0.05	1112
Weighted avg	0.42	0.05	0.08	1112

The following is the confusion matrix for the SVM model developed in **Experiment 2**:

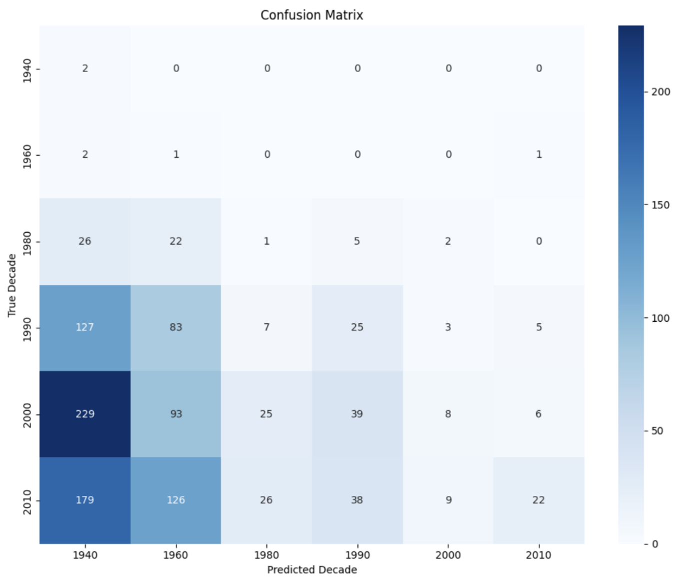

Figure 7: Confusion matrix for SVM Experiment 2

Exactly as before, as the classification report shows, even though the model achieved high training accuracy for some configurations, the performance on the test set remained weak. Again, we theorize that this is likely due to class imbalance and overlapping feature distributions — underscoring the fact that SVM may not be the right choice here.

Experiment 3: Adding in some feature engineering

The third experiment conducted was to add in feature engineering; aggregate features added, as well as ratio features, which we hypothesized — based on research about chroma and mfccs – may capture tonal or timbre relationships.

We used the following 13 additional features:

- 9 features, representing the ratio of a subset of chroma values to a subset of mfcc values.
- 2 features, one for the mean of the chroma values and one for their standard deviation.
- 2 features, one for the mean of the mfcc values and one for their standard deviation.

A separate grid search was conducted using the same 144 combinations of hyperparameters as utilized in Experiment 2. This search, as with its predecessor, again explored various values for C, γ, and the number of features k selected by feature selection.

The best configuration found this time around was:

- `feature_selection__k`: 50
- `svm__C`: 100.0
- `svm__gamma`: 0.1
- `svm__class_weight`: balanced
- `svm__kernel`: rbf

This yielded a best test accuracy of **0.5764**, slightly outperforming the previous configuration; these fluctuations are to be expected in a stochastic system utilizing randomness, the likes of which we developed.

The top-performant parameter combinations are listed below:

C	γ	k	Train Score	Test Score	Fit Time (s)
100.0	0.1	50	0.9993	0.5764	15.86
10.0	0.1	50	0.9993	0.5755	16.02
10.0	scale	150	0.9642	0.5630	21.10
10.0	0.01	100	0.9312	0.5621	18.92
10.0	scale	100	0.9312	0.5621	19.09

Once the best configuration was identified, it was evaluated on the test set. The classification report from this test is presented below:

Decade	Precision	Recall	F1-score	Support
1940	0.00	0.50	0.00	2
1960	0.01	0.50	0.01	4
1980	0.03	0.04	0.03	56
1990	0.21	0.05	0.08	250
2000	0.43	0.01	0.03	400
2010	0.47	0.02	0.04	400
Accuracy			0.03	1112
Macro avg	0.19	0.19	0.03	1112
Weighted avg	0.37	0.03	0.05	1112

We note the slight improvement in test accuracy. However, the classifier continued to struggle with generalization, particularly for the larger and more recent decades. It reinforced our hypothesis that the features may not sufficiently discriminate between all classes, and hence alternative modeling approaches may be required.

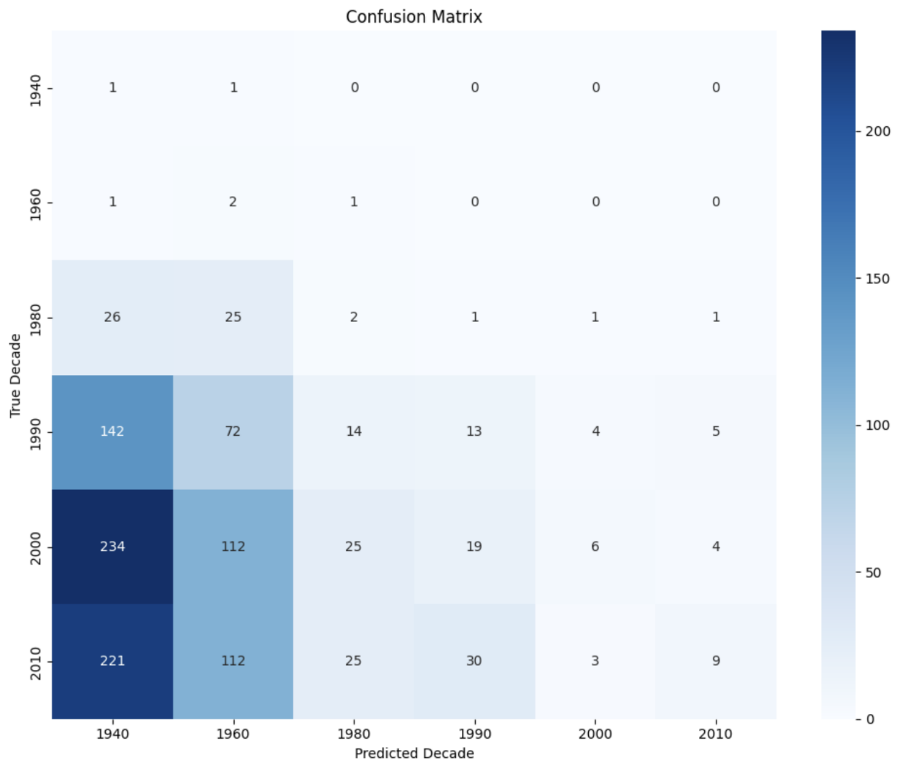

Figure 8: Confusion Matrix for SVM Experiment 3

Experiment 4: Adding an Ensemble Vote

As a last experiment, we further implemented — on top of the feature engineering and expanded hyperparameter search — an **ensemble method** - that is, polling different experts (i.e. different kernels) in aggregate. An ensemble of classifiers was created using the following individual models:

- SVM with RBF kernel
- SVM with polynomial kernel
- SVM with sigmoid kernel
- SVM with linear kernel

Each model was trained independently, and their individual accuracies on the test set were recorded:

Model	Accuracy (on test set)
SVM (RBF)	0.5468
SVM (Poly)	0.5189
SVM (Sigmoid)	0.2230
SVM (Linear)	0.4577
Ensemble	**0.5854**

The ensemble, which aggregated predictions from the base models, achieved the highest accuracy of **0.5854**, outperforming all individual SVMs. The ensemble strategy seems to have successfully leveraged the complementary strengths of the base classifiers.

The classification report for the ensemble is shown below:

Decade	Precision	Recall	F1-score	Support
1940	0.00	0.00	0.00	2
1960	0.00	0.00	0.00	4
1980	0.00	0.00	0.00	56
1990	0.67	0.06	0.10	250
2000	0.54	0.82	0.65	400
2010	0.65	0.77	0.70	400
Accuracy			0.59	1112
Macro avg	0.31	0.27	0.24	1112
Weighted avg	0.57	0.59	0.51	1112

The ensemble improved overall accuracy; however, performance remained skewed toward the classes with more samples (2000s and 2010s). Near-zero precision and recall persisted for earlier decades.

The confusion matrix follows:

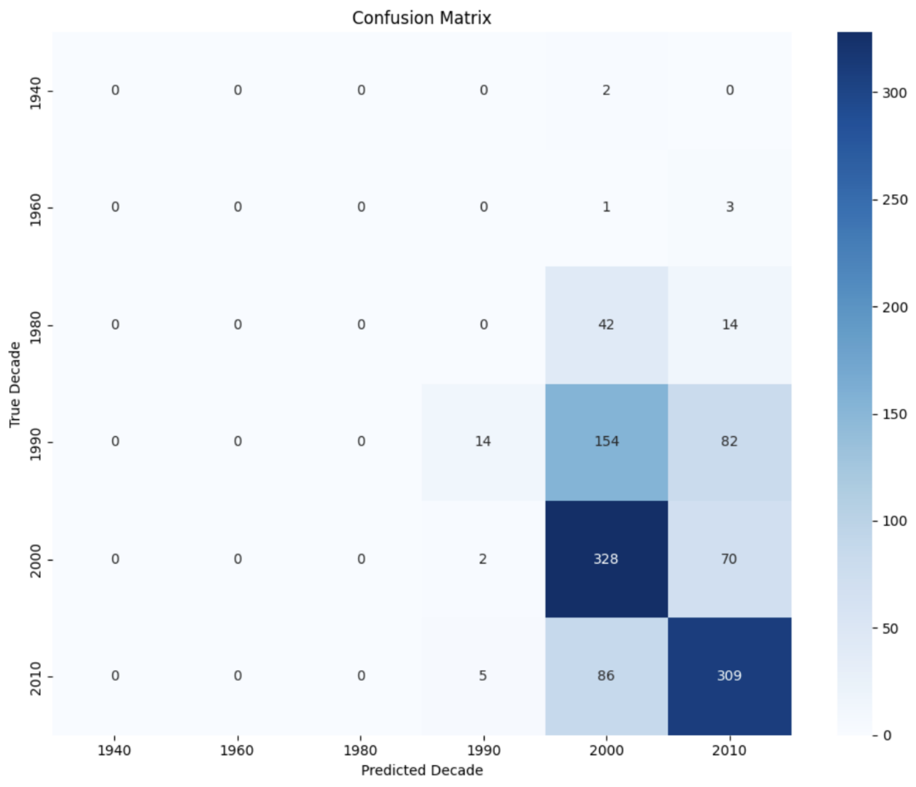

Figure 9: Confusion Matrix for SVM Experiment 4

Experiment 5: Limiting Data to 2009-2013

A further attempt we made with the SVM architecture was to copy what we did with the Kernel-Ridge-Regression and limit the data to 2009-2013. A new data preprocessing pipeline using years rather than decades was constructed for this purpose. The feature engineering and parameter grid search described above were implemented here as well. The various parameters in the chart below are to be interpreted as above:

Kernel	**C**	**Gamma**	**Test Accuracy**
rbf	0.1	scale	0.4880
rbf	0.1	0.01	0.4875
rbf	0.1	0.1	0.3750
rbf	0.1	1.0	0.2110
rbf	1.0	scale	0.6510
rbf	1.0	0.01	**0.6640**
rbf	1.0	0.1	0.5240
rbf	1.0	1.0	0.2055
rbf	10.0	scale	0.6060
rbf	10.0	0.01	0.6460
rbf	10.0	0.1	0.5130
rbf	10.0	1.0	0.2045
rbf	100.0	scale	0.5320
rbf	100.0	0.01	0.5315
rbf	100.0	0.1	0.5145
rbf	100.0	1.0	0.2045
poly	0.1	scale	0.3205
poly	1.0	scale	0.4895
poly	10.0	scale	0.5235
poly	100.0	scale	0.5095
sigmoid	0.1	scale	0.5490
sigmoid	1.0	scale	0.4095
sigmoid	10.0	scale	0.3770
sigmoid	100.0	scale	0.3795

Table 12: SVM performance for various kernels and hyperparameters (2009–2013 year classification)

Year	**Precision**	**Recall**	**F1-score**	**Support**
2009	0.74	0.68	0.71	400
2010	0.58	0.52	0.55	400
2011	0.60	0.52	0.56	400
2012	0.69	0.69	0.69	400
2013	0.68	0.91	0.78	400
Accuracy	0.6640 (on 2000 samples)			
Macro Avg	0.66	0.66	0.66	2000
Weighted Avg	0.66	0.66	0.66	2000

Table 13: Classification report for SVM model (Kernel: RBF, $C = 1.0$, $\gamma = 0.01$) on 2009–2013 year classification

The confusion matrix detailing these results follows:

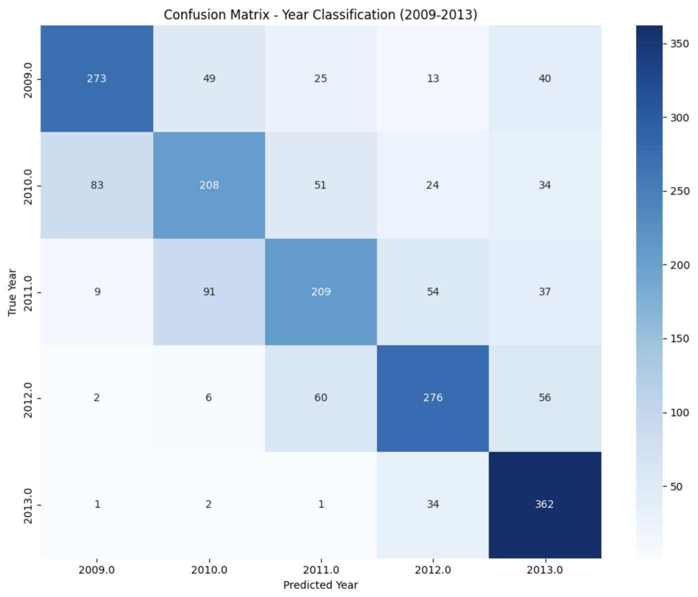

Figure 10: Confusion matrix for SVM Experiment 5

As is abundantly clear, using a dataset far less sparse enables us to raise our accuracy to around 2/3.

4.4 Deep Neural Network on Features

What follows is a description of the data, model, and training regimen.

1. **Data Selection:** For the Neural Network experiment we did not group the data by decade. Initial experiments with a bare-bones neural network showed that there was enough data between 2009 and 2013 to get a better than random accuracy for a granularity of one year. Therefore this model predicts the exact year that a song was created. Due to data scarcity outside of this five year range, we could not extend beyond it. This is our year distribution over the data set of 32,290 total samples.

Year	**Count**
2009	5864
2010	6275
2011	6607
2012	6798
2013	6746

Table 14: Distribution of songs by year

2. **Preprocessing:** Here we apply the feature-engineering function described above (in the SVM section) in order to increase the dimensionality of the data based on relationships between chroma and mfccs. This brought the feature count from 518 to 531. We furthermore split the data into the training set and testing set in a $80 : 20$ ratio. The labels $(2009, 2010, 2011, 2012, 2013)$ were mapped to $(1, 2, 3, 4, 5)$. All of the data is transferred from an ndarray to a Pytorch tensor to ensure compatibility with the torch model.

3. **Hyperparameters**: We established an efficient documentation pipeline to record results for different sets of hyperparameters. As we tuned the hyperparameters we saved the training and test metrics as well as the confusion matrix in order to facilitate easy comparison. The parameters that we chose yielded the best test accuracy and were a result of this trial, error, and comparison process.

Hyperparameter	**Value**
Input dimension	532
Hidden layer sizes	128, 64
Dropout rate	0.30
Learning rate	0.01
Epochs	100
Batch size	64
Scheduler step size	1
Scheduler gamma	0.9
Early stopping patience	20

Table 15: Neural network training hyperparameters

4. **Neural Network:** The network, *SongClassifierADV*, subclasses PyTorch's `nn.Module` base class and implements a fully connected feedforward neural network using `nn.Sequential`. It includes two hidden layers and one output layer. The architecture is designed with regularization and stability in mind. Each hidden layer consists of three key components in addition to a linear transformation: `BatchNorm1d`, `ReLU`, and `Dropout`.

   - **Batch Normalization (`BatchNorm1d`):** This layer normalizes the input features across the batch to have zero mean and unit variance. It helps to mitigate internal covariate shift, speeds up training, and often improves generalization. In this model, batch normalization is applied immediately after each linear transformation in the hidden layers.
   - **ReLU Activation (`ReLU`):** The Rectified Linear Unit introduces non-linearity by applying the function $f(x) = \max(0, x)$ element-wise. This activation function helps the network learn complex functions and relationships in the data while avoiding issues like vanishing

gradients (common in activations functions used in older models, e.g. the sigmoid function or tanh function).

- **Dropout:** Dropout is a regularization technique that randomly sets a fraction of input units to zero during training (controlled by the `dropout` parameter). This prevents the model from becoming overly reliant on particular nodes, thereby reducing overfitting. In our architecture, dropout follows the ReLU activation in each hidden layer.
- **Structure Summary:** The first hidden layer maps the input of dimension `input_dim` to `hidden1` units; the second maps `hidden1` to `hidden2`. Each is followed by batch normalization, ReLU, and dropout. The final output layer projects from `hidden2` to `num_classes` and does not include an activation function, as this is handled by the loss function (in our case, for example, `CrossEntropyLoss`, which applies the `softmax` function internally).

5. **Optimization Architecture:** Since this is a multiclass classification problem we use the CrossEntropyLoss as our loss function. We also utilize an advanced Pytorch optimization algorithm called `Adam`. `Adam` is an extension of SGD (stochastic gradient descent) which includes an implementation of a momentum factor, which helps accelerate gradients in the right direction by maintaining an exponentially decaying average of past gradients, as well as `RMSProp`. `RMSProp` adapts the learning rate for each parameter individually using a moving average of the squared gradients. We also use a scheduler to implement learning rate decay throughout optimization. Lastly, we implement an early stopping mechanism to stop the training loop early, if no improvements are discerned in the model update for an amount of epochs equivalent to the *patience* parameter (in our case, $patience = 20$). Having completed training and evaluation for a specified number of epochs, we then save the best model.

   From the plot of outputs reproduced below (Fig. 9), it is evident that our model does not over-fit; the gap between the training and validation has closed by the time we reach the 40th epoch or so:

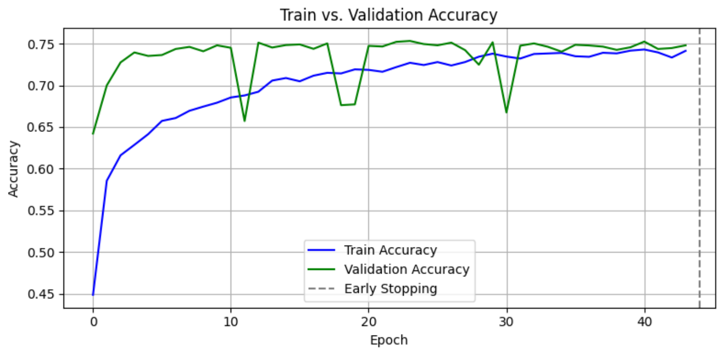

Figure 11: Training and Validation Loss Superimposed

6. **Results:** The best test accuracy was 75.3%; this result, far better than the other models we developed, we consider quite significant. The confusion matrix for this model, reproduced below, provides insight as to the subset of data for which the model is most successful vis a vis predictions. Interestingly the model seems to err on the side of newer, predicting song creation date one year after the true label in many cases. Still, given the relative simplicity of our data (note that 518 features is a relatively high compression from what would have been a large spectrograph), and the relatively high granularity, we are pleased with our current accuracy. This model acts as a proof of concept that would become more generally applicable upon the availability of more data for other years. However, it is not clear to us if, given a more dense and balanced data set, we could achieve similar accuracy at the same level of single year granularity.

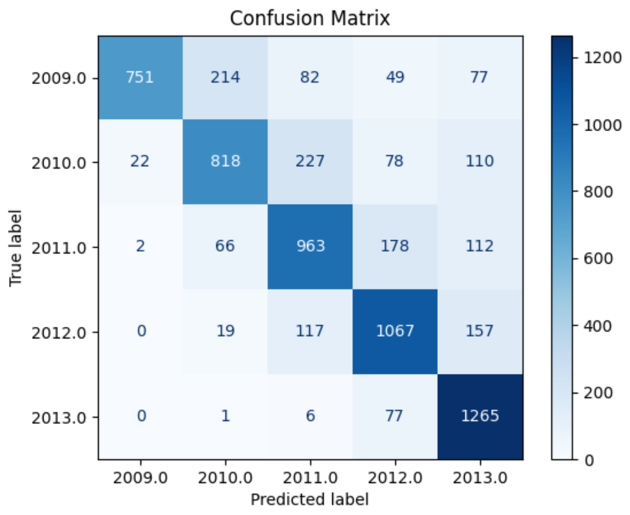

Figure 12: Confusion Matrix for Best Neural Network

5 Conclusion

Having tested multiple frameworks, it is evident that the neural network is superior. The dataset is constrained by legal issues; abiding by copyright regulations means that the amount of music published before the advent of huge open-source libraries of beats, riffs, and general music (which seems to have coincided with the rise of the mp3 player, Napster, and digital music writ large in the mid-2000s) is negligible. This affected each and every one of our decisions. We were forced either to bound our data to a small time span, or to train our models on data heavily skewed towards the mid-2000s and beyond. The difficulty here was compounded by the fact that the relation between these high-level musical features and the dates their songs were created is by no means obvious and not linearly separable by any means whatsoever. We have discovered that basic machine learning models struggle to relate features to labels.

This, we think, is why the neural network is superior. With multiple layers to form more complex relations between audio features and song dates, the neural network significantly outperforms all other models. Despite data limitations and the failure of standard machine learning methods, we still obtain a hopeful result for song dating. With more robust data and even stronger deep learning, who knows what accuracy could be reached. A finely-tuned model could sing in perfect pitch.

Link to code folder.